Redbooks
ibm.com/redbooks

Turning Data into Insight with IBM Machine Learning for z/OS

Samantha Buhler
Guanjun Cai
John Goodyear
Edrian Irizarry
Nora Kissari
Zhuo Ling
Nicholas Marion
Aleksandr Petrov
Junfei Shen
Wanting Wang
He Sheng Yang
Dai Yi
Xavier Yuen
Hao Zhang

IBM

Redbooks

International Technical Support Organization

Turning Data into Insight with IBM Machine Learning for z/OS

August 2018

SG24-8421-00

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (August 2018)

This edition applies to IBM Machine Learning for z/OS version 1.1.0.

Contents

Notices

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

AIX®	IBM Z®	WebSphere®
CICS®	Language Environment®	z Systems®
dashDB®	OMEGAMON®	z/OS®
DB2®	RACF®	z13®
Db2®	Redbooks®	z13s®
developerWorks®	Redbooks (logo) ®	zEnterprise®
IBM®	Tivoli®	
IBM Watson®	Watson™	

The following terms are trademarks of other companies:

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

The exponential growth in data over the last decade coupled with a drastic drop in cost of storage has enabled organizations to amass a large amount of data. This vast data becomes the new natural resource that these organizations must tap in to innovate and stay ahead of the competition, and they must do so in a secure environment that protects the data throughout its lifecycle and data access in real time at any time.

When it comes to security, nothing can rival IBM® Z, the multi-workload transactional platform that powers the core business processes of the majority of the Fortune 500 enterprises with unmatched security, availability, reliability, and scalability. With core transactions and data originating on IBM Z®, it simply makes sense for analytics to exist and run on the same platform.

For years, some businesses chose to move their sensitive data off IBM Z to platforms that include data lakes, Hadoop, and warehouses for analytics processing. However, the massive growth of digital data, the punishing cost of security exposures as well as the unprecedented demand for instant actionable intelligence from data in real time have convinced them to rethink that decision and, instead, embrace the strategy of data gravity for analytics. At the core of data gravity is the conviction that analytics must exist and run where the data resides. An IBM client eloquently compares this change in analytics strategy to a shift from “moving the ocean to the boat to moving the boat to the ocean,” where the boat is the analytics and the ocean is the data.

IBM respects and invests heavily on data gravity because it recognizes the tremendous benefits that data gravity can deliver to you, including reduced cost and minimized security risks. IBM Machine Learning for z/OS® is one of the offerings that decidedly move analytics to Z where your mission-critical data resides. In the inherently secure Z environment, your machine learning scoring services can co-exist with your transactional applications and data, supporting high throughput and minimizing response time while delivering consistent service level agreements (SLAs).

This book introduces Machine Learning for z/OS version 1.1.0 and describes its unique value proposition. It provides step-by-step guidance for you to get started with the program, including best practices for capacity planning, installation and configuration, administration and operation. Through a retail example, the book shows how you can use the versatile and intuitive web user interface to quickly train, build, evaluate, and deploy a model. Most importantly, it examines use cases across industries to illustrate how you can easily turn your massive data into valuable insights with Machine Learning for z/OS.

Authors

This book was written by a team of experts across geographies working on the IBM Machine Learning for z/OS project.

Samantha Buhler serves as an Offering Leader for the IBM analytics software portfolio on System Z. She has over 15 years of experience with software that runs on the IBM Z platform across many functions, including pricing and product management. She holds a Master of Business Administration degree from The College of William and Mary. Her areas of expertise include industry and market analysis, pricing, and go-to-market strategy for new offerings.

Guanjun Cai is an Information Architect for Machine Learning for z/OS and IBM Db2® for z/OS at IBM in the United States. He has 20 years of experience in designing and developing content for IBM analytics, enterprise storage, and z/OS products. He holds a Ph.D. in Rhetoric, Composition, and the Teaching of English from the University of Arizona. His areas of expertise include information architecture, development, and publishing as well as user experience design, markup languages, and automation.

John Goodyear is a Client Technical Specialist at the IBM Washington System Center in the United States. He has 25 years of experience in the computer industry. He holds a BA in Computer Science from University of Maryland. His areas of expertise include Machine Learning and Spark on z/OS, C, C++, Java, and Perl programming, Linux, IBM AIX®, and moderate z/OS administration.

Edrian Irizarry is a z/OS DevOps Architect in the United States. He has 3.5 years of experience in the field of software engineering. He holds a degree in Computer Science from the University of Rochester. His areas of expertise include Package Management, DevOps, and Automation.

Nora Kissari is a Data Science Specialist on the Z Advanced Technical Specialist Team for Europe, Middle-East, and Africa. She has 4 years of experience helping customers start new analytics projects. She holds a degree in Computer Science - Software Architecture and Engineering from the University of Montpellier in France. Her areas of expertise include designing data-centric architecture as well as hands-on data management, data processing, business intelligence and data modeling skills. She has lectured extensively on these topics at international events around the world.

Zhuo Ling is a senior data scientist and an IBM-certified information architect at the IBM China Development Lab and a lead data scientist on the MLz development team. She has over 20 years of experience in the field of data science. She holds a master degree in computer science from Peking University in China. Her areas of expertise include data analytics, data modeling and business intelligence solution architectural design. She has rich domain knowledge on banking, transportation and renewable energy, as well as IT operational analytics on z/OS.

Nicholas Marion is a Staff Software Engineer and Service Team Lead for Open Data Analytics in Poughkeepsie, NY. He has over 3 years of experience in the field of software design, development, and test. He holds a degree in Computer Science from the State University of New York at New Paltz. His areas of expertise include operating system development, development and test automation, and product support. He has previously had publications on IBM developerWorks®.

Aleksandr Petrov is a manager of the Data Science Elite (US) team, IBM Analytics, and specializes in Analytics platforms, Big Data, and Data Mining solutions. Over the years, he has delivered Analytics products that contain the implementation of ETL engine, machine learning, NLP, data visualization, and distributed algorithms implementation. Alexander has more than 20 years of industry experience working as a software architect, data scientist, and manager on creating software products and developing new technologies.

Junfei Shen is a data scientist in China. She has 1.5 years of experience in the field of data science. She holds a Master of Arts degree from Columbia University. Her areas of expertise include data science, machine learning, statistics, and finance.

Wanting Wang is a Data Scientist on the Data Science Elite Team in the United States. She has 2 years of experience working with clients to analyze data and derive business insights. She holds a degree in Quantitative Methods in the Social Sciences from Columbia University, with an interdisciplinary background in social science, statistics, and computer science. Her areas of expertise include statistics, machine learning, and data visualization.

He Sheng Yang is a Data Scientist and Enablement of IBM Analytics in China. He has 15 years of experience in the field of software engineering. He holds a degree in Computer Science from Beijing Institute of Technology. His areas of expertise include data science, solution architect, software enablement, project management in data analytics, healthcare industry, enterprise content management. He has written extensively for developerWorks and software engineering books.

Dai Yi is a Data Scientist in China. He has more than 5 years of experience in software development and data analysis. He holds a doctorate degree in Statistics and a master degree in Computer Science from the Renmin University of China. His areas of expertise include Software Engineer, Machine Learning, Statistics Learning, and Big Data.

Xavier Yuen is an Advisory Engineer in USA. He has 20 years of experience in Software Test. He holds a degree in Computer Science from USC, Sacramento.

Hao Zhang is a Machine Learning for z/OS Architect in China. He has 15 years of experience in the field of software engineering. He holds a master degree in Computer Science from the Chinese Academy of Sciences. His areas of expertise include Kubernetes installation and deployment. He has written extensively on installing and troubleshooting IBM Machine Learning for z/OS.

Acknowledgements

Thanks to the following people for their contributions to the book:

- Ke Wei Wei
- Deng Ke Zhao
- Shuang Yu
- Maggie Lin

Special thanks to Jane Dong for coordinating teams across geographies to ensure timely delivery of the book.

Thanks to Guanjun Cai for providing information architecture and design guidance for the book.

This book was produced by the IBM Redbooks® publication team with valuable contributions from Martin Keen, IBM Redbooks Project Leader.

Now you can become a published author, too!

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks
- Send your comments in an email to:

  redbooks@us.ibm.com
- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks
- Follow us on Twitter:

  http://twitter.com/ibmredbooks
- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm
- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Overview

This chapter introduces IBM Machine Learning for z/OS (MLz) in the context of industry trends in artificial intelligence (AI) that are enabling enterprises to embark on a cognitive journey with confidence. MLz brings advanced IBM predictive capabilities to the mission-critical IBM Z platform. These advanced capabilities enable enterprises to capitalize on actionable insights generated from their transaction processing and to do so without exposing their data in motion.

This chapter includes the following topics:

1.1 Challenges of exponential data growth

The past decade has seen two inversely proportionate trends in digital data growth and data storage cost. The amount of human and machine generated data has grown at an explosive rate, but the storage cost for the data is on a steep decline. In fact, the volume of digital data is so massive and the rate of its growth so fast that John Kelly, IBM SVP and Head of Cognitive and Research, calls this phenomenon the "third exponential curve."

The first exponential curve occurred in the 1960s when Intel co-founder Gordon Moore observed that the number of transistors per square inch on integrated circuits had doubled every year since their invention. He went on to predict that the trend would continue for the foreseeable future and revised the estimate in 1975 to doubling every two years. His prediction, which was proven to true, has become known as *Moore's Law*.[1] The pursuit of Moore's Law to essentially double transistors every 12 to 18 months completely changed the landscape for technology and industries.

The second exponential curve took place in the late 1990s to 2000s in the age of *Metcalfe's Law*, which is sometimes also referred as the *law of networking*. Governed by Metcalfe's Law, social networking sites, such as Facebook and Snapchat, become more valuable as they expand their user base.

The third exponential curve, as Kelly articulates, is occurring right now. Data in the digital world grows at a rate that doubles every 12 to 18 months.[2]

The preponderance of digital data from the exponential growth curves promises business opportunities in the data-driven economy, but it presents profound challenges for industries that seek ways to extract values from the data. The massive volume and the unprecedented rate force organizations to look beyond human competencies and to drive the increasing demand for technologies in artificial intelligence and cognitive systems.

1.2 Trends in artificial intelligence and cognitive systems

The term *artificial intelligence* (AI) has had many meanings over the years, but a common theme appears to be emerging. Forrester research likens AI as "a self-learning system that is able to interact with humans naturally, understand the environment, solve problems, and perform tasks" without the input of instructions or rules.[3]

The term *cognitive systems* implies the application of cognitive science to build systems that simulate human thought processes and behaviors. Kelly refers to cognitive systems as those that "learn at scale, reason with purpose and interact with humans naturally."[4]

Clear trends have emerged in the areas of artificial and cognitive intelligence. One rising trend is the sharp increase in investment in AI systems and the correlated upward growth rate of the industry. Businesses like IBM have heavily invested in the research and development of AI. The AI industry is expected to grow at an annual compound rate of over 50% through 2021, surpassing that of Public cloud services.

[1] Gordon E. Moore, 1975 IEEE Text Speech, "Progress in Digital Integrated Electronics."

[2] IBM's Management Presents at 21st Annual Credit Suisse Technology, Media & Telecom Conference. Transcript provided by http://www.SeekingAlpha.com

[3] Forrester Research, The Top Emerging Technologies to Watch: 2017-2021, https://ibm.northernlight.com/document.php?docid=IA20160913900000032&datasource=IBM

[4] John E Kelly. "Computing, cognition and the future of knowing: How humans and machines are forging a new age of understanding." https://www.research.ibm.com/software/IBMResearch/multimedia/Computing_Cognition_WhitePaper.pdf

The same study reveals another rising trend in the AI space. That is, while the demand for AI solutions comes from across the industries, nearly 60% of the market opportunities for cognitive systems in 2021 is expected to come from banking and financial management, healthcare and life sciences, and retail and consumer packaged goods. It is not surprising given that organizations across these three industries have a large amount of customer transaction data that can be leveraged to simultaneously grow top line revenue and minimize risks and costs.

Near-term use cases in the banking and financial management industry include customized program advisors and recommendations, fraud analysis and detection (including in-transaction scoring for retail credit card businesses), automated claims processing, and regulatory intelligence. The healthcare and life sciences industry holds the opportunities in the areas of diagnostics and treatment, pharma research and development, claims management, as well as personalized health services. In the retail and consumer packaged goods sector, the focus will be on providing customers with that “celebrity” experience through automated customer service agents, expert shopping advisors and recommendations, and Omni-channel marketing and merchandising.[5]

1.3 IBM’s approach to artificial intelligence and cognitive systems

IBM has a different prospective on artificial intelligence and guides its development under the principle of what CEO Ginni Rometty calls “augmented intelligence.” The difference is slight in terms of terminology but critical in terms of meanings. Rather than attempting to replicate all of human intelligence, IBM envisions cognitive systems powered by artificial intelligence (AI) to enhance and scale human expertise (Preparing for the Future of Artificial Intelligence).[6]

For this vision, IBM designs and builds a myriad of capabilities based on machine learning, reasoning and decision technologies, language, speech and vision technologies, human interface technologies, distributed and high-performance computing, and new computing architectures and devices. Specifically, IBM cognitive systems are envisioned to include the following core competencies:

- Deep understanding of domains
- Ability to reason towards specific goals
- Continuously learning from experiences
- Naturally interacting with cognitive agents or applications

These core competencies drive the research, design, and development of cognitive solutions in hardware, software, services, and applications. This is how IBM brings to you IBM Watson® and IBM Machine Learning for z/OS. Coupled with your own data, these solutions can serve as a launching pad for your business on its cognitive journey. They can help you solve a wide range of practical problems, boost productivity, and optimize your business decision-making. When purposefully integrated into your business strategy, these solutions can help you not only disrupt the competition, but also change the entire industry in which you compete similar to how Amazon has completely changed the landscape of retail businesses and how Uber has forever altered the rideshare industry.

[5] IBM Market Development & Insights Analysis, GMV, and IDC Cognitive Spending Guide
[6] IBM Research. Preparing for the Future of Artificial Intelligence, `http://research.ibm.com/cognitive-computing/ostp/rfi-response.shtml`

1.4 IBM Machine Learning for z/OS: An enterprise machine learning solution

Digital data can originate on premise or from private or public cloud. Most cognitive solutions on the market require that data be moved off their premise or private cloud, exposing it to potential breaches. IBM recognizes the imperative of allowing businesses to keep their sensitive, mission-critical data where they want it and provides solutions that bring cognitive capabilities to the data. MLz is the solution that enables you to run machine learning on the IBM Z platform while keeping mission-critical data where it is and using the existing Z capabilities.

MLz provides a complete enterprise grade platform and tooling for you to put your data to use through developing behavioral models that can accelerate and optimize business decisions. MLz can help your organization become much more agile with the ability to anticipate customer trends, minimize risks, and optimize production environments among other potential opportunities.

MLz includes, but is not limited to, the following key features:

- Community and project management feature to enable timely sharing of resources and facilitate close collaboration across teams.
- Support for multiple programming languages, including Scala, Python, and R to enable data scientists to use the languages of their choice.
- Integrated Jupyter Notebook to provide a programmatic approach to model development
- Visual Model Builder to provide a guided approach to model development without requiring extensive programming knowledge and skills.
- Model management dashboard to monitor the health and performance of deployed models.
- Administration dashboard for use by the system programmer to manage system configurations, set up and monitor scoring services, provision user access and privileges, and configure available kernels.

Together with other IBM cognitive technologies, including Db2 Analytics Accelerator for z/OS, MLz can help transform your Z platform into a highly-efficient, hybrid transaction, and analytic processing environment. It can build, deploy and manage behavioral models in real time by directly consuming data that is stored in Db2 for z/OS (or other sources) and transformed in the Accelerator. The data stays securely in the Accelerator as it is being ingested into MLz, removing the latency between data creation and transactions.

1.5 Unmatched capabilities of Machine Learning for z/OS

What sets MLz apart from other machine learning frameworks on the market is its unique ability to work directly with data that originates from IBM Z while keeping that data in place. MLz essentially provides IBM Z clients with a complete enterprise machine learning platform that leverages valuable business data to infuse mission-critical applications with intelligence. Specifically, MLz provides an enterprise grade platform with the following capabilities that are not matched by anything else on the market.

- Secure IBM Z platform for running machine learning with data in place
- Full lifecycle management of models
- Enterprise grade performance and high availability
- Flexible choice of machine learning languages

- Intuitive self-guided model development
- Developer-friendly API interfaces for applications on the Z platform

1.5.1 Secure IBM Z platform for running machine learning with data in place

Data gravity is at the core of big data analytics. Simply stated, data gravity means the movement of data at a large scale. Data movement increases the complexity, risk, and cost of managing the data. This is especially true for data that resides on IBM Z platform, the world's most secure and resilient environment. Moving data off the Z system introduces security risks and operational costs. MLz enables you to keep the sensitive data of your business in the secure Z environment while using the industry leading machine learning capabilities to extract actionable insights from the data.

MLz also enables you to use advanced proprietary cognitive technologies from IBM Research while seamlessly integrating with the most commonly used open source packages with which data scientists are familiar. For example, the technology of cognitive assistant for data scientists (CADS) helps data scientists select the best performing algorithms through an iterative approach applied to a select data set. When an algorithm has been selected, hyper parameter optimization (HPO) is applied to tune the model with the optimal parameters. This essentially condenses what used to take weeks into hours or days.

1.5.2 Full lifecycle management of models

Most machine learning frameworks or tools focus on the support for an extensive set of algorithms and model development. IBM recognizes that machine learning does not end with the creation of models. In fact, model development is just the beginning. A model that is not deployed and put into production essentially has no value.

As Figure 1-1 shows, the complete machine learning workflow starts from data preparation and include steps like feature engineering and model training before the model gets deployed for business applications to consume.

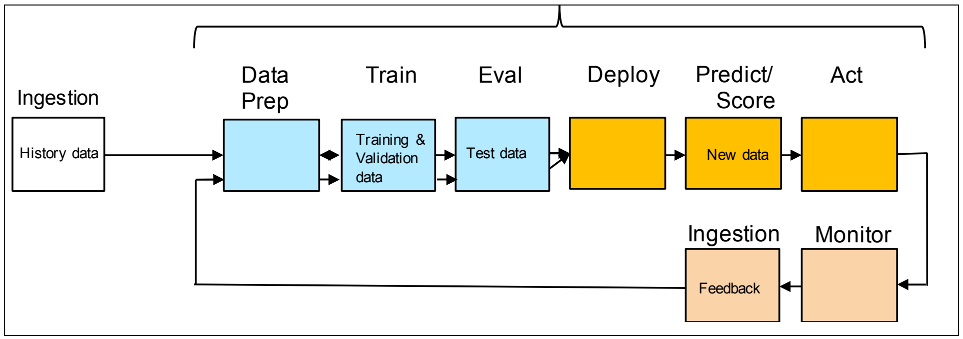


Figure 1-1 Machine learning workflow

The work flow does not end when the model is deployed either. The accuracy of the model may degrade over time with new data. The model needs to be continuously monitored and retrained based on data from live transactions so as to improve its performance or accuracy. This iterative MLz workflow forms a continuous learning loop (see Figure 1-2 on page 6).

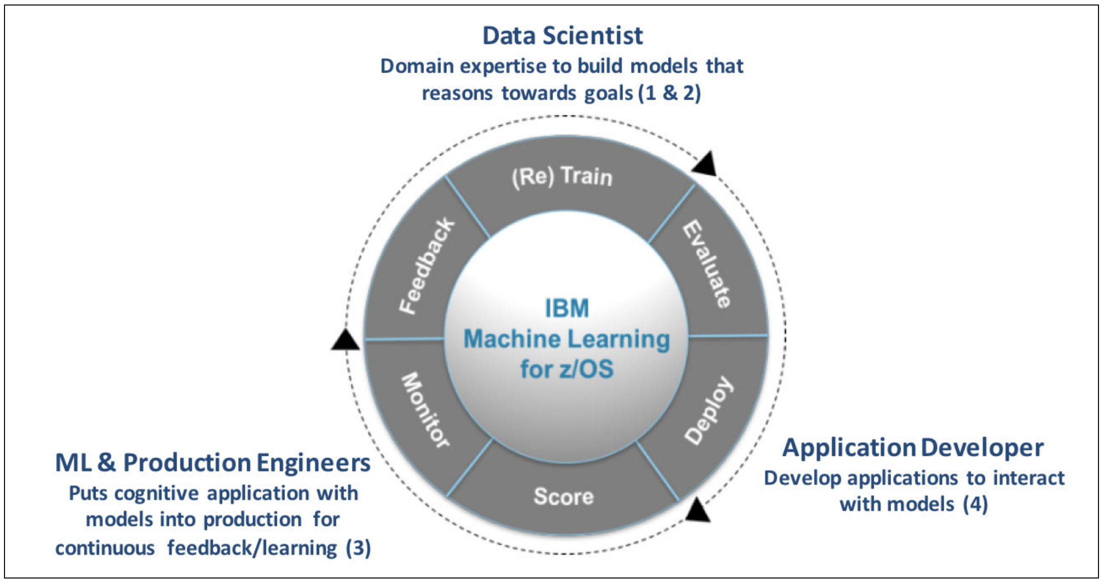

Data Scientist
Domain expertise to build models that reasons towards goals (1 & 2)
(Re) Train
Evaluate
Deploy
Score
Monitor
Feedback
IBM Machine Learning for z/OS
ML & Production Engineers
Puts cognitive application with models into production for continuous feedback/learning (3)
Application Developer
Develop applications to interact with models (4)

Figure 1-2 Continuous learning loop

The complete learning loop architecture of MLz makes possible the full lifecycle management of the model which enables close collaboration between members of your machine learning team in your organization. Data scientists can select data sources for model building, training, and deployment. Production engineers deploy the model into cognitive applications for production use. Application developers develop applications to interact with the model. Deployed models are used to make predictions. The models can be optimized as the data is pushed back into the machine learning workflow in a feedback loop.

1.5.3 Enterprise grade performance and high availability

The vast majority of applications running on IBM Z are processing and generating transactional data. High availability and real-time prediction with minimum impact to transaction processing are the key requirements for enterprise machine learning on IBM Z. MLz takes this requirement to heart in the architecture of the scoring service that runs natively on z/OS. Specifically, while using the advanced tooling that is available through open source, MLz also integrates tightly with IBM Z as follows:

- MLz takes full advantage of the Z sysplex technology. MLz supports scoring service clusters that reside on a single logical partition (LPAR) or span across multiple LPARs to ensure high availability of a standalone scoring service and minimize any downtime. MLz also leverages the Z dynamic VIPA and Shareport technologies in its design and architecture to ensure high availability of a scoring service cluster.
- MLz runs the online scoring service in IBM CICS® where most of data transaction and processing happen on Z. This flexibility ensures the scoring or prediction requests for various models are fast enough to meet service level agreements (or SLAs) for transaction processing. The scoring service can be deployed within a CICS region, where most of the transactions on Z are processed. Transactions running in CICS call scoring service directly through CICS LINK, which rids the latency of network communication.

1.5.4 Flexible choice of machine learning languages and scoring engines

IBM data science and machine learning strategy are built with the understanding that data scientists and users might have preferred programing languages or machine learning frameworks or tools. Similar to other IBM offerings for cognitive systems, MLz supports the vast majority of popular machine learning languages, including Python, Scala, R, and scoring engines, such as SparkML, Scikit-Learn, and PMML, with a development plan in place to include XGboost, LightGBM, H2O, Tensorflow, and Caffe in the near future. This flexibility to choose the language and framework with which data scientists and other users have experience makes MLz easy to consume.

1.5.5 Intuitive self-guided modeling capabilities

MLz provides a myriad of tools imbued with proprietary IBM technology to assist users with varying levels of data science and machine learning skills for model development. For example, Jupyter Notebook is integrated to provide advanced users who have strong programming skills a tool and framework they can use. A visual model builder is developed for beginner business users who have minimal or no programming skills. The visual builder is in essence a wizard that guides these users step by step to create models. For intermediate users, MLz plans to deliver a canvas tool which balances flexibility with ease of use. Users can easily drag and drop data processing elements to create a flow on the canvas and then generate machine learning models when running the flow.

1.5.6 Developer-friendly API interfaces for applications on the Z platform

In addition to RESTful APIs, MLz also provides developer-friendly API interfaces for applications, for example COBOL applications, that run on the Z platform. This capability drastically reduces the amount of effort that are otherwise required for application developers to use machine learning capabilities on the mainframe and allows organizations to derive benefits from applying predictive capabilities to transactions running on IBM Z.

1.6 Value proposition of Machine Learning for z/OS

MLz brings state-of-the-art machine learning technology behind the firewall on IBM Z, the world's most secure and resilient platform. Combined with your own data, MLz can help you achieve a competitive edge by creating behavior models and deriving actionable insights from the models for business decisions. Specifically, MLz provides the development and runtime environment as well as the end-to-end model lifecycle management capability that enable your organization to:

- Identify patterns from your data.
- Build models from those patterns that can be deployed and embedded in applications to predict behavioral outcomes.
- Manage the lifecycle of models to ensure their accuracy over time.

Insights derived from well performing behavior models can yield top line revenue growth while minimizing risks or costs. This benefit can be realized for all lines of businesses from a chat bot providing customized product recommendations to a unique prediction for each transaction to curtail fraud.

In summary, when purposefully integrated into your cognitive strategy and operations on IBM Z, MLz can deliver the following key benefits:

- Gain new insights from current and historical data on IBM Z systems.
- Combine insights from structured and non-structured data from Z and non-Z data sources.
- Deliver in-transaction predictive analytics capability with real-time data while keeping data in place and at the source.
- Minimize the latency, cost, and complexity of data movement.
- Maintain security and governance of data for analytics at a lower cost.
- Use the strength of IBM Z to provide the highest level of availability, reliability, scalability, and high-performance machine learning services.
- Significantly reduce and simplify development and deployment tasks based on IBM Z DevOps infrastructure, which enable mainframe developers and administrators to use machine learning with ease.
- Integrate closely with hardware and software that make up the IBM Z stack, including IBM Open Data Analytics for z/OS, Db2 for z/OS, Db2 Analytics Accelerator for z/OS, IBM WebSphere® Liberty Profile for z/OS, and CICS Transaction Server.
- Support interoperability across IBM Machine Learning and Data Science Experience offerings on Public Cloud, Private Cloud, x86, Power, and Linux on Z.
- Provide built-in solution templates for typical IBM Z mainframe use cases.
- Reduce costs by offloading machine learning workloads to IBM Z Integrated Information Processors (zIIPs). Workloads not eligible for zIIP offload, including training and scoring services for python models, qualify for Container Pricing for IBM Z (IBM United States Software Announcement, 217-519, dated 14 November 2017). You can find more detail about Container Pricing online.

2

Planning

Planning can both minimize costly errors during installation and shorten the lead time to get a new system up and running for production. Thorough planning is imperative to the successful installation of IBM Machine Learning for z/OS, which consists of integrated systems and services on different platforms. This chapter can help you develop a high-level, actionable plan that includes essential tasks, from obtaining product installers and allocating system capacity to provisioning network ports.

This chapter includes the following topics:

2.1 Product installers

Machine Learning for z/OS comes with installers or installation files for both z/OS and Linux or Linux on IBM Z (hereafter, *Linux on Z*) systems. Depending on your business need, choose one of the following combinations of system environment for installation:

- Installation on z/OS and Linux
- Installation on z/OS and Linux on Z

Your planning activity should start with obtaining all the required installation materials based on your decision.

Upon receiving your purchase order from IBM Shopz, verify that it includes the following materials:

- SMP/E for z/OS image
- Program Directory for IBM Machine Learning for z/OS
- License Information for IBM Machine Learning for z/OS DVD
- Accessing Machine Learning Services on Linux DVD
- All available maintenance packages

Maintenance packages are version-specific and are posted as they become available. Make sure that you have all the updates for the version of Machine Learning for z/OS that you install by checking the IBM Support Customer access portal for IBM Machine Learning for z/OS.

The SMP/E image and the maintenance packages, if any, are only part of the installation materials. You need to download the remaining installers and scripts from the IBM Web Membership (IWM) site.

The Accessing Machine Learning Services on Linux DVD includes a “Memo to Users.” The memo contains the product access key and the full URL to the IBM Web Membership site, where you will see the following installation files:

- `IBM_Machine_Learning_Installer_v1.1.0.5_Linux_x86-64` (for installation on Linux)
- `IBM_Machine_Learning_Installer_v1.1.0.5_Linux_s390x` (for installation on Linux on Z)
- `ITOA-Health-Tree-v1.1.0.5.tar` (for installation of ITOA Health Tree application on Linux)
- `ITOA-Health-Tree-v1.1.0.5-s390x.tar` (for installation of ITOA Health Tree application on Linux on Z)
- `iml_utilities-v1.1.0.5.tar` (for SSL certificate generation on Linux or Linux on Z)

Download the installation files for installing Machine Learning for z/OS in the system environments that you decided.

2.2 Hardware and software requirements

Machine Learning for z/OS uses both IBM proprietary and open source technologies and requires the installation of various hardware and software products in the z/OS and Linux or Linux on Z environments. Make sure that you procure all the prerequisite products for installation on the systems that you selected.

Important: Make sure that you install and configure the prerequisite products you select and acquire, with the exception of IBM Open Data Analytics for z/OS. The next chapter guides you through the installation and configuration of the Open Data Analytics for z/OS components.

2.2.1 Prerequisites for z/OS

The following hardware and software are required for installing Machine Learning for z/OS in the Z environment:

- z14, IBM z13®, or IBM zEnterprise® EC12 system
- z/OS 2.1 or later
- Db2 10 for z/OS (with APAR PI13725 applied) or later
- z/OS Integrated Cryptographic Service Facility (ICSF)
- IBM CICS Transaction Server for z/OS 5.4.0 (or 5.3.0 with APAR PI63005 applied)
- IBM Open Data Analytics for z/OS 1.1.0
- z/OS Spark 2.1.1 (FMID HSPK120)
- z/OS Anaconda (FMID HANA110)
- z/OS Mainframe Data Service 1.1 (FMID HMDS120)
- IBM Tivoli® Directory Server for z/OS LDAP
- IBM 64-bit SDK for z/OS, Java Technology Edition, v8 (with Refresh 4 Fix Pack 10) or later
- Gzip 1.6

CICS is required only if you want to install and run Machine Learning for z/OS scoring services in a CICS region. Also, be aware that Java 8 SR5 has a known issue with batch processing during the start-up, such as the start-master and start-slave process. To avoid the problem, plan to use Java 8 SR5 with FP7 or later.

2.2.2 Prerequisites for Linux

The following hardware and software are required for installing Machine Learning for z/OS in the Linux environment:

- Three x86 64-bit servers
- Red Hat Enterprise Linux Server 7.2 or later
- Open JDK 1.8.0 or later

2.2.3 Prerequisites for Linux on Z

The following hardware and software are required for installing Machine Learning for z/OS in the Linux on Z environment:

- Three s390x 64-bit server that runs on an LPAR of a z14, z13, IBM z13s®, zEnterprise EC12, zEnterprise BC12, LinuxOne Emperor, or LinuxOne Rockhopper system
- Red Hat 7.2 or later
- Open JDK 1.8.0 or later

2.3 System capacity

The correct system capacity for the correct workload is quintessential to maximize the value of Machine Learning for z/OS. A workload is typically defined by the number of concurrent model creation jobs run by the Jupyter Notebook or the Machine Learning for z/OS visual model builder, the size of modeling training data set (in GB), and the number of model training data features. Each model creation job includes the tasks for data loading, data transformation, data visualization, feature extraction, feature transformation, model training, and model evaluation. Carefully plan adequate system capacity in hardware, processing power, and disk storage based on the anticipated needs of your enterprise workload.

2.3.1 Basic system capacity

The scoring and training services of Machine Learning for z/OS run on z/OS, and its management services, user interface, and administration dashboard run on Linux or Linux on Z. These services require a minimum of system capacity. Although you can use the basic system capacity to run any reasonable workload, the rule of thumb is that the heavier the workload is, the more capacity you need to allocate.

If you choose the combination of z/OS and Linux for installation, ensure that the systems have the basic capacity listed in Table 2-1.

Table 2-1 Basic system capacity for installation on z/OS and Linux

Hardware	Number of LPAR/Server	CPU (Per LPAR/Server)	Memory (GB) (Per LPAR/Server)	DASD/Disk Space (GB) (Per LPAR/server)
IBM z Systems®	1 LPAR	4 zIIP processors, 1 general purpose processor	100	50
Linux system	3 x86 64-bit servers	8 cores	48	250 (plus a minimum of 650 GB secondary storage for each server)

If you choose the combination of z/OS and Linux on Z for installation, ensure that the systems have the basic capacity listed in Table 2-2.

Table 2-2 Basic system capacity for installation on z/OS and Linux on Z

Hardware	Number of LPAR/Server	CPU (Per LPAR/Server)	Memory (GB) (Per LPAR/Server)	DASD/Disk Space (GB) (Per LPAR/server)
z Systems	1 LPAR	4 zIIP processors, 1 general purpose processor	100	50
Linux on Z system	3 s390x 64-bit servers	3 IFL processors	48	250 (plus a minimum of 650 GB secondary storage for each server)

For best performance in either installation scenario, consider dedicating the LPAR for Machine Learning for z/OS. Also, allocate a secondary storage to each Linux or Linux on Z server and configure it with two mount points in XFS format with the `ftype` option enabled. Make sure that one mount point is appropriated a minimum of 300 GB for installer files and the other a minimum of 350 GB for data storage.

2.3.2 Capacity considerations for training services

Machine learning models are trained with data and algorithms. In general, model training is CPU intensive and can consume most of the CPU available on a given LPAR. The heavier the training workload is, the more CPU is needed. So, allocate enough processors based on your projected workload for the LPAR where Machine Learning for z/OS training services run.

The type of models and algorithms also affects the type of processors you need for model training on z/OS. For example, Spark and MLeap models are typically trained on zIIP processors, and Scikit-learn models are processed primarily on the general processors. Increase the number of processors to process the type of models you build and the type of algorithms you plan to use.

Last but not least, the size of the training data itself constitutes a significant factor in memory usage. Results of repeated tests indicate that memory usage tends to be two to three times of the size of the training data, and that number bumps up when training jobs are executed concurrently. Therefore, the preferred practice is to allocate adequate memory based on both the size of your training data and the number of concurrent training jobs.

2.3.3 Capacity considerations for scoring services

Machine Learning for z/OS processes scoring requests on different processors depending on the type of models. For example, while scoring requests for Scikit-learn models are processed on the general processor, those for Spark, MLeap, and PMML models are handled on zIIP processors. So, take into account the type of models that you develop and allocate the appropriate type of processors for the LPAR where scoring services run.

If high availability is essential to your business, consider using a scoring service cluster. In such a cluster, multiple instances of a scoring service share the same URI, with each running on a different LPAR of a sysplex. The cluster uses a round-robin algorithm of the sysplex distributor (SD) to dispatch scoring requests across the LPARs. The cluster processes all the scoring requests as long as one of the LPARs is operational.

2.3.4 Capacity considerations for performance

System response time is a key performance indicator in machine learning operations. The response time of Machine Learning for z/OS services generally corresponds to the availability of system capacity, as evidenced by test results in the following example (see Table 2-3).

Table 2-3 System response time corresponds to availability of system capacity

Processors	System response time (minutes)					
	Number of concurrent model creation jobs				Size of data set (GB)	Number of data features
	8	16	32	64		
4 zIIP 1 GCP	13	27	56	135	2	100
	21	46	94	238	4	200

8 zIIP 1 GCP	9	15	31	68	2	100
	14	21	44	101	4	200

In this example, if four zIIPs and one GCP are allocated to run the workload of 16 concurrent jobs, one training data set of 2 GB in size, and 100 data features, it can take the system up to 27 minutes to complete the jobs. If eight zIIPs are allocated for the same workload, system response time can be reduced to 15 minutes. Although the actual results of your system performance can vary, the example demonstrates the positive correlation between system capacity and system response time. In other words, adequate allocation of system capacity improves system response time when the same or similar workload is being processed. Consider increasing your system capacity to improve the response time and thus the overall performance of Machine Learning for z/OS services.

2.4 Installation options on z/OS

Machine Learning for z/OS offers flexibility in terms of where to install the training and scoring services on z/OS. Depending on your business need and the system capacity you plan to allocate, carefully assess the following installation options and choose one that satisfies your machine learning workload while achieving the best performance without exceeding system capacity:

- Training and scoring services on the same LPAR
- Training and scoring services on different LPARs
- Training services on an LPAR and scoring services on an LPAR cluster

Machine Learning for z/OS uses z/OS Mainframe Data Service (MDS) as both a data connector and a data source. MDS must be on the same LPAR where the machine learning training and scoring services run. If you use MDS in your setup, make sure that MDS and the training and scoring services are installed on the same LPAR or sysplex.

2.4.1 Option 1: Training and scoring services on the same LPAR

This option suggests the installation of both training and scoring services on a single LPAR that is dedicated to Machine Learning for z/OS. The sysplex in Figure 2-1 on page 15 includes multiple LPARs with one handing exclusively machine learning workload and others executing existing applications for production. At run time, data is ingested from one or more production systems to the dedicated LPAR for training and scoring services.

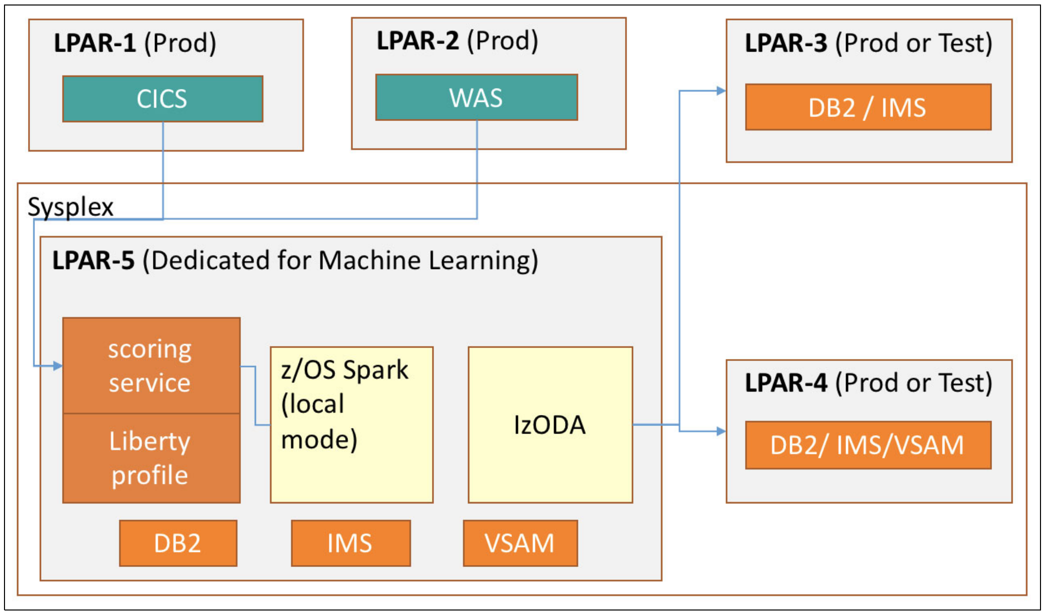


Figure 2-1 Installing training and scoring services on the same LPAR

There are several advantages to this option. The installation is straightforward with all the machine learning component systems and services going to the same location on z/OS. Also, the option does not impact the performance of the existing production systems in the sysplex. Most importantly, with careful workload balancing for training and scoring requests, the services can share and maximize the use of system resources for better performance.

The disadvantage of this option is the potentially negative impact on the performance of scoring services. Both data ingestion for training and scoring requests come from other production systems, and heavy network traffic between the LPARs might slow down the responses of those services. Consider this option if your machine learning workload is not heavy and if you want to keep the production systems for machine learning and other applications separate.

2.4.2 Option 2: Training and scoring services on different LPARs

This option suggests the installation of machine learning training and scoring services on separate LPARs with existing production systems. Figure 2-2 shows an example of this installation option.

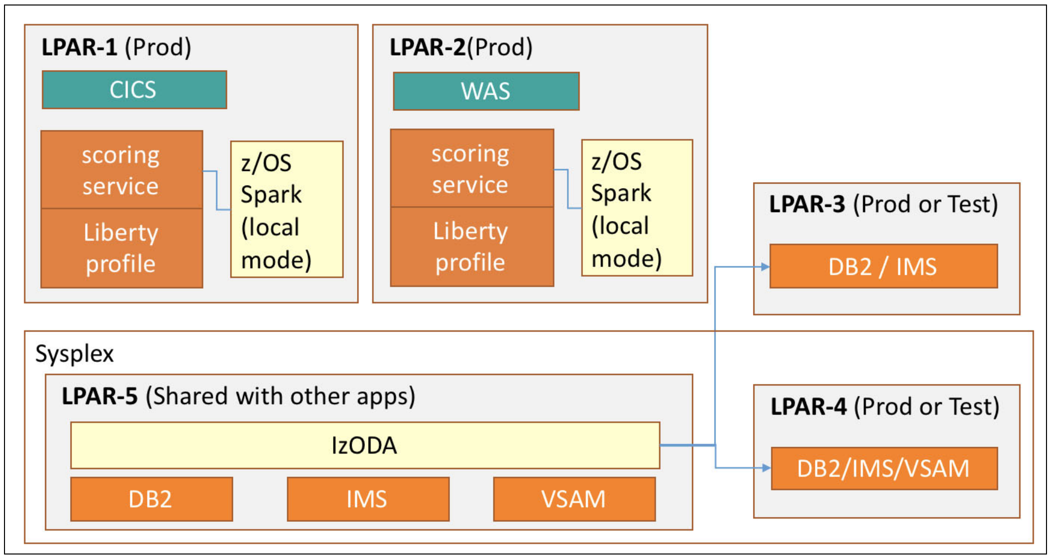


Figure 2-2 Figure 2: Installing training and scoring services on different LPARs

In this example, the installation of training and scoring services spreads across three different LPARs in the sysplex. All of them coexist with other applications on their respective production systems where data lives. The training services run on the same LPAR along with Db2, IMS, or VSAM which holds the data for model training. The scoring services run on the same LPARs where scoring requests originate and can respond to those requests with minimal performance impact.

This installation option addresses the shortcomings in the first option. The biggest upside is that it uses and optimizes the use of existing system resources on each LPAR while eliminating the potential performance impact due to heavy network traffic. Consider the option particularly when fast elapsed time for both scoring and training services is essential to the operation of your production systems.

2.4.3 Option 3: Training services that are on an LPAR and scoring services that are on an LPAR cluster

This option is similar to the second one in terms of installing the training and scoring services on separate LPARs. The difference, which is significant, lies in the suggestion that the scoring services be installed on an LPAR cluster. Figure 2-3 on page 17 shows the layout of this installation option.

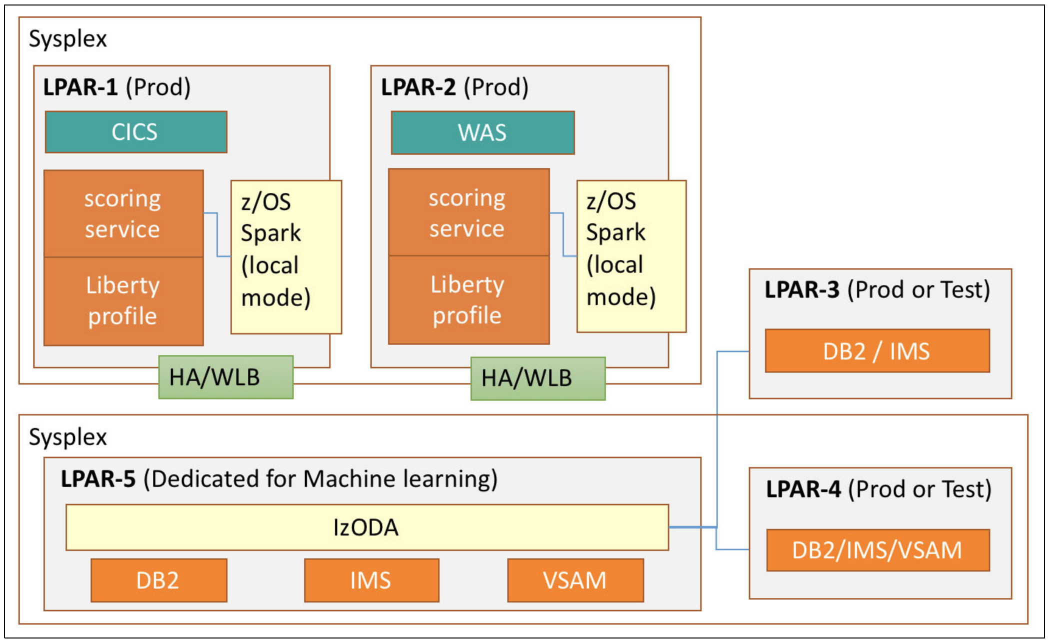


Figure 2-3 Installing training services on an LPAR and scoring services on an LPAR cluster

In this example, the training services run on a dedicated LPAR in a sysplex, and the scoring services are installed on multiple LPARs in another sysplex which is configured as a scoring service cluster. The sysplex distributor (SD) is used to balance and distribute scoring workloads among multiple instances of scoring services in the cluster. All scoring requests are processed as long as one LPAR in the cluster is up and running. This option delivers high availability and scalability of Machine Learning for z/OS services. Consider this option if your machine learning workload is significantly heavy and high availability and stability are top priorities of your business.

2.5 User IDs and permissions

The Linux or Linux on Z installer of Machine Learning for z/OS uses the default user of each node to install component systems and services but requires user-defined IDs with proper permissions for installation on z/OS. Dedicated user IDs are also required for Machine Learning for z/OS to access Db2 for z/OS and z/OS LDAP with the SDBM backend. Make sure that you identify or create all the required IDs and assign them sufficient privileges, as listed in Table 2-4, before you start the installation.

Table 2-4 User IDs and permissions required for installing Machine Learning for z/OS

Type of User ID	Description	Required Privileges or Permission
Db2 for z/OS authorization (<*db2_auth_id*>)	This authorization ID is used by the Machine Learning services to access Db2 for z/OS.	`DBADM` authority, which is granted when you run the ALNMLEN sample JCL job
z/OS LDAP user ID (<*zldap_userid*>)	This user ID is used by the Machine Learning services to access z/OS LDAP.	`RACF SPECIAL` authority for validating a new user that you want to add

z/OS Spark, Jupyter kernel gateway, Apache Toree, and MLz operation handling service user ID (<*spark_jupyter_toree_userid*>)	This user ID is used for installing and configuring z/OS Spark, Jupyter kernel gateway, and Apache Toree. This ID is also used for creating, configuring, and starting the operation handling service on z/OS.	► Member of IBM RACF® user group <*spark-GRP*>.
		► `$SPARK_HOME and $SPARK_OPTS` (`$SPARK_OPTS="–master spark://<ip_address>:<port>"`) Environment variables included in the user's profile (`$HOME/.profile`)
		► `$IML_HOME` environment variable included in the user's profile, which points to <*install_dir_zos*>.
		► Inclusion of the following environment variables in the user's profile:
		`export ANACONDA_ROOT="<install_dir_anaconda>"`
		`export PATH=$ANACONDA_ROOT/bin:$PATH`
		`export PYTHONHOME=$ANACONDA_ROOT`
		`export FFI_LIB=$PYTHONHOME/lib/ffi`
		`export LIBPATH=$PYTHONHOME/lib:$LIBPATH`
		► Permission to read and write to `<install_dir_zos>/configuration` and subdirectories
		► Permission to read and write to `<install_dir_zos>/iml-library/tmp`
		► Permission to read and write to `<install_dir_zos>/imlpython` and subdirectories
		► Permission to read and write to `<install_dir_zos>/ophandling` and subdirectories
		► Permission to write to `<install_dir_zos>/iml-library/output`
		► Permission to read `<install_dir_zos>/iml-library`
		► Permission to read `<install_dir_zos>/iml-library/brunel`
		► Permission to write to `<install_dir_anaconda>`

CICS region owner or user ID (<*cics_region_userid*>)	This user ID is used to start and run the scoring service in a CICS region.	► Permission to read and write to `<install_dir_zos>/cics-scoring` and subdirectories ► `<JVMPROFILEDIR>/ALNSCSER.jvmprofile`
Machine Learning scoring service user ID (<*mlz_scoring_userid*>)	This user ID is used for installing and configuring the scoring service and for starting the service servers.	► Member of RACF user group <*spark-GRP*> ► `$SPARK_HOME` and `$SPARK_CONF_DIR` environment variables included in the user's profile ► `$PYTHONHOME` environment variable included in the user profile ► `$JAVA_HOME/bin` defined in the `$PATH` environment variable in the user's profile ► READ access to `BPX.FILEATTR.APF` and `BPX.FILEATTR.PROGCTL` facilities ► Permission to write to `<install_dir_zos>`

For ease of installation and post-installation access control, consider using the same user ID for installing Machine Learning for z/OS operation handling services, z/OS Spark, z/OS Anaconda, Jupyter kernel gateway, and Apache Toree. If you prefer to use different IDs, consider applying the same naming convention, such as `MLZ(TYPE)`, to create MLZSPARK, MLZLDAP, and MLZSCORS. The naming convention helps make it easier to administer and monitor the activities of these IDs.

2.6 Networks, ports, and firewall configuration

Machine Learning for z/OS implements SSL/TLS protocols to secure network communications across component systems and uses Kubernetes to manage security policies in a cluster. The networks use dedicated ports, some of which are predefined. Make sure that you reserve the required ports for Machine Learning for z/OS and configure your network firewall accordingly.

2.6.1 Network requirements

The Linux or Linux on Z installer sets up a Kubernetes cluster. The cluster is configured to provide high availability to Machine Learning for z/OS services, including the primary web user interface and the administration dashboard. Make sure that you meet the following network requirements for this cluster:

- All nodes in the cluster run in the same subnet, with each assigned a private static IP address.
- Each node is associated with a gateway within the subnet, regardless whether or not the gateway allows outbound network access.
- The subnet itself is assigned a private static IP address that is to be used as a proxy server address. The IP address must be offline during the installation.
- The SELinux module on each node is set to “permissive” or “enforcing” (`SELINUX=permissive` or `SELINUX=enforcing`) in the `/etc/selinux/config` file. Restart the node after any change to the setting.
- The cluster requires two unique IP ranges in CIDR format, one to be used by the Kubernetes service network and the other by the cluster overlay network.
  - Kubernetes service network: A Kubernetes service is an abstraction which defines a logical set of pods and a policy. It redirects the network traffic to each of the pods at the service's backend. Kubernetes manages the IP range and assigns an IP address to each service. You need to assign an IP range for the Kubernetes service network.
  - Cluster overlay network: A pod is the basic building block of Kubernetes, which encapsulates an application container. Kubernetes relies on an overlay network to manage how groups of pods are allowed to communicate with each other and other endpoints. You need to assign an IP range for the cluster overlay network.

Make sure that the IP ranges are represented by a CIDR notation. CIDR specifies an IP address range by the combination of an IP address and its associated network mask. Take the range of 192.168.0.0/16 as an example. Although 192.168.0.0 is the network IPv4 address itself, the number 16 indicates that the first 16 bits are the network part of the address, and the remaining 16 bits are for host addresses. If the subnet mask is 255.255.0.0, the range can start from 192.168.0.0 to 192.168.255.255.

Carefully select the required IP ranges. The ranges must not overlap with each other. The IP addresses in the ranges must not conflict with those used by the Machine Learning for z/OS proxy server or your local networks.

Table 2-5 shows an example for selecting an internal IP range.

Table 2-5 Example of internal IP ranges

	Host Network/IP	Cluster Overlay Network	Kubernetes Service Network
Host has a single IP	172.16.x.x	192.168.0.0/16	10.0.0.0/16
Host IP conflicts with the overlay network default	192.168.x.x	172.16.0.0/16	10.0.0.0/16
Host has more than one IP address	192.168.x.x, 10.3.x.x	172.16.0.0/16	172.17.0.0/16

2.6.2 Ports

Machine Learning for z/OS requires dedicated ports for network communication across component systems and services. Some ports are predefined, and others can be user defined. Make sure that you configure the required ports and open them in your firewall, as listed in Table 2-6.

Table 2-6 Ports for systems and services on z/OS and Linux or Linux on Z

System or Service	Port Number	Outbound	Inbound	Note
Db2 for z/OS	User defined	Linux or Linux on Z system	Db2 subsystem	The assignment of this port depends on your Db2 configuration.
LDAP	User defined default: 636	Linux or Linux on Z system	z/OS system	
z/OS Spark Master	User defined default: 7077	Linux or Linux on Z system	z/OS system	
z/OS Spark Master REST API	User defined default: 6066	Linux or Linux on Z system	z/OS system	
Operation Handling Service	User defined default: 10080	Linux system	z/OS system	
Scoring Service	User defined	Linux or Linux on Z system	Liberty Profile for z/OS system	The assignment of this port depends on the configuration of the Liberty Profile server and the scoring service by default.
Jupyter kernel gateway	1 user defined default: 8889	Linux or Linux on Z system	Apache Toree kernel	
Apache Toree kernel	User defined (A range of port numbers in consecutive order)	None	z/OS system	Each Toree kernel must be assigned 5 port numbers in consecutive order. All port numbers in the range must be in consecutive order. Example: If you use eight Toree kernels in your setup, you must prepare a total of 40 ports starting from the first port number.

Repository service	12501	Linux system, Liberty Profile for z/OS system	Linux system	
Deployment service	14150	Linux system, Liberty Profile for z/OS system, Python run time for z/OS	Linux system	
Batch scoring service	12200	Linux system, z/OS Spark system	Linux system	
RabbitMQ service	5671, 5672	Linux system	Linux system	
Kubernetes ETCD	2379	Linux system	Linux system	
Feedback service	14350	Linux system	Linux system	
Ingestion service	13100	Linux system	Linux system	
Pipeline service	13300	Linux system	Linux system	
Machine Learning for z/OS UI	443	Your network	Linux system	

2.7 Firewall configuration

Instead of a traditional server firewall, Kubernetes uses IP tables for cluster communication. So, disable the cluster firewall. If an extra firewall must be in place, set it up around the cluster, and open the ports in your local network that need to interact with the cluster, such as port 443 for web access.

Ensure that every node in the cluster has a single local host entry in the `/etc/hosts` file that corresponds to the 127.0.0.1 address. Do not allow any daemon or script process or any `cron` job to modify the hosts file, IP tables, routing rules, or firewall settings during or after the installation.

3

Installation and customization

This chapter guides you through the installation and configuration of Machine Learning for z/OS and the prerequisite IBM Open Data Analytics for z/OS (IzODA). It also shows you how to configure for security, high availability, and scalability. Ultimately, the step-by-step instructions help you get Machine Learning for z/OS up and running, ready for use.

Important: The discussion in this chapter assumes that you already installed and configured all the prerequisite products, except IzODA, as described in Chapter 2, “Planning” on page 9. Do not proceed until you complete the installation and configuration of the prerequisites for the system environments that you selected.

This chapter includes the following topics:

3.1 Installation roadmap

Machine Learning for z/OS consists of integrated component systems and services that run on different platforms. The installation and configuration involve multiple sequences of tasks that might be performed by people in different roles, each with a unique set of skills and authorities. The roles of z/OS system programmer and Linux or Linux on Z system administrator are required. While the roles of database administrator, security administrator, network administrator, and UNIX shell programmer are optional, their skills and knowledge are much wanted.

Given the number of tasks and roles that are involved, close collaboration is key to successfully getting Machine Learning for z/OS up and running. Use the high-level roadmap that is shown in Table 3-1 to coordinate, organize, and track all installation and configuration tasks.

Table 3-1 Planning checklist for a first-time installation

√	**Task**	**IT Role / Skills**
◊	Planning installation options on z/OS	z/OS system administrator
◊	Creating user IDs and assigning permissions	z/OS system administrator
◊	Configuring network, ports, and firewalls	Network administrator or engineer
◊	Procuring, installing, and configuring prerequisite products, except IzODA	z/OS, Linux, and Linux on Z system administrator or programmer
◊	Installing IzODA	z/OS system programmer with UNIX skills
◊	Configuring IzODA	z/OS system programmer with UNIX skills
◊	Verifying IzODA installation and configuration	z/OS system programmer with UNIX skills
◊	Authoring and authenticating users	z/OS system programmer with UNIX skills, security administrator
◊	Creating, distributing, and installing an SSL certificate	z/OS system programmer with UNIX skills, Linux or Linux on Z system administrator, security administrator
◊	Configuring LDAP with SDBM for user authentication	z/OS system programmer with UNIX skills, security administrator
◊	Installing machine learning services on z/OS	z/OS system programmer with UNIX skills
◊	Installing machine learning scoring service in a CICS region	z/OS system programmer with UNIX skills
◊	Installing machine learning services on Linux or Linux on Z	Linux or Linux on Z system administrator
◊	Configuring TCP/IP for port sharing and load balancing	z/OS system programmer with UNIX skills
◊	Configuring an application cluster with extra compute nodes	Machine Learning for z/OS administrator

3.2 Installing and configuring IzODA

IzODA 1.1.0 consists of z/OS Spark 2.1.1, z/OS Anaconda and Python 3.6.1, and z/OS MDS 1.1.0. Machine Learning for z/OS uses the high-performance, general execution engine technology of z/OS Spark. Built on Apache Spark, z/OS Spark can perform large-scale data processing and in-memory computing.

Machine Learning for z/OS also takes full advantage of z/OS Anaconda that includes various Python packages and provides data scientists with a comprehensive machine learning solution. You must install and configure Spark and Anaconda before you install Machine Learning for z/OS

z/OS MDS provides integration facilities for IBM Z data sources and other off-platform data sources. As a data connector, MDS provides Spark applications with optimized, virtualized, and parallelized access to varied data sources. Install MDS only if you use it as a data source, a data connector, or both. This section assumes that you want to install MDS.

3.2.1 Installing IzODA

Complete the following steps to install IzODA:

1. Locate the SMP/E image, the Preventive Service Planning (PSP) bucket, and the *Program Directory for IBM Open Data Analytics for z/OS* (5655-OD1).
2. Follow the instructions in the *Program Directory* to install any IzODA prerequisite.
3. Run the SMP/E program and the sample JCL jobs to install Spark, Anaconda, and MDS. For more information, see *IBM Open Data Analytics for z/OS Installation and Customization Guide*.
4. Update the permissions of the IzODA installation directories. When the sample jobs are done, Spark, Anaconda, and MDS are installed. However, they are not ready for use yet. You must update the permissions of the IzODA installation directory and assign the directory ownership to user <*spark_jupyter_toree_userid*> and associated user group.

   Attention: For the ease of reference, the remaining instructions use the following user ID, user group, and directories for installing and configuring IzODA:

   - `MLZSPARK` is user `<spark_jupyter_toree_userid>`
   - `IZODADEV` is the user group that includes MLZSPARK
   - `/usr/lpp/IBM/izoda` is the root IzODA installation directory
   - `/usr/lpp/IBM/izoda/spark` is the Spark installation directory
   - `/usr/lpp/IBM/izoda/anaconda` is the Anaconda installation directory

   Complete the following steps:

   a. Log in to OMVS with the authority to change directory ownership.

   b. Issue the **`chown`** command, as shown in the following example, to make `MLZSPARK` the new owner of the `izoda` directory:

   ```
   chown -R MLZSPARK:IZODADEV /usr/lpp/IBM/izoda
   ```

   c. Verify that the `MLZSPARK` is the new owner by issuing the **`ls`** command. If the change occurs, you should see an output similar to the output that is shown in Example 3-1 on page 26.

Example 3-1 ls command

```
TUSER01@BJC08:> ls -la /usr/lpp/IBM/izoda
total 64
drwxr-xr-x   4 MLZSPARK  IZODADEV     8192 Dec  6 10:01 .
drwxr-xr-x   8 MLZSPARK  IZODADEV     8192 Dec  6 10:01 ..
drwxr-xr-x  15 MLZSPARK  IZODADEV     8192 Dec  6 10:01 anaconda
drwxr-xr-x   5 MLZSPARK  IZODADEV     8192 Dec  6 10:01 spark
```

5. Ensure that the `env` program is in the correct path and provides a correct list of the environment variables. Spark shell scripts require that the `env` program runs in the `/usr/bin` directory.

   a. Issue the following command to check whether `/usr/bin/env` exists and if yes, correctly lists all the environment variables:

      `/usr/bin/env`

      If `/usr/bin/env` exists, you should see all the environment variables, some of which are visible in Example 3-2.

      Example 3-2 Environment variables

      ```
      TUSER01@BJC08:> /usr/bin/env
      _BPX_SHAREAS=
      _BPXK_AUTOCVT=
      IBM_JAVA_OPTIONS=
      STEPLIB=
      ANACONDA_HOME=/usr/lpp/IBM/izoda/anaconda
      SPARK_HOME=/usr/lpp/IBM/izoda/spark
      ...
      ```

      If `/usr/bin/env` does not exist, check to see where the `env` program is installed on your system. One common location is the `/bin` directory.

   b. Assuming that the `env` program is in the `/bin/env` directory on your system, create a symbolic link from `/usr/bin/env to /bin/env` by issuing the following command:

      `ln -s /bin/env /usr/bin/env`

      If the symlink is successfully created, you should see the message that is shown in Example 3-3 when you issue the **`ls -la /usr/bin/env`** command.

      Example 3-3 env command

      ```
      TUSER01@BJC08:> ls -la /usr/bin/env
      lrwxrwxrwx   1 ADFAF  1 8 Dec  4 14:13 /usr/bin/env -> /bin/env
      ```

   c. Issue the **`/usr/bin/env`** command again to confirm that `/usr/bin/env` is successfully resolved to `/bin/env`. The command should return the same list of environment and value pairs that are contained in `/bin/env`.

Leading practice: Depending on how `/usr/bin` is configured on your system, the symlink for `/usr/bin/env` might not persist across IPL sessions. In that case, ensure that you add the creation of this symlink to your IPL setup.

3.2.2 Configuring IzODA

Before you can use IzODA, customize your system environment variables based on the installation. Some of the customizations are common to z/OS Spark, Anaconda, and MDS, while others are unique to each.

Complete the following steps to configure IzODA:

1. Add IzODA configuration settings to your user profiles. When you log on to UNIX System Services, the system automatically loads and applies the environment settings that are specified in the following files:

   – `/etc/profile` sets system-wide environment variables on shells for any user.

   – `$HOME/.profile` sets or changes the values of environment variables for an individual user with a `.profile` in the `$HOME` directory. The file is loaded when the user logs in.

   – `$HOME/.bashrc` starts an interactive shell session. The file is loaded when an individual user with a `.bashrc` in the `$HOME` directory starts the bash shell interactively.

   The files are loaded in the order of `/etc/profile`, `$HOME/.profile`, and `$HOME/.bashrc`. If you set the default shell to bash, all three files are loaded when you log in. Otherwise, the `.bashrc` file does not run until you start a bash shell.

   To make it easier to set up and use Spark, Anaconda, and MDS with Machine Learning for z/OS, add the environment variables that are shown in Example 3-4 to your user-specific `.profile` or `.bashrc` file or your global `/etc/profile` file.

Example 3-4 Sample configuration file

```
# Machine Learning for z/OS configuration
export IML_HOME=/usr/mlz_install/mlz_services
export IML_INSTALL=/usr/mlz_install/
# IBM Open Data Analytics for z/OS Spark configuration
export JAVA_HOME="/usr/lpp/java/J8.0_64"
export IBM_JAVA_OPTIONS="-Dfile.encoding=UTF8"
export SPARK_HOME="/usr/lpp/IBM/izoda/spark/spark211"
export SPARK_CONF_DIR="/etc/spark/conf"
export SPARK_MASTER_PORT=7077
export SPARK_LOCAL_IP=""
export SPARK_WORKER_DIR="/var/spark2/work"
export SPARK_LOCAL_DIRS="/tmp/spark2/scratch"
export SPARK_LOG_DIR="/var/spark2/logs"
export SPARK_PID_DIR="/tmp/spark2/pid"
# IBM Open Data Analytics for z/OS Anaconda and Python configuration
export ANACONDA_HOME="/usr/lpp/IBM/izoda/anaconda"
export PYTHON_HOME="$ANACONDA_HOME"
export FFI_LIB="$PYTHON_HOME/lib/ffi"
export LIBPATH="$PYTHON_HOME/lib:$LIBPATH"
# IBM Open Data Analytics for z/OS MDS configuration
export STEPLIB=hlq.SAZKLOAD:$STEPLIB
# IBM Open Data Analytics for z/OS common configuration
export _BPXK_AUTOCVT="ON"
export _BPX_SHAREAS="NO"
export PATH="$ANACONDA_HOME/bin:$JAVA_HOME/bin:$PATH"
# IBM Open Data Analytics for z/OS optional configuration
export TERM="xterm"
export _CEE_RUNOPTS="FILETAG(AUTOCVT,AUTOTAG) POSIX(ON)"
```

Leading practice: Place the sample configuration file in an area, such as `/etc/spark`, that can be accessed by all users.

The following environment variables are included in the sample configuration file:

– `_BPX_SHAREAS` specifies whether the created child process is run in a separate address space from the login shell's address space or in the same address space. The default is `YES`. Do not specify the variable if your login shell is tcsh.

– `_BPXK_AUTOCVT` enables processes that automatically convert EBCDIC to ASCII. The default is OFF. IzODA requires the variable set to `ON`.

– `_CEE_RUNOPTS` enables you to specify the IBM Language Environment® runtime option. To make file tagging not apparent to Python, consider setting the variable to "`FILETAG(AUTOCVT,AUTOTAG) POSIX(ON)`". This variable is optional.

– `ANACONDA_HOME` specifies the root installation directory of Anaconda. The default is `/usr/lpp/IBM/izoda/anaconda`.

– `FFI_LIB` specifies the absolute or relative path to Python's Foreign Function Interface.

– `IBM_JAVA_OPTIONS` specifies IBM JVM runtime options. Spark requires that the variable is set to "`-Dfile.encoding=UTF*`".

– `IML_HOME` specifies the location where the configuration files for Machine Learning for z/OS scoring services reside.

– `IML_INSTALL` specifies the location where Machine Learning for z/OS configuration files is installed.

– `JAVA_HOME` specifies the location where IBM 64-Bit SDK for z/OS Java Technology Edition V8 is installed.

– `LIBPATH` specifies the absolute or relative path to Python's C dependencies.

– `PATH` specifies the location where the shell and bash programs can find executable files for Python and Conda.

– `PYTHON_HOME` specifies the path to the Python executable file. The default location is `/usr/lpp/IBM/izoda/anaconda/bin`.

– `SPARK_CONF_DIR` specifies the location of Spark configuration files. The default is `$SPARK_HOME/conf`.

– `SPARK_HOME` specifies the Spark installation directory. The default is `/usr/lpp/IBM/izoda/spark/spark211`.

– `SPARK_LOCAL_DIRS` specifies the directory that stores Spark shuffle and RDD data. The default is `/tmp`.

– `SPARK_LOCAL_IP` specifies the consistent IP address to which Spark processes bind when listening ports are created.

– `SPARK_LOG_DIR` specifies the directory that stores Spark log files. The default is `$SPARK_HOME/logs`. You must configure this location if you install Spark on an R/O file system.

– `SPARK_MASTER_PORT` specifies the listening port to which the Spark master process binds. The default is `7077`.

– `SPARK_PID_DIR` specifies the directory that stores Spark PID files. The setting is `/tmp`.

– `SPARK_WORKER_DIR` specifies the directory that stores Spark working data for the worker process. The default is `$SPARK_HOME/work`. You must configure this location if you install Spark on an R/O file system.

– STEPLIB is specified when connecting to MDS from Python. The DSDBC package explicitly requires that this variable is set to the MDS data set on z/OS.

– TERM enables MacOS users to edit files with vi.

2. Configure the bash shell for issuing Python and Anaconda (or **conda**) commands. With the ownership of the IzODA installation directories and the proper setup of the environment variables, you can complete Anaconda configuration by enabling the execution of python and conda commands from your bash shell:

   a. Log in to OMVS as MLZSPARK and browse to the /usr/lpp/IBM/izoda/anaconda directory where Bash 4.2.53 is installed as a prerequisite of IzODA.

   b. From the directory, issue the following command to start a bash session:

   ```
   ./bin/bash
   ```

   c. Issue the following command to run the install_ensure_scripts_are_in_ebcdic script, which tags all executable scripts inside of the /bin/ directory:

   ```
   ./bin/install_ensure_scripts_are_in_ebcdic
   ```

   Now you can issue both python and conda commands from your bash shell.

Leading practice: If bash is not the default shell on the system, consider making it your login shell by updating the OMVS segment information or by adding the **/usr/lpp/IBM/izoda/anaconda/bin/bash** command to your $HOME/.profile file. When you log in, the new command starts a bash shell.

3. Update Spark configuration settings. By default, Spark configuration files are stored in $SPARK_HOME/conf. Spark does not write to this directory for its operations, but any modification to the configuration files inside is lost during a Spark service update. The best way is to copy the configuration files to the $SPARK_CONF_DIR directory that you specified in the sample configuration file and make necessary changes there.

   Complete the following steps:

   a. Log in to a shell session as MLZSPARK and create the new directory as specified for the SPARK_CONF_DIR variable by issuing the following command:

   ```
   mkdir -p /etc/spark/conf
   ```

   Make sure that user MLZSPARK and user group IZODADEV have R/W access to the new /etc/spark/conf directory.

   b. Copy all the configuration files from $SPARK_HOME/conf to the new /etc/spark/conf directory by issuing the following command:

   ```
   cp $SPARK_HOME/conf/* $SPARK_CONF_DIR
   ```

   c. Browse to the new directory and open the spark-defaults.conf file that includes the default Spark configuration settings, some of which are shown in Example 3-5.

Example 3-5 spark-defaults.conf

```
# Default system properties included when running spark-submit.
# This is useful for setting default environmental settings.

# spark.master                     spark://master:7077
# spark.eventLog.enabled           true

# This option is the prefix of the driver jobname and applicable in cluster
deploy mode only.
spark.zos.driver.jobname.prefix    ODASD
```

```
# Set this option to false if you want to disable client authentication on the
master port. The default is true, which requires the enablement of AT-TLS on
z/OS. This option applies to client deploy mode only.
spark.zos.master.authenticate      false

# The REST server does not support client authentication or application-layer
TLS. Enable this option only when you have adequate security in place for the
REST port.
spark.master.rest.enabled          true
```

d. Set `spark.zos.master.authentication` to `false` and `spark.master.rest.enabled` to `true`. Spark client authentication is enabled by default. Spark uses AT-TLS with Level 2 client authentication to secure communications between the Spark master and its clients, including the Spark worker and driver.

   Although the REST option is enabled by default, Spark disables the port in the configuration templates because it currently does not include the required security. Make sure that you enable this option after you configure adequate security in place for the REST port.

4. Update the MDS configuration by adding the following environmental variable in the `$HOME/profile` file:

```
export STEPLIB=hlq.SAZKLOAD:$STEPLIB
```

3.2.3 Verifying IzODA installation and configuration

Before you start to use IzODA, perform some basic tasks to verify that the installation and configuration are successful and that Spark, Anaconda, and MDS work individually or together. Complete the following steps:

1. Verify that Anaconda is correctly installed and configured and that the python and conda executable files are accessible.:

   a. Start a bash shell session as `MLZSPARK` and enter the following command to verify the installation of Python 3.6.1:

   ```
   python
   ```

   The installation is successful when you see a message similar to Example 3-6.

   Example 3-6 python command

   ```
   MLZSPARK@BJC08:>python
   Python 3.6.1 (tags/HANA110:7960479, Aug 29 2017, 23:30:12) [C] on zos
   Type "help", "copyright", "credits" or "license" for more information
   ```

   b. Run the following **`print`** command to ensure that the python interpreter works correctly:

   ```
   print('Machine Learning for z/OS')
   ```

   The python interpreter is working if you see a message that is similar to the message that is shown in Example 3-7.

Example 3-7 Python interpreter

```
>>> print('Machine Learning for z/OS')
Machine Learning for z/OS
```

   c. Press CRTL+D or run the **`exit()`** command to quit the bash shell.

d. Issue the **`conda list`** command to ensure that Anaconda is properly configured and all installed packages are accessible:

```
conda list scikit-learn
```

The command should list all versions of scikit-learn, which is a popular machine learning package. If you see a message similar to the following example, conda is correctly configured and that the conda package manager works properly (see Example 3-8).

Example 3-8 Conda list command

```
MLZSPARK@BJC08:>conda list scikit-learn
# packages in environment at /usr/lpp/IBM/izoda/anaconda:

scikit-learn   0.18.  np112py36_1IzODA
```

2. Verify that Spark is correctly installed and configured by testing the Spark shell, an interactive Scala environment that runs on Java JVM. You can use the shell to access Spark API or analyze data interactively:

   a. Start the spark shell by issuing the following **`spark-shell`** command:

   ```
   $SPARK_HOME/bin/spark-shell
   ```

   If you see a message that is similar to Example 3-9, Spark shell is properly started, and a Spark context is obtained.

Example 3-9 spark-shell command

```
MLZSPARK@BJC08:>$SPARK_HOME/bin/spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
...
Spark context Web UI available at http://<ip>:<port>
Spark context available as 'sc' (master = local[*], app id = local-1513132304712).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.1
      /_/
Using Scala version 2.11.8 (IBM J9 VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.
Scala>
```

Note: Spark shell writes three files into the directory where it is started. Ensure that user MLZSPARK has read/write permissions to the `$SPARK_HOME/bin/` directory where the Spark-shell script is stored. You can redirect the output files to another location.

For more information, see *IBM Open Data Analytics for z/OS Installation and Customization Guide*.

   b. Issue more commands, including **`sc.parallelize`** and **`help`**, if necessary, to confirm that Spark shell works properly.

The **sc.parallelize(1 to 1000).count()** command should return the value of 1000 and the **:help** command should display the help information, as shown in Example 3-10.

Example 3-10 Help command

```
scala> :help
All commands can be abbreviated, e.g. :he instead of :help.
Those marked with a * have more detailed help, e.g. :help imports.
:help [command]           Print this summary or command-
specific help
:history [num]            Show the history (optional num
is commands to show)
:h? <string>              Search the history
:imports [name name ...]  Show import history, identifying sources of names
:implicits [-v]           Show the implicits in scope
:javap <path|class>       Disassemble a file or class name
:load <path>              Load and interpret a Scala file
```

c. Press CRTL+D or enter the **:quit** command to close the Spark shell.

3. Verify that the **spark-submit** command can complete successfully, which starts an application in Spark. The easiest way to test the command is to run `SparkPi`, which is one of the sample programs that starts the spark-submit script in the background. Issue the following command to run the SparkPi sample program:

   ```
   $SPARK_HOME/bin/run-example SparkPi
   ```

   The **spark-submit** command works properly if you see a message that is similar to the message that is shown in Example 3-11.

Example 3-11 run-example command

```
MLZSPARK@BJC08:> $SPARK_HOME/bin/run-example SparkPi
17/12/12 21:38:51 INFO SparkContext: Running Spark version 2.1.1
17/12/12 21:38:52 WARN NativeCodeLoader: Unable to load native-hadoop library for
your platform… using builtin-java classes where applicable
17/12/12 21:38:53 INFO SecurityManager: Changing view acls to: MLZSPARK
17/12/12 21:38:53 INFO SecurityManager: Changing modify acls to: MLZSPARK
17/12/12 21:38:53 INFO SecurityManager: Changing view acls groups to:
17/12/12 21:38:53 INFO SecurityManager: Changing modify acls groups to:
17/12/12 21:38:53 INFO SecurityManager: SecurityManager: z/OS client
authentication is disabled
17/12/12 21:38:53 INFO SecurityManager: SecurityManager: authentication disabled;
ui acls disabled; users with view
17/12/12 21:39:01 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have
all completed, from pool
17/12/12 21:39:01 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38)
finished in 3.223 s
17/12/12 21:39:01 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38,
took 3.890919 s
Pi is roughly 3.138155690778454
17/12/12 21:39:01 INFO SparkUI: Stopped Spark web UI at
http://<host_IP_address>:4040
```

The following line at the end of the example indicates that the SparkPi sample was successfully started and that the **spark-submit** command worked properly:

```
Pi is roughly 3.138155690778454
```

4. Verify that a Spark cluster can be started. A spark cluster consists of a Spark master and a Spark worker that manage resources for all submitted Spark applications:

   a. Start the Spark master by issuing the following command:

   ```
   $SPARK_HOME/sbin/start-master.sh -h <host_IP_address>  -p <sparkMaster-port>
   ```

   Where `<host_IP_address>` is the IP address of the z/OS system and `<sparkMaster-port>` is the Spark master daemon port. The default port is 7077. The Spark master daemon attempts to define the port that you specify for the `-p` parameter. If the default port is not available, it increments the `-p` value by one and attempts to bind to the next available port.

   The **`start-master`** command generates a Spark master daemon log file in the `$SPARK_LOG_DIR` directory. The log file includes output that is similar to the output that is shown in Example 3-12.

Example 3-12 Log file output

```
MLZSPARK@BJC08:>$SPARK_HOME/sbin/start-master.sh -h <host_IP_address> -p 7077
Starting org.apache.spark.deploy.master.Master, logging to
/var/spark/logs/spark-MLZSPARK-org.apache.spark.deploy.master.Master-1-BJC08.out
```

   Check the log to make sure that the Spark master is started successfully and to determine the port number on which the Spark master daemon is listening.

   Leading practice: You do not need to specify the host (`-h`) and port (`-p`) parameters for the **`spark-master`** command if you set the `SPARK_LOCAL_IP` and `SPARK_MASTER_PORT` variables in the IzODA configuration file.

   b. Start the Spark worker by issuing the following command:

   ```
   $SPARK_HOME/sbin/start-slave.sh spark://<host_IP_address>:<sparkMaster-port>
   ```

   The **`start-slave`** command generates a Spark worker daemon log file in the `$SPARK_LOG_DIR` directory. The output of the command is similar to the output that is shown in Example 3-13.

Example 3-13 start-slave command

```
MLZSPARK@BJC08:>$SPARK_HOME/sbin/start-slave.sh spark://<host_IP_address>:7077
Starting org.apache.spark.deploy.worker.Worker, logging to
/var/spark/logs/spark-MLZSPARK-org.apache.spark.deploy.worker.Worker-1-BJC08.out
```

   Check the log to make sure that the Spark worker is started successfully.

   Note: The **`start-master`** and **`start-slave`** commands might return an error that states “FSUMA904 no matching processes found.” This error occurs when the commands check for but cannot find an active master or worker in the current user’s processes. Ignore the error if it occurs once or twice. More frequent occurrences might indicate that the Spark master or worker daemon process cannot start. Check the appropriate log file for more information.

c. Submit an application to the Spark cluster that you just started. Issue the following the **spark-submit** command on a single line to run the same SparkPi sample that you ran earlier by way of the interactive **run-example** command:

```
$SPARK_HOME/bin/spark-submit --class org.apache.spark.examples.SparkPi
--master spark://<host_IP_address>:<sparkREST-port>
--deploy-mode cluster
$SPARK_HOME/examples/jars/spark-examples_2.11-2.1.1.jar
```

Where `<sparkREST-port>` is the Spark master REST port (the default is 6066). Successful execution of the command returns a message similar to the message that is shown in Example 3-14.

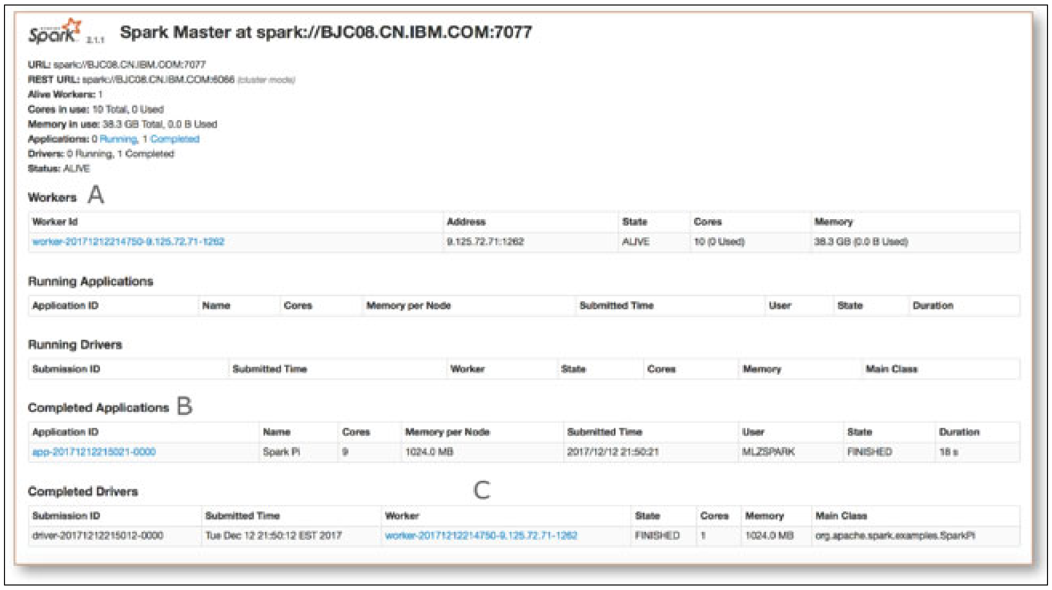
Example 3-14 spark-submit command

```
MLZSPARK@BJC08:>$SPARK_HOME/bin/spark-submit --class
org.apache.spark.examples.SparkPi --master
spark://<host_IP_address>:<sparkREST-port> --deploy-mode cluster
$SPARK_HOME/examples/jars/spark-examples_2.11-2.1.1.jar
Running Spark using the REST application submission protocol.
{
  "action" : "CreateSubmissionResponse",
  "message" : "Driver successfully submitted as driver-20171212215012-0000",
  "serverSparkVersion" : "2.1.1",
  "submissionId" : "driver-20171212215012-0000",
  "success" : true
}
```

d. Verify the submitted application by using the Spark WebUIs. When the Spark master is started, a master WebUI is opened on port 8080 by default. You can specify another port if needed. View the application by starting your browser and entering the following URL:

`http://<host_IP_address>:8080`

Your browser should display a page similar to what is shown in Figure 3-1.



Spark Master at spark://BJC08.CN.IBM.COM:7077

URL: spark://BJC08.CN.IBM.COM:7077
REST URL: spark://BJC08.CN.IBM.COM:6066 (cluster mode)
Alive Workers: 1
Cores in use: 10 Total, 0 Used
Memory in use: 38.3 GB Total, 0.0 B Used
Applications: 0 Running, 1 Completed
Drivers: 0 Running, 1 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20171212214750-9.125.72.71-1262	9.125.72.71:1262	ALIVE	10 (0 Used)	38.3 GB (0.0 B Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

Running Drivers

Submission ID	Submitted Time	Worker	State	Cores	Memory	Main Class

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20171212215021-0000	Spark Pi	9	1024.0 MB	2017/12/12 21:50:21	MLZSPARK	FINISHED	18 s

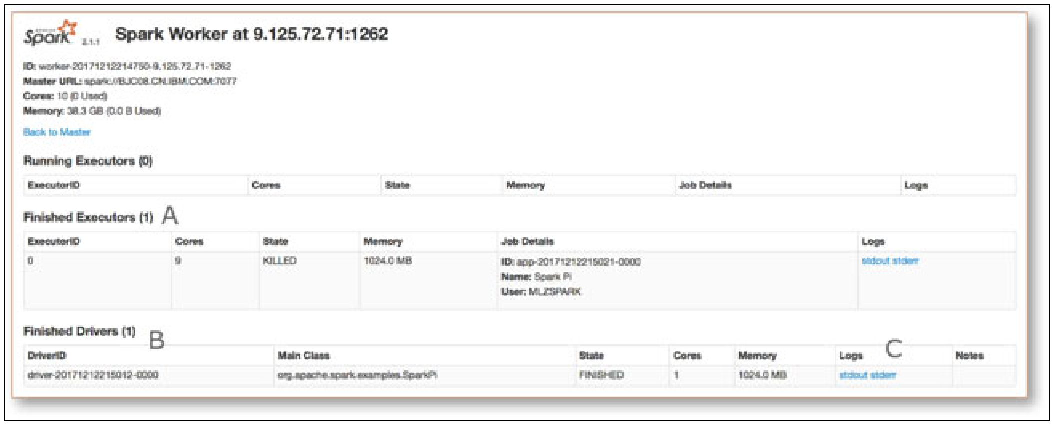
Completed Drivers

Submission ID	Submitted Time	Worker	State	Cores	Memory	Main Class
driver-20171212215012-0000	Tue Dec 12 21:50:12 EST 2017	worker-20171212214750-9.125.72.71-1262	FINISHED	1	1024.0 MB	org.apache.spark.examples.SparkPi

Figure 3-1 Spark WebUIs

Notice the following annotations in Figure 3-1 on page 34:

- A shows that the worker daemon process started and completed successfully.
- B shows that the SparkPi application was submitted and started successfully.
- C shows that the link for the worker page is active. Click the link to open the worker page. Check the driver output and see whether it provides an estimate of Pi (see Figure 3-2).



Spark Worker at 9.125.72.71:1262

ID: worker-20171212214750-9.125.72.71-1262
Master URL: spark://BJC08.CN.IBM.COM:7077
Cores: 10 (0 Used)
Memory: 38.3 GB (0.0 B Used)

Back to Master

Running Executors (0)

ExecutorID	Cores	State	Memory	Job Details	Logs

Finished Executors (1)

ExecutorID	Cores	State	Memory	Job Details	Logs
0	9	KILLED	1024.0 MB	**ID:** app-20171212215021-0000 **Name:** Spark Pi **User:** MLZSPARK	stdout stderr

Finished Drivers (1)

DriverID	Main Class	State	Cores	Memory	Logs	Notes
driver-20171212215012-0000	org.apache.spark.examples.SparkPi	FINISHED	1	1024.0 MB	stdout stderr	

Figure 3-2 Spark WebUIs

Note the following annotations in Figure 3-2:

- A shows the finished executors.
- B shows the finished drivers.
- C shows the active link to the stdout of the driver. Click the link to open and view the output.

3.3 Configuring security

Machine Learning for z/OS draws on the combined strengths of IBM Security solutions, including z/OS Lightweight Directory Access Protocol (LDAP) and Resource Access Control Facility (RACF), and industry security standards, including the Secure Sockets Layer (SSL) protocol, the Application Transparent-Transport Layer Security (AT-TLS) protocol, and the JSON Web Token (JWT). To secure Machine Learning for z/OS, authorize and authenticate all users, create and install an SSL/TLS certificate for network communication between all component systems, and configure LDAP with SDBM to access RACF for user authentication.

3.3.1 Authorizing and authenticating users

The Machine Learning for z/OS user management service uses LDAP for user authorization and RACF for user authentication. A new user with certain privileges is authorized in LDAP, and the password and other authentication information are stored in RACF. After the LDAP server is configured with SDBM as the backend, the user management service can access RACF remotely to authenticate the user.

When an authorized user signs in from a browser, the user management service sends the request to LDAP for authentication. If the user is successfully validated, the user management service generates a token and returns it to the browser through the proxy service. The user can sign in with the token. The browser then uses the same token to allow the user access to the IBM Machine Learning for z/OS UI and services. When an access request comes through, the proxy service validates the token and redirects to the right component and service.

The token that the user management service issues is the implementation of the JSON Web Token (JWT) standard. JWT is an open, industry standard RFC 7519 method for representing claims securely between two parties. A JWT token is signed by a user's private key that is provided by the user management service and the token expiry time is 12 hours. The machine learning services that receive the token can perform self-validation with the user's public key.

3.3.2 Creating and distributing an SSL certificate

IBM Machine Learning for z/OS uses the SSL/TLS protocols to secure network communications across the component systems. All services on the component systems must be configured to use the same CA-signed or self-signed SSL certificate. The certificate must contain the IP addresses of all the systems in the installation, including the IP address for the Machine Learning for z/OS proxy server.

If you set up an LDAP server with an SSL certificate for your applications, you might want to continue to use it for Machine Learning for z/OS. In that case, provide the public key with full certificate chain for your LDAP server (named `ldapcert.pem`) and copy it to the certificate directory that Machine Learning for z/OS uses. Otherwise, complete the following steps to create an SSL certificate on Linux or Linux on Z and distribute it to z/OS:

1. Collect the IP addresses (and optionally, host names) of all the systems in the installation of Machine Learning for z/OS, including the following addresses:
   - Virtual IP address of a Linux or Linux on Z server
   - LDAP server
   - Machine Learning for z/OS proxy server.

   For more information, see 2.6.1, "Network requirements" on page 19.
2. Create a directory on your Linux or Linux system to contain the SSL certificate for Machine Learning for z/OS:
   a. Log on to one of your Linux or Linux on Z nodes.
   b. If you have not done so, create a root installation directory `<`*`install_dir_linux`*`> or <`*`install_dir_zlinux`*`> ($IML_INSTALL)` for Machine Learning for z/OS.
   c. If you have not done so, retrieve and transfer the `iml_utilities-v1.1.0.5.tar` file to this root installation directory. The `.tar` file contains scripts for generating a self-signed SSL certificate on Linux or Linux on Z. For more information, see 2.1, "Product installers" on page 10.
   d. Create a subdirectory called `certs` to contain the SSL certificate you are going to create.

   Important: Ensure that your z/OS system administrator can access the `certs` directory. All Machine Learning for z/OS component systems must use the same certificate in the `certs` directory for SSL handshakes. Your z/OS system administrator needs access to the `certs` directory to import the certificate that is used during Step 6 of this process.

3. Choose the type of SSL certificates to use. You can use a CA-signed or a self-signed certificate to secure the Machine Learning for z/OS network communication. The two certificate types are compared in Table 3-2.

Table 3-2 Comparison between CA-signed and self-signed SSL certificate

Certificate type	Advantages	Disadvantages
Self-signed certificate	No cost	Requires that you distribute the certificate, minus the private key, to each trading partner in a secure manner
	Easy to generate	Requires that you redistribute the certificate to all clients every time it changes
	Self-validated	Not validated by a third-party entity
	Efficient for a few trading partners	Inefficient for many trading partners
CA-signed certificate	Eliminates the need that you send the certificate to each trading partner	Trading partners must download the digital CA-signed certificate that is used to verify the digital signature of trading partner public keys
	No changes are required on the trading partner's system if you re-create the digitally signed certificate by using the same CA	Must be purchased from a third-party vendor

If you do not have a CA-signed certificate in your organization, consider the use of a self-signed certificate. This option is satisfying, given the simplicity of how it is generated and the low number of entities on which it is installed.

Important: Proceed to Step 4 if you decide to use a CA-signed SSL certificate or Step 5 if you choose to go with a self-signed SSL certificate.

4. Configure a CA-signed SSL certificate on Linux or Linux on Z. Skip to Step 5 if you choose to use a self-signed certificate:
   a. Browse to the `certs` directory.
   b. Copy a CA-signed certificate into the directory.
   c. Convert the certificate to the PEM format.
   d. Rename `cert.pem` to `mycert.pem` and `cert.key` to `mykey.key`.
   e. Choose a password <*yourSSLPassword*> that you or your z/OS system administrator will use to install and use the certificate.
   f. Issue the following command to generate the required mlpubkey.pub file:
```
openssl rsa -pubout -in mykey.key -out mlpubkey.pub
```
   g. Issue the following command to generate the required certkey.pfx file:
```
openssl pkcs12 -export -in mycert.pem -inkey mykey.key -out certkey.pfx
-passout pass:<yourSSLPassword> -name selfsigned
```
   h. Issue the following command to generate the required keystore.jks file:
```
keytool -importkeystore -deststorepass <yourSSLPassword> -destkeypass
<yourSSLPassword> -destkeystore keystore.jks -srckeystore certkey.pfx
-srcstoretype PKCS12 -srcstorepass <yourSSLPassword> -alias selfsigned
```

5. Generate a self-signed SSL certificate on Linux or Linux on Z. If you decide to use a CA-signed SSL certificate, go back and follow the instructions in Step 4 to configure a CA-signed certificate:

   a. Browse to the `$IML_INSTALL` directory (<*install_dir_linux*> or <*install_dir_zlinux*>) and locate the `iml_utilties-v1.1.0.5.tar` file.

   b. Issue the following command to extract the `gen-cert.sh` and `openssl.cnf.multiple` files into the `misc` directory:

      ```
      tar -xvf iml_utilities-v1.1.0.5.tar
      ```

   c. Run the gen-cert script and specify the following information when prompted:

      - Keystore database password (required): Specify a password that for the keystore database that you or your z/OS system administrator creates for storing the certificate on z/OS. Ensure that you provide this password when distributing the certificate.
      - LDAP server IP address (required): Specify the IP address of the LDAP server. The host name of the server is optional.
      - Proxy server IP address (required): Specify the extra private static IP address that is assigned to the subnet for high availability and used by the Machine Learning for z/OS proxy server. The host name of the server is optional.
      - Additional IP address and DNS (optional): Specify more IP addresses and DNS for the certificate. Leave blank if you do not want to do so.
      - User Information: Specify information about you or the certificate requester, including your email address and the name of your country, state or province, locality, organization, and organization unit. It also includes a “common name,” which is the host name of your domain server, such as *.ibm.com. The information is incorporated into the certificate (see Example 3-15).

Example 3-15 gen-cert.sh

```
[root@ga311-master-1 misc]# ./gen-cert.sh
...
Generating a 2048 bit RSA private key
...
Country Name (2 letter code)[XX]:FR
State or Province Name(full name)[]:Herault
Locality Name (eg, city [Default City]:Montpellier
Organization Name (eg, company)[Default Company Ltd]:IBM
Organization Unit Name (eg, section):ibmccmpl
Common Name (eg, your name or your server’s
hostname):*.ibm.com
Email Address[]:username@fr.ibm.com
Writing RSA key
```

   d. Issue the following command to verify that the certificate was successfully created. This command prints the information of your certificate:

      ```
      openssl x509 -in certs/mycert.pem -text –noout
      ```

   e. The certificate you generated is stored in the default `certs` directory where you run the gen-cert.sh script.

   f. Verify that the `certs` directory contains the following files for the certificate:

      - mlpubkey.pub
      - certkey.pfx
      - keystore.jks

- mycert.pem
- mykey.key

6. Create a keystore database on your z/OS system where you installed LDAP. Use the database to store the SSL certificate after it is imported from your Linux or Linux on Z system:
   a. Log on to your z/OS system as a system administrator.
   b. Create a <*install_dir_zos*> directory (`$IML_INSTALL`) as the root installation directory for Machine Learning for z/OS.
   c. Create a `certs` subdirectory to store the keystore database files you are going to create and the SSL certificate files you are going to import from Linux or Linux on Z.
   d. Issue the **`gskkyman`** command from the `certs` directory to start the Database menu.
   e. Select **Option 1** to create a keystore database and specify the following information when prompted:
      - Key database name (required): Specify a database name of your choice.
      - Key database password (required): If you use a self-signed certificate, specify the keystore password that you defined in Step 5 during the certificate creation process. If necessary, ask your Linux or Linux on Z system administrator for the password.
      - Password expiration date: Do not set any password expiration date.
      - Database record length: Use the default database record length, which sets a non-expiring password for the database.
      - FIFS or non-FIFS mode: Select 0 for non-FIFS mode for the database.
   f. Press Enter to create the database.
   g. Verify that the `<mykeystore_name>` and `<mykeystore_name>.rdb` files are present.
7. Import the SSL certificate from your Linux or Linux on Z system to the new keystore database on your z/OS system:
   a. Verify that you can access the `certs` directory on your Linux or Linux on Z system.
   b. Set the FTP program on your z/OS system to run in binary mode. The FTP program might return an error message that is similar to the message that is shown in Example 3-16 if you do not set it in binary mode.

      Example 3-16 FTP error message

      ```
      ---------------------------------------------------------
      Unable to import certificate and key.
      Status 0x03353020 – Unrecognized file or message encoding.
      ---------------------------------------------------------
      ```

   c. Use the FTP program to copy all the files in the certs directory on your Linux or Linux on Z system to the corresponding certs directory on your z/OS system, including `certkey.pfx`, `mlpubkey.pub`, and `keystore.jks`.
   d. Browse to the `certs` directory on your z/OS system and issue the **`gskkyman`** command to start the Key and Certificate menu.
   e. Select Option 8 to import a certificate and a private key and specify the following information when prompted:
      - Import file name: Specify `certkey.pfx` as the import file.
      - Import file password: Specify the password for the SSL certificate.

   - Label: Specify “mlz” as the label. A certificate label is a unique identifier that represents the digital certificate stored in your key repository. It provides a convenient human-readable name for the key management functions.

   f. Select **Option 1** from the Key and Certificate menu to display the list of keys and certificates.
   g. Select **Option 1** to display the menu for the “mlz” key.
   h. Select **Option 3** to set “mlz” as the default key.
   i. Select **Option 4** to set the “mlz” certificate status as trusted.
   j. Enter `0` to exit the menu.

3.3.3 Configuring LDAP with SDBM for user authentication

Machine Learning for z/OS uses RACF to protect the password and other authentication for a user that is defined and authorized in LDAP. You can use and configure SDBM (or a similar program) as the backend of the LDAP server to manage its access to RACF.

Complete the following steps:

1. Prepare for LDAP configuration with SDBM:
   a. Ensure that you know the (absolute or relative) path to the directory on your z/OS system that contains the LDAP schema database and checkpoint files, including `LDBM-1.db`, `LDBM-2.db`, `LDBM.ckpt`, and `schema.db`.
   b. Verify that user <*zldap_userid*> is created and assigned proper permissions. For more information, see 2.5, “User IDs and permissions” on page 17. The discussion in this section uses and references MLZLDAP as <*zldap_userid*>.
   c. Start the RACF service if you have not done so yet.
   d. Browse to the $IML_INSTALL directory (`<install_dir_zos>`) and create a `dsconfig` subdirectory by issuing the following command:

      `mkdir $IML_INSTALL/dsconfig`

2. Copy the following configuration files from the default `/usr/lpp/ldap/etc` directory to the new `dsconfig` directory:
   – `ds.profile` contains the configuration options for the LDAP server.
   – `ds.slapd.profile` contains the configuration options for the LDAP server backend.
   – `ds.db2.profile` contains Db2-specific configuration options for TDBM or GDBM.
   – `ds.racf.profile` contains RACF-specific configuration options for SDBM.

   Issue the commands that are shown in Example 3-17 to copy the files.

   Example 3-17 Copy file commands

```
cp /usr/lpp/ldap/etc/
{ds.profile,ds.slapd.profile,ds.db2.profile,ds.racf.profile}
$IML_INSTALL/dsconfig
```

   You can customize these files in the dsconfig folder when you configure LDAP with SDBM.

3. Use an editor of your choice, such as vi, to customize the `ds.profile` file in the `dsconfig` directory by setting proper values to the following options:
   – ADMINDN specifies the distinguished name (DN) of a user in SDBM (RACF).
   – SDBM_SUFFIX specifies the suffix for the SDBM backend.

– SCHEMAPATH specifies the name of the file system directory that contains the LDAP schema database file.

– ADDRMODE enables 31-bit or 64-bit addressing mode.

– PROG_SUFFIX specifies the suffix of the PROG member to be created in the output data set.

– LDAPUSRID specifies the user ID under which the LDAP server runs. This value is the <*zldap_userid*> you created in 2.5, “User IDs and permissions” on page 17.

– OUTPUT_DATASET specifies the name of the data set that contains the output from the configuration utility and JCL jobs. The data set must have the appropriate format to receive and submit JCL jobs.

– OUTPUT_DATASET_VOLUME specifies the name of the volume for the output data set. Use the SMS keyword to indicate that the volume should be SMS-managed.

– APF_JOBCARD_1, PRGCTRL_JOBCARD_1, DB2_JOBCARD_1, RACF_JOBCARD_1 specify the job cards for the output JCL jobs that is produced.

– SLAPD_PROFILE specifies the path to the `ds.slapd.profile` file in the `dsconfig` directory.

– DB2_PROFILE specifies the path to the `ds.db2.profile` file in the `dsconfig` directory.

– RACF_PROFILE specifies the path to the `ds.racf.profile` file in the `dsconfig` directory.

Example 3-18 shows part of a customized `ds.profile` file that contains these configuration options.

Example 3-18 ds.profile file

```
ADMINDN = "racfid=racf000,profiletype=user,o=IBM"
SDBM_SUFFIX = "o=IBM"
SCHEMAPATH = /var/ldap/schema
ADDRMODE = 31
PROG_SUFFIX = ML
LDAPUSRID = MLZLDAP
OUTPUT_DATASET = GLD.CNFOUT
OUTPUT_DATASET_VOLUME = SMS
APF_JOBCARD_1 = //LDAPAPF JOB MSGCLASS=H,NOTIFY=&SYSUID
PRGCTRL_JOBCARD_1 = //LDAPPC JOB MSGCLASS=H,NOTIFY=&SYSUID
DB2_JOBCARD_1 = //LDAPDB2 JOB MSGCLASS=H,NOTIFY=&SYSUID
RACF_JOBCARD_1 = //LDAPRACF JOB MSGCLASS=H,NOTIFY=&SYSUID
SLAPD_PROFILE = /usr/mlz_install/dsconfig/ds.slapd.profile
DB2_PROFILE = /usr/mlz_install/dsconfig/ds.db2.profile
RACF_PROFILE = /usr/mlz_install/dsconfig/ds.racf.profile
```

Review the volume names and the high-level qualifier that is specified for SSL, RACF, and the Language Environment. Ensure that they match your system configuration.

4. Customize the `ds.slapd.profile` file in the `dsconfig` directory by setting proper values to the following options:

   – LISTEN specifies the URL format of the LDAP server and the port number you set.

   – SDBM_ENABLERESOURCES specifies whether the SDBM backend supports operations on RACF resources and classes.

   – SERVERCOMPATLEVEL specifies the LDAP server compatibility level.

   – SSL_AUTH specifies the SSL/TLS authentication method.

– SSL_KEYRINGFILE specifies the path and name of the SSL certificate keystore database you created in the previous section.

– SSL_KEYRINGFILEPW specifies the password of the SSL certificate keystore database you created.

– GSK_PROTOCOL_TLSV1_1, GSK_PROTOCOL_TLSV1_2 are ENVAR variables that enable or disable TLS1.1 or TLS1.2.

Example 3-19 shows part of a customized `ds.slapd.profile` file that contains these configuration options.

Example 3-19 ds.slapd.profile file

```
#LISTEN = ldap://:389
#LISTEN = ldap://hostname:389
LISTEN = ldaps://:636
#LISTEN = ldaps://hostname:636
#LISTEN = ldap://:pc
SDBM_ENABLERESOURCES = on
SERVERCOMPATLEVEL = 7
SSL_AUTH = serverAuth
SSL_KEYRINGFILE = /usr/mlz_install/certs/keystore
SSL_KEYRINGFILEPW = <mykeyringfilepassword>
ENVVAR = GSK_PROTOCOL_TLSV1_1=ON
ENVVAR = GSK_PROTOCOL_TLSV1_2=ON
```

5. Customize the configuration options in the `ds.db2.profile` file to match your system Db2 version.

6. Customize the configuration options in the `ds.racf.profile` file to specify the appropriate UID and GID.

   By default, the UID and GID for *<zldap_userid>* are set respectively to 2 and 1 in the `ds.racf.profile file`. If your installation does not allow shared UID values, ensure that the specified value is unique or that the LDAPGID and LDAPUID is replaced with AUTOUID and AUTOGID.

7. Issue the **`dsconfig`** command to create a set of JCL members, configuration files, and the LDAP server start-up procedure:

   a. Issue the following command to export the LDAP `sbin` directory to the PATH environment variable:

   ```
   export PATH=$PATH:/usr/lpp/ldap/sbin
   ```

   b. Issue the following command from the `dsconfig` directory and specify the customized ds.profile file as the input file:

   ```
   dsconfig -i ds.profile -a yes -d error
   ```

   The command should generate the set of JCL members that is shown in Example 3-20.

   Example 3-20 JCL members

   ```
   APF        //APF authorizations
   DSCONFIG   //Main configuration file
   DSENVVAR   //LDAP environment variables
   MLZLDAP    //Procedure for starting LDAP
   PRGMCTRL   //Program control on the libraries
   PROGLP     //Copy to SYS1.PARMLIB
   ```

```
RACF       //RACF update
```

You use this set of output files to configure your LDAP server and set up the z/OS system to run the server. You can start the LDAP server after the JCL jobs are successfully submitted and run.

You can manually update the output DSCONFIG and DSENVAR files, but any manual change is lost when you run the **dsconfig** command again.

c. Review the DSCONFIG file to ensure that the values that are shown in Example 3-21 are set correctly.

Example 3-21 DSCONFIG file

```
adminDN
commThreads 10
listen ldaps://:636
maxConnections 65535
schemaPath /var/ldap/schema
sendV3StringsOverV2As UTF-8
serverCompatLevel 7
sendV3StringsOverV2As
sizeLimit 500
timeLimit 3600
validateIncomingV2strings on
sslAuth serverAuth
sslKeyRingFile /usr/mlz_install/certs/keystore
sslKeyRingFilePW <mykeyringfilepassword>
database SDBM GLDBSD31/GLDBSD64
suffix "o=yourCompanyName"
enableResources on
database CDBM GLDBCD31/GLDBCD64 cdbm
```

Uncomment the "`database CDBM GLDBCD31/GLDBCD64 cdbm`" line if it is commented out.

d. Review the DSENVVAR file to ensure that the values that are shown in Example 3-22 are set correctly.

Example 3-22 DSENVVAR file

```
NLSPATH=/usr/lpp/ldap/lib/nls/msg/%L/%N
LANG=En_US.IBM-1047
GSK_PROTOCOL_TLSV1_1=ON
GSK_PROTOCOL_TLSV1_2=ON
```

8. Add the LDAP started task procedure to the procedure library of the target system:

   a. Locate the started task procedure in the output data set. The name of the started task procedure is the name of the LDAP user ID that is specified on the LDAPUSRID statement in the `ds.profile` file. The pre-assigned name of the LDAP user ID is MLZLDAP.

   b. Copy the MLZLDAP started task procedure from the output data set to the procedure library of the target system.

9. Update RACF by running the RACF and PRGMCTRL JCL jobs in the output data set:

   a. Before you run the JCL jobs, meet the following system requirements:

      - The profile data set in the JCL job is SYS1.** must match your system profile data set .* or .**

- The DSNR class must contain the DSN9_BATCH resource.
- The JCL must match the version profile of Db2 that runs on your system. The JCL job uses UID(1) and GID(2). Make sure those UID and GID are available on your system. Otherwise, update the JCL with AUTOID and AUTOGID.

b. Run the following JCL job in the RACF member that allows the LDAP server to run as a started task:

```
RACF member
```

c. Run the following JCL job in the PRGMCTRL member that sets Program Control on libraries used by the LDAP server. The PRGMCTRL member is required only if Program Control is active:

```
PRGMCTRL member
```

10. Start the LDAP server in SDSF (/s <*zldap_userid*>) or from the operator's console (s <*zldap_userid*>).

11. Verify the LDAP server configuration by issuing the **`ldapsearch`** command, as shown in Example 3-23

Example 3-23 LDAP server configuration

```
ldapsearch -Z -K /usr/mlz_install/certs/keystore –P <keystorepass> -h
<host_ip_address> -p 636
-D racfid=mlzldap,profiletype=user,o=IBM -w <mlzlda_password>
-b "profiletype=user,o=IBM" "racfid=tuser01"
racfid=TUSER01,profiletype=USER,o=IBM
```

where:

– `Z` indicates that the SSL certificate is used, and `keystore` is the keystore database that you created.

– `K` is your keystore database name.

– `P` is your keystore database password.

– `h` is the host name or IP address of your LDAP server (the same z/OS host on which the imported SSL certificate resides).

– `p` is the port of your LDAP server.

– `D` is the LDAP administrator ID that can be used to search other users. It refers to LDAP user MLZLDAP.

– `w` is the password for the LDAP administrator ID.

– `b` is the user search base.

– `racfid` is the user that you want to search. It can be any of the existing users on your system.

3.4 Installing and configuring Machine Learning for z/OS

Machine Learning for z/OS consists of scoring, training, management, and other services and processes that run on different platforms. You must install these services on z/OS and Linux or Linux on Z systems based on the choice you made in 2.1, “Product installers” on page 10.

3.4.1 Installing machine learning services on z/OS

Machine Learning for z/OS scoring and training services run on z/OS. Complete the following procedure to install and configure those services on z/OS:

1. Prepare for the installation and configuration of Machine Learning for z/OS:
   a. Log on to your z/OS system by using your <*mlz_scoring_userid*>. For more information about this user ID, see 2.5, “User IDs and permissions” on page 17.
   b. Create a `<install_dir_zos>` directory for storing z/OS system configuration files and a `<mlz_home>` directory for storing service configuration files.
   c. For quick and easy access, set $IML_INSTALL and $IML_HOME environment variables for the new directories and add them to your `/etc/profile` file.
   d. Ensure that you authorize the RACF user group and the Spark user group (<spark-GRP>) access to the new directories.
   e. If you have not done so, retrieve and transfer the *Machine Learning for z/OS Program Directo*ry, SPM/E image, and other installation scripts you received or downloaded from IBM to the z/OS system.
2. Run the SMP/E program to install the base code of Machine Learning for z/OS and apply any available maintenance package:
   a. Follow the instructions in the *Program Directory* to run the SMP/E program and install the machine learning services.
   b. The SMP/E program places the source code in the default `/usr/lpp/IBM/aln/v1r1m0/` directory. If you specify a different directory, ensure that it contains the `/usr` subdirectory.
   c. Follow the instructions at the Customer access portal for Machine Learning for z/OS to download and apply the latest product updates (APARs).
   d. Verify that the `/usr/lpp/IBM/aln/v1r1m0/` directory contains the files that are shown in Example 3-24.

Example 3-24 /usr/lpp/IBM/aln/v1r1m0/ directory

```
bash-4.2# ls /usr/lpp/IBM/aln/v1r1m0
ALNSAMP.pax        IMLzOP.tar          WLP17002.pax
IMLPython.pax      IMLzOS.properties   iml-install.sh
IMLzCICS.tar       IMLzOS.tar.         IMLzOSMF.tar
README
```

3. Extract the `.pax` and `.tar` files by issuing the following command:

   ```
   ./iml-install.sh $IML_HOME
   ```

   You should see a directory structure in $IML_HOME similar to the structure that is shown in Example 3-25.

   Example 3-25 Directory structure

   ```
   +- alnsamp/
   +- bin/
   |  +- server.sh
   +- cics-scoring/
   +- configuration/
   |  +- defaults/
   |  |  +- bootstrap.properties
   |  |  +- iml/
   |  |  |  +- server.xml
   |  +- generated/
   |  +- log4j.properties
   |  +- scoring.cfg
   +- extra/
   |  |  +- examples
   |  |  +- log4j2.properties
   +- iml-library/
   |  +- library/
   |  +- runner/
   +- imlpython/
   +- ophandling/
   +- output/
   +- usr/
   |  +- extension/
   |  |  +- lib
   |  |  |  +- feature
   |  |  |  |  +- scoring-1.0.mf
   |  +- servers/
   +- wlp/
   +-
   zosmf-wf/
   ```

   The script process might prompt you to overwrite the `/etc/spark/conf/log4j2.properties` file. You can confirm the overwrite because Spark was configured with the options specified in the `spark-defaults.conf` file, instead of the `log4j2` file.

4. Copy the `mlpubkey.pub` and `keystore.jks` files from the `$IML_INSTALL/certs` directory to your `$IML_HOME/configuration` directory.

5. Install, configure, and start the Jupyter kernel gateway and Apache Toree:

   a. Log on to your z/OS system as user <*spark_jupyter_toree_userid*>.

   b. Run the following `kg2at-install.sh` script in the `$IML_HOME/imlpython/bin` directory to install the Jupyter kernel gateway and Apache Toree:

      ```
      ./kg2at-install.sh
      ```

      This script runs the setupEcosystem.sh script and creates a configuration file.

c. Run the `kg2at-config.sh` script in the `$IML_HOME/imlpython/bin` directory to configure the gateway. A default port 8889 is assigned to the gateway. If you want to use a different port, specify the port number, as shown in the following example:

```
./kg2at-config.sh <port>
```

d. Run the `kg2at-start.sh` script in the `$IML_HOME/imlpython/bin` directory with a source command to start the gateway:

```
source kg2at-start.sh
```

e. Check the `gateway.out` log in the `$IML_HOME /imlpython/logs` directory to verify that the gateway is started. You should see a message similar to the following example if the start was successful:

```
[KernelGatewayApp] Jupyter Kernel Gateway at http://<ip_address>:8889
```

If necessary, run the `kg2at-stop.sh` script to stop the gateway and then run the `kg2at-start.sh` script to restart it.

6. Install the Python packages that are required by Machine Learning for z/OS to train and score Scikit-learn models. Run the `setupDependencies.sh` script in the `$IML_HOME/imlpython/bin` directory:

```
./setupDependencies.sh
```

It might take a few minutes for the script process to complete.

7. Create, configure, and start a new scoring server for the Machine Learning for z/OS scoring service. The `$IML_HOME/bin` directory contains the `server.sh script`, which you can use to create, start, stop, or remove a scoring server with the appropriate parameter:

a. Log on to your z/OS system as user <*mlz_scoring_userid* >.

b. Issue the following command to create a scoring service server:

```
bin/server.sh create <serverName>
```

The command generates a `scoring.cfg.<serverName>` configuration file in the `$IML_HOME/configuration` directory

c. Customize the scoring `scoring.cfg.<serverName>` file by setting appropriate values to the following configuration options for the scoring service host (IP addresses or host names and port numbers):

- `scoring_ip` specifies the host name or IP address of the system where the scoring service runs.
- `http_port` and `https_port` specify the HTTP and HTTPS ports of the scoring service.
- `admin_http_port` and `admin_https_port` specify the administrative HTTP and HTTPS ports of the scoring service.
- `flask_http_port` and `flask_https_port` specify the HTTP and HTTPS ports for the Python uWSGI server of the scoring service. This port is used by a Python uWSGI server for scoring Scikit-learn models.

  After it is started, the uWSGI server listens on the specified port. Each scoring service server is defined with a uWSGI server. The scoring server routes a Scikit-learn model scoring request to the attached uWSGI server for processing. If you do not need Scikit-learn model scoring, specify the `flask_http_port -1` and `flask_https_port -1` parameters to disable the ports when the scoring service server is created. If you do need any Scikit-learn model scoring, define the ports of your choice.

The following configuration options are available for the SSL certificate:

- `public_key_path` specifies the location where you store the public key `mlpubkey.pub` file on your system. This value should be in the `certs` folder that you imported from the Linux system (`/usr/mlz_install/certs/mlpubkey.pub`).
- `ssl_cert_file_location` specifies the location where you store the certificate (`mycert.pem`) file on your z/OS system (`/usr/mlz_install/certs/ mycert.pem`).
- `ssl_key_store_location` specifies the location where you store the SSL keystore (*keystore.jks*) file on your z/OS system (`/usr/mlz_install/certs/ keystore.jks`).
- `ssl_key_store_type` specifies the type of the SSL keystore file.
- `ssl_private_key_location` specifies the location where you store the private key (`mykey.key`) file on your z/OS system (`/usr/mlz_install/certs/`).
- `ssl_trust_store_location` specifies the location of the SSL truststore file (`/usr/mlz_install/certs/`).
- `ssl_key_store_password` specifies the encrypted password of the SSL keystore file.
- `ssl_trust_store_password` specifies the encrypted password of the SSL truststore file.

For better security, encrypt the passwords of the SSL keystore and truststore files. Complete the following steps to encrypt the passwords:

i. Start a second shell session and browse to the `$IML_HOME` directory.

ii. Issue the following command to encrypt your SSL keystore password:

```
wlp/bin/securityUtility encode --encoding=xor <your_keystore_password>
```

iii. Set the ssl_key_store_password option to the encrypted password.

iv. Repeat Steps ii and iii to encrypt the password for the ssl_trust_store_password option.

The following configuration options are available for the repository service:

- `repository_host` specifies the IP address of the Linux or Linux on Z system where the repository service runs. It is the virtual IP address that is used to set up the Kubernetes cluster as defined in `kube_service_ip` in the `deploy.cfg` file.
- `repository_port` specifies the port that is used by the repository service on the Linux or Linux on Z system. The default value is 12501.
- `repository_ssl_protocol` specifies whether HTTPS protocol is used to connect to the repository service. By default, the HTTP protocol is used to connect to the repository service.

The following configuration options are available for the deployment service:

- `deployment_host` specifies the IP address of the Linux System where the deployment service run, as defined in `kube_service_ip` on the `deploy.cfg` file. This address should be the Virtual IP Address that was used to set up the Linux cluster.
- `deployment_port` specifies the port that is used by the deployment service on the Linux system. This port is defined during the prerequisite step. The default value is 14150.
- `deployment_ssl_protocol` specifies whether HTTPS protocol is used to connect to the Machine Learning deployment service. By default, HTTP protocol is used to connect to the deployment service.
- `enable_monitor` enables the monitoring by way of the administration dashboard if it is set to `true`.

   - `upload_interval` specifies the interval of data uploading by the monitoring service.

d. Issue the following command to start the new scoring server that you created and configured:

```
bin/server.sh start <serverName>
```

It might take a few minutes for the scoring server to start completely. The server runs in the background.

If `flask_http_port` or `flask_https_port` is specified, start the Python uWSGI server by issuing the following command:

```
bin/server.sh start-python <serverName>
```

e. Verify that the scoring server is successfully started by checking the server status message in the console.log file in the `output/<serverName>/logs` directory. If the server is successfully started, you should see a message similar to the following example:

```
com.ibm.ml.scoring.online.service.ServiceManager | Scoring server started
...
```

If the Python uWSGI server is configured and started, check the server status message in the `uwsgi.log` file in the `output/<serverName>/logs` directory. The server is successfully started if you see a message similar to the following example:

```
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI master process (pid: 6849)
spawned uWSGI worker 1 (pid: 7067, cores: 10)
```

f. Issue the following command to list all the scoring servers that are currently running:

```
bin/server.sh list
```

g. If you need to stop a scoring server or the Python uWSGI server, issue the following command:

```
bin/server.sh stop <serverName>
```

h. If a uWSGI server is attached, issue the following command to stop the server:

```
bin/server.sh stop-python <serverName>
```

i. If you need to remove a scoring server, issue the following command:

```
bin/server.sh remove <serverName>
```

8. Create, configure, and start a new server to handle Machine Learning for z/OS operations:

a. Create and customize a new operation handling service server by issuing the following command:

```
bin/ophandling.sh create
```

The command generates the `ophandling.cfg` server configuration file in the `$IML_HOME/configuration` directory.

b. Edit the `ophandling.cfg` file to customize the IP address and port numbers for the operation handling service. If necessary, specify `--http-port -1` or `--https-port -1` to disable either port.

c. Start the operation handling service by issuing the following command:

```
bin/ophandling.sh start
```

The operation handling service server starts and runs in the background.

d. Verify that the operation handling service server is successfully started by checking the server status message:

```
bin/ophandling.sh status
```

e. If you need to stop the operation handling server, issue the following command:

```
bin/ophandling.sh remove
```

f. If you need to remove the operation handling server, issue the following command:

```
bin/ophandling.sh remove
```

9. Add the new scoring service you created to the Machine Learning for z/OS administration dashboard. The service must be added to the `Scoring Services` page of the dashboard for it to become available for model deployment in the Machine Learning for z/OS user interface.

10. Create the database objects that are required by the Machine Learning for z/OS repository service. The repository service provides important metadata about models. The metadata is stored in Db2 tables that must be created before any service can be started.

    A sample ALNMLEN JCL job is provided in the `$IML_HOME/alnsamp` directory. You can run the sample job to create the required database and other database objects in the ALN schema:

    a. Log on to your Db2 subsystem as user <*db2_auth_id*> with the privileges required CREATE DATABASE and CREATE SEQUENCE. For more information about this user ID, see 2.5, “User IDs and permissions” on page 17.

    b. Locate the ALNMLEN file in the `$IML/HOME/alnsamp` directory and issue the following command to move it into a PS data set in the ISPF shell:

    ```
    oget '/usr/mlz_install/mlz_services/alnsamp/ALNMLEN'
    'TUSER01.ALNSAMP(ALNMLEN)' TEXT convert(NO)
    ```

    c. Follow the instructions in the ALNMLEN file to customize the sample JCL job.

    d. Submit the customized ALNMLEN job to create the required database objects, including a database, table spaces, sequences, tables, and indexes.

    e. Verify that the job runs successfully (with a return code of 0).

3.4.2 Installing the scoring service in a CICS region

If you use CICS to run your online transactions and have the business need for real-time scoring within your CICS transactions, you can install and deploy the Machine Learning for z/OS scoring service in your CICS regions (see Figure 3-3).

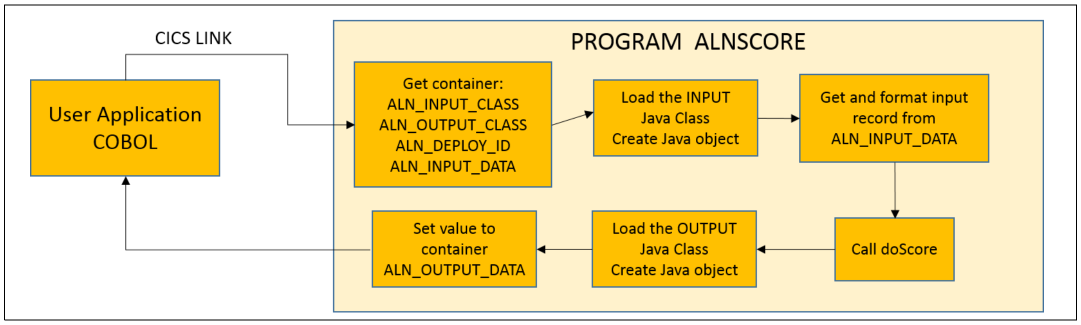

CICS LINK
PROGRAM ALNSCORE
User Application COBOL
Get container: ALN_INPUT_CLASS ALN_OUTPUT_CLASS ALN_DEPLOY_ID ALN_INPUT_DATA
Load the INPUT Java Class Create Java object
Get and format input record from ALN_INPUT_DATA
Call doScore
Load the OUTPUT Java Class Create Java object
Set value to container ALN_OUTPUT_DATA

Figure 3-3 Scoring service with CICS

As shown in Figure 3-3 on page 50, you can start the scoring service and run online scoring of Spark, MLeap, and PMML models. To complete this process, you use the EXEC CICS LINK API in your COBOL application, instead of the RESTful API:

1. Ensure that you completed the installation of machine learning services on z/OS or at least Steps 1 - 3 as described in 3.4.1, “Installing machine learning services on z/OS” on page 45.
2. Verify that the `$IML_HOME/cics-scoring` directory exists. If the subdirectory does not exist, rerun the `iml-install.sh` script to extract the `.pax` and `.tar` files. The `IMLzCICS.tar` file contains the source materials for installing and configuring the Machine Learning scoring service in a CICS region.
3. Copy the `mlpubkey.pub` and `keystore.jks` files from the `$IML_INSTALL/certs` directory to your `$IML_HOME/cics-scoring/configuration` directory.
4. Create and configure a new Liberty Profile server named ALNSCSER in a CICS region for the Machine Learning for z/OS scoring service. The `$IML_HOME/cics-scoring/bin` directory contains the `server.sh` script. Use the script to create, configure, start, stop, or remove a scoring server in a CICS region.

   You can create and start one scoring server per CICS region. If you run multiple CICS regions on the same LPAR, you can create a server for each region. Regardless the region, the server must be named ALNSCSER:

   a. Log on to your z/OS system as user <*mlz_scoring_userid* >.

   b. Issue the following command from the $IML_HOME directory to create an ALNSCSER server:

      ```
      Cics-scoring/bin/server.sh create <cics_region_name>
      ```

      The command generates a `scoring.cfg.<cics_region_name>` server configuration file in the `$IML_HOME/cics-scoring/configuration` directory.

   c. Customize the `scoring.cfg.<cics_region_name>` file by setting appropriate values to the following configuration options, in addition to the ones that you specified in the `scoring.cfg.<serverName>` file as described in 3.4.1, “Installing machine learning services on z/OS” on page 45:

      - `-Dcom.ibm.ws.logging.console.log.level` specifies the error condition level at which the Liberty Profile server writes a message to the JVM server stdout stream. The default level is INFO.
      - `JAVA_HOME` specifies the location of the Java directory on your z/OS system.
      - `WLP_INSTALL_DIR` specifies the directory of the Liberty Profile installation in your CICS environment. For example, in the IBM CICS Transaction Server for z/OS 5.4.0 installation, the default Liberty Profile directory is `/usr/lpp/cicsts/cicsts54/wlp`.

   d. Configure the ALNSCSER scoring server for CICS region <*cics_region_name*> by issuing the following command:

      ```
      cics-scoring/bin/server.sh config <cics_region_name>
      ```

      The **`config`** command reads the options in the `scoring.cfg.<cics_region_name>` configuration file and generates the `ALNSCSER.jvmprofile` and `server.xml` files for the ALNSCSER server.

      You can find the ALNSCSER.jvmprofile file in `$IML_HOME/cics-scoring/configuration/generated/<cics_region_name>` and the `server.xml` file in `$IML_HOME/cics-scoring/usr/<cics_region_name>/servers/ALNSCSER`.

e. Copy the `ALNSCSER.jvmprofile` file from `$IML_HOME/cics-scoring/configuration/generated/<cics_region_name>` to `<JVMPROFILEDIR>` that is used by your CICS region.

f. Verify that your CICS region user <*cics_region_userid*> can read and write to the following directories and files:

- $IML_HOME/cics-scoring/bundle
- $IML_HOME/cics-scoring/configuration
- $IML_HOME/cics-scoring/usr/<cics_region_name>
- $IML_HOME/cics-scoring/workdir
- <JVMPROFILEDIR>/ALNSCSER.jvmprofile

5. Define the ALNSCSER JVM server to CICS by issuing a **`CEDA`** command, as shown in Example 3-26.

Example 3-26 ALNSCSER JVM server

```
CEDA DEFINE JVMSERVER(ALNSCSER) GROUP(ALNSCGRP)
DESCRIPTION(JVM SERVER FOR MLZ SCORING SERVER)
JVMPROFILE(ALNSCSER)
STATUS(ENABLED)
```

You can also run the sample JCL ALNSCDEF job in the `$IML_HOME/cics-scoring/extra/jcllib` directory to define the JVM server to your CICS region.

6. Define the Machine Learning scoring service bundle to CICS by issuing a CEDA command, as shown in Example 3-27.

Example 3-27 Define scoring service bundle

```
CEDA DEFINE BUNDLE(ALNSCBDL) GROUP(ALNSCGRP)
DESCRIPTION(CICS BUNDLE FOR MLZ SCORING SERVER)
BUNDLEDIR(/$IML_HOME/cics-scoring/bundle)
STATUS(ENABLED)
```

You can also run the sample JCL ALNSCDEF job in the `$IML_HOME/cics-scoring/extra/jcllib` directory to define the scoring service bundle to your CICS region.

7. Start the ALNSCSER JVM server and then the Machine Learning scoring service bundle by issuing the **`CEDA`** commands in the order that is shown in Example 3-28.

Example 3-28 Start the ALNSCSER JVM server

```
CEDA INSTALL JVMSERVER(ALNSCSER) GROUP(ALNSCGRP)
CEDA INSTALL BUNDLE(ALNSCBDL) GROUP(ALNSCGRP)
```

8. Verify that the ALNSCSER JVM server and the Machine Learning scoring service bundle are successfully started. It takes several minutes for the scoring service to start.

9. Check `scoring-all.log` for server status in the `$IML_HOME/cics-scoring/workdir/<cics_region_name>/ALNSCSER` directory. The server is successfully started if you see a message similar to the following example:

```
|com.ibm.ml.scoring.online.service.ServiceManager| Scoring server started...
```

Alternatively, you can issue the CICS CEMT INQUIRE commands to query the status of the JVM server, the scoring service bundle, and the scoring service program (xx), as shown in Example 3-29 on page 53.

Example 3-29 CICS CEMT INQUIRE commands

```
CEMT INQURIY JVMSERVER(ALNSCSER)
Jvm(ALNSCSER) Ena Prf(ALNSCSER) Ler(DFHAXRO)
   Threadc(011) Threadl(015) Cur(964350144)
CEMT INQURIY BUNDLE(ALNSCBDL)
Bun(ALNSCBDL) Ena  Par(00001) Tar(00001)
   Enabledc(00001) Bundlei(scoring-service)

CEMT INQURIY PROGRAM(ALNSCORE)
Prog(ALNSCORE) Pro Ena   Ced
   Resc(0000) Use(0000000000) Any Cex Dpl Ore Ope Jvm
```

If needed, stop the scoring service and the ALNSCSER server by issuing the CICS command that is shown in Example 3-30.

Example 3-30 Stopping service and server

```
CEMT SET JVMSERVER(ALNSCSER) DISABLED
CEMT SET BUNDLE(ALNSCBDL) DISABLED
```

Restart the ALNSCSER server and the scoring service by issuing the CICS command that is shown in Example 3-31.

Example 3-31 Restarting service and server

```
CEMT SET JVMSERVER(ALNSCSER) ENABLED
CEMT SET BUNDLE(ALNSCBDL) ENABLED
```

If needed, remove the ALNSCSER server installation and configuration from a CICS region by issuing the following command:

```
cics-scoring/bin/server.sh remove <cics_region_name>
```

10. Add the new ALNSCSER scoring service as a native CICS scoring service to the Scoring Services page of the Machine Learning for z/OS administration dashboard. You can then select and use the new scoring service when creating the deployment of a model. For more information, see Chapter 4, “Administration and operation ” on page 63.

3.4.3 Installing machine learning services on Linux or Linux on Z

Machine Learning for z/OS scoring and training services also run on Linux or Linux on Z. You can install these services on x86 64-bit Linux systems or s390x 64-bit Linux on Z systems. The following instructions apply to Linux and Linux on Z (differences are noted wherever they occur):

Note: The overall installation and configuration process takes about 120 minutes to complete, which starts at the shell command line and continues from a web interface. Ensure that you plan ahead and allocate enough time for the entire process.

1. Gather the following materials and information from your z/OS and Linux or Linux on Z system administrators:
   - Network IP ranges, IP addresses, and port numbers that are allocated and configured for Machine Learning for z/OS. For more information, see 2.6.1, “Network requirements” on page 19 and 2.6.2, “Ports” on page 20.

   - Repository service settings, including Db2 subsystem IP address, location, port number, and authorization ID. For more information about `<db2_auth_id>`, see 2.5, “User IDs and permissions” on page 17.
   - MDSS JAR files if you plan to use MDS as a data source, data connector, or both.
   - The `ldapsearch` script that is used to test LDAP installation.

2. Prepare for the installation and configuration of Machine Learning for z/OS:

   a. Log on to one of the cluster nodes as the default user with at least sudo access. The Machine Learning for z/OS installer uses the default user of each node to upload installation materials and configure the node. The user or user name must have at least sudo access, and the password must not contain a single quotation ('), double quotation ("), pound sign (#), or white space ( ).

   b. Browse to the $IML_INSTALL directory (`<install_dir_linux>` or `<install_dir_zlinux>`) that you or your system administrator created during the creation of an SSL certificate. For more information, see 3.3.2, “Creating and distributing an SSL certificate ” on page 36.

      Ensure that the `$IML_INSTALL/certs` directory exists and that the directory contains the required SSL certificate files. The installer fails if it cannot find the `certs` directory.

      If you plan to use MDSS as a Machine Learning data source, you must also create a `mdss_ext_lib` subdirectory in the `$IML_INSTALL` directory and copy the MDSS jars into it.

   c. If it is not done yet, transfer the following Machine Learning for z/OS installer files to the $IML_INSTALL directory:

      - IBM_Machine_Learning_Installer_v1.1.0.5_Linux_x86-64 (for installation on Linux)
      - IBM_Machine_Learning_Installer_v1.1.0.5_Linux_s390x (for installation on Linux on Z)

   d. Configure the secondary storage that is required for each node. As described in 2.3.1, “Basic system capacity” on page 12, each node requires a secondary storage of at least 650 GB. Configure the secondary storage into two partitions in XFS format with the `ftype=1` option enabled.

      Ensure that one partition has a minimum of 300 GB disk space for installation files and the other a minimum of 350 GB for data storage. Both partitions must be mounted to the path that is used by the Machine Learning for z/OS installer:

      ```
      mkfs.xfs -f -n ftype=1 /dev/sdb1
      mkfs.xfs -f -n ftype=1 /dev/sdb2
      ```

      Leading practice: Consider configuring the mount points for automatic remount in case of a system restart.

   e. Ensure that the secondary storage has good I/O performance. Conduct the following simple latency tests and ensure that the performance is better or comparable to 512000 bytes (512KB) copied, 1.7912 s, and 286 KB/s:

      ```
      dd if=/dev/zero of=/root/testfile bs=512 count=1000 oflag=dsync
      ```

      Conduct the following simple throughput tests and make sure that the performance is better or comparable to 1073741824 bytes (1.1 GB) copied, 5.14444 s, and 209 MBps:

      ```
      dd if=/dev/zero of=/root/testfile bs=1G count=1 oflag=dsync
      ```

   f. Verify that the required network and firewall are properly configured. For more information, see 2.6, “Networks, ports, and firewall configuration” on page 19.

g. Disable the RHN plug-in on all nodes if the nodes do not have access to the Red Hat network. Keeping the RHN plug-in enabled without giving the nodes access to the RHN network prolongs the installation process considerably. Set enabled=0 in the `/etc/yum/pluginconf.d/rhnplugin.conf` file on all nodes. If necessary, enable the plug-in after the installation is completed.

3. Start the installation and configuration at the shell command line. Issue the following command to run the installer.

   – For Linux:

   ```
   sh IBM_Machine_Learning_Installer_v1.1.0.5_Linux_x86-64
   ```

   – For Linux on Z:

   ```
   sh IBM_Machine_Learning_Installer_v1.1.0.5_Linux_s390x
   ```

   The installer runs a sequence of eight steps to prepare for installation, including the creation of an installation log file and loading IBM Data Platform docker images and containers. After the initial setup is completed successfully, the installer prepares for you to continue the installation from a web interface.

4. Continue the installation, configuration, and service deployment from the Machine Learning for z/OS installer web UI:

   a. When prompted, open your browser and enter following link with the IP address and the port number shown at the shell command line:

   `http://<`*`ip_address`*`>:<`*`port`*`>`

   b. Select the **I agree** option to accept the terms and conditions and then, click **Continue** to start the Machine Learning for z/OS installer web UI (see Figure 3-4).

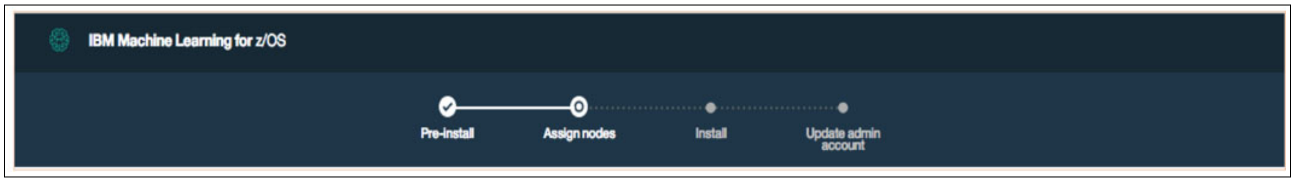


Figure 3-4 Machine Learning for z/OS installer web UI

The UI guides you through a four-step process to complete the installation and configuration, with the preinstall task already done at the shell command line.

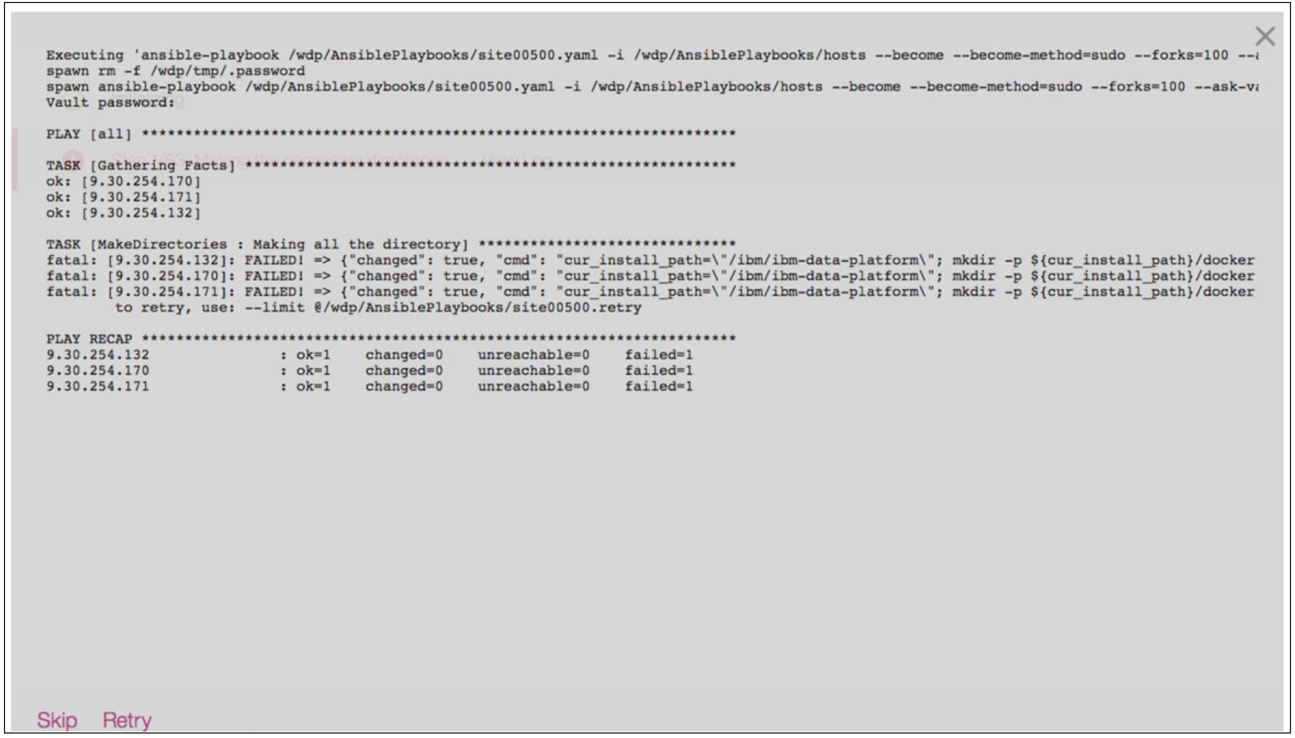
c. On the Assign nodes page, specify physical or virtual nodes for the cluster that will be managed by Kubernetes. The installer pings the nodes that you assign and validate the information that you provide. Consider the following points:

   - A private key can be used as an alternative for sudo credentials.
   - The cluster overlay network IP is the IP range that you or your system administrator already assigned. For more information, see 2.6.1, “Network requirements” on page 19.
   - The portable static proxy IP address must be unique.
   - The nodes serve as a control, compute, and storage plane for the entire Kubernetes cluster. Each node is configured to work as the master and worker. If you specify host names instead of IP addresses for the nodes, ensure that the host names are specified in lowercase.

     For each node, provide a user name that has sudo access, and the path of the directory or directories for the primary and secondary storage disks.

After all values of all fields are validated, click **Next** to assign nodes and continue the installation.

d. On the Install page, monitor the progress of the remaining installation. The installer runs a sequence of 62 steps. Respond to any on-screen instructions. If necessary, check the installation log (as shown in Figure 3-5) to understand and address any issues that might occur.



```
Executing 'ansible-playbook /wdp/AnsiblePlaybooks/site00500.yaml -i /wdp/AnsiblePlaybooks/hosts --become --become-method=sudo --forks=100 --
spawn rm -f /wdp/tmp/.password
spawn ansible-playbook /wdp/AnsiblePlaybooks/site00500.yaml -i /wdp/AnsiblePlaybooks/hosts --become --become-method=sudo --forks=100 --ask-v
Vault password:

PLAY [all] *********************************************************************

TASK [Gathering Facts] *********************************************************
ok: [9.30.254.170]
ok: [9.30.254.171]
ok: [9.30.254.132]

TASK [MakeDirectories : Making all the directory] ******************************
fatal: [9.30.254.132]: FAILED! => {"changed": true, "cmd": "cur_install_path=\"/ibm/ibm-data-platform\"; mkdir -p ${cur_install_path}/docker
fatal: [9.30.254.170]: FAILED! => {"changed": true, "cmd": "cur_install_path=\"/ibm/ibm-data-platform\"; mkdir -p ${cur_install_path}/docker
fatal: [9.30.254.171]: FAILED! => {"changed": true, "cmd": "cur_install_path=\"/ibm/ibm-data-platform\"; mkdir -p ${cur_install_path}/docker
        to retry, use: --limit @/wdp/AnsiblePlaybooks/site00500.retry

PLAY RECAP *********************************************************************
9.30.254.132               : ok=1    changed=0    unreachable=0    failed=1
9.30.254.170               : ok=1    changed=0    unreachable=0    failed=1
9.30.254.171               : ok=1    changed=0    unreachable=0    failed=1
```

Figure 3-5 Installation log

When the progress indicator is at 100%, the installer completed the installation. Click **Next** to start the system configuration.

e. On the Update Admin Account page, specify a new password for the installation administrator account. This “admin” user is a default “super” user that cannot be removed. The user has the privileges to install and configure Machine Learning for z/OS, deploy all services, and manage other users.

   Click **Next** to update the administrator account and start the IBM Machine Learning for z/OS

5. Complete the system configuration and service deployment:

   a. On the Sign in page, enter `admin` as the user name and the new password you created for the installation administrator account.

      Click **Sign In** to start the IBM Machine Learning for z/OS administration dashboard, where you can easily configure your system, deploy or redeploy services, add or remove users, and manage other system resources.

   b. From the sidebar of the administration dashboard, select **System configuration** to set the required parameters to the values based on your installation and configuration. The specified parameters, along with some optional parameters, are stored in a `deploy.cfg` file in the `/iml/scripts` directory.

A typical `deploy.cfg` file can include some of the following parameters:

- `certificate_alias` specifies the alias name of the certificate when it is generated. If you generate the certificate by running the `gen_cert.sh` script, the default value of certificate_alias is set to “selfsigned”.
- `certs_location` specifies the location of the `certs` directory where the SSL key certificate files are stored on your Linux on Z system.
- `image_registry` specifies the name of the Docker Registry domain and the port number (for example, `image_registry = <myregistrydomain.com:5000>`).
- `image_version` specifies the version of the Machine Learning images that are used (for example, `image_version = 1.1.0`, which is the default).
- `jdbc_password` specifies the Db2 password.
- `jdbc_username` specifies the Db2 authorization ID.
- `jdbc_url` specifies where your Db2 is installed (for example, `jdbc:db2://<host>:<port>/<db2_subsystem_name>`).
- `jupyter_kernal_gateway_host` specifies the host where the Jupyter kernel gateway is installed.
- `jupyter_kernal_gateway_port` specifies the port that is used by the Jupyter kernel gateway.
- `keystore_password` specifies the password that you used for creating the SSL key database.
- `kube_service_ip` specifies the IP address of the host where the Kubernetes master runs.
- `ldap_cansearch` specifies whether the LDAP user ID (`<zldap_userid>`) has the RACF SPECIAL authority and can be used for validating a new user to be added.
- `ldap_host` specifies the name or IP address of the host where the LDAP server is installed. The specified host name or IP must match that of the system on which the SSL certificate was generated.
- `ldap_password` specifies the password for the LDAP credentials used to search the directory.
- `ldap_port` specifies the port that is used by the LDAP server.
- `ldap_searchbase` specifies the base distinguished name for the LDAP credentials that are used to search the directory (for example, `profiletype=user,o=IBM`, where `o=IBM` is the SDBM_SUFFIX parameter of your LDAP configuration).
- `ldap_searchfilter` specifies the filter set for the directory search (for example, `ldap_searchfilter = racfid = {0}`, where {0} represents the user ID used to log in at the UI for authentication).
- `ldap_userdn` specifies the user distinguished name for the LDAP credentials that are used to search the directory (for example, `racfid=racf000,profiletype=user,o=IBM`, where `racf000` is OPERATOR).

Note: If you specify a racfid with the SPECIAL attribute for the UserDN field, it implies that when you try to add a user, Machine Learning for z/OS uses this racfid as an administration ID to validate whether the new added user is defined. If you cannot provide a racfid with the SPECIAL attribute, the value that you specify for the UserDN field is for the purpose for testing LDAP connectivity only.

- `mdss_ext_lib` specifies the absolute path to the MDSS library. For Linux on Z, the path is `iml/deployfactory/k8s_deploy_script/ibmml-ga/iml_deploy_scripts`. Set this parameter only if you want to use MDSS as a data source; otherwise, leave it blank.
- `nfs_host` specifies the IP address of the Linux on Z host where the NFS runs.
- `nfs_volume_path` specifies the location of the directory to be mounted on the Linux on Z system.
- `spark_host` specifies the host where z/OS Spark runs (for example, `spark_host = spark://<host_ip>`).
- `spark_port` specifies the port that is used by z/OS Spark.
- `spark_web_port` specifies the port that is used by the z/OS Spark web user interface.
- `spark_rest_port` specifies the port that is used by the z/OS Spark REST API.
- `scoring_host` specifies the host where the scoring service runs. This parameter is required for installation on Linux on Z only.
- `scoring_port` specifies the port that is used by the scoring service. The default is 10080. This parameter is required for installation on Linux on Z only.
- `zos_installation_path` specifies the absolute path to the `$IML_INSTALL` directory on your z/OS system.

After entering valid values for all the fields, click **Next** to save the configuration settings and prepare for the services deployment process.

c. On the Deploy window, click **Next** to deploy the machine learning services.

After all services are successfully deployed, click **Next** to start the Users page of the administration dashboard. You can start to add users and assign them appropriate access level. For more information, see Chapter 4, “Administration and operation ” on page 63.

6. Verify that Machine Learning for z/OS was successfully installed and configured. Complete one or more of the following tasks to verify:
   - Launch and sign onto the Machine Learning for z/OS web user interface.
   - Attempt to create and deploy a test model by using the sample. For more information, see Chapter 5, “Model development and deployment: A retail example” on page 93.
   - Go to the Machine Learning for z/OS administration dashboard to ensure that all nodes, pods, and clusters are up and running. For more information, see Chapter 4, “Administration and operation ” on page 63.

3.5 Configuring high availability and scalability

Machine Learning for z/OS is highly available and scalable, with multiple scoring services on a single LPAR or spanning across different APARs. One way that Machine Learning for z/OS achieves high availability and scalability is to use the proven technologies of the IBM Z sysplex distributor.

You can configure TCP/IP with the SHAREPORT or distributed dynamic VIPA (DDVIPA) option. Although port sharing can route requests to multiple scoring services on the same LPAR, DDVIPAs work as a gateway to distribute requests to scoring services across different LPARs. Another way is to scale up your application cluster by adding compute nodes.

3.5.1 Configuring TCP/IP for port sharing and load balancing

You can configure a TCP/IP profile for port sharing and load balancing to achieve high availability. As shown in Figure 3-6, consider setting up port sharing if you run multiple scoring services on the same LPAR. With port sharing, all instances run on the same TCP port and requests from applications are routed to the instances by using a weighted, round-robin distribution. As a result, online scoring services remain available, even when some of the instances are down.

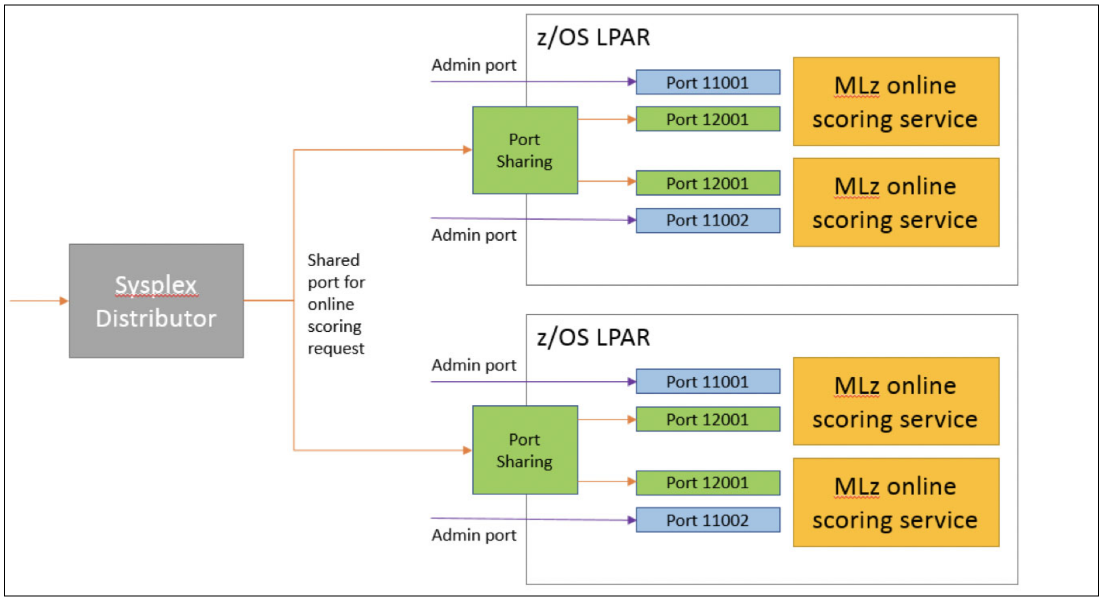


Figure 3-6 Port sharing

If your scoring service instances span across different LPARs, consider configuring the sysplex distributor to use DDVIPAs. A DDVIPA address and port also use a round-robin policy to route requests to scoring service instances on different LPARs. With DDVIPAs, online scoring services are still available even when some LPARs are lost.

Complete the following steps:

1. Define or update the TCP/IP profile in the TCP communication layer of your network configuration:

   a. If you run multiple scoring service instances on the same LPAR, specify the SHAREPORT parameter on the PORT statement in the TCP/IP profile to enable port sharing, as shown in the following example:

   ```
   PORT
         12001 TCP SPARK*  SHAREPORT ;
   ```

   Where `SPARK*` is the user or user ID that starts the scoring process. A wildcard is allowed.

   All scoring services on the same LPAR can now share and listen on the same port 11180 for scoring requests from applications. For high availability, each scoring service must be defined with a second but unique port, such as 11001, which is called the *administrative port*. The administrative port is required for processing deployment requests.

When a model is updated, the deployment service must notify all scoring services on the same LPAR so they can pick up the right version of the deployed model. Every scoring service must have its unique endpoint to receive the notification and every endpoint must be defined to a unique port.

b. If you run multiple scoring services across different LPARs, define the VIPADEFine and VIPABackup keywords for the VIPADynamic block in the TCP/IP profile to enable DDVIPA for workload distribution, as shown in Example 3-32. The VIPADISTribute keyword adds the required sysplex distributor definitions, which keeps all scoring services sysplex-aware.

Example 3-32 VIPADynamic block

```
VIPADYNAMIC
  VIPADEFINE TCP SPARK* SHAREPORT;
  VIPADISTRIBUTE DEFINE
        DISTMETHOD ROUNDROBIN <ip_address>
        PORT 13330
  DESTIP ALL
ENDVIPADYNAMIC
```

Consider setting up a backup DDVIPA on a separate LPAR by specifying the VIPABackup keyword, as shown in Example 3-33. This configuration helps ensure that the scoring services are available when one LPAR is down.

Example 3-33 Backup DDVIPA

```
VIPADYNAMIC
   VIPABACKUP 1 MOVE IMMED 255.255.255.255 <ip_address>
   VIPADISTRIBUTE DEFINE
       DISTMETHOD ROUNDROBIN <ip_address>
       PORT 13330
       DESTIP ALL
 ENDVIPADYNAMIC
```

2. Update the configuration of the scoring services through the Machine Learning for z/OS administration dashboard. The services can be stand-alone, in a scoring cluster, or created by a CICS-integrated scoring service. For more information, see Chapter 4, “Administration and operation ” on page 63.

3.5.2 Scaling an application cluster with extra compute nodes

The Machine Learning for z/OS application cluster uses a two-layer architecture for load balancing and high availability. As shown in Figure 3-7, the first layer the Keepalived and HAProxy technologies to monitor application cluster nodes. Keepalived performs load balancing and failover tasks, while HAProxy provides load balancing and high-availability services to the application cluster nodes. The second layer is a Kubernetes cluster that is made of master and worker nodes.

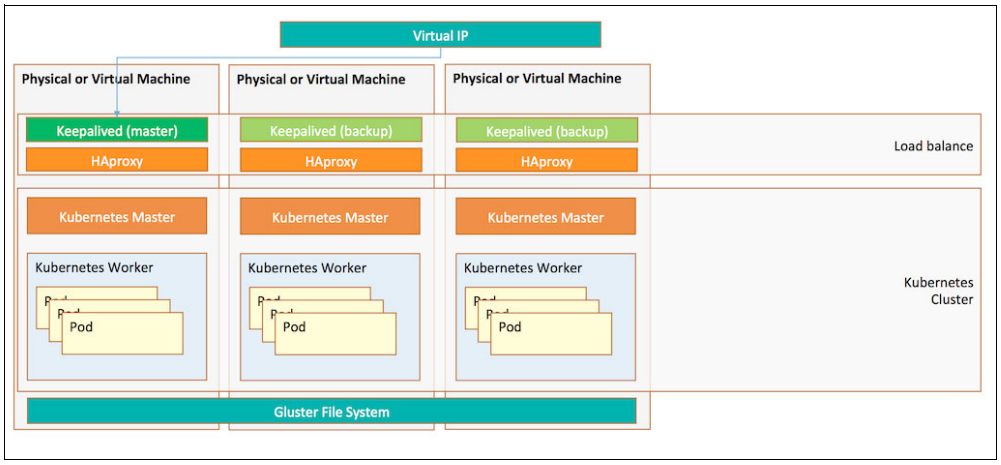


Figure 3-7 Two layer architecture for load balancing and high availability

With the layered architecture, you can scale out the application cluster by adding new compute nodes to satisfy the need of any growing workload. As shown in Figure 3-8, the Machine Learning for z/OS services with the maximum number of replicas (pods) are not automatically deployed onto the new node. If you redeploy these services, some of them might be pushed onto the new node.

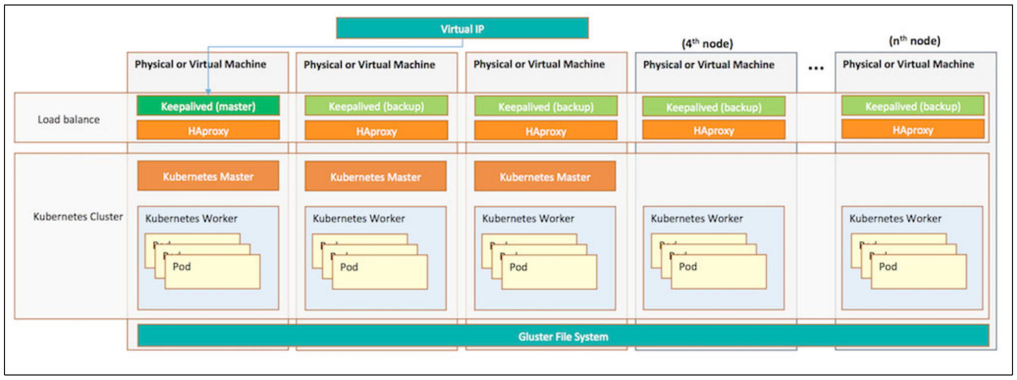


Figure 3-8 Scaling the cluster

For more information about how to add compute nodes to an application cluster, see Chapter 4, "Administration and operation " on page 63.

4

Administration and operation

After Machine Learning for z/OS is up and running, it is important to keep it that way. You can manage Machine Learning for z/OS from the web and command-line interfaces.

This chapter introduces the web-based administration dashboard and describes the UNIX commands that you can use to keep Machine Learning for z/OS systems healthy and its services uninterrupted.

This chapter includes the following topics:

4.1 Administration dashboard

The Machine Learning for z/OS administration dashboard enables system administrators and operators to easily monitor and manage various backend systems (for example, Kubernetes pods and nodes and Spark clusters), services, users, and other system resources through a web user interface. The interface consists of the pages or views that are listed in Table 4-1, all of which can be accessed through a sliding sidebar.

Table 4-1 Dashboard pages and views

Dashboard page	Description
Dashboard	Presents a visualization of network, CPU, disk, and memory usage at the cluster level.
Nodes	Presents a detailed view of network, CPU, disk, and memory usage at the node level.
Cluster Logs	Provides an entry point to Kubernetes cluster logs where you can view, search, and filter logs by node, container, time and date, and error level.
Services	Provides a view of all Kubernetes services, including the administration dashboard UI service, its status, and the pod and image in which it runs.
Pods	Displays the status of all pods that are defined in a cluster, with which you can view the details of a node and when necessary, and redeploy the node.
Users	Lists all current users and their permissions, which allows you to add users and assign privileges, change permissions, and remove users, if necessary.
System Configuration	Allows you to view or update the configuration settings of systems, including LDAP, Jupyter kernel gateway and Spark, and services, including the repository service.
Scoring Services	Displays the status of all current scoring services, which allows you to add services and start, stop, update, or remove services.
Spark Clusters	Displays the details of current Spark clusters, which allows you to add or remove remote clusters.
Alerts	A filterable list of alerts that are generated by the Kubernetes cluster in response to changes of state in the various services and pods.

4.1.1 Accessing the administration dashboard

Complete the following steps to access the administration dashboard pages:

1. Sign in the administration dashboard by using one of the following methods:
   - Log in to the IBM Machine Learning for z/OS user interface by using your user ID and password. Then, select **IBM Machine Learning for z/OS administration dashboard** from the application drop-down menu, as shown in Figure 4-1.

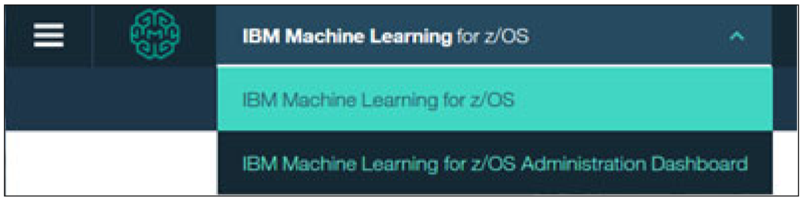

IBM Machine Learning for z/OS
IBM Machine Learning for z/OS
IBM Machine Learning for z/OS Administration Dashboard

Figure 4-1 Selecting IBM Machine Learning for z/OS administration dashboard

2. Enter the following administration dashboard URL, and then, authenticate by using your Machine Learning for z/OS user ID and password:

   `https://<ip_address>/admin-dashboard/`

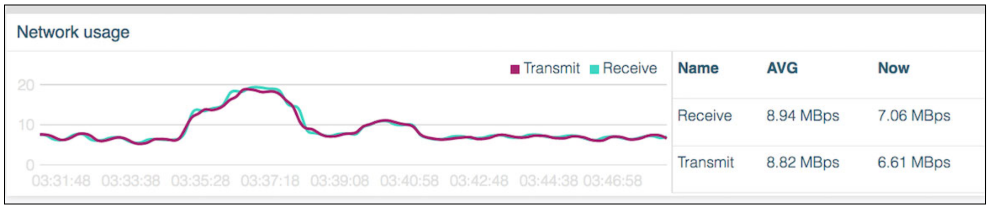
   Where `<ip_address>` is the IP address or host name of your Machine Learning for z/OS proxy server.
3. In the upper left corner of the Welcome page, click the three vertically stacked bars icon (☰) to activate the administration dashboard sidebar.
4. From the sidebar, select and start any of the administration dashboard pages or views.

Some of the administration dashboard pages are described next.

4.1.2 Dashboard page

The Dashboard page is the default page when you sign in the administration dashboard application. It features the visualization of CPU, memory, disk, and network utilization at the cluster level.

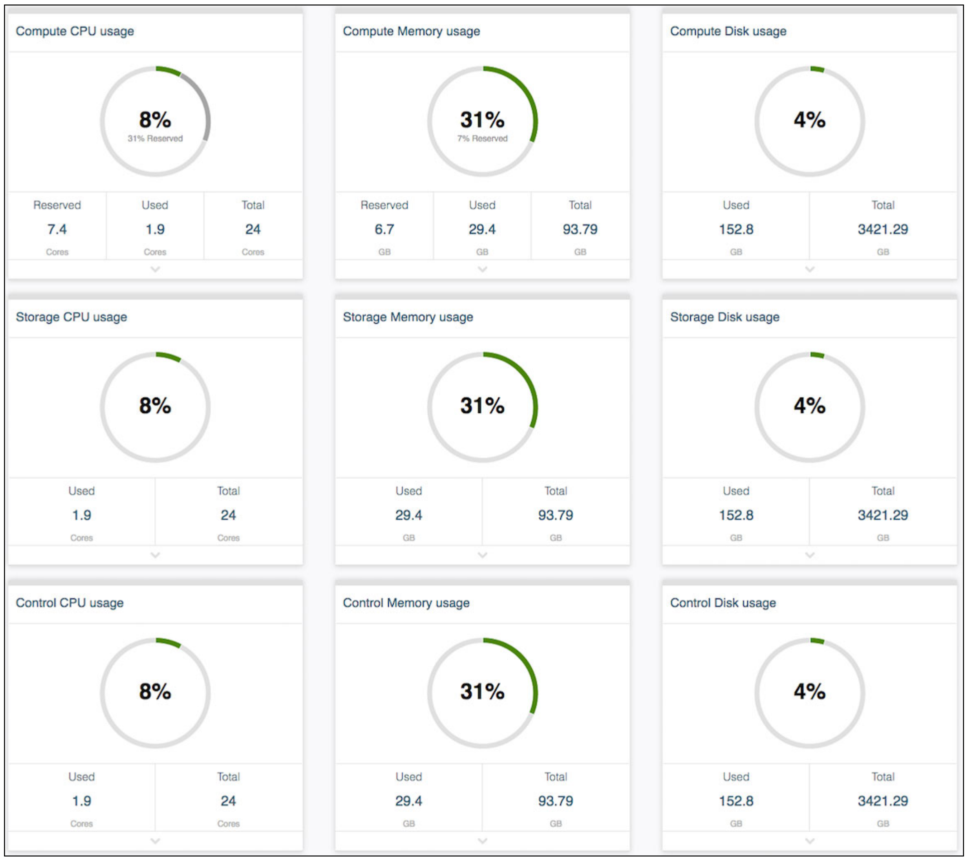
As shown in Figure 4-2, the visualization uses percentage (%) to measure resource usage, the colors of a traffic light (green, yellow, and red) to callout potential problems, and the continuing timeline to show the network data transmission rate. You can select any point on the timeline to view the number of bytes read and written across the nodes in the cluster.



Name	AVG	Now
Receive	8.94 MBps	7.06 MBps
Transmit	8.82 MBps	6.61 MBps

Figure 4-2 Network usage visualization

The visualization also shows the three types of nodes that Machine Learning for z/OS supports in a cluster: control, compute, and disk nodes. To maximize usage and reduce costs, each physical node is configured for multiple purposes and serves as a control, compute, and disk node, as shown in Figure 4-3 on page 66.


Compute CPU usage
8%
31% Reserved
Reserved 7.4 Cores | Used 1.9 Cores | Total 24 Cores

Compute Memory usage
31%
7% Reserved
Reserved 6.7 GB | Used 29.4 GB | Total 93.79 GB

Compute Disk usage
4%
Used 152.8 GB | Total 3421.29 GB

Storage CPU usage
8%
Used 1.9 Cores | Total 24 Cores

Storage Memory usage
31%
Used 29.4 GB | Total 93.79 GB

Storage Disk usage
4%
Used 152.8 GB | Total 3421.29 GB

Control CPU usage
8%
Used 1.9 Cores | Total 24 Cores

Control Memory usage
31%
Used 29.4 GB | Total 93.79 GB

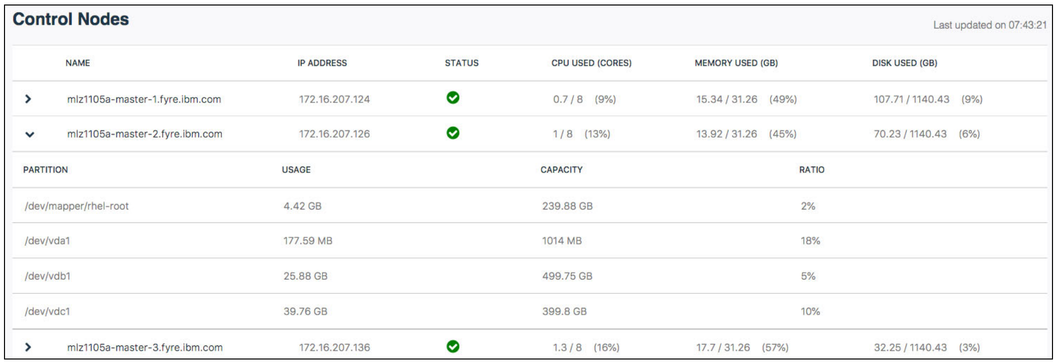
Control Disk usage
4%
Used 152.8 GB | Total 3421.29 GB

Figure 4-3 Cluster controls

The cluster that is shown in Figure 4-3 consists of three physical nodes. The physical nodes are represented by the columns in the visualization and the multipurpose configuration of each node is indicated by the rows. You can display more details about a node by clicking the down arrow at the bottom of each cell block.

4.1.3 Control Nodes page

The Control Nodes page (see Figure 4-4) uses a different set of metrics to present the nodes in a cluster. The view includes such details as the host name, IP address, and status, and its CPU, memory, and storage usage information. As with the Dashboard, the details are listed under the control, compute, and storage categories.


Control Nodes

Last updated on 07:43:21

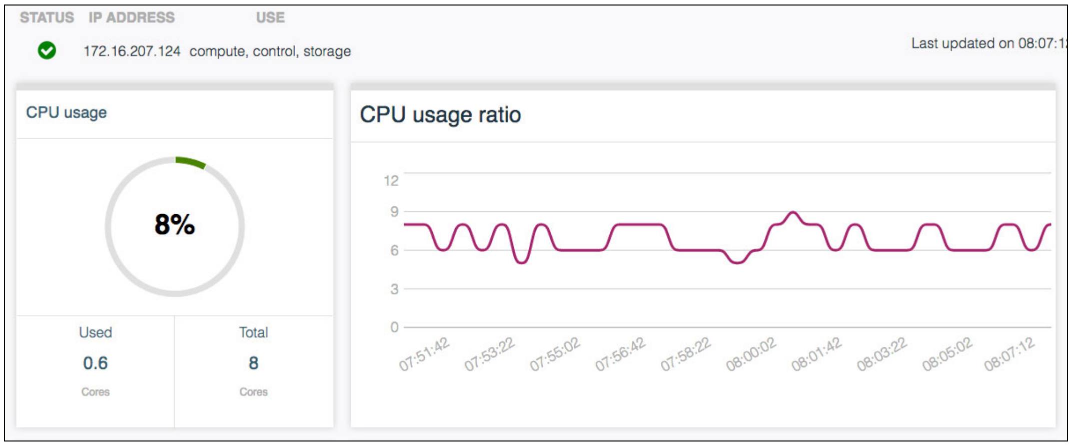
	NAME	IP ADDRESS	STATUS	CPU USED (CORES)	MEMORY USED (GB)	DISK USED (GB)
›	mlz1105a-master-1.fyre.ibm.com	172.16.207.124	✓	0.7 / 8 (9%)	15.34 / 31.26 (49%)	107.71 / 1140.43 (9%)
⌄	mlz1105a-master-2.fyre.ibm.com	172.16.207.126	✓	1 / 8 (13%)	13.92 / 31.26 (45%)	70.23 / 1140.43 (6%)

PARTITION	USAGE	CAPACITY	RATIO
/dev/mapper/rhel-root	4.42 GB	239.88 GB	2%
/dev/vda1	177.59 MB	1014 MB	18%
/dev/vdb1	25.88 GB	499.75 GB	5%
/dev/vdc1	39.76 GB	399.8 GB	10%

›	mlz1105a-master-3.fyre.ibm.com	172.16.207.136	✓	1.3 / 8 (16%)	17.7 / 31.26 (57%)	32.25 / 1140.43 (3%)

Figure 4-4 Control Nodes page

Clicking the right arrow shows the partitions that are attached to the node and the capacity of each partition. Clicking the host name starts a separate page with the visualization of CPU, memory, and storage usage of the node, as shown in Figure 4-5.

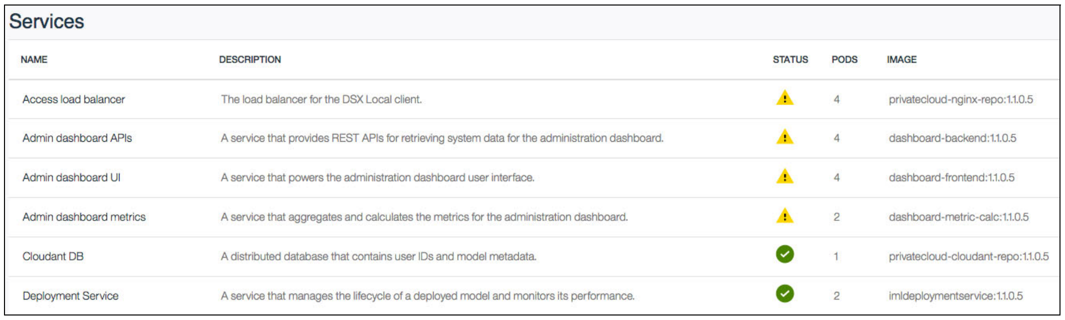


STATUS	IP ADDRESS	USE
✓	172.16.207.124	compute, control, storage

Figure 4-5 CPU usage example

4.1.4 Services page

As shown in Figure 4-6, the Services page lists and describes all of the different types of Kubernetes services in a Machine Learning for z/OS installation. It also displays the status of each service and identifies the pod in which the service deployed.

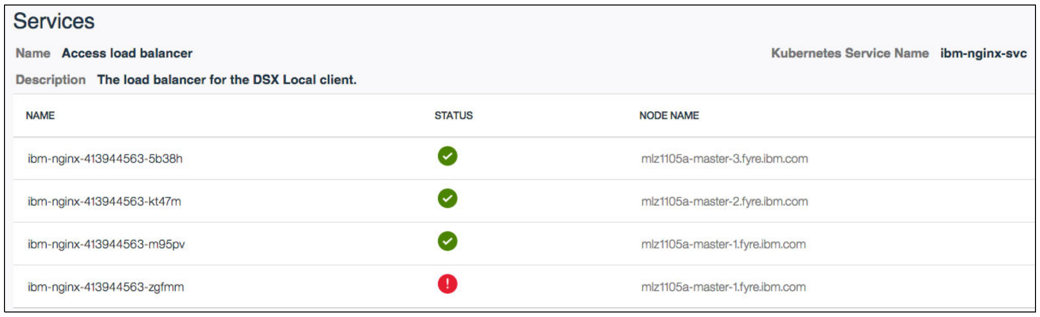


Services

NAME	DESCRIPTION	STATUS	PODS	IMAGE
Access load balancer	The load balancer for the DSX Local client.	⚠	4	privatecloud-nginx-repo:1.1.0.5
Admin dashboard APIs	A service that provides REST APIs for retrieving system data for the administration dashboard.	⚠	4	dashboard-backend:1.1.0.5
Admin dashboard UI	A service that powers the administration dashboard user interface.	⚠	4	dashboard-frontend:1.1.0.5
Admin dashboard metrics	A service that aggregates and calculates the metrics for the administration dashboard.	⚠	2	dashboard-metric-calc:1.1.0.5
Cloudant DB	A distributed database that contains user IDs and model metadata.	✓	1	privatecloud-cloudant-repo:1.1.0.5
Deployment Service	A service that manages the lifecycle of a deployed model and monitors its performance.	✓	2	imldeploymentservice:1.1.0.5

Figure 4-6 Services page

You can use the Services page (see Figure 4-7) as a problem diagnosis aid. Clicking the name of any service starts a separate page that includes more information, such as the pod in which the service is deployed and the node on which the service runs.



Services

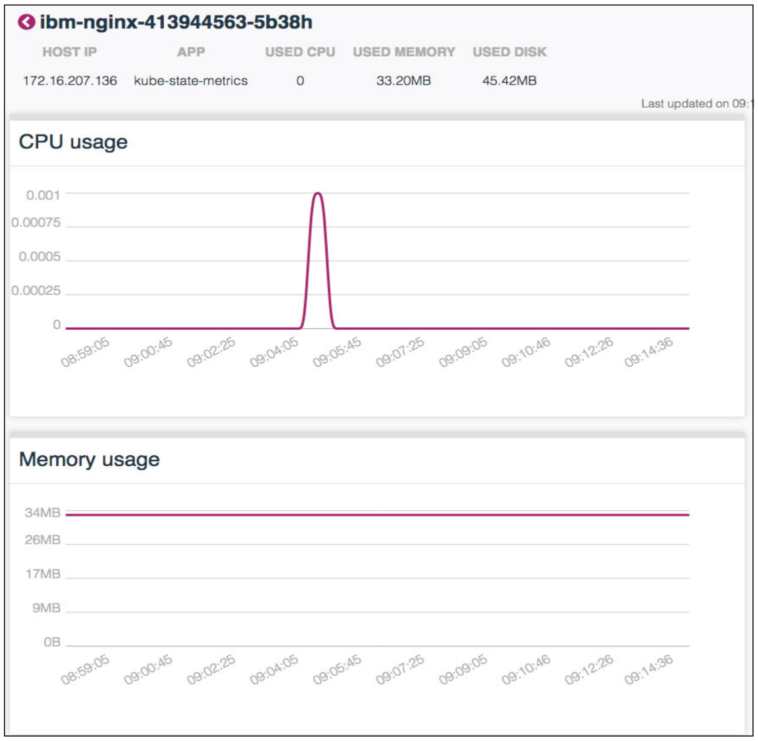
Name Access load balancer **Kubernetes Service Name** ibm-nginx-svc

Description The load balancer for the DSX Local client.

NAME	STATUS	NODE NAME
ibm-nginx-413944563-5b38h	✓	mlz1105a-master-3.fyre.ibm.com
ibm-nginx-413944563-kt47m	✓	mlz1105a-master-2.fyre.ibm.com
ibm-nginx-413944563-m95pv	✓	mlz1105a-master-1.fyre.ibm.com
ibm-nginx-413944563-zgfmm	!	mlz1105a-master-1.fyre.ibm.com

Figure 4-7 More information in Services page

Clicking the name of any pod starts another page with the visualization of resource usage by the specified pod, as shown in Figure 4-8.



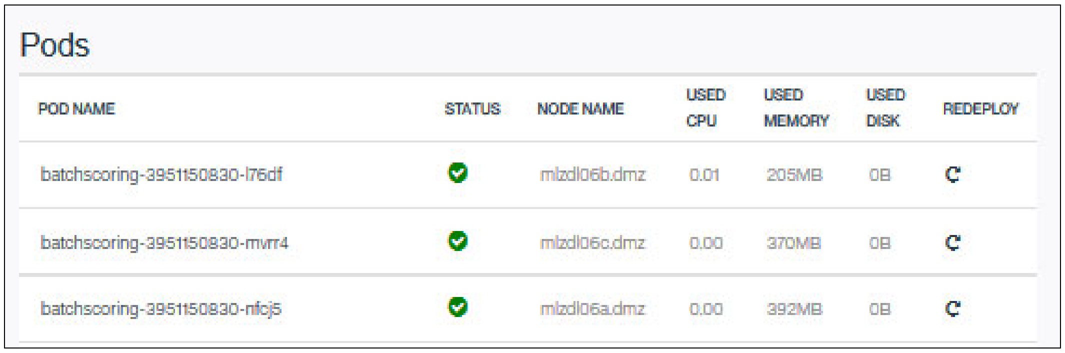
HOST IP	APP	USED CPU	USED MEMORY	USED DISK
172.16.207.136	kube-state-metrics	0	33.20MB	45.42MB

Figure 4-8 Pod-specific visualization of resource usage

Note: Although the memory and storage usage graphs label the units for the y-axis, the CPU usage graph does not. A value of 1.0 equals one thread on a core/IFL, or one virtual processor, depending on the type of CPU that the Linux or Linux on Z system uses.

4.1.5 Pods page

As shown in Figure 4-9, the Pods page flattens the hierarchy of services and pods so that you can scan the status of all pods at once and quickly spot the pods that are in trouble. You can click the name of any pod to display the resource consumption information. You also can click the play button to redeploy a problematic pod after the issue is resolved.



Pods

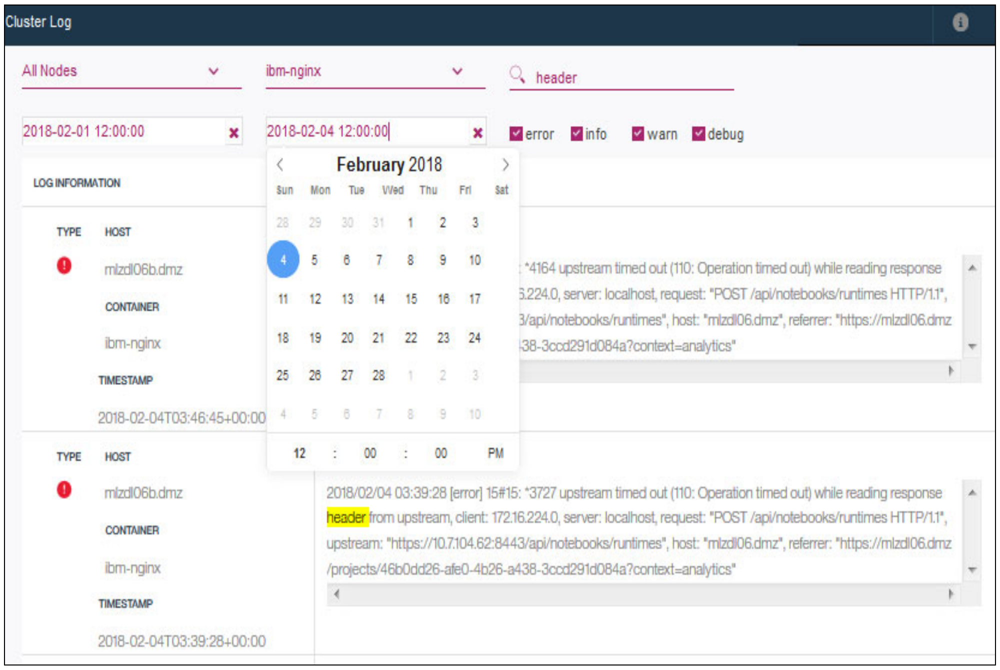
POD NAME	STATUS	NODE NAME	USED CPU	USED MEMORY	USED DISK	REDEPLOY
batchscoring-3951150830-l76df	✓	mlzdl06b.dmz	0.01	205MB	0B	C
batchscoring-3951150830-mvrr4	✓	mlzdl06c.dmz	0.00	370MB	0B	C
batchscoring-3951150830-nfcj5	✓	mlzdl06a.dmz	0.00	392MB	0B	C

Figure 4-9 Pods page

4.1.6 Cluster Logs page

The Cluster Log page (see Figure 4-10) provides you the flexibility to display all or some of the logs that Machine Learning for z/OS services generate in a cluster. You can use keywords to search for specific logs or apply the following filters to narrow your search:

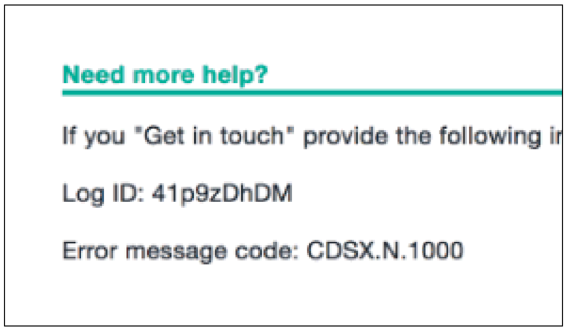
- Node
- Container
- Start and end date and time
- Severity of error condition (info, debug, warn, and error)



```
TYPE  HOST
      mlzdl06b.dmz
      CONTAINER
      ibm-nginx
      TIMESTAMP
      2018-02-04T03:46:45+00:00

: *4164 upstream timed out (110: Operation timed out) while reading response
5.224.0, server: localhost, request: "POST /api/notebooks/runtimes HTTP/1.1",
3/api/notebooks/runtimes", host: "mlzdl06.dmz", referrer: "https://mlzdl06.dmz
38-3ccd291d084a?context=analytics"

TYPE  HOST
      mlzdl06b.dmz
      CONTAINER
      ibm-nginx
      TIMESTAMP
      2018-02-04T03:39:28+00:00

2018/02/04 03:39:28 [error] 15#15: *3727 upstream timed out (110: Operation timed out) while reading response
header from upstream, client: 172.16.224.0, server: localhost, request: "POST /api/notebooks/runtimes HTTP/1.1",
upstream: "https://10.7.104.62:8443/api/notebooks/runtimes", host: "mlzdl06.dmz", referrer: "https://mlzdl06.dmz
/projects/46b0dd26-afe0-4b26-a438-3ccd291d084a?context=analytics"
```

Figure 4-10 Cluster Log page

Performing a keyword search is an especially useful feature. For example, you can quickly retrieve the specific log if you search “41p9zDhDM”, as shown in Figure 4-11.



Need more help?

If you "Get in touch" provide the following i

Log ID: 41p9zDhDM

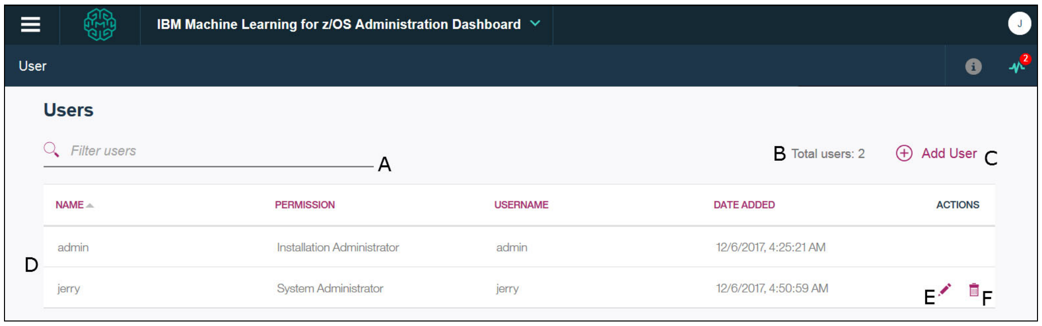
Error message code: CDSX.N.1000

Figure 4-11 Keyword search

The information in the log with the “41p9zDhDM“ ID can help you quickly identify the pod that is associated with the error and eventually resolve the error.

4.1.7 Users page

The Users page (see Figure 4-12) enables you to manage user access and privileges. In addition to listing all currently authorized users and their permission level, you can add a user and update or remove a user, as shown in Figure 4-12.



NAME	PERMISSION	USERNAME	DATE ADDED	ACTIONS
admin	Installation Administrator	admin	12/6/2017, 4:25:21 AM	
jerry	System Administrator	jerry	12/6/2017, 4:50:59 AM	

Figure 4-12 Users page

The Users page includes the following elements, as shown in Figure 4-12:

- A: Filters the view of users
- B: Total number of users
- C: Add a user
- D: List of users
- E: Edit a user
- F: Delete a user

4.1.8 System Configuration page

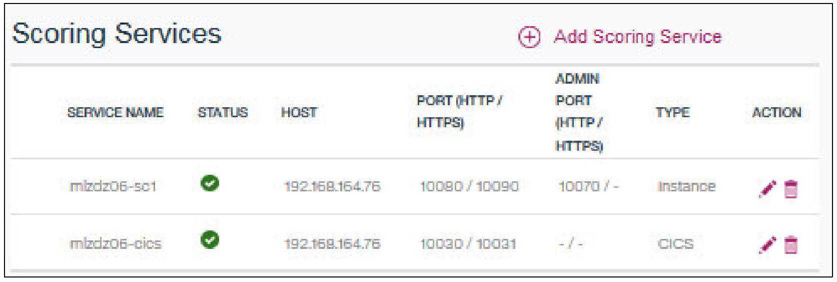
By using the System Configuration page, you can view or update the configuration settings of services (for example, the repository service) and systems (for example, LDAP, Jupyter kernel gateway, and Spark) in a cluster. The System Configuration page also can be used to change the port number of the operation handling service on z/OS.

To ensure security and privacy, the password fields are left blank on the System Configuration page. You must complete those fields with the correct passwords when you update the following configuration settings:

- LDAP
- Repository service
- SSL certificate
- Kubernetes master login

4.1.9 Scoring Services page

The Scoring Services page (see Figure 4-13) displays the status and other details (including port numbers) of all the scoring services that are available in a cluster. You can add a scoring service with z/OSMF and start, stop, update, or remove a service.

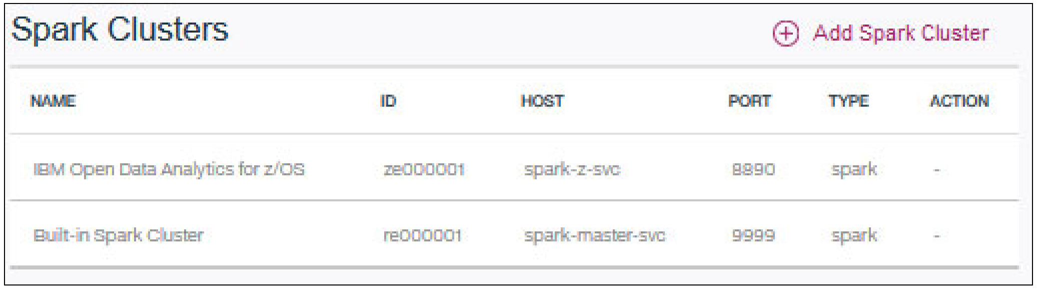


SERVICE NAME	STATUS	HOST	PORT (HTTP / HTTPS)	ADMIN PORT (HTTP / HTTPS)	TYPE	ACTION
mlzdz06-sc1	✓	192.168.164.76	10080 / 10090	10070 / -	Instance	
mlzdz06-cics	✓	192.168.164.76	10030 / 10031	- / -	CICS	

Figure 4-13 Scoring Services page

4.1.10 Spark Clusters page

The Spark Cluster page (see Figure 4-14) lists and displays the information about built-in, local, and remote clusters in the Machine Learning for z/OS installation. The built-in and local (IzODA Spark) clusters are defined and enabled during the Machine Learning for z/OS installation process, which cannot be modified or removed. You also can add, update, or remove a remote cluster.



NAME	ID	HOST	PORT	TYPE	ACTION
IBM Open Data Analytics for z/OS	ze000001	spark-z-svc	8890	spark	-
Built-in Spark Cluster	re000001	spark-master-svc	9999	spark	-

Figure 4-14 Spark clusters page

4.2 Administering by using the administration dashboard

You can use the administration dashboard to perform some common administration and operation tasks, including managing users, scoring services, compute nodes, and remote clusters.

4.2.1 Managing users and privileges

All users must be authorized and authenticated to access the Machine Learning for z/OS user interface and administration dashboard. You can use the administration dashboard to manage these users and their privileges.

Complete the following steps to add a user and assign privileges:

1. Browse to the Users page of the administration dashboard. For more information, see 4.1.1, “Accessing the administration dashboard” on page 65.
2. Click **Add User** and specify the full name and valid user name or user ID of the new user. A user name is valid if it is defined in the LDAP user directory for Machine Learning for z/OS.
3. Choose the appropriate Access Level for the user. An access level is determined by the combined permissions of one or more user roles. Each user role is predefined with a specific set of privileges. You can assign a user one or more of the roles that are listed in Table 4-2.

Table 4-2 Privileges and user roles

Privilege	**User role**				
	Installation administrator (installadm)	**System administrator (sysadm)**	**Machine Learning administrator (mladm)**	**Model developer (devuser)**	**Application developer (appuser)**
Monitoring nodes (administration dashboard)	Yes	Yes			
Managing kernels (administration dashboard)	Yes	Yes			
Deploying services (administration dashboard)	Yes	Yes			
Managing users and granting privileges (administration dashboard)	Yes	Yes			
Granting self privileges (administration dashboard)		Yes			

Using visual model builder (Notebook)		Yes	Yes	Yes	
Using canvas (Notebook)		Yes	Yes	Yes	
Viewing, importing, and deleting self-owned models (Notebook)		Yes	Yes	Yes	
Viewing, importing, and deleting any model		Yes	Yes		
Creating and deleting model deployments		Yes	Yes		
Testing model predictions		Yes	Yes		
Running model evaluations		Yes	Yes		
Scoring any model		Yes	Yes	Yes	Yes

4. Click **Add** to add the new user.
5. Verify that the new user appears in the list that is shown on the Users page.

Editing a user

Complete the following steps to edit a user:

1. On the Users page, select a user that you want to update.
2. From the ACTIONS menu for the selected user, click the **Edit** (pencil) icon.
3. Modify the Access Level of the user. The user name cannot be edited before it is predefined in LDAP.
4. Click **Save** to update the user.

Remove a user

Complete the following steps to remove a user:

1. On the Users page, select a user that you want to delete.
2. From the ACTIONS menu for the selected user, click the **Delete** (trash bin) icon.
3. Click **Delete** to confirm the removal.

4.2.2 Managing scoring services

You can create a scoring service by running the supplied deployment scripts during the installation process or by using the z/OSMF workflow service through the administration dashboard.

After a scoring service is created, you must add it to the Scoring Services page of the administration dashboard so that the new service is available for model deployment in the Machine Learning for z/OS user interface.

Add a scoring service

Complete the following steps to add a scoring service:

1. Ensure that the target server is started so that the management services can connect to and validate it.
2. Browse to the Scoring Services page of the administration dashboard. For more information, see 4.1.1, “Accessing the administration dashboard” on page 65.
3. Click **Add Scoring Service**. In the Service type and cluster configuration panel, select one of the following service types:
   - Scoring service refers to the scoring service that you created (or will create) on z/OS.
   - CICS-integrated scoring service refers to the scoring service that you created in a CICS region.

   If you select Scoring service, indicate whether you want to add the scoring service to a cluster (the default is No). If you choose **Yes**, select whether you want to add the scoring service to an existing or new cluster:

   a. To add to an existing cluster, select a cluster from the list.

   b. To add to a new cluster, specify a name, host, and HTTP or HTTPS port to create the cluster.
4. In the Service host and port configuration panel, specify a name and host for the scoring service.

For the Scoring Service Host field, the host name or IP address must be consistent with how the `scoring_ip` property is defined in the configuration file for the target scoring server. The two values must match for the scoring service to work correctly.

If the scoring service is added to a cluster, the Scoring Service HTTP Port and Scoring Service HTTPS Port fields are automatically populated with the values from the cluster. Otherwise, specify the HTTP or HTTPS port.

Optionally, specify the scoring service administrative HTTP or HTTPS port. Although a scoring service port number can be common and used by multiple scoring services, an administrative port number must be unique.

Specify the HTTP or HTTPS port for individual scoring services only if those services run on the same z/OS LPAR and are instances of the same cluster. If the scoring service is added to a cluster, you must specify the common ports and the administrative ports. For more information, see 3.5.1, “Configuring TCP/IP for port sharing and load balancing ” on page 59.

Attention: Complete steps 5 and 6 only if you want to use z/OSMF to create a scoring service or manage a scoring service.

5. In the z/OSMF configuration panel, select the option to expand the panel.

   Specify the name of the z/OS system, location of the Machine Learning for z/OS installation on the system, and URL, user ID, and password for remotely accessing z/OSMF. Ensure that the user ID includes the privileges to use the z/OSMF workflow service and to create, configure, start, and stop the new scoring service.

   Click **Validate** to verify the information that you provided. The z/OSMF workflow service also uses the information to verify whether the scoring service that you specified in the previous panel exists on the z/OS system. Consider the following points:

   – If the validation fails, resolve the issues that are identified in the error message and then, click **Validate** again.

   – If the validation is successful and the scoring service that you specified exists, go to Step 7.

   – If the validation is successful and the scoring service that you specified does not exist, continue to the next configuration panel to create the service.

6. In the Additional configuration for scoring service “serviceName” panel, specify the Flask service, authentication key, SSL certificate you use for Machine Learning for z/OS, JVM memory sizes, and PMML pool sizes:

   – For Flask service, specify the Python Flask HTTP or HTTPS port.

   – For Authentication, specify the path to where the public key for authentication is located.

   – For SSL, specify the keystore location and password, the truststore location and password, and the private key location and password.

   – For JVM, specify the initial and maximal memory sizes.

   – For PMML, specify the initial and maximal pool sizes.

7. Click **Add** at the bottom of the page to add the service to the Scoring Services page.

8. Verify that the scoring service appears on the Scoring Services page. The scoring service is now available for use when you create a model deployment in the Machine Learning for z/OS user interface.

9. Start the scoring service by clicking the **Play** icon under the ACTIONS menu on the Scoring Services page.

Editing a scoring service

Complete the following steps to edit a scoring service:

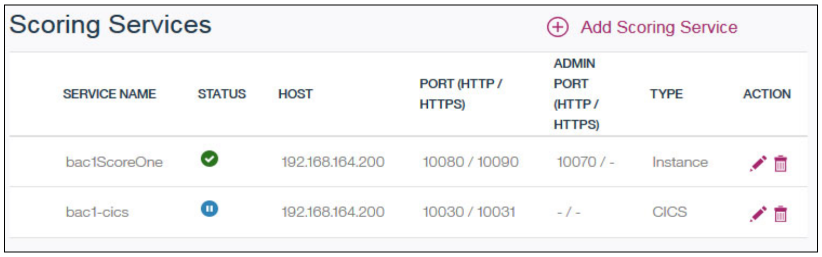
1. On the Scoring Services page, select a service that you want to update.
2. From the Actions menu for the selected service, click the **Edit** (pencil) icon.
3. Modify the fields as needed and click **Save** to update the service.

Removing a scoring service

Complete the following steps to remove a scoring service:

1. On the Scoring Services page, select a service that you want to delete.

2. Ensure that the server for the scoring service is stopped and that the server status is Paused on the services list as shown in Figure 4-15 on page 76. The server must be stopped before the corresponding service can be deleted.


Scoring Services

SERVICE NAME	STATUS	HOST	PORT (HTTP / HTTPS)	ADMIN PORT (HTTP / HTTPS)	TYPE	ACTION
bac1ScoreOne	✓	192.168.164.200	10080 / 10090	10070 / -	Instance	
bac1-cics	⏸	192.168.164.200	10030 / 10031	- / -	CICS	

Figure 4-15 Scoring Services

3. From the ACTIONS menu for the selected service, click the **Delete** (trash bin) icon.
4. Click **Delete** to confirm the removal.

4.2.3 Managing remote Spark clusters

Machine Learning for z/OS uses built-in, local (z/OS IzODA Spark), and remote Spark clusters. You can use the administration dashboard to add a remote Spark cluster.

Adding a remote Spark cluster

Complete the following steps to add a remove Spark cluster:

1. Browse to the Spark Clusters page of the administration dashboard. For more information, see 4.1.1, "Accessing the administration dashboard" on page 65.
2. Click **Add Spark Cluster** and specify a name for the new cluster. The name must be alphanumeric and can be up to 26 characters in length. Special characters other than "_" and "-" are not allowed.

   Specify an IP address and a port number for the cluster. The IP address and port number are used by the Livy service of the Spark cluster (the Livy API).
3. Click **Add** to add the cluster.
4. Verify that the new cluster appears on the cluster list on the Spark Clusters page. The new Spark cluster is now available when you create or update a notebook in the Machine Learning for z/OS user interface.
5. Enable the new cluster to access the Machine Learning for z/OS library. Issue the following commands to mount the Machine Learning for z/OS library on every master and worker node in the cluster:

   ```
   mkdir /<iml-home>
   mount -t glusterfs <MLZIP>:<iml-home> /<iml-home>
   ```

   Where:

   – *<iml-home>* is a Gluster volume that contains the Machine Learning for z/OS library
   – *<MLZIP>* is the IP address of your Machine Learning for z/OS proxy server

Removing a remote Spark cluster

Complete the following steps to removing a remote Spark cluster:

1. In the Spark Clusters page, select a cluster that you want to delete.
2. From the ACTIONS menu for the selected cluster, click the **Delete** (trash bin) icon.
3. Click **Remove** to confirm the removal.

4.2.4 Adding compute nodes

As described in 4.1.2, “Dashboard page” on page 65 and 4.1.3, “Control Nodes page” on page 66, a node in a Machine Learning for z/OS cluster is configured to run as a storage, control, and compute node. However, you can use the administration dashboard to add a dedicated compute node to run machine learning services only.

Complete the following steps:

1. Allocate and install a Linux or Linux on Z server that meets the basic system capacity requirement as described in 2.3.1, “Basic system capacity” on page 12. Consider the following points:
   - Ensure that the system meets all other requirements, such as user name and password, access authority, auxiliary storage, network ports, cluster subnet and firewall configuration, and RHN plug-in disablement. If you install the new node outside of the subnet, ensure that the new node and the existing Kubernetes cluster are authorized to communicate with each other.
   - Ensure that the server is up and running.
2. Browse to the Nodes page of the administration dashboard. For more information, see 4.1.1, “Accessing the administration dashboard” on page 65.
3. In the Compute Nodes panel, click **Add Node**, and enter the IP address of the node that you want to add, specify the path to the mounted auxiliary storage disk, and enter the user name and password. The user name must have sudo or root authority.
4. Click **Add** to add the node. A progress window shows the completion status of all nine installation steps. You can exit the window by clicking **Done** at the completion of the steps or **Close** at any time. Clicking **Close** does *not* interrupt or end the installation process.
5. Verify that the new node is successfully added and listed under Compute nodes.

4.2.5 Updating system configurations

After the initial Machine Learning for z/OS installation, you update your system configuration if any setting is changed. Complete the following steps:

1. Browse to the System Configuration page of the administration dashboard. For more information, see 4.1.1, “Accessing the administration dashboard” on page 65.

   For security and privacy, all password fields are left blank when you access the System Configuration page. You must reenter all of the required passwords.
2. Follow the instructions as described in Step 5 in 3.4.3, “Installing machine learning services on Linux or Linux on Z” on page 53 to complete the system configuration update.

4.3 Administering by using commands on Linux or Linux on Z

Although the administration dashboard provides a convenient way to manage aspects of the Machine Learning for z/OS cluster, the command-line interface presents another (and for some, the preferred) way to check system health.

Because many Machine Learning for z/OS services and processes run on Linux or Linux on Z, you might find UNIX-based commands the quickest way to monitor and manage those processes and services. For example, the services on Linux or Linux on Z are deployed as Docker containers in the multi-node cluster that is managed by Kubernetes and the container services all use the GlusterFS file system. You can use UNIX or Linux commands to monitor and check the health of the operating system, nodes, cluster, GlusterFS, and Kubernetes services.

The services on Linux or Linux on Z are implemented in a multi-node cluster. It is important to perform system health checks on all the nodes in the cluster. Consider the use of a separate terminal for each node or setting up password-less authentication in SSH so that commands can be issued from the master node to the worker nodes in the cluster. Use the root user ID for all system checks unless other IDs are deemed necessary.

4.3.1 Monitoring system resource usage

File system storage, memory, and CPU are key resources that can negatively affect the performance of any operating system. Regularly check the storage, memory, and CPU use on the Linux or Linux on Z operating system in the Machine Learning for z/OS environment, which ensures the adequacy of those key resources to support the changing workload.

File system storage

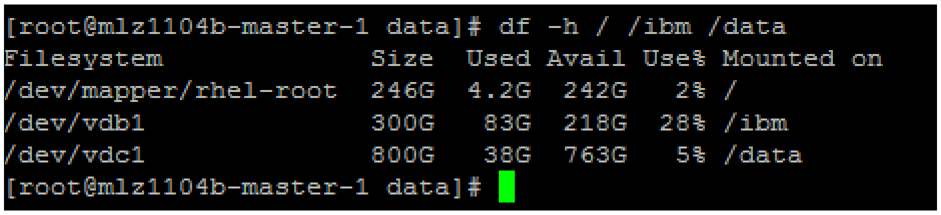
Consider the use of LVM-based file systems for the operating system mount points and Machine Learning for z/OS storage needs. An LVM file system allows you to increase the storage without requiring a system outage.

Typically, monitor the file systems that are listed in Table 4-3 (names can vary).

Table 4-3 File systems

File system	Description
	Typically contains the operating system files and log files (in the `/var/log/` directory).
/ibm	Contains Machine Learning for z/OS runtime code and overlay-mounted file systems.
/data	Contains the data for various Kubernetes pods and containers. Each of the subdirectories represents bricks in the Gluster clustered file system. Each brick from the three nodes comprise a Gluster volume, which is mounted by the various Docker containers that are used in the IBM MLz solution.

Run the **df** command to check and ensure enough space in each file system, as shown in Figure 4-16.



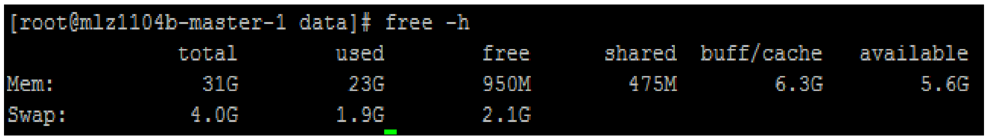
```
[root@mlz1104b-master-1 data]# df -h / /ibm /data
Filesystem              Size  Used Avail Use% Mounted on
/dev/mapper/rhel-root   246G  4.2G  242G   2% /
/dev/vdb1               300G   83G  218G  28% /ibm
/dev/vdc1               800G   38G  763G   5% /data
[root@mlz1104b-master-1 data]#
```

Figure 4-16 Using the df command

If necessary, increase the storage in each file system to meet the demand of your workload.

Memory

Memory usage varies greatly, depending on your workload. Run the **free -h** command, as shown in Figure 4-17, to ensure that sufficient memory is available for processing your workload.



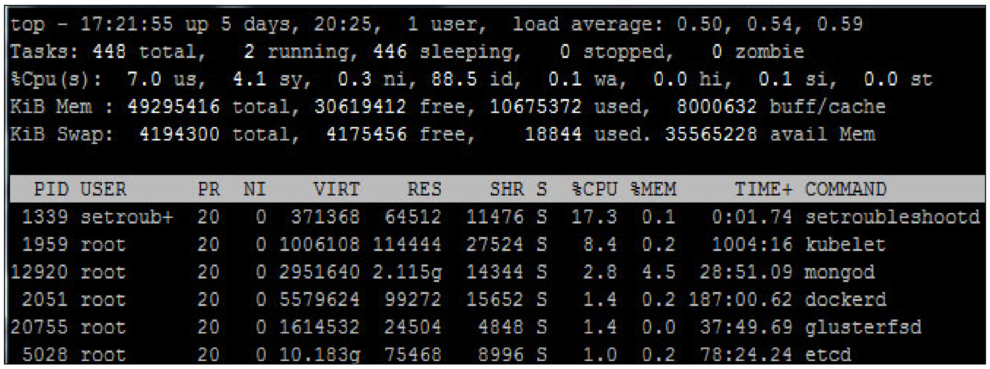
```
[root@mlz1104b-master-1 data]# free -h
              total        used        free      shared  buff/cache   available
Mem:            31G         23G        950M        475M        6.3G        5.6G
Swap:          4.0G        1.9G        2.1G
```

Figure 4-17 Using the free -h command

Consider adding memory if significant use of the swap file exists or more paging space if the system must support heavier workload.

CPU

System activities that use CPU are always running in the background. For example, a typical 8-CPU Linux configuration uses approximately up to 7-10% CPU when the system is idle. Run the **top** command, as shown in Figure 4-18, to monitor the system background activities and minimize their CPU use



```
top - 17:21:55 up 5 days, 20:25,  1 user,  load average: 0.50, 0.54, 0.59
Tasks: 448 total,   2 running, 446 sleeping,   0 stopped,   0 zombie
%Cpu(s):  7.0 us,  4.1 sy,  0.3 ni, 88.5 id,  0.1 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem : 49295416 total, 30619412 free, 10675372 used,  8000632 buff/cache
KiB Swap:  4194300 total,  4175456 free,    18844 used. 35565228 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1339 setroub+  20   0  371368  64512  11476 S  17.3  0.1   0:01.74 setroubleshootd
 1959 root      20   0 1006108 114444  27524 S   8.4  0.2   1004:16 kubelet
12920 root      20   0 2951640 2.115g  14344 S   2.8  4.5  28:51.09 mongod
 2051 root      20   0 5579624  99272  15652 S   1.4  0.2 187:00.62 dockerd
20755 root      20   0 1614532  24504   4848 S   1.4  0.0  37:49.69 glusterfsd
 5028 root      20   0 10.183g  75468   8996 S   1.0  0.2  78:24.24 etcd
```

Figure 4-18 Use of top command

System logs

In addition to the commands, you can check how the key system resources are used in the system log files. For example, you can find the log file that Kubernetes generates in the `/var/log/messages` directory. The volume of the messages in the log varies, depending on the system workload. The heavier the workload, the longer the log file, which can make the file difficult to parse and use.

Consider splitting the kublet and docker messages into separate log files by adding new rsyslog rules. To do so, complete the following steps:

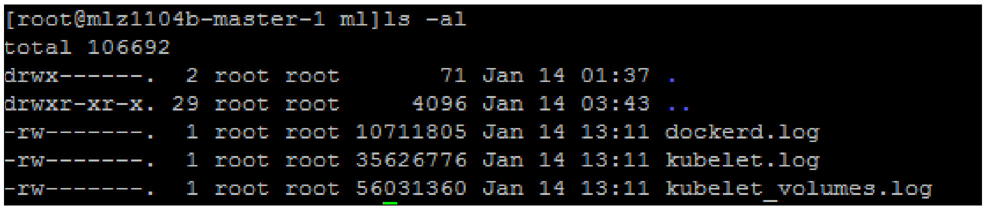
1. Add the following lines to the top of the RULES section in `/etc/rsyslog.conf` file to divest Kubernetes messages into three different log files:

   ```
   :msg, contains, "kubelet_volumes.go"   -/var/log/ml/kubelet_volumes.log

   & ~

   :syslogtag, isequal, "kubelet:" -/var/log/ml/kubelet.log

   & ~

   :syslogtag, isequal, "dockerd:" -/var/log/ml/dockerd.log

   & ~
   ```

2. Save the changes and restart rsyslog by running the following command:

   ```
   systemctl restart rsyslog
   ```

3. Very that the `/var/log/ml` directory contains the three new log files, as shown in Figure 4-19.



```
[root@mlz1104b-master-1 ml]ls -al
total 106692
drwx------.  2 root root       71 Jan 14 01:37 .
drwxr-xr-x. 29 root root     4096 Jan 14 03:43 ..
-rw-------.  1 root root 10711805 Jan 14 13:11 dockerd.log
-rw-------.  1 root root 35626776 Jan 14 13:11 kubelet.log
-rw-------.  1 root root 56031360 Jan 14 13:11 kubelet_volumes.log
```

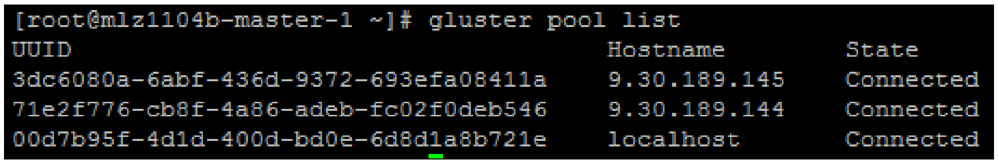
Figure 4-19   Three new log files

4.3.2 Checking the status of the GlusterFS file system

Each node in the Machine Learning for z/OS cluster acts as a GlusterFS storage server to provide the ‘bricks” for implementing GlusterFS replicated volumes. The GlusterFS replicated volumes provide redundancy in the event of a lost node. These volumes are mounted by various Kubernetes services. Periodically check the status of the GlusterFS pool and volumes and ensure that the volumes are available to the Kubernetes services.

GlusterFS pool

A GlusterFS pool is made of the nodes in a cluster. Run the **`gluster pool list`** command to display the status of the nodes, which should be in “Connected” state (see Figure 4-20).



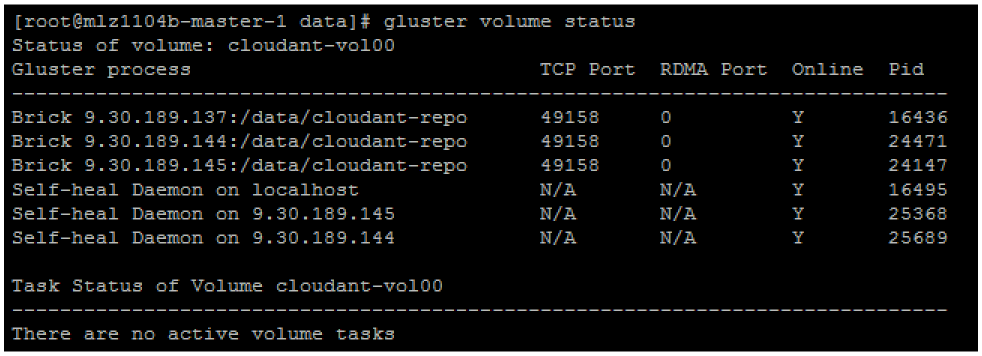
```
[root@mlz1104b-master-1 ~]# gluster pool list
UUID                                    Hostname        State
3dc6080a-6abf-436d-9372-693efa08411a    9.30.189.145    Connected
71e2f776-cb8f-4a86-adeb-fc02f0deb546    9.30.189.144    Connected
00d7b95f-4d1d-400d-bd0e-6d8d1a8b721e    localhost       Connected
```

Figure 4-20 Node status

If any of the nodes is not in “Connected” state, check the logs in the `/var/log/glusterfs` directory for more information and correct the problem.

GlusterFS volume

To check the status of all the volumes, run the **`gluster volume status`** command. The command returns an output that segments the status information about each volume. The segment regarding the cloudant-vol00 volume is shown in Figure 4-21.



```
[root@mlz1104b-master-1 data]# gluster volume status
Status of volume: cloudant-vol00
Gluster process                             TCP Port  RDMA Port  Online  Pid
------------------------------------------------------------------------------
Brick 9.30.189.137:/data/cloudant-repo      49158     0          Y       16436
Brick 9.30.189.144:/data/cloudant-repo      49158     0          Y       24471
Brick 9.30.189.145:/data/cloudant-repo      49158     0          Y       24147
Self-heal Daemon on localhost               N/A       N/A        Y       16495
Self-heal Daemon on 9.30.189.145            N/A       N/A        Y       25368
Self-heal Daemon on 9.30.189.144            N/A       N/A        Y       25689

Task Status of Volume cloudant-vol00
------------------------------------------------------------------------------
There are no active volume tasks
```

Figure 4-21 Cloudant-vol00 volume segment

Look for any line in the output that indicates that a brick is down or a self-heal daemon cannot be reached. The status for these conditions appears as a single ‘N’ character in the online column. If you need more information, review the log file in the `/var/log/glusterfs` directory.

If you want to check the status of a specific volume, append the volume name at the end of the command as shown in the following example:

```
gluster volume status <volume_name>
```

4.3.3 Checking the status of machine learning services

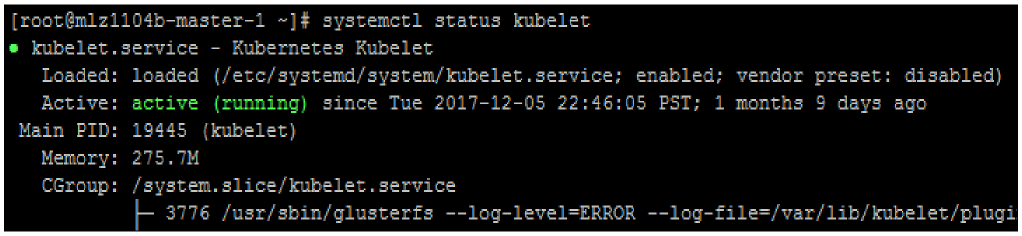
Key system and Machine Learning for z/OS services run in the Kubernetes cluster. Periodically Check the status of the cluster and the services.

Key Linux or Linux on Z services

Docker, Kubelet, and Haproxy are key Linux or Linux on Z services that are essential to the Kubernetes cluster operation. Run the **`systemctl status`** command as shown in the following example to check the status of each service:

```
systemctl status <service_name>
```

Where <*service_name*> is “docker”, “kubelet”, or “haproxy”. For example, you can issue the **systemctl status kubelet** command and see the output that is shown in Figure 4-22.



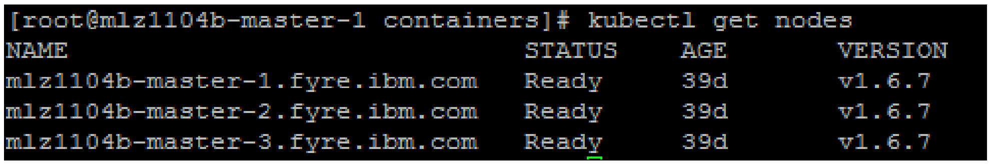
```
[root@mlz1104b-master-1 ~]# systemctl status kubelet
● kubelet.service - Kubernetes Kubelet
   Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2017-12-05 22:46:05 PST; 1 months 9 days ago
 Main PID: 19445 (kubelet)
   Memory: 275.7M
   CGroup: /system.slice/kubelet.service
           ├─ 3776 /usr/sbin/glusterfs --log-level=ERROR --log-file=/var/lib/kubelet/plugi
```

Figure 4-22 Output of systemctl status kubelet command

The output shows the “active(running)” status and a list of recent events that are useful in diagnosing problems.

Kubernetes nodes

Kubernetes does not dispatch work to a node that it is not in the “Ready” state. Ensure that you regularly check that all nodes are up and running. Run the **kubectl get nodes** command to list the state of each nodes in a cluster, as shown in Figure 4-23.

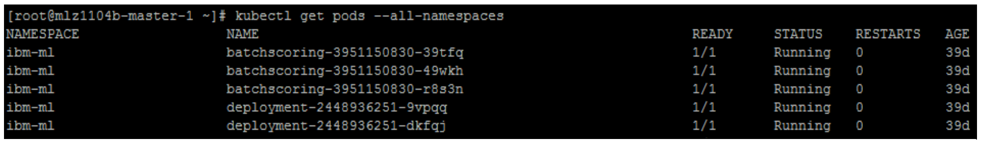


```
[root@mlz1104b-master-1 containers]# kubectl get nodes
NAME                           STATUS    AGE       VERSION
mlz1104b-master-1.fyre.ibm.com  Ready     39d       v1.6.7
mlz1104b-master-2.fyre.ibm.com  Ready     39d       v1.6.7
mlz1104b-master-3.fyre.ibm.com  Ready     39d       v1.6.7
```

Figure 4-23 Nodes state

Kubernetes pods

A Kubernetes pod is a group of one or more containers and their required storage and networking resources that work together to provide a service. Run the **kubectl get pods --all-namespaces** command (see Figure 4-24) to check the status of all pods. Run the **kubectl get pods -n <namespace>** command for only the pods in a specific namespace.



```
[root@mlz1104b-master-1 ~]# kubectl get pods --all-namespaces
NAMESPACE          NAME                                   READY     STATUS    RESTARTS   AGE
ibm-ml             batchscoring-3951150830-39tfq          1/1       Running   0          39d
ibm-ml             batchscoring-3951150830-49wkh          1/1       Running   0          39d
ibm-ml             batchscoring-3951150830-r8s3n          1/1       Running   0          39d
ibm-ml             deployment-2448936251-9vpqq            1/1       Running   0          39d
ibm-ml             deployment-2448936251-dkfqj            1/1       Running   0          39d
```

Figure 4-24 Checking the status of all pods

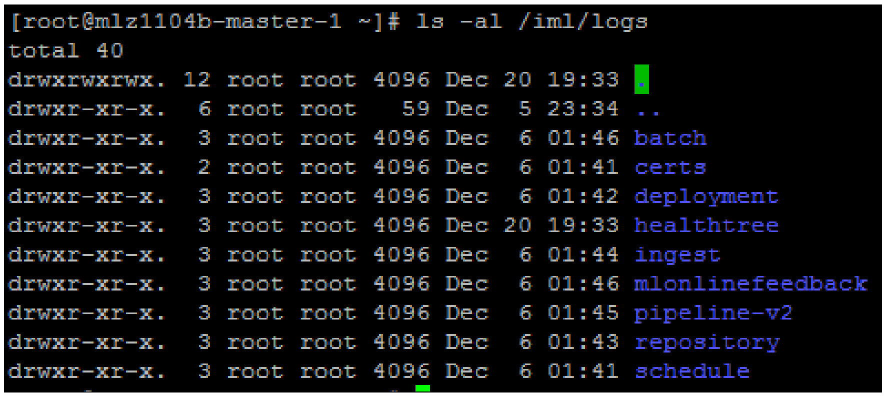
The “Running” status indicates that a pod is functioning normally. The number pairs, such as 1/1 or 2/2, in the READY column means that all containers in the pod are running. Because a high numeric value in the RESTARTS column suggests that the operation of the pod is not stable, investigate the potential problem before the pod goes offline.

You can get a quick count of the pods by running the following command:

```
kubectl get pods --all-namespaces | wc -l
```

The number of pods that is running varies depending on the Machine Learning for z/OS release. It is a good idea to note the total number immediately after the initial installation and use that number as a baseline. A number that is lower than the baseline during regular operation indicates potential problems.

Kubernetes logs provide another important source of information about the containers and pods and other machine learning services. Access to the logs is simplified and available in the GlusterFS-mounted `/iml/logs` directory, as shown in Figure 4-25.



```
[root@mlz1104b-master-1 ~]# ls -al /iml/logs
total 40
drwxrwxrwx. 12 root root 4096 Dec 20 19:33 .
drwxr-xr-x.  6 root root   59 Dec  5 23:34 ..
drwxr-xr-x.  3 root root 4096 Dec  6 01:46 batch
drwxr-xr-x.  2 root root 4096 Dec  6 01:41 certs
drwxr-xr-x.  3 root root 4096 Dec  6 01:42 deployment
drwxr-xr-x.  3 root root 4096 Dec 20 19:33 healthtree
drwxr-xr-x.  3 root root 4096 Dec  6 01:44 ingest
drwxr-xr-x.  3 root root 4096 Dec  6 01:46 mlonlinefeedback
drwxr-xr-x.  3 root root 4096 Dec  6 01:45 pipeline-v2
drwxr-xr-x.  3 root root 4096 Dec  6 01:43 repository
drwxr-xr-x.  3 root root 4096 Dec  6 01:41 schedule
```

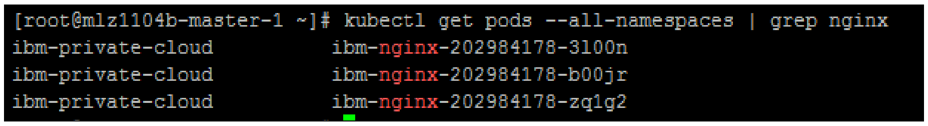
Figure 4-25 GlusterFS-mounted /iml/logs directory

If the directory is empty, remount the `/im` directory (and subdirectories) by running the `/wdp/scripts/mount_iml_gluster_volumes.sh` script. Kubernetes also generates and stores a series of symlinked log files in the `/var/log` directory.

Run the **`kubectl get logs`** command to list all of the Kubernetes logs, as shown in the following example:

```
kubectl get pods --all-namespaces
```

As shown in Figure 4-26, the command returns the following output with the pod name in the left column and the log file name in the right column.



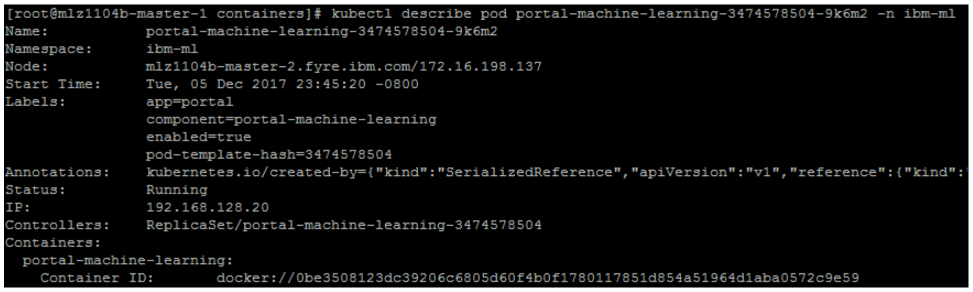
```
[root@mlz1104b-master-1 ~]# kubectl get pods --all-namespaces | grep nginx
ibm-private-cloud           ibm-nginx-202984178-3100n
ibm-private-cloud           ibm-nginx-202984178-b00jr
ibm-private-cloud           ibm-nginx-202984178-zq1g2
```

Figure 4-26 Output of kubectl get logs command

It can be required to "describe" a pod to get important status and event information. For example, run the following command to describe a pod:

```
kubectl describe pod <pod_name> -n <namespace>
```

The output is shown in Figure 4-27.



```
[root@mlz1104b-master-1 containers]# kubectl describe pod portal-machine-learning-3474578504-9k6m2 -n ibm-ml
Name:           portal-machine-learning-3474578504-9k6m2
Namespace:      ibm-ml
Node:           mlz1104b-master-2.fyre.ibm.com/172.16.198.137
Start Time:     Tue, 05 Dec 2017 23:45:20 -0800
Labels:         app=portal
                component=portal-machine-learning
                enabled=true
                pod-template-hash=3474578504
Annotations:    kubernetes.io/created-by={"kind":"SerializedReference","apiVersion":"v1","reference":{"kind":
Status:         Running
IP:             192.168.128.20
Controllers:    ReplicaSet/portal-machine-learning-3474578504
Containers:
  portal-machine-learning:
    Container ID:       docker://0be3508123dc39206c6805d60f4b0f1780117851d854a51964d1aba0572c9e59
```

Figure 4-27 Command output

The event information in the output is helpful if the pod does not start correctly or at all.

Kubernetes services

Kubernetes services can discover each other and enable multiple pods to be logically grouped. As a single logical entity, the pods provide a highly available and scalable service.

4.3.4 Stopping and starting nodes

The use of Kubernetes to manage Machine Learning for z/OS services provides availability and resilience. If a node or cluster fails, the node or cluster must be restarted. Use the following guidelines when you must stop and restart one or more nodes while keeping the Machine Learning for z/OS services available:

- Instead of rebooting a node, consider a managed shutdown of Kubernetes and GlusterFS services before rebooting the node. For more information, see “Restarting a node”.
- Always restart one node at a time. If multiple nodes are restarted at the same time, the files that are hosted on the Gluster volumes can be damaged, which causes inconsistencies in data or metadata.
- When a node is restarted and comes back online, verify the following information:
  - All GlusterFS volumes are started, all bricks are online, and no active volume tasks exist.
  - All Kubernetes pods are running, and all containers are ready.
- Allow the cluster sufficient time to stabilize before another node is restarted. The GlusterFS cluster needs time to synchronize files on each of the bricks for each of the volumes that are managed by the cluster. Unsynchronized files can result in I/O errors when the GlusterFS client attempts to read them, which can cause an outage in the container that requires the files.

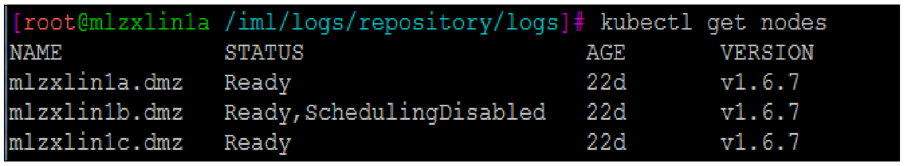
Restarting a node

Complete the following steps to restart a node:

1. Select and stop a node by running a sequence of Kubernetes commands in the following order (wait for one command to complete before the next command is run):
   - `kubectl drain <node_name>`

     The command directs Kubernetes to stop scheduling work on the node. You might see warnings or errors that indicate some of the pods do not fully support the drain operation. However, you can proceed because the drain command places the node in the wanted state to leave the cluster.
   - `kubectl get nodes`

     The command displays that the status of the targeted node changed, as shown in Figure 4-28.



```
[root@mlzxlin1a /iml/logs/repository/logs]# kubectl get nodes
NAME            STATUS                       AGE       VERSION
mlzxlin1a.dmz   Ready                        22d       v1.6.7
mlzxlin1b.dmz   Ready,SchedulingDisabled     22d       v1.6.7
mlzxlin1c.dmz   Ready                        22d       v1.6.7
```

Figure 4-28 Status of targeted node

   - `systemctl stop kubelet`
   - `systemctl stop docker`
   - `killall glusterfs`

- `killall glusterfsd`
- `killall glusterd`

2. Restart the node. After the system is back on, mark it schedulable.
3. Run the following command to uncordon the node so that Kubernetes can resume the scheduling of work to run on it:

   ```
   kubectl uncordon <node_name>
   ```
4. Run the following command to verify that the node is ready to resume work:

   ```
   kubectl get nodes
   ```

Restarting a cluster

If you must stop and restart an entire cluster, make sure to shut down each service in parallel across all the nodes. Complete the following steps:

1. Stop all of the nodes in the cluster individually by running the following commands. Ensure to run one command at a time across the nodes and wait for the command to complete. Run the commands in the following order:
   - `systemctl stop kubelet`
   - `systemctl stop docker`
   - `killall glusterfs`
   - `killall gluster`
   - `killall glusterd`
2. Restart the nodes individually. When restarting the nodes, ensure that master nodes are restarted in any order before compute-only nodes.
3. Very that all nodes in the cluster are successfully restarted and in Ready state.

4.4 Administering by using commands on z/OS

As with Linux or Linux on Z, you might find the use of UNIX-based commands is an efficient way to monitor and manage the many Machine Learning for z/OS processes and services that run on z/OS or in a CICS region.

4.4.1 Stopping and starting a Spark cluster

As described in 3.2.3, “Verifying IzODA installation and configuration” on page 30, a Spark cluster consists of a Spark master and a Spark worker that manages resources for your submitted Spark applications. You can quickly start and stop the cluster at the command line by completing the following steps:

1. Log in to a UNIX shell session as MLZSPARK.
2. Start a Spark master by running the following command:

   ```
   $SPARK_HOME/sbin/start-master.sh -h <host_IP_address>  -p <sparkMaster-port>
   ```

   Where `<host_IP_address>` is the IP address of the z/OS system and `<sparkMaster-port>` is the Spark master daemon port. The default port is 7077.
3. Start a Spark worker by running the following command:

   ```
   $SPARK_HOME/sbin/start-slave.sh spark://<host_IP_address>:<sparkMaster-port>
   ```
4. Stop the Spark worker by running the following command:

   ```
   $SPARK_HOME/sbin/stop-slave.sh spark://<host_IP_address>:<sparkMaster-port>
   ```

5. Stop the Spark master by running the following command:

   ```
   $SPARK_HOME/sbin/stop-master.sh spark://<host_IP_address>:<sparkMaster-port>
   ```

Alternatively, you can use batch jobs or z/OS started tasks to control the Spark master and worker and manage a Spark cluster.

4.4.2 Checking Spark Java processes

You can check Spark Java processes by defining an alias for spark jobs, or `sjobs`, to the spark user `.bashrc` profile. Add the entire entry as one line inside a pair of single quotation marks, as shown in the following example:

```
alias sjobs='COLUMNS=340 ps -A -o jobname,pid,ppid,xasid,thdcnt,vsz=VIRTUAL -o
stime,ruser,etime,args | grep /java | grep -v -e grep -e "ps -A" -e bash -e sshd |
sort'
```

You can customize the COLUMNS display by specifying the `-o` option or changing the COLUMNS values to suit your needs. Ensure that the values are large enough for you to receive adequate information about each Java process, as shown in Example 4-1.

Example 4-1 Java process

```
SPARK:/:> sjobs
SPARK1     67178357          1   67    75  224452   Apr 18    SPARK  4-08:11:50
/shared/java/J8.0_64/bin/java -cp
/var/sparkserver2/conf211/:/shared/IBM/izoda/spark/spark211/jars/*
-Dfile.encoding=UTF8 -Xmx1g org.apache.spark.deploy.worker.Worker --webui-port
8081 spark://MLZDZ07.DMZ:7077
SPARK8     33621533   33621527   5f    64  228264 20:54:19    SPARK 00:06:13
/shared/java/J8.0_64/bin/java -cp
/usr/lpp/aln/IMLzOS/iml-library/library/*:/var/sparkserver2/conf211/:/shared/IBM/i
zoda/spark/spark211/jars/* -Xmx1g org.apache.spark.deploy.SparkSubmit --master
spark://MLZDZ07.DMZ:7077 --conf spark.executor.extraClassPath=/u
SPARK9     16846706          1   59    83  224656   Apr 18    SPARK  4-08:13:30
/shared/java/J8.0_64/bin/java -cp
/var/sparkserver2/conf211/:/shared/IBM/izoda/spark/spark211/jars/*
-Dfile.encoding=UTF8 -Xmx1g org.apache.spark.deploy.master.Master --host
MLZDZ07.DMZ --port 7077 --webui-port 8080
```

4.4.3 Managing the Jupyter kernel gateway

The Jupyter kernel gateway runs on z/OS and is used to allow off-platform clients to run Spark and other analytic workloads on z/OS. To manage the gateway, log on to the z/OS system as `<spark_jupyter_toree_userid>`, which is the user that installed the gateway to establish a UNIX shell session.

Checking the gateway status

Complete the following steps to check the gateway status by verifying that the gateway process is running and that the gateway is listening on its designated port:

1. Log on to z/OS as `<spark_jupyter_toree_userid>` to start a UNIX shell session.
2. Change to the gateway log directory by running the following command:

   ```
   SPARK:> cd $IML_HOME/imlpython/logs
   ```

3. List all of the processes by running the following **grep** command:

```
SPARK:/usr/lpp/aln/IMLzOS/imlpython/logs:> ps -ef | grep `cat gateway.pid`
```

The command searches the PID of the gateway, which is saved in the `gateway.pid` file when the gateway is started. If the gate is up and running, the command returns a message that is similar to the following example:

```
SPARK   33622609      68129  - 23:48:07 ttyp0000  0:00 python
/shared/IBM/izoda/anaconda/bin/jupyter-kernelgateway --KernelSpecManage
```

4. Verify whether the gate is listening on the port it was configured with by running the **onetstat** command, as shown in Example 4-2.

Example 4-2 Running the onetstat command

```
SPARK:/usr/lpp/aln/IMLzOS/imlpython/bin:> onetstat -P 8889
MVS TCP/IP NETSTAT CS V2R2       TCPIP Name: TCPIP
User Id  Conn     State
-------  ----     -----
SPARK6   029334A7 Listen
  Local Socket:   192.168.164.77..8889
  Foreign Socket: 0.0.0.0..0
SPARK6   029334C0 Establsh
  Local Socket:   192.168.164.77..8889
  Foreign Socket: 192.168.164.37..58240
```

The default port is 8889. If the gateway is available, you see at least the local socket 8889 in listen state. If notebooks or other jobs are connected to the gateway, you see the connection information for each job. If the gateway is not running, review the `$IML_HOME/imlpython/log/gateway.out` log file for more information.

Restarting the gateway

The gateway must be running always to ensure the normal operation of Jupyter Notebooks and other machine learning. Complete the following steps to restart the gateway:

1. Log on to the z/OS system as `<spark_jupyter_toree_userid>` to start a UNIX shell session.
2. Change to the kernel gateway bin directory:

```
cd $IML_HOME/imlpython/bin
```

3. Stop the gateway by running the following command:

```
./kg2at-stop.sh
```

The script does not usually print any message.

4. Verify that the gateway is stopped by checking the gateway.out log.
5. Restart the gateway by running the kg2at-start.sh script with a source command to start the gateway:

source kg2at-start.sh

6. Follow the instructions as described in Step 5 in 3.4.1, “Installing machine learning services on z/OS” on page 45.
7. Verify that the gateway is started by checking the `gateway.out` log.
8. Verify that the gateway is started on the correct port and listening on the port.

If the status shows that the gateway is not listening on port 8889 or a configured port, check for another running gateway process. You might have to stop the gateway by its PID number and then restart the gateway to ensure it is started on the right port.

4.4.4 Managing stand-alone scoring servers

Machine Learning for z/OS scoring servers run inside a WebSphere Liberty container on z/OS. This configuration provides an environment for models to be scored by way of REST calls.

Checking the status of a scoring server

Complete the following steps to determine the status of a scoring server by monitoring the associated WLP process or checking the log files:

1. Log on to the z/OS system as `<spark_jupyter_toree_userid>` to start a UNIX shell session.
2. Change to the scoring service bin directory where the `server.sh` script is stored:

   ```
   cd $IML_HOME/bin
   ```

3. List all known scoring servers by running the following list command:

   ```
   ./server.sh list
   ```

   The command should return a list of all known scoring servers in the INSTANCE column. For each scoring server instance, the command returns information in HOST, HTTP, HTTPS, ADMIN_HTTP, ADMIN_HTTPS, FLASK_HTTP, FLASK_HTTPS, LIBERTY_PID, and PYTHON_PID columns.

   Most of the columns in the output correspond to the configuration options you specified for a scoring server. For example, INSTANCE is the *<serverName>* that you specified in the `scoring.cfg.`*<serverName>* file. HOST is the `scoring_ip` that you specified for the system where the scoring service runs. LIBERTY_PID is the ID of the WLP server process; ADMIN_PID is the ID of the Python process. For more information, see Step 7 in 3.4.1, “Installing machine learning services on z/OS” on page 45.

   A “-1” port value means that the port is not configured or disabled. A “--" PID value means that no process is running.

If more information is needed, check the `console.log`, `flask.log`, `uwsgi.log`, and other log files in the `$IML_HOME/output/`*<serverName>* directory.

The base WLP server is started and running if you see a message in the `console.log` file that is similar to the following example:

```
Server iml started with process ID 50400611.
 [AUDIT] CWWKF0011I: The server iml is ready to run a smarter planet.
```

The scoring service is ready to handle scoring requests if you see a message similar to the following example:

```
[AUDIT] CWWKT0016I: Web application available (default_host):
http://mlzdz07.dmz:10080/iml/
```

The flask server generates `flask.log` and `uwsgi.log` files. The server is started and ready for scoring requests if you see a message in `uwsgi.log` that is similar to the message that is shown in Example 4-3.

Example 4-3 Log message

```
WSGI app 0 (mountpoint='') ready in 2 seconds on interpreter 50088F7FF0 pid: 69021
(default app)
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI master process (pid: 69021)
spawned uWSGI worker 1 (pid: 69022, cores: 10)
*** Stats server enabled on /shared/IBM/aln/IMLzOS/output/iml/logs/stats.socket
fd: 14 ***
```

Restarting a scoring server

If a scoring server is not running or responding to scoring requests, consider stopping and restarting it. Complete the following steps:

1. Log on to the z/OS system as `<spark_jupyter_toree_userid>` to start a UNIX shell session.
2. Change to the scoring service bin directory where the `server.sh` script is stored:

   ```
   cd $IML_HOME/bin
   ```

3. Stop a scoring server by running the following stop command:

   ```
   ./server.sh stop <serverName>
   ```

4. If a flask server is attached and running, stop it by running the stop-python command:

   ```
   ./server.sh stop-python <serverName>
   ```

   The commands are successful and the servers are stopped if you see messages that are similar to the messages that are shown in Example 4-4.

Example 4-4 Resulting messages

```
[MLZSCOR@mlzdz07 /usr/lpp/aln/IMLzOS/bin]$ ./server.sh stop iml
Stopping Liberty server.
Stopping server iml.
Server iml stopped.
Cleaning temporary files.
[MLZSCOR@mlzdz07 /usr/lpp/aln/IMLzOS/bin]$ ./server.sh stop-python iml
Stopping uWSGI server iml.
This will take some time, please check the file
/shared/IBM/aln/IMLzOS/output/iml/logs/uwsgi.log until see "goodbye to uWSGI"
Cleaning temporary files.
```

5. Start the scoring server by running the **`start`** command:

   ```
   ./server.sh start <serverName>
   ```

   The command is successful, and the scoring server is started if you see a message that is similar to the message that is shown in Example 4-5.

Example 4-5 start command message

```
[MLZSCOR@mlzdz07 /usr/lpp/aln/IMLzOS/bin]$ ./server.sh start iml
Starting server iml.
Updating Python configuration file encoding.
Starting server iml.
```

```
Server iml started with process ID 50400347.
iml is starting in the background, it may take about 3-4 minutes to finish this
process, please check the log file
/shared/IBM/aln/IMLzOS/output/iml/logs/console.log for more details.
```

6. Start the flask server by running the **`start-python`** command:

   ```
   ./server.sh start-python <serverName>
   ```

The command is successful and the flask server is started if you see a message that is similar to the message that is shown in Example 4-6.

Example 4-6 start-python command

```
[MLZSCOR@mlzdz07 /usr/lpp/aln/IMLzOS/bin]$ ./server.sh start-python iml
Starting uWSGI server iml.
Updating Python configuration file encoding.
…
…
iml uWSGI server is starting in the background with PID 68746.
For scoring service log, please check
/shared/IBM/aln/IMLzOS/output/iml/logs/flask.log.
For uWSGI server log, please check
/shared/IBM/aln/IMLzOS/output/iml/logs/uwsgi.log.
```

4.4.5 Managing a scoring server in a CICS region

If you installed a Machine Learning for z/OS scoring service in a CICS region, the service is embedded in a Liberty Profile (JVM) server in a CICS resource bundle.

Checking the status of a CICS-integrated scoring server

You can check the status of a CICS-integrated scoring server by looking up the status of the JVM process, the status of the WAR bundle, and the log files. Complete the following steps:

1. Log on to a CICS terminal or a z/OS system console as a user who is authorized to run CICS commands.
2. Check the JVM process by running the following command:

   ```
   CEMT INQUIRE JVMSERVER(ALNSCSER)
   ```

   Where `ALNSCSER` is the name of the scoring server. The command returns a message similar to the message that is shown in Example 4-7.

Example 4-7 Returned message of CEMT INQUIRE JVMSERVER(ALNSCSER) command

```
CEMT INQUIRE JVMSERVER(ALNSCSER)
RESPONSE=MLZ1
   Jvmserver(ALNSCSER)
   Enablestatus(Enabled)
   Purgetype(         )
   Prfile(ALNSCSER)
   Lerunopts(DFHAXRO)
   Threadcount(015)
   Threadlimit(015 )
   Currentheap(351917528)
   Initheap(2G)
```

The JVM is up and running when `Enablestatus` is “Enabled.”

3. Check the status of the WAR bundle by running the following command:

   ```
   CEMT INQUIRE BUNDLE(ALNSCBDL)
   ```

   The resource bundle is available if `Enablestatus` is “Enabled” in the response.

If more information is needed, check the scoring server log and trace records in the `$IML_HOME/cics-scoring/workdir/<`*`cics_region_name`*`>/ALNSCSER` directory for more information. Use a UNIX **`more`** command to view the files.

Restarting a scoring server in a CICS region

If the scoring JVM server and the CICS resource bundle are configured during the installation to start at system start, you do not need to manually start the scoring server. However, if the server is down or not responding, consider stopping and restarting it. Complete the following steps:

1. Log on to a CICS terminal or a z/OS system console as a user who is authorized to run CICS commands.
2. Disable the JVM server by running the following command:

   ```
   CEMT SET JVMSERVER(ALNSCSER) DISABLED
   ```

3. Verify that the JVM server is stopped.
4. Enable the JVM server by running the following command:

   ```
   CEMT SET JVMSERVER(ALNSCSER) ENABLED
   ```

5. Verify that the JVM server is enabled and running.

5

Model development and deployment: A retail example

Machine Learning for z/OS provides an end-to-end enterprise machine learning framework through a simple, easy to use web user interface (UI). This chapter introduces the UI and its many features. Through a retail example, we describe how the UI can be used to manage the entire machine learning workflow, from creating a project to training, evaluating, saving, publishing, and deploying a model.

This chapter includes the following topics:

5.1 Machine Learning for z/OS web UI

The Machine Learning for z/OS web UI enables business analysts, data engineers, data scientists, and application developers to collaborate and manage the entire machine learning workflow from data preparation to model training, evaluation, deployment, and retraining. In addition to the sign-in and welcome pages, the interface consists of the main pages that are listed in Table 5-1, all of which can be accessed through a sliding sidebar.

Table 5-1 UI main pages

UI page or view	Description
Community	Contains a list of links to useful machine learning resources, including sample notebooks and Jupyter and Spark basics learning materials.
Projects	Lists all machine learning projects and links to each individual project, and a self-contained workspace for you to work alone or collaborate with others on data and models.
Model Management	Provides a workspace in which you can manage all of the models that are published or saved to the central repository and their deployments.
Environments	Provides a view of the status of runtime engines across all projects.

The main UI pages contain multiple tabs or links to subpages, and some subpages also contain multiple tabs.

5.1.1 Signing in the Machine Learning for z/OS web UI

To access the Machine Learning for z/OS UI, you must have pre-authorization and the URL from your system or machine learning administrator. The administrator can authorize you by creating a user name in the LDAP user directory and then assigning the user name privileges through the administration dashboard. With the required authorization, you can sign in the interface by completing the following steps:

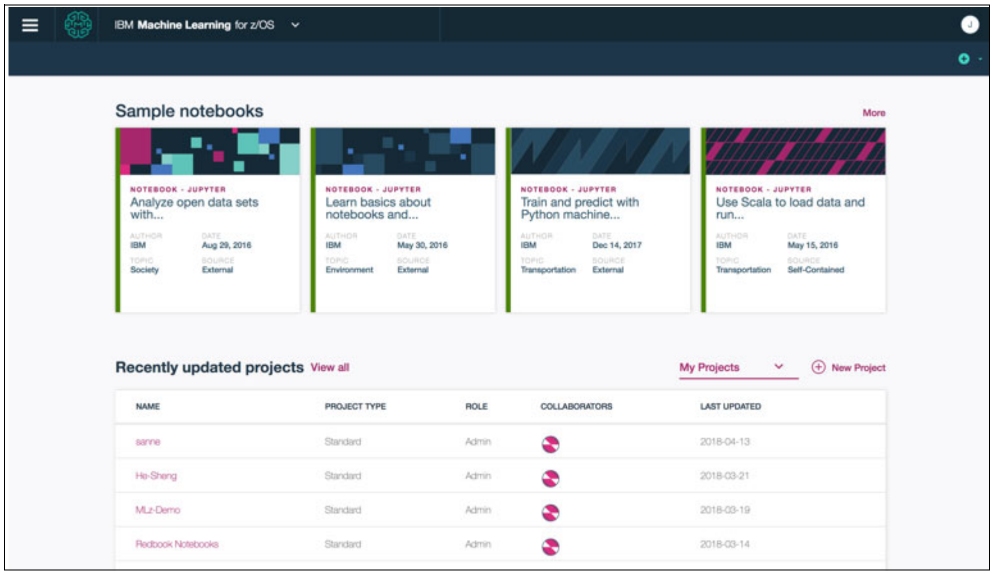
1. Open a web browser and enter the URL for the Machine Learning for z/OS UI, as shown in the following example:

   ```
   https://<ip_address>
   ```

   Where `<ip_address>` is the IP address or host name of your Machine Learning for z/OS proxy server.

2. Enter the user name and password that was provided by the administrator and click sign in to authenticate yourself.

   Upon successful authentication, the Welcome window opens (see Figure 5-1).



NOTEBOOK - JUPYTER
Analyze open data sets with...
AUTHOR IBM DATE Aug 29, 2016
TOPIC Society SOURCE External

NOTEBOOK - JUPYTER
Learn basics about notebooks and...
AUTHOR IBM DATE May 30, 2016
TOPIC Environment SOURCE External

NOTEBOOK - JUPYTER
Train and predict with Python machine...
AUTHOR IBM DATE Dec 14, 2017
TOPIC Transportation SOURCE External

NOTEBOOK - JUPYTER
Use Scala to load data and run...
AUTHOR IBM DATE May 15, 2016
TOPIC Transportation SOURCE Self-Contained

NAME	PROJECT TYPE	ROLE	COLLABORATORS	LAST UPDATED
sanne	Standard	Admin	●	2018-04-13
He-Sheng	Standard	Admin	●	2018-03-21
MLz-Demo	Standard	Admin	●	2018-03-19
Redbook Notebooks	Standard	Admin	●	2018-03-14

Figure 5-1 Welcome page

   The page acts as an information hub that includes sample notebooks, tutorials, and other resources for machine learning and for using Machine Learning for z/OS. The page also lists recently created or updated projects, which provides a fast path to projects and a shortcut to creating projects.

3. In the upper left corner of the Welcome page, click the icon of three vertically stacked bars to open the sidebar of the UI.
4. From the sidebar, select and start any of the UI main pages.
5. From a main page, browse to any tab or subpage.

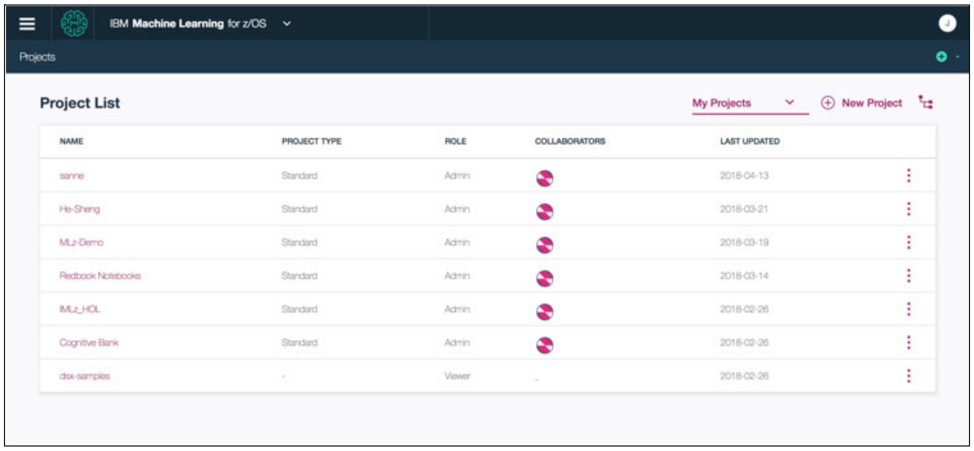
Next, we describe some of the UI main pages, subpages, and tabs.

5.1.2 Community

As with the Welcome page, the Community page is a repository of links to useful machine learning resources, including sample notebooks, tutorials, and basic machine learning education materials.

5.1.3 Projects

The Projects page collects and lists all the Machine Learning for z/OS projects by name, type, and (last updated) date, as shown in Figure 5-2.



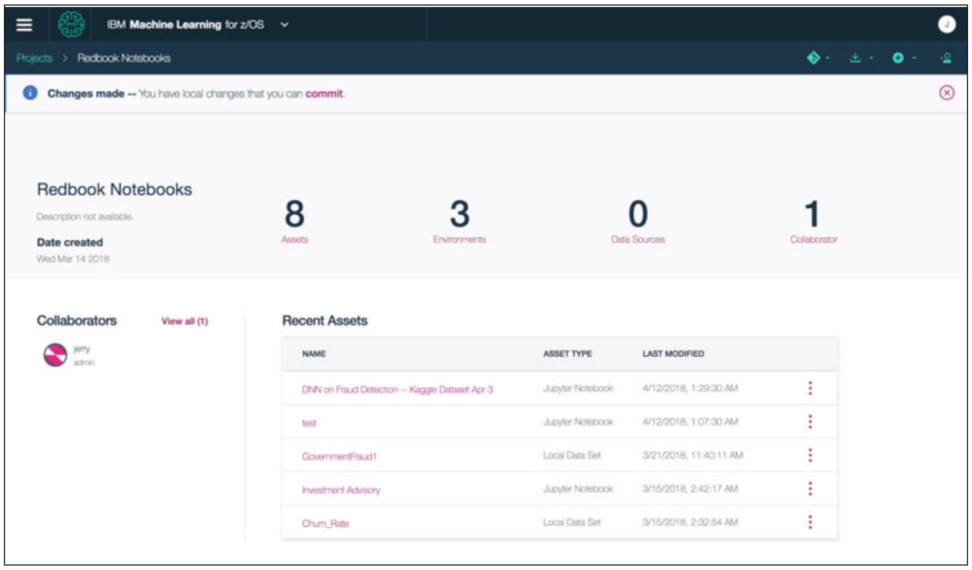
Project List

NAME	PROJECT TYPE	ROLE	COLLABORATORS	LAST UPDATED
sanne	Standard	Admin	●	2018-04-13
He-Sheng	Standard	Admin	●	2018-03-21
MLz-Demo	Standard	Admin	●	2018-03-19
Redbook Notebooks	Standard	Admin	●	2018-03-14
IMLz_HOL	Standard	Admin	●	2018-02-26
Cognitive Bank	Standard	Admin	●	2018-02-26
dsx-samples	-	Viewer	-	2018-02-26

Figure 5-2 Projects page

Each project provides business analysts, data engineers, data scientists, and application developers a self-contained workspace to collaborate on data preparation and analysis, model training and evaluation, model deployment and retraining.

Click the name of a project to open the subpage for the project, as shown in Figure 5-3.



Changes made -- You have local changes that you can commit.

Redbook Notebooks

Description not available.

Date created
Wed Mar 14 2018

8 Assets · 3 Environments · 0 Data Sources · 1 Collaborator

Collaborators — View all (1)

jerry
admin

Recent Assets

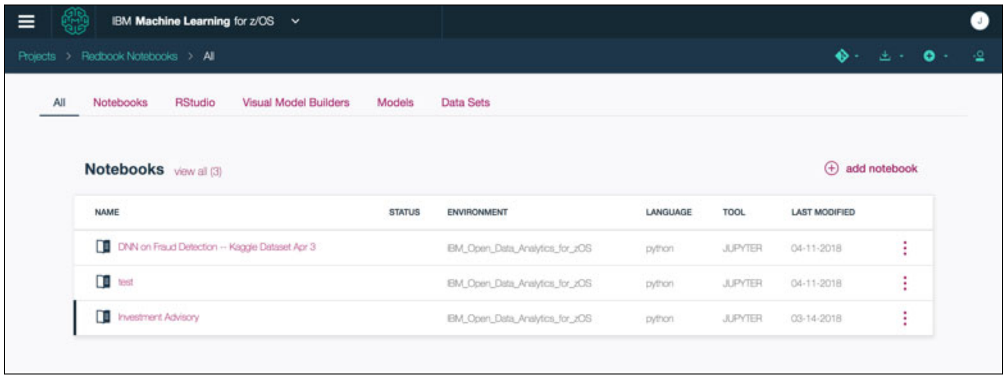
NAME	ASSET TYPE	LAST MODIFIED
DNN on Fraud Detection -- Kaggle Dataset Apr 3	Jupyter Notebook	4/12/2018, 1:29:30 AM
test	Jupyter Notebook	4/12/2018, 1:07:30 AM
GovernmentFraud1	Local Data Set	3/21/2018, 11:40:11 AM
Investment Advisory	Jupyter Notebook	3/15/2018, 2:42:17 AM
Churn_Rate	Local Data Set	3/15/2018, 2:32:54 AM

Figure 5-3 Project subpage

The page acts as a gateway to more pages in which you can manage the objects and resources of the project. The objects and resources are tallied and grouped into the categories of Assets, Environments, Data Sources, and Collaborators. Click each category to open the category page and manage the associated objects and resources.

Assets

The Assets of a project consist of data sets that are used for analysis, integrated tools or frameworks for model development, and models. As shown in Figure 5-4, the tools include Notebook, RStudio, and Visual Model Builder.



Notebooks

NAME	STATUS	ENVIRONMENT	LANGUAGE	TOOL	LAST MODIFIED
DNN on Fraud Detection -- Kaggle Dataset Apr 3		IBM_Open_Data_Analytics_for_zOS	python	JUPYTER	04-11-2018
test		IBM_Open_Data_Analytics_for_zOS	python	JUPYTER	04-11-2018
Investment Advisory		IBM_Open_Data_Analytics_for_zOS	python	JUPYTER	03-14-2018

Figure 5-4 Project assets

Different assets are listed in their sections or tabs on the Assets page.

Notebook

The integrated Jupyter Notebook provides a programmatic approach to model development. You can use the interactive Notebook interface to create models in Python and Scala. Python is a popular programming language among data scientists. It includes many handy packages for data analysis, and Scala is a language native to Spark. You can add, edit, or delete a notebook.

RStudio

The integrated RStudio provides another programmatic approach to model development. You can use the RStudio IDE for data preprocessing, visualization, and modeling with open source R packages. You can build a model in R and then define and share the model by way of PMML.

Visual Model Builder

The Visual Model Builder provides a guided approach to model development. You can use the step-by-step wizard to perform quick data analysis and create simple models without the need for extensive programming knowledge and skills. You also can add or delete a visual model builder.

Models

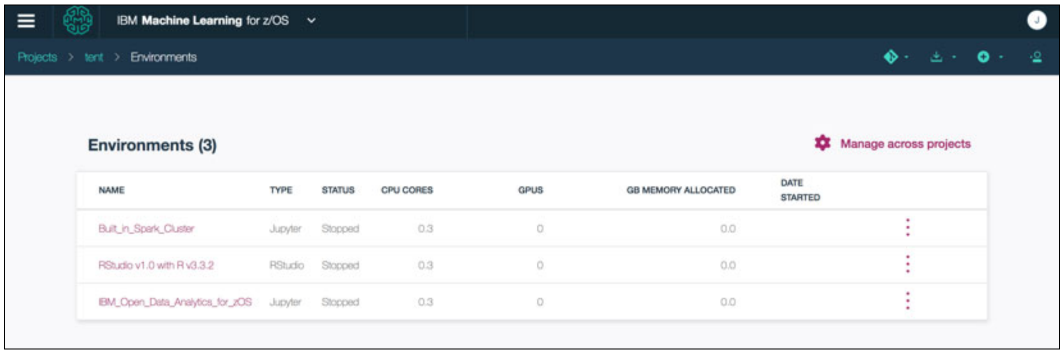
Models are created with the Notebook, RStudio, and Visual Model Builder within the project. The models are available and visible to the project owner and collaborators. You can publish the models to the central repository service so that they become available to all users in the system. You also can add, publish, or delete a model.

Data Sets

Data Sets are the data that is prepared and analyzed for model training and development. The data sets can be imported from remote sources, such as Db2, or uploaded from local storage in `.csv` format. You also can add, export, preview, or delete a data set.

Environments

Environments are the runtime engines that are used by the project. You can select the Built-in Spark Cluster, RStudio, and IBM Open Data Analytics for z/OS as the Environment when you create a model (see Figure 5-5).



Environments (3)

NAME	TYPE	STATUS	CPU CORES	GPUS	GB MEMORY ALLOCATED	DATE STARTED
Built_in_Spark_Cluster	Jupyter	Stopped	0.3	0	0.0	
RStudio v1.0 with R v3.3.2	RStudio	Stopped	0.3	0	0.0	
IBM_Open_Data_Analytics_for_zOS	Jupyter	Stopped	0.3	0	0.0	

Figure 5-5 Environments

You can also add environments to the project. Click the **Manage across projects** button to see a complete list of the environments that are used by all projects.

Data Sources

Data Sources are the databases or repositories that provide input data for creating models. You can connect Machine Learning for z/OS to and import data from IBM dashDB®, Db2, Db2 for z/OS, HDFS – HDP, Oracle, and Mainframe Data Service (MDS). Data sets that are created from the data sources are listed on the Assets page and available for model development. You also can add or delete a data source.

Collaborators

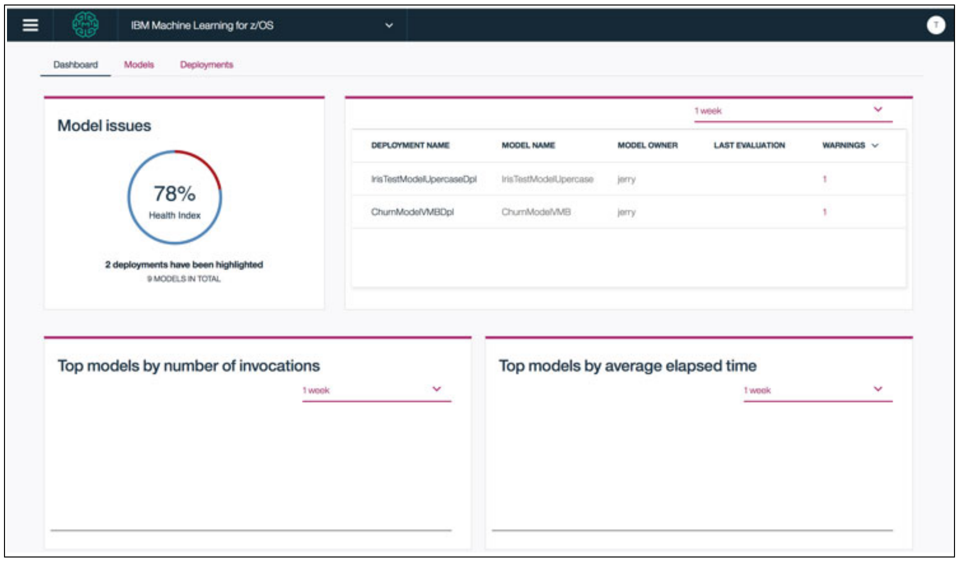
Collaborators are the users whom the owner of a project invites to collaborate as a viewer, editor, or administrator of the project. A collaborator must be an authorized Machine Learning for z/OS user first. A collaborator with the administrator permission can invite other authorized users to collaborate on the project. You also can add or remove a collaborator.

5.1.4 Model Management

The Model Management page provides a workspace for you to manage all the models and their deployments. The models can be developed in individual projects and published to the central repository service. They also can be trained on another platform and imported by way of PMML. The page consists of the Dashboard, Models, and Deployments tabs.

Dashboard

The Dashboard tab is the default view of the Model Management page. It features a visual index of the overall health of deployed models and a list view of high-performing and under-performing models, as shown in Figure 5-6 on page 99.


IBM Machine Learning for z/OS

Dashboard Models Deployments

Model issues

78%
Health Index

2 deployments have been highlighted
9 MODELS IN TOTAL

1 week

DEPLOYMENT NAME	MODEL NAME	MODEL OWNER	LAST EVALUATION	WARNINGS
IrisTestModelUpercaseDpl	IrisTestModelUpercase	jerry		1
ChurnModelVMBDpl	ChurnModelVMB	jerry		1

Top models by number of invocations

1 week

Top models by average elapsed time

1 week

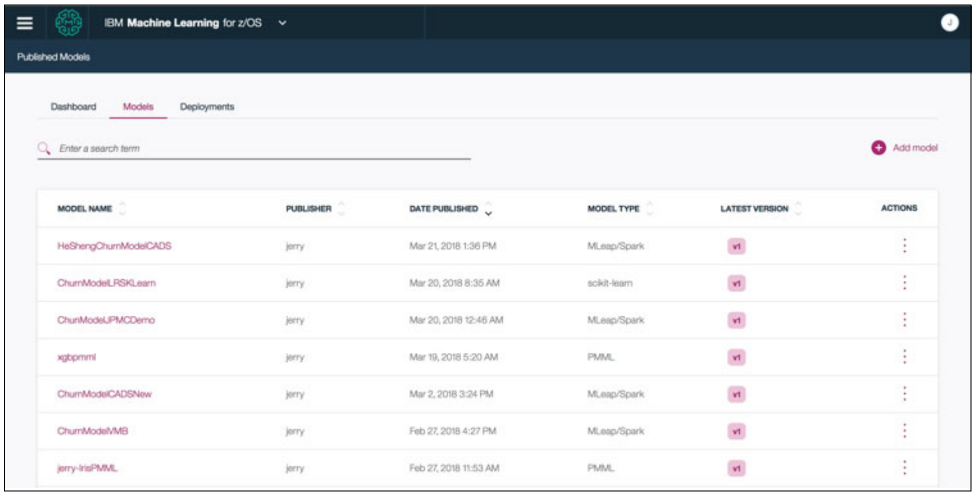
Figure 5-6 Dashboard

The model health index presents a metric of the deployments that are evaluated as normal, contain warnings or errors, and are not evaluated. Top-performing models are ranked by the number of API calls and the average of response time.

Under-performing models are listed by warnings or errors. When the performance of a model drops below a specified threshold, the owner or users are alerted, and the model is marked as under-performing.

Models

The Models tab lists all models in the repository service. These models are also referred as *published models*. They are published from individual projects or imported from external sources by way of PMML, as shown in Figure 5-7.


IBM Machine Learning for z/OS

Published Models

Dashboard Models Deployments

Enter a search term

Add model

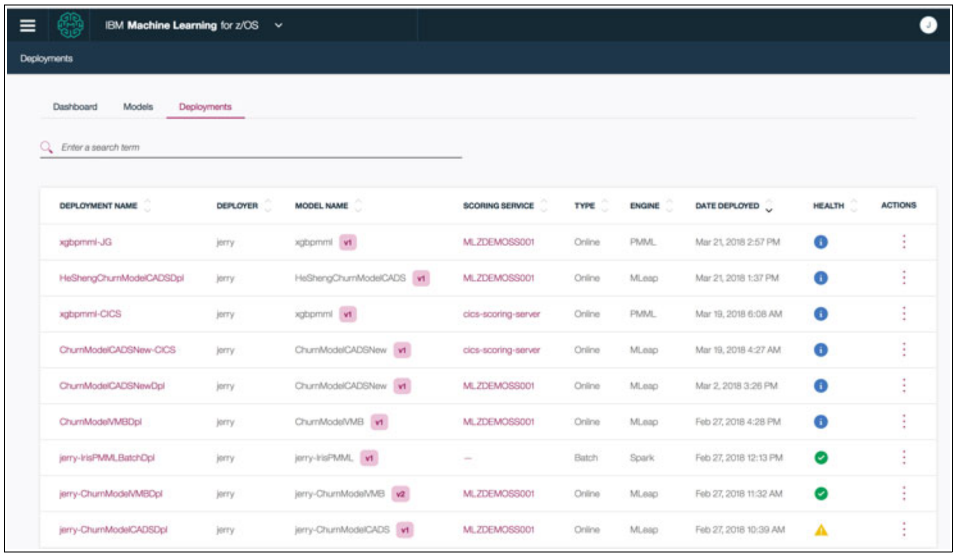
MODEL NAME	PUBLISHER	DATE PUBLISHED	MODEL TYPE	LATEST VERSION	ACTIONS
HeShengChurnModelCADS	jerry	Mar 21, 2018 1:36 PM	MLeap/Spark	v1	⋮
ChurnModelLRSKLearn	jerry	Mar 20, 2018 8:35 AM	scikit-learn	v1	⋮
ChunModelJPMCDemo	jerry	Mar 20, 2018 12:46 AM	MLeap/Spark	v1	⋮
xgbpmml	jerry	Mar 19, 2018 5:20 AM	PMML	v1	⋮
ChurnModelCADSNew	jerry	Mar 2, 2018 3:24 PM	MLeap/Spark	v1	⋮
ChurnModelVMB	jerry	Feb 27, 2018 4:27 PM	MLeap/Spark	v1	⋮
jerry-IrisPMML	jerry	Feb 27, 2018 11:53 AM	PMML	v1	⋮

Figure 5-7 Models tab

You can add, deploy, retrain, publish (to WML), export, or delete a model. You can also view the details of a model.

Deployments

After models are created, they cannot be used by external applications unless they are deployed. Deployed models are generally referred as *deployments*, which are listed on the Deployments tab (see Figure 5-8).



DEPLOYMENT NAME	DEPLOYER	MODEL NAME	SCORING SERVICE	TYPE	ENGINE	DATE DEPLOYED	HEALTH	ACTIONS
xgbpmml-JG	jerry	xgbpmml v1	MLZDEMOSS001	Online	PMML	Mar 21, 2018 2:57 PM	ℹ	⋮
HeShengChurnModelCADSDpl	jerry	HeShengChurnModelCADS v1	MLZDEMOSS001	Online	MLeap	Mar 21, 2018 1:37 PM	ℹ	⋮
xgbpmml-CICS	jerry	xgbpmml v1	cics-scoring-server	Online	PMML	Mar 19, 2018 6:08 AM	ℹ	⋮
ChurnModelCADSNew-CICS	jerry	ChurnModelCADSNew v1	cics-scoring-server	Online	MLeap	Mar 19, 2018 4:27 AM	ℹ	⋮
ChurnModelCADSNewDpl	jerry	ChurnModelCADSNew v1	MLZDEMOSS001	Online	MLeap	Mar 2, 2018 3:26 PM	ℹ	⋮
ChurnModelVMBDpl	jerry	ChurnModelVMB v1	MLZDEMOSS001	Online	MLeap	Feb 27, 2018 4:28 PM	ℹ	⋮
jerry-IrisPMMLBatchDpl	jerry	jerry-IrisPMML v1	—	Batch	Spark	Feb 27, 2018 12:13 PM	✓	⋮
jerry-ChurnModelVMBDpl	jerry	jerry-ChurnModelVMB v2	MLZDEMOSS001	Online	MLeap	Feb 27, 2018 11:32 AM	✓	⋮
jerry-ChurnModelCADSDpl	jerry	jerry-ChurnModelCADS v1	MLZDEMOSS001	Online	MLeap	Feb 27, 2018 10:39 AM	⚠	⋮

Figure 5-8 Deployments tab

You can test, update, or schedule deployment evaluations. You can also view information about a deployment, including the scoring endpoint and the online feedback endpoint. The scoring endpoint is a RESTful API that can be called directly for scoring requests for the model.

5.1.5 Environments

Machine Learning for z/OS environments are the runtime engines that are used by projects. The Environments tab within a project lists the runtime engines that are used by the project, and the Environments main page collects the runtime engines that are used in the entire system. If necessary, you can start an environment on this page.

5.2 Machine learning workflow for model development and deployment

A typical machine learning workflow involves the following tasks:

- Goal setting
- Data collection
- Data preprocessing
- Model training
- Model evaluation
- Model deployment

After the initial deployment, a model is monitored and retrained if its performance degrades. Machine Learning for z/OS implements the workflow through the UI. Some of these stages are described next.

5.2.1 Data collection

Data scientists work to extract insights from business-relevant data and facilitate decision making. Therefore, the first step in performing enterprise level machine learning is to clarify the business question and collect data that might help answer the question.

In the Machine Learning for z/OS UI, data collection is implemented through the Data Source page and the Data Sets tab. You can connect to a remote data source, such as Db2, IMS, and SMF, or import local data and then, create data sets as input data.

5.2.2 Data preparation

At the time of acquisition, data might not be immediately ready as input to train a machine learning model. The data might contain categorical features, such state, but it might lack the numeric inputs that a model accepts. It is also possible that the data is missing some columns. In these cases, the data must be carefully prepared before it can be used.

The Machine Learning for z/OS UI makes data preparation process easy. In the Visual Model Builder, you can use the automatic data preparation feature in a Scala package. In the integrated Notebook editor, you can preprocess data by using SparkSQL, Python pandas, or scikit-learn. If you prefer, you also can analyze data and scale features by defining your own functions.

5.2.3 Model training

When the data becomes ready as input, data scientists can start to build the model. Models are selected depending on the business questions that they are expected to address.

One key to building a successful model is the selection of machine learning algorithms. Supervised learning and unsupervised learning are two types of frequently used algorithms. Unsupervised learning explores the pattern of the data without any specific labels; supervised learning deals with data with labels, which specifically shows the outcomes of the records.

Supervised learning algorithms are divided to classification (binary and multiclass) and regression. Classification determines the pattern of how data fits to a discrete label, and regression identifies the pattern of how to fit a continuous number.

In the Machine Learning for z/OS UI, you can select data and then algorithm to train a model through the Visual Model Builder or the integrated Notebook editor.

5.2.4 Model evaluation

After the model is trained, you must use different metrics to evaluate how well the model is performing on the specific data. Model evaluation helps data scientists improve model selection because different models are compared against each other based on the evaluation metrics. The better a model performs during the evaluation, the more likely it helps answer the business question.

In the Machine Learning for z/OS UI, model evaluation occurs immediately after data fitting and algorithm selection through the Visual Model Builder or the integrated Notebook editor.

5.2.5 Model deployment

After a model meets the evaluation criteria, you can deploy it to make predictions for new records and produce business values. A deployed model is monitored because as the pattern of the data changes, the model that is based on old data can become not predictive over time when new data is imported. Any under-performing model must be retrained with new data so that it matches the new pattern.

You can deploy, evaluate, and retrain a model through the Model Management page of the Machine Learning for z/OS UI.

5.3 Developing and deploying a model to predict tent sales

An outdoor equipment e-commerce retailer is developing a marketing strategy for a new tent product. It wants to use machine learning to predict potential customers and understand their purchase behavior so that it can tailor its promotional activities and ultimately maximize the sales of the new product. Y

As the data scientist who works on the mandate, you use IBM Machine Learning for z/OS to create, evaluate, deploy, and retain a model to generate the predictions and the underlying intelligence that help the retailer with their important decisions. The input data that you use includes, but is not limited to, the features of gender, age, profession, and marital status of the potential customers.

5.3.1 Creating a project

After you set the goal for the model that you are going to create, the first task is to create a project in the Machine Learning for z/OS UI if you do not have one. The project provides the workspace for you and your collaborators and all the required resources.

Complete the following steps to create a project:

1. Browse to the Projects – View all Projects page of the UI. For more information about how to sign in to the UI, see 5.1.1, “Signing in the Machine Learning for z/OS web UI” on page 94.
2. Click the **New Project** (⊕) button to open in the upper right corner to start the Create Project page.
3. Select **Blank** to create a project from scratch or select **From File** to create a project from a file.

   If you select **Blank**, enter `Tent` as the name for the project and optionally a short description. The new project is empty until you add content to it later.

   If you select **From File**, enter `Tent` as the name for the project and drop the project file in `.zip`, `.jar`, or `.gz` format into the Project File box. Alternatively, you can click **Browse** to locate and upload the project file.
4. Click **Create** to create the project.
5. Verify that the Tent project shows up in the Project List.

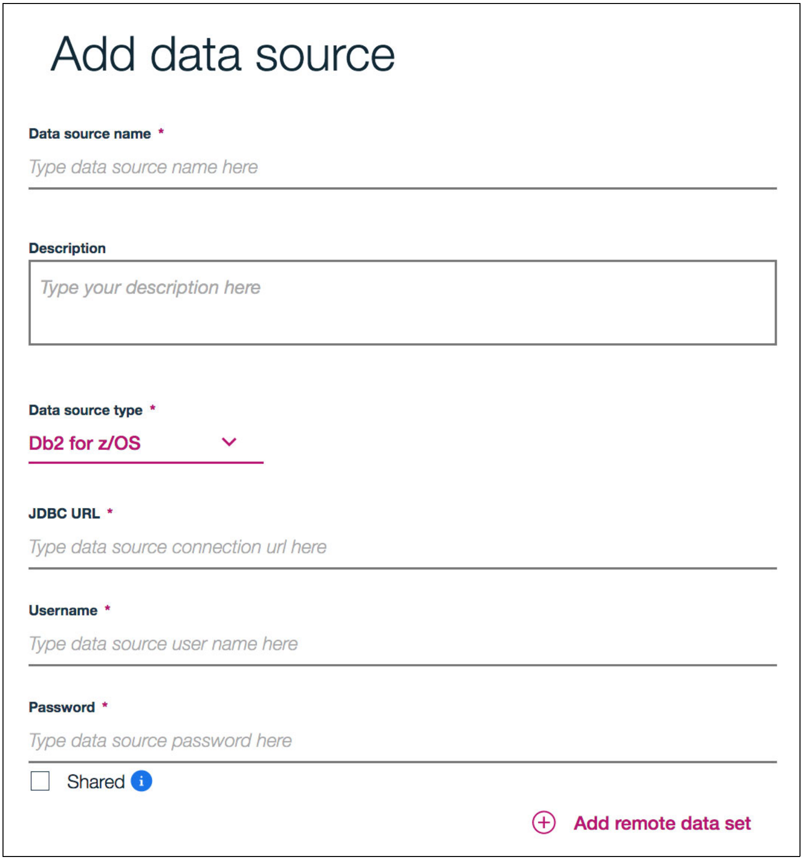
5.3.2 Adding a data set

After you create a project, you must add resources to it that are required for creating a model. Complete the following steps:

1. From the Project List, click **Tent** to open the new project.
2. Click **Assets** to open the Assets page for the project. It is normal that the asset category is empty.
3. From the Data Sets section, click **add data set** (⊕) to display the panel in which you can add a data set. (You can create a data set by importing local data files or connect to remote data sources.)

   If you select Local File, drop your files into the file loading box or click **Select from your local file system** to choose your files.

   If you connect to Remote Data Set, click **Select a data source** to choose a data source. If no data source exists or fits your current purpose, click **Add data source** to create a data source.
4. Specify a data source name and select a data source type. Supply any other required information, depending on the data source type. For example, if you select Db2 for z/OS as the data source type, ensure that you provide the required JDBC URL, Db2 user name, and password, as shown in Figure 5-9.



Add data source

Data source name *
Type data source name here

Description
Type your description here

Data source type *
Db2 for z/OS

JDBC URL *
Type data source connection url here

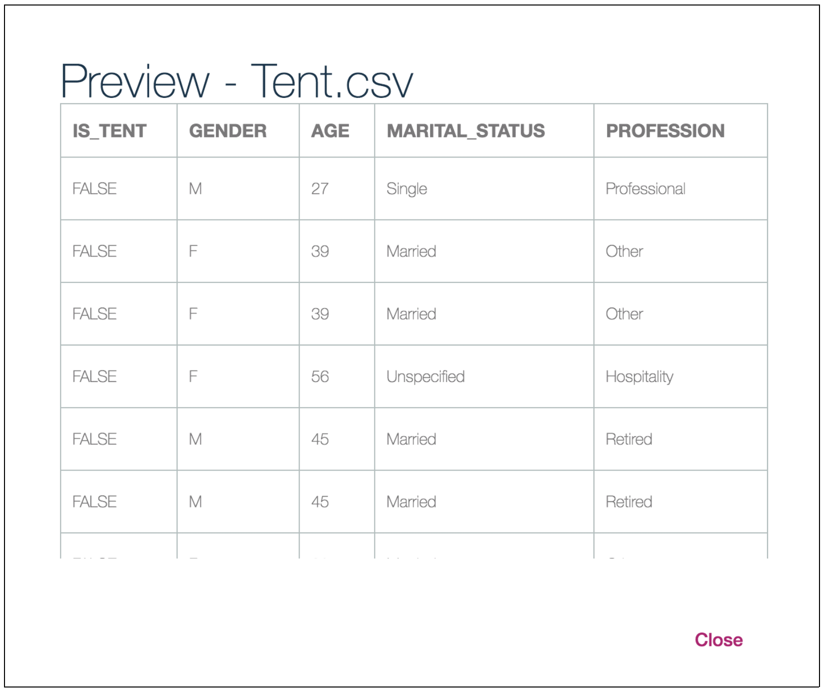
Username *
Type data source user name here

Password *
Type data source password here

Shared

Figure 5-9 Adding a data source

5. Enter `TentConn1` as the name for the new data set and optionally a short description. Complete the following steps:
   a. Specify a schema and enter `MLZ.TENTDATA_N_4000` as the table for the new data set.
   b. Click **Save** to create the TentConn1 data set.
   c. Verify that the new data set appears in the Data Sets section.
   d. Optionally, select **Preview** from the ACTIONS menu for TentConn1 to preview the data set in `.csv` format, as shown in Figure 5-10.


Preview - Tent.csv

IS_TENT	GENDER	AGE	MARITAL_STATUS	PROFESSION
FALSE	M	27	Single	Professional
FALSE	F	39	Married	Other
FALSE	F	39	Married	Other
FALSE	F	56	Unspecified	Hospitality
FALSE	M	45	Married	Retired
FALSE	M	45	Married	Retired

Figure 5-10 Data set preview

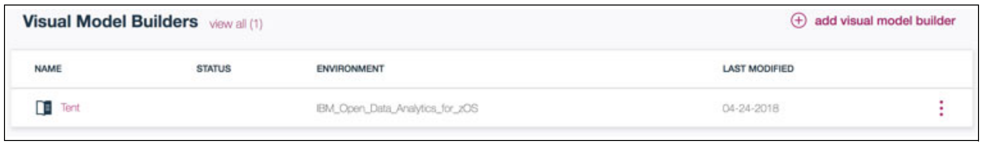
5.3.3 Training and saving a model

Now that the data is ready, you can build a model from the data by using the Visual Model Builder or the Notebooks editor or writing your own code.

Creating a model by using the Visual Model Builder

If you want to create a model by using the Visual Model Builder, you must first create a visual builder and then, select data, train, evaluate, and save the actual model. Complete the following steps:

1. From the Visual Model Builder section, click **add visual model builder** (⊕) to open the Add Visual Model Builder panel.
2. Enter `Tent` as the name and optionally a short description of the new visual model builder.
3. Select **IBM_Open_Data_Analytics_for _z/OS** as the Environment, which is the runtime engine.
4. Click **Create** to create the visual model builder.
5. Verify that the new Tent visual model builder appears in the Visual Model Builder section, as shown in Figure 5-11 on page 105.

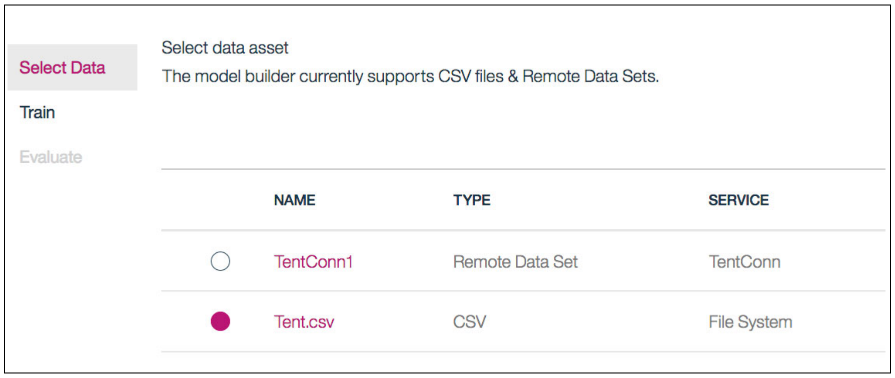


Visual Model Builders view all (1) add visual model builder

NAME	STATUS	ENVIRONMENT	LAST MODIFIED
Tent		IBM_Open_Data_Analytics_for_zOS	04-24-2018

Figure 5-11 Visual Model Builders section

6. Click **Tent** in the list to open the container for the visual model builder.
7. For Select Data, select **Tent.csv** as the data asset (see Figure 5-12). Click **Next** to continue.

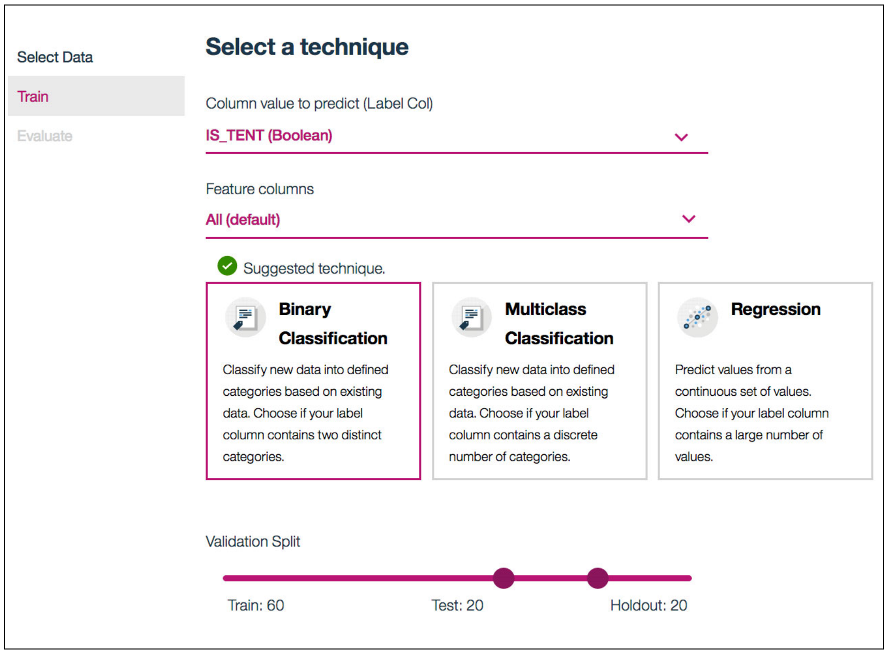


Select data asset
The model builder currently supports CSV files & Remote Data Sets.

	NAME	TYPE	SERVICE
○	TentConn1	Remote Data Set	TentConn
●	Tent.csv	CSV	File System

Figure 5-12 Selecting Tent.csv

8. For Train, select **IS_TENT** for the required label column and a feature for the optional feature column (see Figure 5-13). If a feature column is not selected, all columns are used by default.



Select a technique

Column value to predict (Label Col): IS_TENT (Boolean)

Feature columns: All (default)

✓ Suggested technique.

Binary Classification: Classify new data into defined categories based on existing data. Choose if your label column contains two distinct categories.

Multiclass Classification: Classify new data into defined categories based on existing data. Choose if your label column contains a discrete number of categories.

Regression: Predict values from a continuous set of values. Choose if your label column contains a large number of values.

Validation Split: Train: 60 Test: 20 Holdout: 20

Figure 5-13 Selecting a technique

A suitable technique (or algorithm) is suggested that is based on the label column selection. It is best to use the suggested algorithm.

9. Set the data split ratio for test, training, and holdout purposes. The default ratio is 60% for training, 20% for test, and 20% for holdout. Data for “Test” is used to test a model, data for “Train” is used to train the model and generate evaluation metrics, and data for “Holdout” is left out or not used. This data split ratio is recommended, particularly when the data set is large and you do not want to feed all of the data to the estimator that slows down model training.
10. Select an estimator for the model from the Configured estimators list. If no configured estimator is available, click **Add Estimators** to choose one or more algorithms from the selection panel:
    - For binary classification, choose from logistic regression, decision tree, random forest, and gradient boosted tree.
    - For multiclass classification, choose from decision tree, random forest, and naïve bayes.
    - For regression, choose from linear regression, decision tree, random forest, gradient boosted tree, and Isotonic regression.

    The chosen estimators are listed under the Configured estimators section as shown in Figure 5-14.

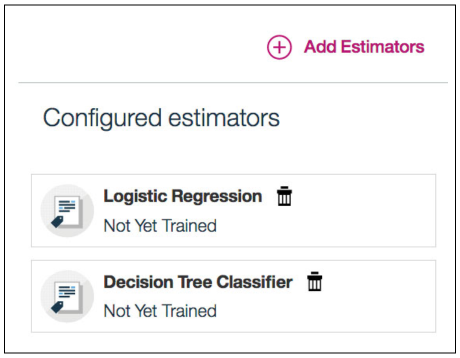
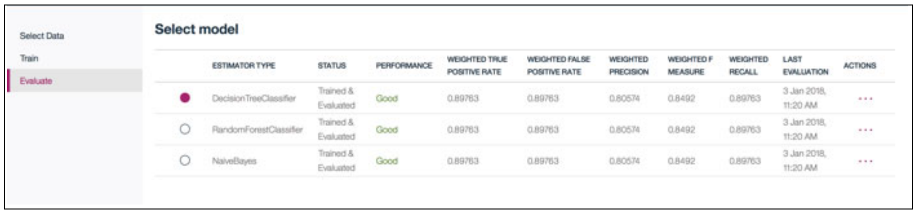


Figure 5-14 Configured estimators

11. Click **Next** to start the process of training the model.
12. For Evaluate, resolve any training error and review the performance indicators to see whether the model meets the performance goals.

    Different estimators use different evaluation measures. For example, binary classification includes an area under ROC curve and another under PR curve. Multiclass classification includes weighted true positive rate, weighted false positive rate, weighted precision, weighted F measure, and weighted recall, as shown in Figure 5-15.



Select model

	ESTIMATOR TYPE	STATUS	PERFORMANCE	WEIGHTED TRUE POSITIVE RATE	WEIGHTED FALSE POSITIVE RATE	WEIGHTED PRECISION	WEIGHTED F MEASURE	WEIGHTED RECALL	LAST EVALUATION	ACTIONS
●	DecisionTreeClassifier	Trained & Evaluated	Good	0.89763	0.89763	0.80574	0.8492	0.89763	3 Jan 2018, 11:20 AM	...
○	RandomForestClassifier	Trained & Evaluated	Good	0.89763	0.89763	0.80574	0.8492	0.89763	3 Jan 2018, 11:20 AM	...
○	NaiveBayes	Trained & Evaluated	Good	0.89763	0.89763	0.80574	0.8492	0.89763	3 Jan 2018, 11:20 AM	...

Figure 5-15 Selecting a model

13. Check to see whether the performance of the model meets the goal of the evaluation. The overall performance can be rated as Excellent, Good, Fair, or Poor.
14. If you are satisfied with the performance, click **Save** to save the model.

15. Verify that the Tent model appears in the Models section or tab on the Assets page of the project.

Creating a model by using the Notebook

If you decide to create a model by using the Notebook editor, you must first create a notebook and then, select data, train, evaluate, and save the actual model. The following example shows how to create a model by using Python in the Notebooks editor:

1. From the Notebooks section, click **add notebook** (⊕) to open the Create Notebook page.
2. Choose one of the following methods to create a notebook:
   - **Blank** for creating a notebook from scratch.
   - **From File** for creating a notebook by importing a local notebook file.
   - **From URL** for creating a notebook by importing a remote notebook file.

   If you are creating a model From File, browse and select a local notebook file to import. You can also drop the file into the Notebook File box.

   If you are creating a model From URL, enter the URL to the remote server where the notebook file is stored.
3. Enter `Tent_Python` as the name and optionally a short description for the new notebook.
4. Select **IBM_Open_Data_Analytics_for _z/OS** as the Environment, which is the runtime engine.
5. Select **Python** as the Language if you are creating a model from scratch. If you want to create a notebook from a local or remote file, the Language field is automatically detected based on the specification in the file.
6. Click **Create** to add the new notebook. The kernel for the notebook is automatically started.

   For the next part of the process, we assume that you created the notebook from Blank.
7. Verify that the Tent_Python notebook appears in the Notebooks section.
8. Click **Tent_Python** from the Notebooks list to start the integrated Jupyter Notebook interface.

   You can use the Tent_Python notebook in the same way that you use any notebook that was created in the open source Jupyter Notebook. However, notebooks that are created by using the integrated Notebook in Machine Learning of z/OS are supplemented with more functions, including the ability to quickly insert project context code and automatic generation of data from data sets.
9. Add the required project context code by selecting **Inset project context** from the Create New icon in the Notebook framework (see Figure 5-16).

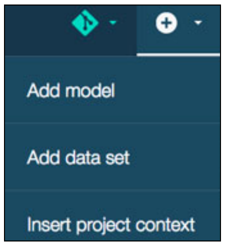


Figure 5-16 Selecting Insert project context option

A block of code is added to the first cell of the notebook. For security, you must insert project context into a notebook whenever you run it.

10. Select any cell that contains code and click **Run cell** from the CellToolbar to train, evaluate, and save the model (see Figure 5-17).

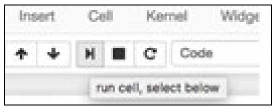
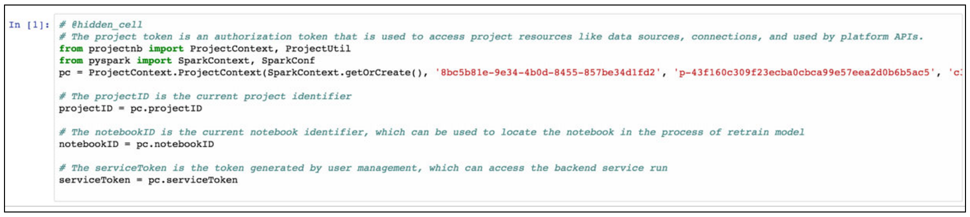


Figure 5-17 Selecting Run cell

The code in the cell is run. An asterisk is displayed in the square bracket at the upper left corner of a cell. This asterisk changes into a number when the cell run completes. The output from the run is added below the cell, as shown in Figure 5-18.



```
In [1]: # @hidden_cell
        # The project token is an authorization token that is used to access project resources like data sources, connections, and used by platform APIs.
        from projectnb import ProjectContext, ProjectUtil
        from pyspark import SparkContext, SparkConf
        pc = ProjectContext.ProjectContext(SparkContext.getOrCreate(), '8bc5b81e-9e34-4b0d-8455-857be34d1fd2', 'p-43f160c309f23ecba0cbca99e57eea2d0b6b5ac5', 'c

        # The projectID is the current project identifier
        projectID = pc.projectID

        # The notebookID is the current notebook identifier, which can be used to locate the notebook in the process of retrain model
        notebookID = pc.notebookID

        # The serviceToken is the token generated by user management, which can access the backend service run
        serviceToken = pc.serviceToken
```

Figure 5-18 Run output

11. If necessary, click the **Find data** icon to access the local and remote data sets available to the current project.

    From Local, click **Insert to code** and select **Insert Pandas DataFrame** or **Insert Spark DataFrame in Python** (see Figure 5-19) to generate a block of code for loading data as Pandas or PySpark DataFrame.

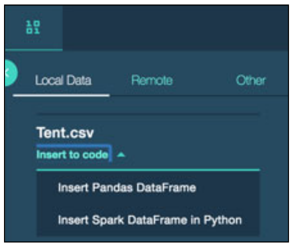


Figure 5-19 Selecting Insert to code option

12. From Remote, click **Insert to code** and then, **Insert Spark DataFrame in Python**, as shown in Figure 5-20.

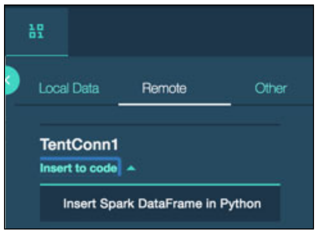
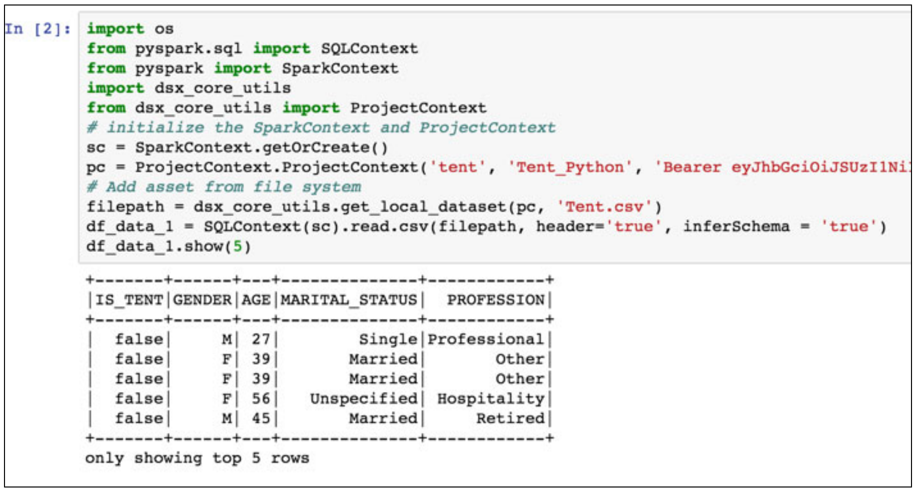


Figure 5-20 Selecting Insert Spark DataFrame in Python

13. Run the automatically generated code to load and preview the data (see Figure 5-21).



```
In [2]: import os
        from pyspark.sql import SQLContext
        from pyspark import SparkContext
        import dsx_core_utils
        from dsx_core_utils import ProjectContext
        # initialize the SparkContext and ProjectContext
        sc = SparkContext.getOrCreate()
        pc = ProjectContext.ProjectContext('tent', 'Tent_Python', 'Bearer eyJhbGciOiJSUzI1Ni
        # Add asset from file system
        filepath = dsx_core_utils.get_local_dataset(pc, 'Tent.csv')
        df_data_1 = SQLContext(sc).read.csv(filepath, header='true', inferSchema = 'true')
        df_data_1.show(5)

        +-------+------+---+--------------+------------+
        |IS_TENT|GENDER|AGE|MARITAL_STATUS|  PROFESSION|
        +-------+------+---+--------------+------------+
        |  false|     M| 27|        Single|Professional|
        |  false|     F| 39|       Married|       Other|
        |  false|     F| 39|       Married|       Other|
        |  false|     F| 56|   Unspecified| Hospitality|
        |  false|     M| 45|       Married|     Retired|
        +-------+------+---+--------------+------------+
        only showing top 5 rows
```

Figure 5-21 Data preview

The data is now in the Notebook workspace and ready for use later.

Creating a model by writing your own code

If you prefer, you can write your own code in Python to preprocess data and train, evaluate, and save a model from the Tent_Python notebook you created. The following example shows how you can manually build a model in Python:

1. Import the tools for splitting data and running logistic regression:

   ```
   from sklearn.model_selection import train_test_split
   from sklearn import preprocessing
   from sklearn.linear_model import LogisticRegression
   ```

   The call for data preprocessing is needed when you want to change the format of features and modify their values or construct new features. In the case of the Tent model that you are creating, you must include the code for data preprocessing because several categorical variables cannot be directly input into the model.

2. Apply label encoding in Scikit Learn to convert a categorical feature to integers between 0 and n - 1, where n is the number of unique values the feature takes. A sequence of integers from 0 to n - 1 exactly correspond to n classes of the categorical feature, as shown in the following example:

   ```
   df1_pd = df_data_1.toPandas()
   labelEncoder = preprocessing.LabelEncoder().fit(df1_pd["IS_TENT"])
   genderEncoder =  preprocessing.LabelEncoder().fit(df1_pd["GENDER"])
   maritalEncoder = preprocessing.LabelEncoder().fit(df1_pd["MARITAL_STATUS"])
   professionEncoder = preprocessing.LabelEncoder().fit(df1_pd["PROFESSION"])
   df1_pd["marital_status"] = maritalEncoder.transform(df1_pd["MARITAL_STATUS"])
   df1_pd["profession"] = professionEncoder.transform(df1_pd["PROFESSION"])
   ```

3. Combine label encoding with one hot encoding when more than two classes are in a categorical feature. If the data is in the correct format, label encoding is sufficient for data preprocess as input to a model. For example, label encoding is sufficient when only two classes are in a categorical feature, such as True and False or male and female. The two classes are transformed into integers 0 and 1, and the parameter of this feature quantifies the difference between class 0 and 1.

In some cases, label encoding alone is not enough. In this instance, you must combine label encoding with one hot encoding. One hot encoding helps measure the differences between classes more accurately.

Label encoder transforms a four-class feature to values of 0, 1, 2, 3, which implies that certain quantitative relations exist between different classes: 2 is twice as 1, and the difference between 3 and 2 are the same as difference between 0 and 1, which are not necessarily true.

In the case of the Tent model, columns IS_TENT and GENDER take binary values, and MARITAL_STATUS and PROFESSION are multiclass features. These four columns need label encoding, and MARITAL_STATUS and PROFESSION need one hot encoding in addition to label encoding, as shown in the following example:

```
Enc = preprocessing.OneHotEncoder().fit(df1_pd[["marital_status",
"profession"]])

import pandas as pd

colNames = ["marital_" + x for x in list(maritalEncoder.classes_)] +
["profession_" + y for y in list(professionEncoder.classes_)]
X = pd.DataFrame(Enc.transform(df1_pd[["marital_status",
"profession"]]).toarray(), columns = colNames)
X["gender"] = genderEncoder.transform(df1_pd["GENDER"])
X["AGE"] = df1_pd["AGE"]
y = labelEncoder.transform(df1_pd["IS_TENT"])
```

Now that categorical features of marital status, profession, and gender are all transformed and the label is specified, the data is ready as input to the estimator.

4. Split the features and label into training and test sets before estimation:

   ```
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
   random_state=0)
   ```

5. Train the model by specifying logistic regression as the binary classifier and starting the logistic regression estimator:

   ```
   lr = LogisticRegression()
   lr.fit(X_train, y_train)
   ```

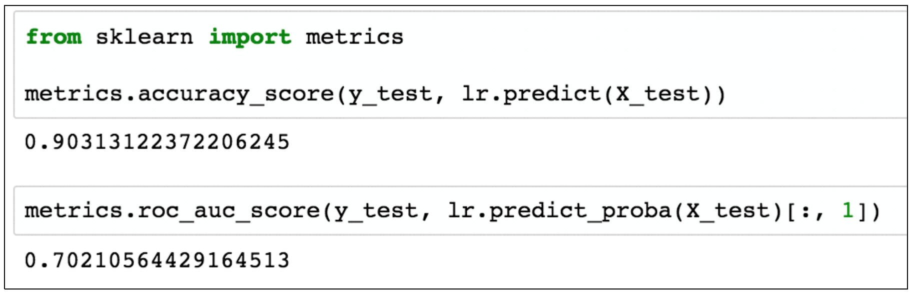
6. Evaluate the model. The estimator includes a scoring method that evaluates the model with specific features and labels. Different estimators can use different evaluation measures in their scoring methods. The logistic regression estimator uses accuracy in the scoring method:

   ```
   lr.score(X_train, y_train)
   lr.score(X_test, y_test)
   ```

   In addition to the scoring method, the metrics package in sklearn provides various model evaluation metrics, such as accuracy and AUC:

   ```
   from sklearn import metrics
   metrics.accuracy_score(y_test, lr.predict(X_test))
   metrics.roc_auc_score(y_test, lr.predict_proba(X_test)[:, 1])
   ```

When the code is run, the evaluation metrics produce the accuracy results that are shown in Figure 5-22.



```
from sklearn import metrics

metrics.accuracy_score(y_test, lr.predict(X_test))
0.90313122372206245

metrics.roc_auc_score(y_test, lr.predict_proba(X_test)[:, 1])
0.70210564429164513
```

Figure 5-22 Accuracy results

The 0.90313122372206245 accuracy means that the model that is trained from the data gets 0.90313122372206245 correct prediction on new data. The 0.70210564429164513 AUC means that the area under the Character Receiver Operator Curve is 0.70210564429164513 on new data that is different from training data. These results are estimates of how the trained model will perform on other future out-of-sample data.

The cross-validation score is also available, as shown in the following example:

```
from sklearn.model_selection import cross_val_score
cross_val_score(LogisticRegression(), X, y, scoring='accuracy', cv=10)
```

7. Save the model after the performance goal is obtained, as shown in the following example:

```
from repository.mlrepository import MetaNames
from repository.mlrepository import MetaProps
from repository.mlrepositoryclient import MLRepositoryClient
from repository.mlrepositoryartifact import MLRepositoryArtifact
ml_repository_client = MLRepositoryClient("http://<metaservice_ip>:12501")
ml_repository_client.authorize_with_token(authToken)
props1 = MetaProps({MetaNames.AUTHOR_NAME: "author", MetaNames.AUTHOR_EMAIL:
"author@example.com", MetaNames.MODEL_META_PROJECT_ID: projectName,
MetaNames.MODEL_META_ORIGIN_TYPE: "notebook", MetaNames.IS_EXTERNAL_CALL:
"True", MetaNames.MODEL_META_ORIGIN_ID: notebookName})
input_artifact = MLRepositoryArtifact(lr, name="Tent", meta_props=props1,
training_data=X_train, training_target=y_train)
saved_artifact1 = ml_repository_client.models.save(input_artifact)
print("model saved successfully")
```

8. Verify that the Tent model shows up in the `Models` section or tab.

5.3.4 Publishing a model

The models that are in the project are visible to users within the project only. When you are satisfied with the model and want to share it with other users, you can publish the Tent model to the central repository service. Verify that the model is listed with all other published models on the Models tab of the Model Management page.

5.3.5 Deploying a model

After the Tent model is available in the repository service, you can deploy it to be used by applications. Complete the following steps:

1. From the Models list, select the Tent model and from the ACTIONS menu (see Figure 5-23). Select **Deploy**.

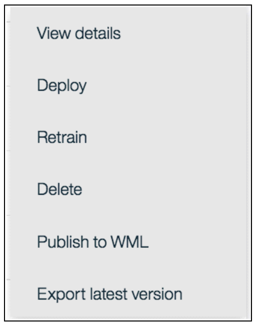
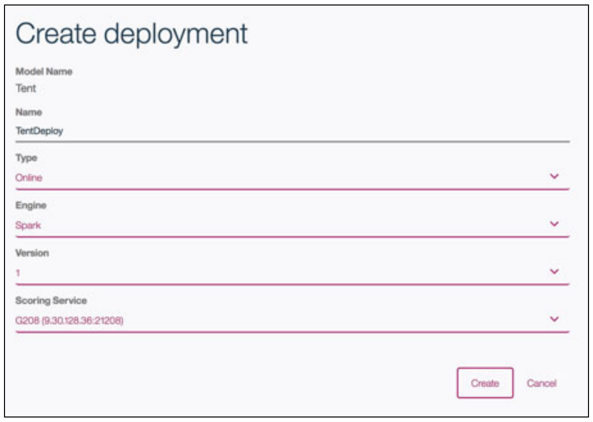


Figure 5-23 Actions menu

2. On the Create deployment page, enter `TentDeploy` as the deployment name and then, select the deployment type, engine, version, and scoring service, as shown in Figure 5-24.



Create deployment

Model Name
Tent

Name
TentDeploy

Type
Online

Engine
Spark

Version
1

Scoring Service
G208 (9.30.128.36:21208)

Figure 5-24 Create deployment page

Online and batch are the two available deployment types. Although online deployment is for real-time scoring with faster response time, it scores one record at a time. Batch scoring processes a batch of records for prediction, but it often takes slightly longer time to complete the scoring.

3. Verify that the deployed model TentDeploy appears on the Deployments tab of the Model Management page. You can view details, test, evaluate, update, or delete the deployment (see Figure 5-25).

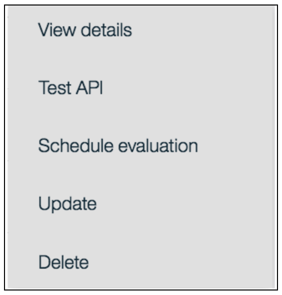
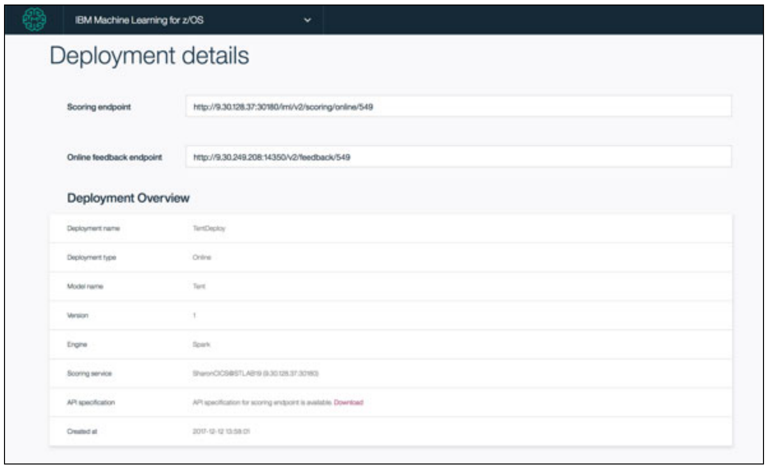


Figure 5-25 Deployment options

4. From the ACTIONS menu, select **View details** to configure the deployed model (see Figure 5-26).



Deployment details

Scoring endpoint: http://9.30.128.37:30180/iml/v2/scoring/online/549

Online feedback endpoint: http://9.30.249.208:14350/v2/feedback/549

Deployment Overview

Deployment name	TentDeploy
Deployment type	Online
Model name	Tent
Version	1
Engine	Spark
Scoring service	SharonCICS@STLAB19 (9.30.128.37:30180)
API specification	API specification for scoring endpoint is available. Download
Created at	2017-12-12 13:58:01

Figure 5-26 Deployment details

The URL for the scoring endpoint is a RESTful API that can be called directly to perform the process of scoring the model. The scoring results can help the retailer determine who are the potential customers of the new tent product, what the marketing strategy is based on the likely buyers, how its promotional activities are run, and the methods that can be used to maximize sales.

5.4 Preparing a model for online scoring by using CICS program ALNSCORE

If you installed the Machine Learning for z/OS scoring service in a CICS region, you can prepare a model for online scoring with the integrated CICS program called ALNSCORE. You can use the `CICS LINK` command in your CICS COBOL application to call ALNSCORE for online scoring. The call uses special containers to transfer the scoring input and output between the COBOL application and the ALNSCORE program. Because the input and output schemas are model-specific, you must carefully prepare each model to ensure appropriate mapping and interpretation of the input and output data structures for deployment.

Complete the following steps to prepare a model for online scoring by using ALNSCROE:

1. Extract the input and output JSON schemas of a model that is deployed. Complete the following steps:
   a. Log on to the Machine Learning for z/OS web UI by using your user name and password.
   b. From the sidebar, browse to the Deployments tab of the Model Management page.
   c. Select a deployment and then **View details** from the ACTIONS menu.
   d. On the Deployment details page, scroll down to the Schema section, and click **JSON Schema** on the right side to display the Input Schema and Output Schema columns.
   e. Copy the input and output schemas into two separate JSON files. Name the files, such as `modelInput.json` and `modelOutput.json`, and store them on your z/OS UNIX System Services.
2. Generate COBOL copybooks for the input and output schemas for the model by using the CICS DFHJS2LS utility.
   a. Locate the sample JCL ALNJS2LS job file in the following directory:

      `<install_dir_zos>/cics-scoring/extra/jcllib`
   b. Customize the JCL job by following the instructions in the ALNJS2LS file to generate two COBOL copybooks.
   c. Copy and customize the ALNJS2LS job and set JSON-SCHEMA to the path of input JSON schema file `modelInput.json`.
   d. Run the customized ALNJS2LS job to generate a PDS member that contains the COBOL copybook for the model's input.

      The JCL job reads the JSON schema from the `modelInput.json` file and creates a PDS member that is named MODELIxx that contains the COBOL copybook for the model's input.
   e. Customize the ALNJS2LS job again and set JSON-SCHEMA to the path of output JSON schema file `modelOutput.json`.
   f. Run the customized ALNJS2LS job and generate another PDS member that contains the copybook for the model's output.
3. Use the new COBOL copybooks to create Java helper classes for the input and output of the model. The ALNSCSER scoring server in the CICS region uses the Java helper classes to interpret the input data structure that is transmitted from the COBOL application and then passes the scoring result back to the COBOL application. Complete the following steps:
   a. Copy the copybook to a PDS member and add the 01 layer to the data structure and the required sections to create a COBOL program.
   b. Locate the sample JCL ALNJCGEN job file in the following directory:

      `<install_dir_zos>/cics-scoring/extra/jcllib`
   c. Follow the instructions in the ALNJCGEN file to customize the JCL job.
   d. Run the customized JCL job to create the Java helper classes. The job compiles the MODELIN COBOL program that first creates an ADATA file and then uses the ADATA file as the input to generate Java helper class ModelInWrapper.
   e. Repeat Steps 3.c - 3.d to create Java helper classes for the input and the output of the model.

4. Use special CICS containers and channels in your COBOL application to call the Machine Learning scoring program ALNSCORE. If the Type column is a string, you must set the correct length to the corresponding fieldname-length parameter. Each user COBOL application that calls ALNSCORE must create its own channel with a unique name for passing the input parameters.

ALNSCORE does not support a model if its input schema contains field names that are COBOL-reserved words. If the field names in your training data contain COBOL-reserved words, you must rename them during the model training.

6

Use cases: Applying Machine Learning for z/OS in business

The e-commerce tent trader that was described in Chapter 5, “Model development and deployment: A retail example” on page 93 represents just one segment of the massive retail industry. It epitomizes the many businesses that are at the crossroads of the existing process of labor-intensive decision-making and the fast-changing landscape of market opportunities.

With the new development in AI research and cognitive systems, data scientists across the industries increasingly turned to machine learning for answers. As a complete enterprise machine learning solution, Machine Learning for z/OS can help you optimize decision-making, solve business problems, minimize risks and costs, and grow top-line revenue by capitalizing on insights from real-time transaction data.

This chapter describes how you can apply Machine Learning for z/OS in several use cases to answer specific business questions. It also provides high-level procedural guidance for managing the full lifecycle of building a predictive model from data analysis to model training, evaluation, deployment, monitoring, and retraining.

This chapter includes the following topics:

6.1 Customer churn: Reducing customer attrition in banking

Customer churn or *attrition* in banking refers to the situation when customers end their relationship with a bank, close their accounts, and discontinue other services. The high churn rate can cause severe financial loss and challenges for banks viability. Although losing existing customers is costly, attracting new customers is even more so. To retain customers and reduce costs, it is important for banks to predict and identify customers with high attrition risk and take proactive actions to mitigate this risk.

Assume that a US-based cognitive bank wants to predict potential churn rate, analyze customer behavioral tendencies, and identify the demographics of likely churners. For this process, data scientists at the bank use Machine Learning for z/OS to develop a solution that consists of visual exploration of historical churn data and building the churn data model. This model can predict the probability for churning for a customer with a specific profile.

Bank IT wants to integrate the developed model with the bank enterprise information system. For this purpose, the data scientist deploys the prepared model in Machine Learning for z/OS to make it available for integration.

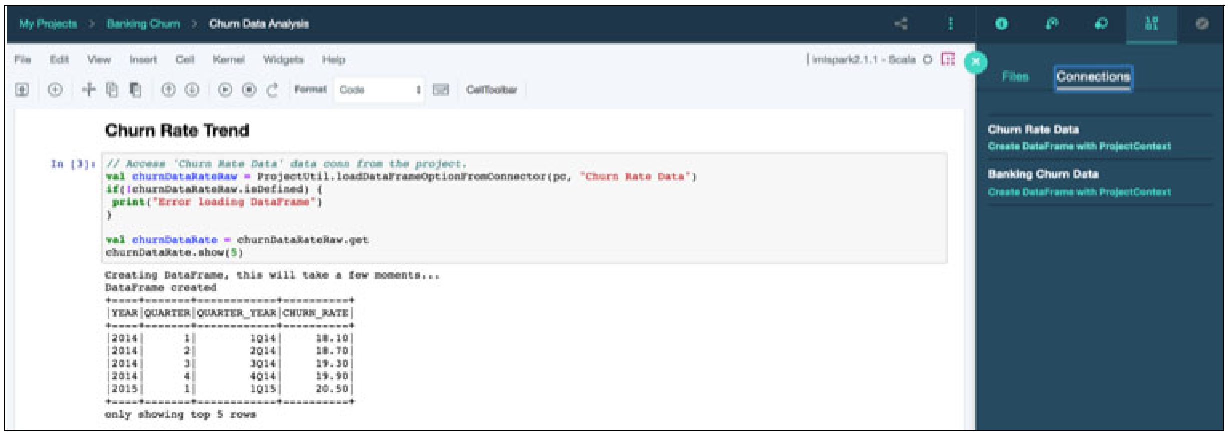
In this example, the historical churn data is stored in a Db2 for z/OS subsystem. In Machine Learning for z/OS UI, you created a “Banking Churn” project, connected to the Db2 subsystem, and defined “Churn rate data” from the data source.

6.1.1 Analyzing historical churn data

The first step in data analysis is to understand the historical churn data and identify the current trend in customer attrition. This analysis can be implemented by using a Scala-based notebook.

To accomplish this task, you create a “Banking churn analysis” notebook and specify Scala as the programming language.

To analyze the data in the notebook, you define a Spark data frame object first. This object is needed to support data transformations and to supply data for visualizations. The integrated Notebook editor supports generating code for creating data frame objects from the data sources that are defined in the project, which can be done with a few clicks (see Figure 6-1).


My Projects > Banking Churn > Churn Data Analysis
File Edit View Insert Cell Kernel Widgets Help
imlspark2.1.1 - Scala
Format Code
CellToolbar
Files
Connections
Churn Rate Data
Create DataFrame with ProjectContext
Banking Churn Data
Create DataFrame with ProjectContext
Churn Rate Trend
In [3]:

```
// Access 'Churn Rate Data' data conn from the project.
val churnDataRateRaw = ProjectUtil.loadDataFrameOptionFromConnector(pc, "Churn Rate Data")
if(!churnDataRateRaw.isDefined) {
 print("Error loading DataFrame")
}

val churnDataRate = churnDataRateRaw.get
churnDataRate.show(5)
```

```
Creating DataFrame, this will take a few moments...
DataFrame created
+----+-------+------------+----------+
|YEAR|QUARTER|QUARTER_YEAR|CHURN_RATE|
+----+-------+------------+----------+
|2014|      1|        1Q14|     18.10|
|2014|      2|        2Q14|     18.70|
|2014|      3|        3Q14|     19.30|
|2014|      4|        4Q14|     19.90|
|2015|      1|        1Q15|     20.50|
+----+-------+------------+----------+
only showing top 5 rows
```

Figure 6-1 Integrated Notebook editor

Visualization on the loaded data set is helpful for you to understand the variables and the relationship between variables. Brunel is an innovative visualization tool for analysts and business users that is used to create highly professional interactive visualizations with a few lines of descriptive code. It also can be used to visualize the data with the churn rate trend.

You might want to review the churn trend, which is the target of your model building effort. To build an interactive bar chart that represents the churn trend with Brunel, you must use the Brunel notebook extension (“Brunel magic” %%brunel). Brunel visualization language includes the following key elements that you must specify for your visualization:

- Name of the data frame (keyword: data)
- Type of the visualization (for example, bar in this case)
- Dimensions and measures of the resulting graph; for example, x(<attribute>) y(<attribute>)
- Sorting (sort(<attribute>:<ascending|descending>, <secondary attribute>: <ascending|descending>))
- Coloring (color(<attribute>))
- Tool tips (tooltip(#all))
- Axis names (axes(x:'Name for X-axis':[grid], y:'Name for Y-axis':[grid])

To generate a bar chart to show the churn rate average and trend over the course of the last 12 quarters, you can create a cell in the analytical notebook with the code that is shown in Example 6-1.

Example 6-1 Code to create a cell

```
%%brunel data('churnDataRate') x(QUARTER_YEAR) y(CHURN_RATE) bar tooltip(#all)
axes(x:'Quarters', y:'CHURN RATE':grid) sort(YEAR:ascending, QUARTER:ascending)::
width=800, height=500
```

Running the cell generates the bar chart that is shown in Figure 6-2. The average churn rate for credit card customers is approximately 20% in banking industry. As the chart in Figure 6-2 on page 120 shows, the initial churn rate was within industry standards a couple of years ago. However, the churn rate was growing from 18% to 25% over the last three years, which is higher than the industry standard.

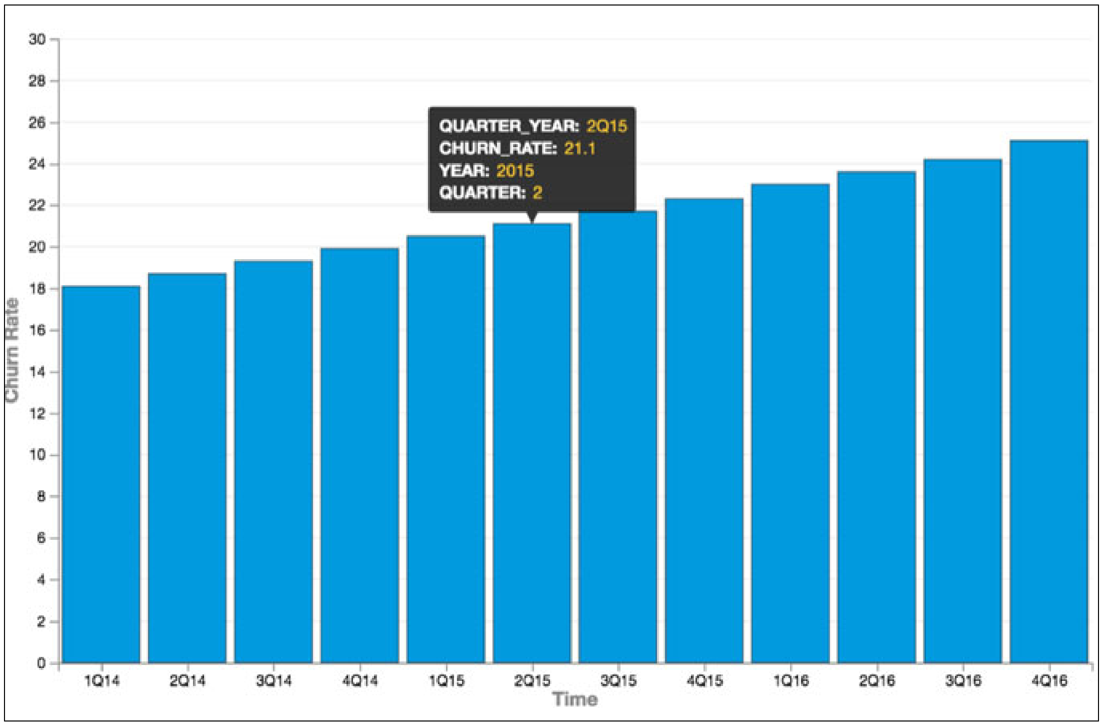


Figure 6-2 Churn rate bar chart

By visualizing various features, you might choose to continue exploring the data that is behind this churn trend to understand how a customer's demographics, tendencies, and behaviors can affect their banking decisions

Income distribution for the bank's customers can be influential. To generate the corresponding map graph, you can enter the following code:

```
val groupedChurnByState = churnData.groupBy("STATE"). agg(avg("INCOME") as
"mean_income")

%%brunel data('groupedChurnByState') map key(STATE) x(STATE) color(mean_income)
label(STATE) tooltip(#all) :: width=800, height=500
```

The results show the average income distribution by state (see Figure 6-3).

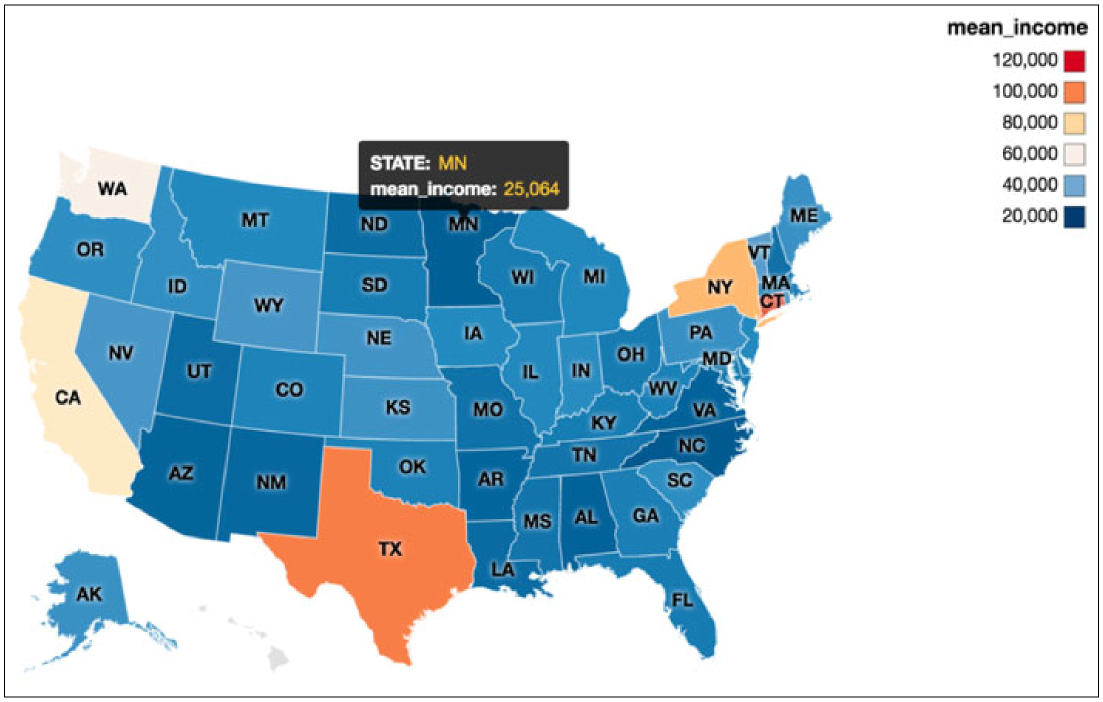


Figure 6-3 Average income distribution by state results

The colors that are used in the map that is shown in Figure 6-3 indicate the mean income levels. For example, states in dark blue have the lowest average income; states in red bring in the highest. States, such as Connecticut, New York, Texas, and California are home to customers who on average earn more than residents in all other states.

If the distribution of churn rate by state is similar to the average income map, income might be a good predictor. To validate this hypothesis, you can review the geographic distribution of the churn rate by state by entering the following code:

```
val groupedChurnByState = churnData.groupBy("STATE").
           agg(avg("CHURN") as "mean_churn")

%%brunel data('groupedChurnByState') map key(STATE) x(STATE) color(mean_churn)
label(STATE) tooltip(#all) :: width=800, height=500
```

The resulting graph shows that the bank's customers in Oregon have the highest risk to churn. Taking the average income map into consideration, most states that are suffering from high churn rate (states in red in Figure 6-4 on page 122) have relatively high average income (states that are not in blue in Figure 6-3).

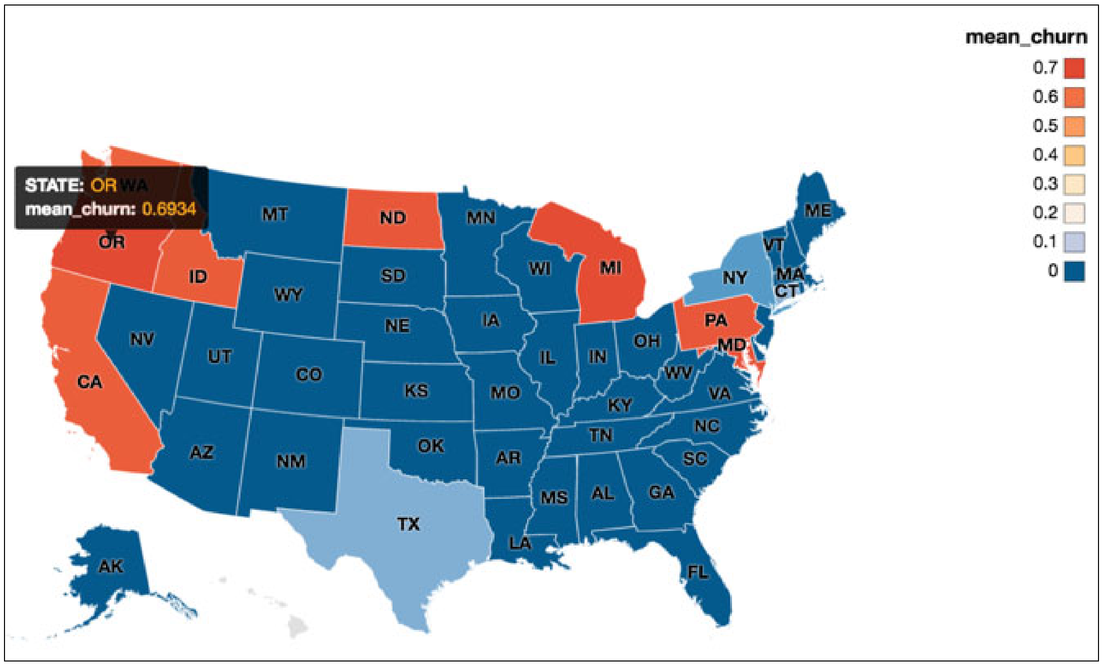


Figure 6-4 Highest churn rate risks

Apart from income, other demographic variables are likely to be influential. The next step you want to take might be to explore churners distributions by using the dynamic dashboard: churner data is color in red and non-churners are in blue.

To explore the details about the banks customers by using interactive dashboard, run the following code to generate a dashboard of charts to show the effect of age, income, and transaction activities on the churn rate:

```
%%brunel data('ChurnData')
   x(AGE) y(#count:linear) color(CHURN_LABEL) bin(AGE) interaction(select) stack
   bar tooltip(#all) filter(CHURN_LABEL) legends(none) |

   x(AVG_DAILY_TX) y(#count:linear) color(CHURN_LABEL) opacity(#selection)
   bin(AVG_DAILY_TX) stack bar tooltip(#all) axes(x:10:'AVG DAILY TX', y) |

   x(AVG_TX_AMT) y(#count:linear) color(CHURN_LABEL) bin(AVG_TX_AMT) stack bar
   tooltip(#all) axes(x, y) legends(none) |

   x(INCOME) y(#count:linear) color(CHURN_LABEL) bin(INCOME) stack bar
   tooltip(#all) axes(x, y) legends(none)
:: width=800, height=500
```

Running the code generates the interactive dashboard that is shown in Figure 6-5 on page 123, where red indicates churners and blue represents non-churners. The dashboard contains rich information. For example, the chart of customer age compared to number of churners (upper left) indicates that the churner distribution is skewed towards the millennials, while the non-churners distribution is centered around those customers at the age of 45 - 50 years old, which implies that the bank must focus on millennials to address their needs.

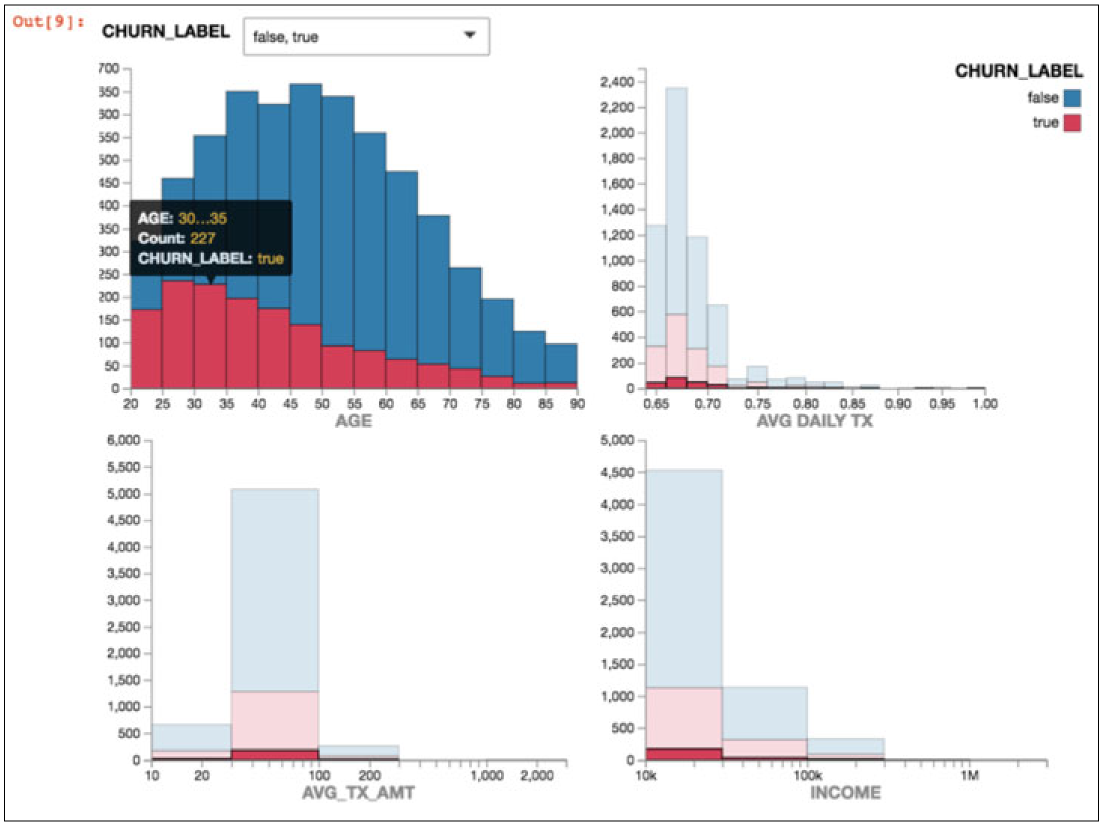


Figure 6-5 Interactive dashboard

To further explore the effect of gender, card activity, and educational background on churn rate, you can create another interactive dashboard, which is defined by using the following code:

```
%%brunel data('ChurnData')

   x(SEX) y(#count:linear) color(CHURN_LABEL) stack bar tooltip(#all) sort(SEX)
   interaction(select) filter(CHURN_LABEL) axes(x:'GENDER', y) legends(none) |

   x(ACTIVITY) y(#count:linear) color(CHURN_LABEL) stack bar tooltip(#all)
   sort(ACTIVITY) opacity(#selection) axes(x:'CARD ACTIVITY', y) |

   x(EDUCATION_GROUP) y(#count:linear) color(CHURN_LABEL) stack bar tooltip(#all)
   sort(#count) opacity(#selection) axes(x:'', y)

:: width=800, height=500
```

Running the cell generates the interactive dashboard of charts that is shown in Figure 6-6.

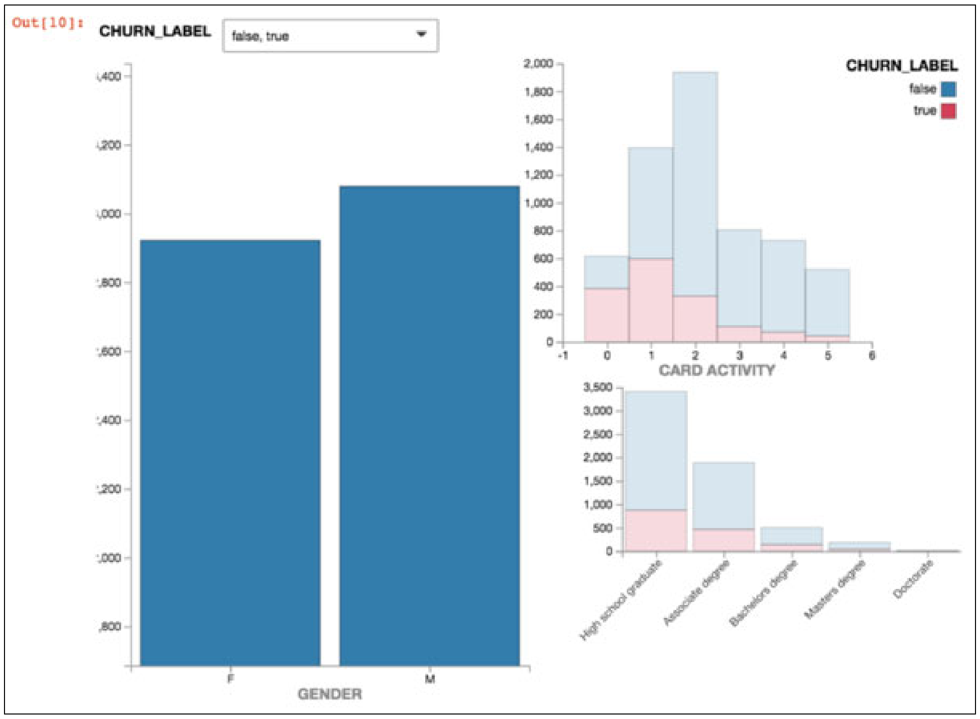


Figure 6-6 Interactive dashboard of charts

The dashboard shows that gender does not play a role in the customer churn of the cognitive bank while the frequency of bank card use does play a role. Churners tend to use their credit cards much less frequently than non-churners. In addition, the use of a bank card once versus twice per day makes a large difference in the potential of churning.

The quick analysis of the historical churn data through Brunel visualizations in the notebook finds that age, card use, income, and the state in which their customers live contribute to the current churn rate and trend. Consider selecting these features to train to build a churn model for predicting the churn, as described in the next section.

6.1.2 Building and deploying a churn model

You can train and build a model by using the Visual Model Builder or the integrated Jupyter Notebook of Machine Learning for z/OS. Regardless of the methods you use, you must explore and transform data, select features and algorithms, train and evaluate the model, and publish and deploy the model.

Creating the churn model by using the Visual Model Builder

The Visual Model Builder provides a quick and easy way to create a model because Visual Model Builder does not require any manual coding and uses a self-guided wizard.

The Visual Model Builder guides you through several phases: data preparation, defining features, specifying algorithms, training models, selecting the best performing model, and publishing the model for use in the applications.

Complete the following steps:

1. Select the data that you bring from the data source.

   For this model, the data set that contains the historical churn data is used. Machine Learning for z/OS automatically applies any necessary transformations for categorical data, although you can also choose to add transformers manually.

2. Select the following data features that can help predict customer churn:

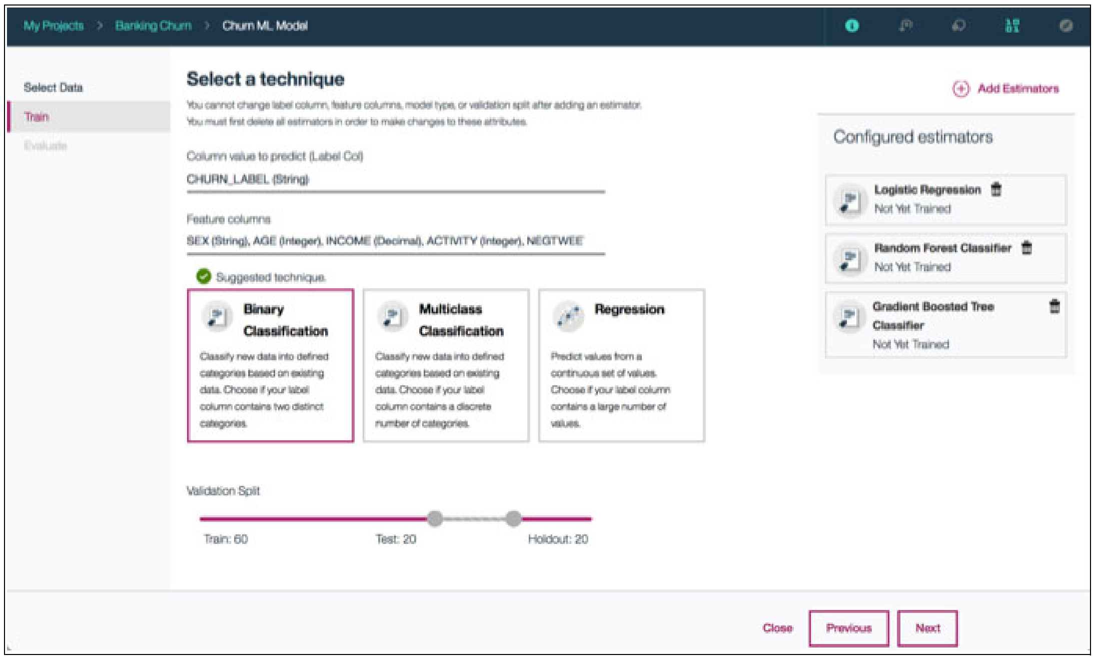
   – Choose the following fields that are to be used by the algorithm for predicting the outcome:

     - GENDER (String)
     - AGE (Integer)
     - INCOME (Decimal)
     - ACTIVITY (Integer)
     - NEGTWEETS (Integer)
     - EDUCATION_GROUP (String)

   – Choose CHURN_LABEL for the field that is used as the Target or outcome of the prediction, also known as the *label*. The target field has a value of true or false in this case.

3. Choose the algorithms that are most appropriate to train a churn model. Of the available types of algorithm, binary classification is automatically suggested when CHURN_LABEL is picked as the label column. Select binary classification as the technique and then logistic regression, random forest classifier, and gradient boosted tree classifier as the estimators.

4. Split the historical churn data into the subsets of training, validation, and testing. By default, 60% of data is used for training, 20% percent for validation, 20% for testing. The ratios can be visually adjusted by using the interface (see Figure 6-7).



Select a technique

You cannot change label column, feature columns, model type, or validation split after adding an estimator.
You must first delete all estimators in order to make changes to these attributes.

Column value to predict (Label Col)
CHURN_LABEL (String)

Feature columns
SEX (String), AGE (Integer), INCOME (Decimal), ACTIVITY (Integer), NEGTWEE

Suggested technique

Binary Classification: Classify new data into defined categories based on existing data. Choose if your label column contains two distinct categories.

Multiclass Classification: Classify new data into defined categories based on existing data. Choose if your label column contains a discrete number of categories.

Regression: Predict values from a continuous set of values. Choose if your label column contains a large number of values.

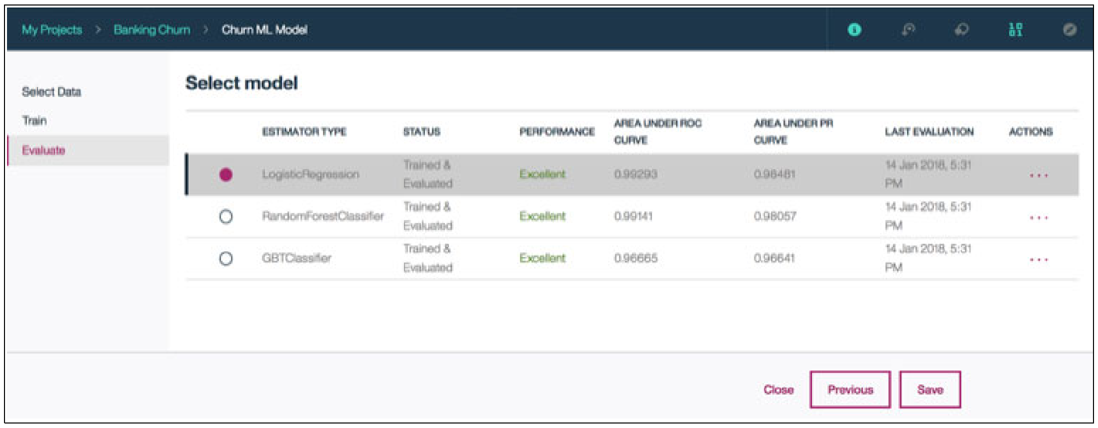
Validation Split
Train: 60 Test: 20 Holdout: 20

Configured estimators
Logistic Regression — Not Yet Trained
Random Forest Classifier — Not Yet Trained
Gradient Boosted Tree Classifier — Not Yet Trained

Figure 6-7 Selecting a technique

The churn model is now defined and ready for training.

5. Train the churn model. After the training is done, Machine Learning for z/OS presents in intuitive terms whether the user built effective models. In the case of the binary classifier, a receiver operating characteristics (ROC) curve can be used for machine learning model performance. The performance of binary models is frequently assessed by the area under the ROC curve, as shown in Figure 6-8.


My Projects > Banking Churn > Churn ML Model

Select Data
Train
Evaluate

Select model

	ESTIMATOR TYPE	STATUS	PERFORMANCE	AREA UNDER ROC CURVE	AREA UNDER PR CURVE	LAST EVALUATION	ACTIONS
●	LogisticRegression	Trained & Evaluated	Excellent	0.99293	0.98481	14 Jan 2018, 5:31 PM	...
○	RandomForestClassifier	Trained & Evaluated	Excellent	0.99141	0.98057	14 Jan 2018, 5:31 PM	...
○	GBTClassifier	Trained & Evaluated	Excellent	0.96665	0.96641	14 Jan 2018, 5:31 PM	...

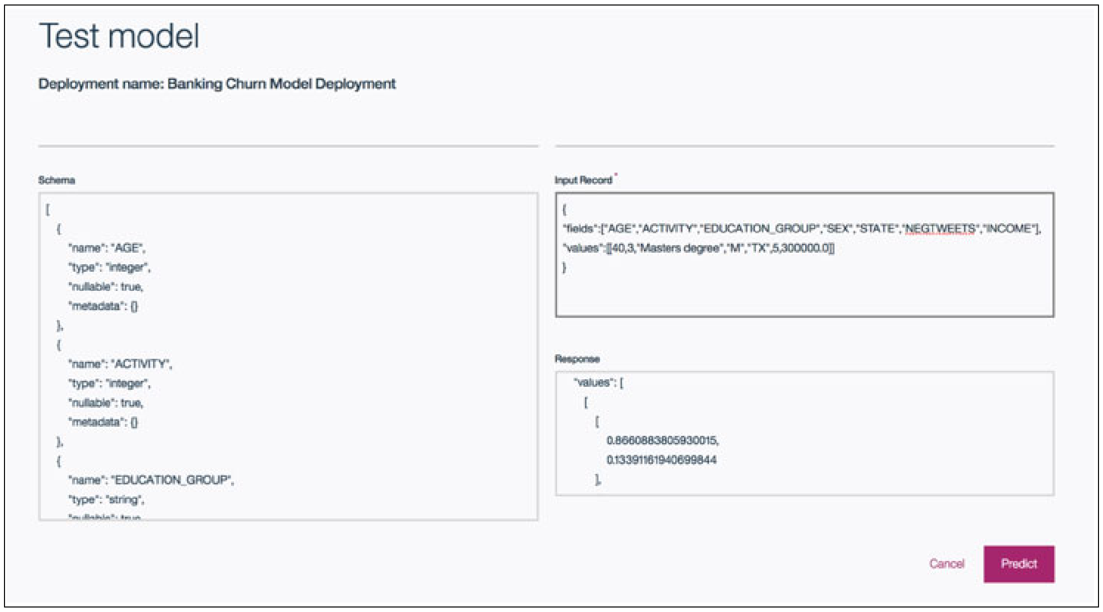
Figure 6-8 Performance assessment

6. Choose and save the model. In this case, the version that is based on the logistic regression algorithm delivers an excellent performance with the highest area under ROC curve of 0.99293. You can select this model as final and save it as the Banking churn model.

7. Select and deploy the Banking churn model. After the churn model is deployed, you can test the model (see Figure 6-9).

   For input, specify the following fields and values that characterize a customer of the bank:

   `Gender - M, Age - 40, INCOME - 300000.0, Activity - 3, NEGTWEETS - 5, STATE - TX`


Test model

Deployment name: Banking Churn Model Deployment

Schema

```
[
  {
    "name": "AGE",
    "type": "integer",
    "nullable": true,
    "metadata": {}
  },
  {
    "name": "ACTIVITY",
    "type": "integer",
    "nullable": true,
    "metadata": {}
  },
  {
    "name": "EDUCATION_GROUP",
    "type": "string",
```

Input Record *

```
{
"fields":["AGE","ACTIVITY","EDUCATION_GROUP","SEX","STATE","NEGTWEETS","INCOME"],
"values":[[40,3,"Masters degree","M","TX",5,300000.0]]
}
```

Response

```
  "values": [
    [
      [
        0.8660883805930015,
        0.1339116194069844
      ],
```

Figure 6-9 Testing the model

Note: The Input Record field accepts key-value pairs in the format of schema. The Predict button is active only when the input pairs are valid.

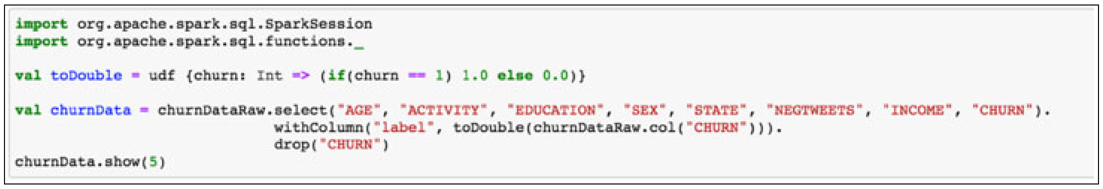
The response includes two numbers that should add up to 1. The numbers represent the predicted possibilities of two classes based on the input data. In binary classification, the first class is negative and the second is positive. The second number of 0.12391161940699844 indicates that the customer has the 13.39% probability to churn.

Creating the churn model by using the integrated Notebook editor

You can programmatically code the Scala-based notebook with APIs to create the same or similar churn model as you did visually by using the Visual Model Builder. The integrated notebook can run code in different languages, including Scala. When you run the code in a cell, the notebook appends the output right after the original code, with the top five rows shown.

The following process that uses the integrated Netbook editor is similar to building the model visually with Visual Model Builder:

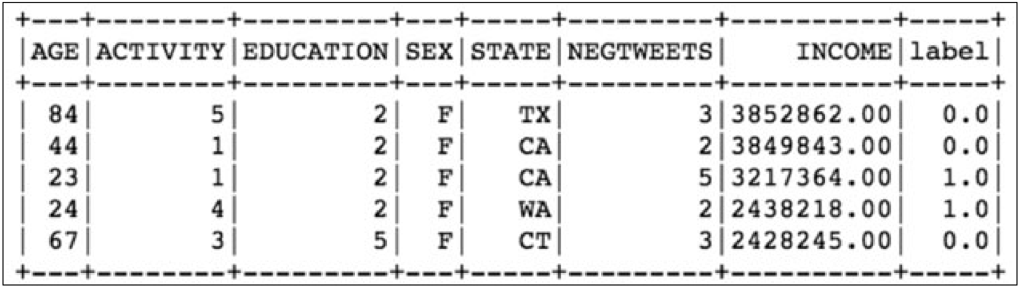
1. Prepare and preview the churn data by entering the code that is shown in Figure 6-10.



```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._

val toDouble = udf {churn: Int => (if(churn == 1) 1.0 else 0.0)}

val churnData = churnDataRaw.select("AGE", "ACTIVITY", "EDUCATION", "SEX", "STATE", "NEGTWEETS", "INCOME", "CHURN").
                          withColumn("label", toDouble(churnDataRaw.col("CHURN"))).
                          drop("CHURN")
churnData.show(5)
```

Figure 6-10 Code to enter

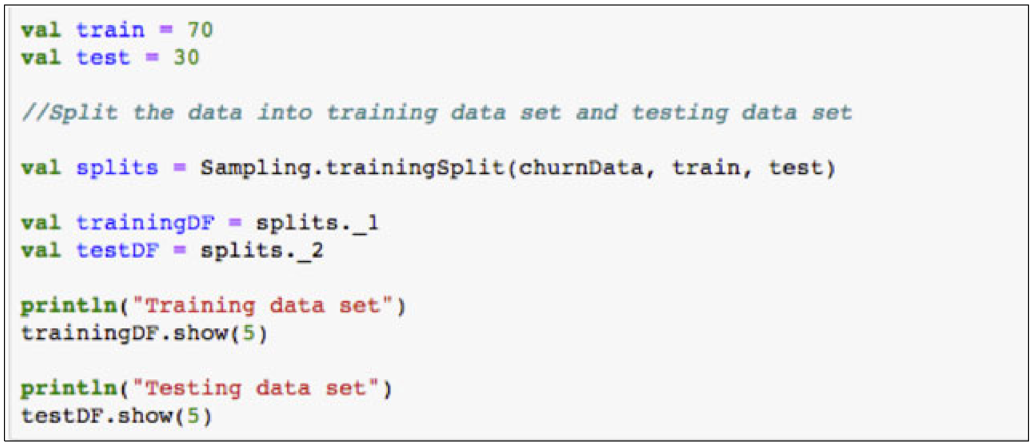
Running the code that is shown in Figure 6-10 in the cell generates the output that shows the historical churn data in categories or table view (see Figure 6-11).



```
+---+--------+---------+---+-----+---------+----------+-----+
|AGE|ACTIVITY|EDUCATION|SEX|STATE|NEGTWEETS|    INCOME|label|
+---+--------+---------+---+-----+---------+----------+-----+
| 84|       5|        2|  F|   TX|        3|3852862.00|  0.0|
| 44|       1|        2|  F|   CA|        2|3849843.00|  0.0|
| 23|       1|        2|  F|   CA|        5|3217364.00|  1.0|
| 24|       4|        2|  F|   WA|        2|2438218.00|  1.0|
| 67|       3|        5|  F|   CT|        3|2428245.00|  0.0|
+---+--------+---------+---+-----+---------+----------+-----+
```

Figure 6-11 Historical churn data results

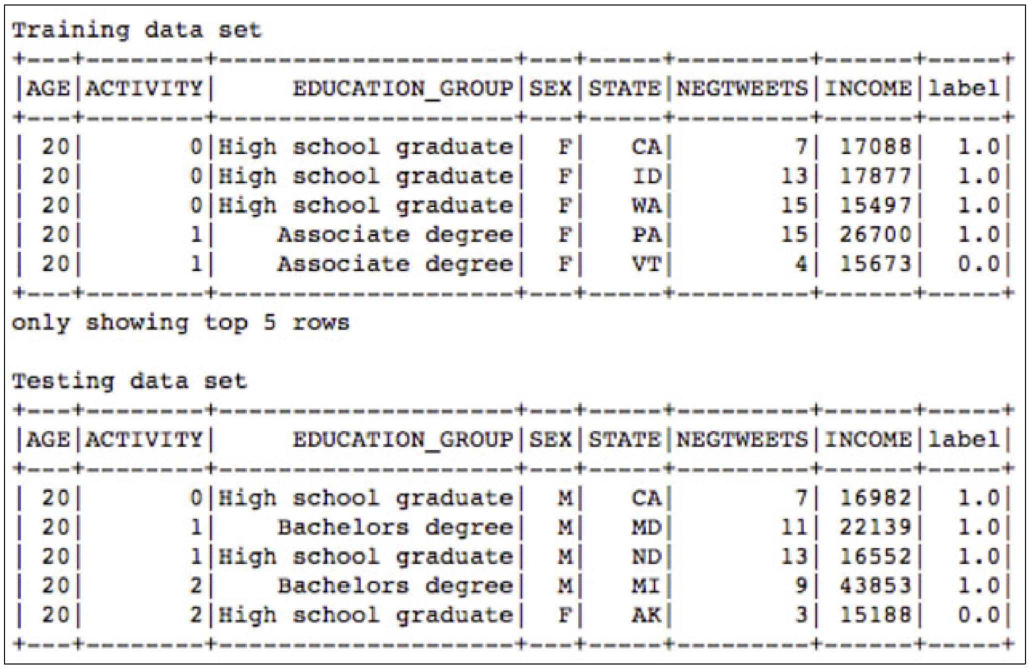
2. Split the data into two subsets (70% for training and 30% for testing) by entering the code that is shown in Figure 6-12.



```
val train = 70
val test = 30

//Split the data into training data set and testing data set

val splits = Sampling.trainingSplit(churnData, train, test)

val trainingDF = splits._1
val testDF = splits._2

println("Training data set")
trainingDF.show(5)

println("Testing data set")
testDF.show(5)
```

Figure 6-12   Splitting data into two subsets

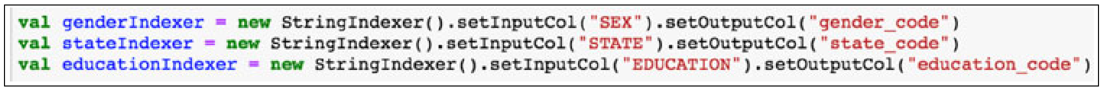
Running the cell generates the previews of the training and testing data sets that are shown in Figure 6-13.



```
Training data set
+---+--------+--------------------+---+-----+---------+------+-----+
|AGE|ACTIVITY|     EDUCATION_GROUP|SEX|STATE|NEGTWEETS|INCOME|label|
+---+--------+--------------------+---+-----+---------+------+-----+
| 20|       0|High school graduate|  F|   CA|        7| 17088|  1.0|
| 20|       0|High school graduate|  F|   ID|       13| 17877|  1.0|
| 20|       0|High school graduate|  F|   WA|       15| 15497|  1.0|
| 20|       1|    Associate degree|  F|   PA|       15| 26700|  1.0|
| 20|       1|    Associate degree|  F|   VT|        4| 15673|  0.0|
+---+--------+--------------------+---+-----+---------+------+-----+
only showing top 5 rows

Testing data set
+---+--------+--------------------+---+-----+---------+------+-----+
|AGE|ACTIVITY|     EDUCATION_GROUP|SEX|STATE|NEGTWEETS|INCOME|label|
+---+--------+--------------------+---+-----+---------+------+-----+
| 20|       0|High school graduate|  M|   CA|        7| 16982|  1.0|
| 20|       1|    Bachelors degree|  M|   MD|       11| 22139|  1.0|
| 20|       1|High school graduate|  M|   ND|       13| 16552|  1.0|
| 20|       2|    Bachelors degree|  M|   MI|        9| 43853|  1.0|
| 20|       2|High school graduate|  F|   AK|        3| 15188|  0.0|
+---+--------+--------------------+---+-----+---------+------+-----+
```

Figure 6-13   Training and testing data sets preview

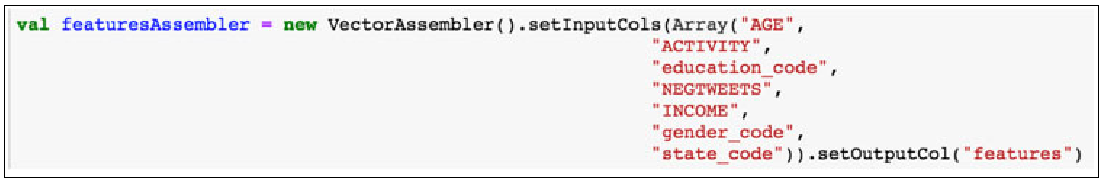
3. Build features by using transformers (see Figure 6-14). As opposed to the UI-based approach, you must code every transformation.



```
val genderIndexer = new StringIndexer().setInputCol("SEX").setOutputCol("gender_code")
val stateIndexer = new StringIndexer().setInputCol("STATE").setOutputCol("state_code")
val educationIndexer = new StringIndexer().setInputCol("EDUCATION").setOutputCol("education_code")
```

Figure 6-14   Coding transformations

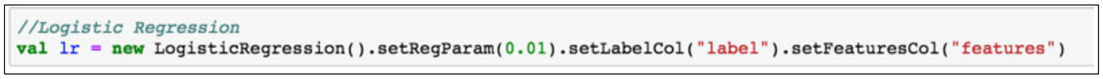
4. Assemble the features to be used in model training by entering the code that is shown in Figure 6-15.



```
val featuresAssembler = new VectorAssembler().setInputCols(Array("AGE",
                                                          "ACTIVITY",
                                                          "education_code",
                                                          "NEGTWEETS",
                                                          "INCOME",
                                                          "gender_code",
                                                          "state_code")).setOutputCol("features")
```

Figure 6-15 Assembling model training features

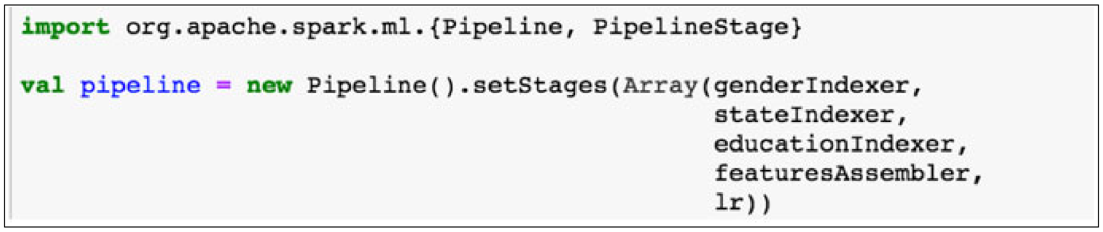
5. Define the algorithm for the churn model. The logistic regression that is used is shown in Figure 6-16.



```
//Logistic Regression
val lr = new LogisticRegression().setRegParam(0.01).setLabelCol("label").setFeaturesCol("features")
```

Figure 6-16 Logical regression algorithm

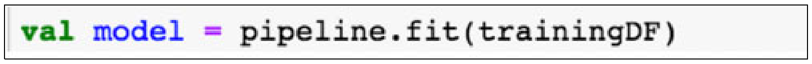
6. Create a complete pipeline for the logistic regression churn model (see Figure 6-17).



```
import org.apache.spark.ml.{Pipeline, PipelineStage}

val pipeline = new Pipeline().setStages(Array(genderIndexer,
                                              stateIndexer,
                                              educationIndexer,
                                              featuresAssembler,
                                              lr))
```

Figure 6-17 Logical regression churn model

7. Train the churn model by passing the training data into the pipeline's fit method (see Figure 6-18).



```
val model = pipeline.fit(trainingDF)
```

Figure 6-18 Pipeline fit method

8. Evaluate the churn model. The ROC curve indicates the top performance, as shown in Figure 6-19.

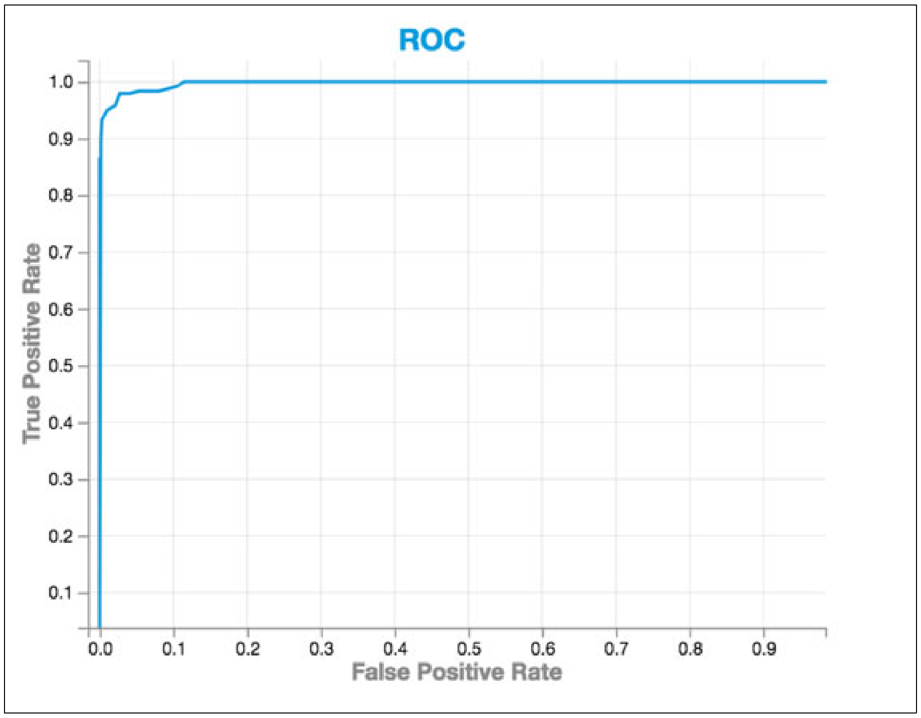
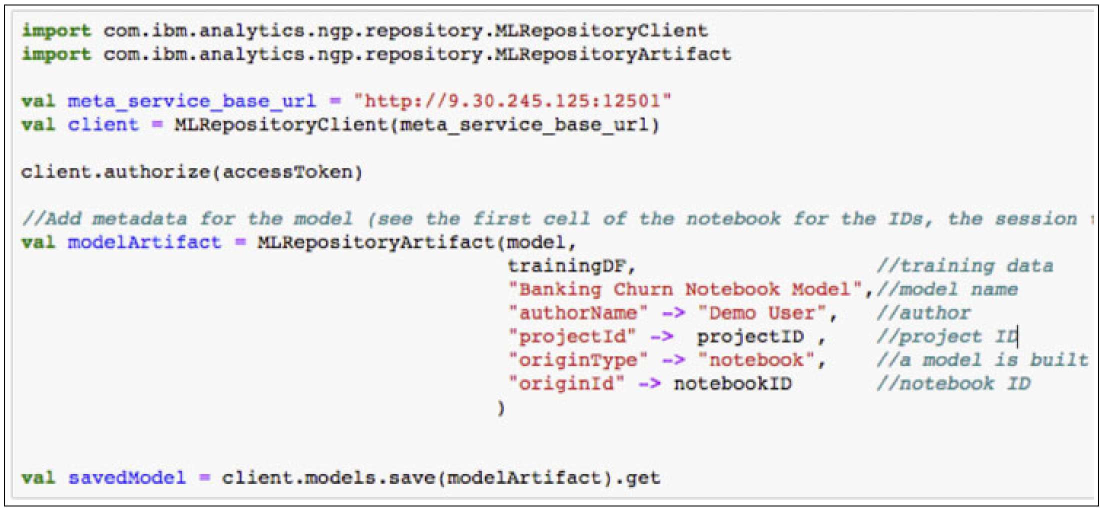


Figure 6-19 Churn model evaluation

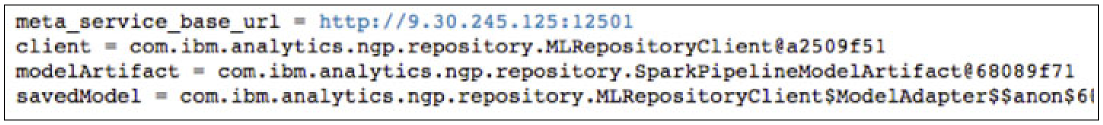
9. Save the model and then, publish it to the repository service (see Figure 6-20).



```
import com.ibm.analytics.ngp.repository.MLRepositoryClient
import com.ibm.analytics.ngp.repository.MLRepositoryArtifact

val meta_service_base_url = "http://9.30.245.125:12501"
val client = MLRepositoryClient(meta_service_base_url)

client.authorize(accessToken)

//Add metadata for the model (see the first cell of the notebook for the IDs, the session
val modelArtifact = MLRepositoryArtifact(model,
                                         trainingDF,                    //training data
                                         "Banking Churn Notebook Model",//model name
                                         "authorName" -> "Demo User",   //author
                                         "projectId" ->  projectID ,    //project ID
                                         "originType" -> "notebook",    //a model is built
                                         "originId" -> notebookID       //notebook ID
                                        )


val savedModel = client.models.save(modelArtifact).get
```

Figure 6-20 Publishing model to repository service

Running the code generates the output that is shown in Figure 6-21 that shows a new churn model is successfully saved and published.

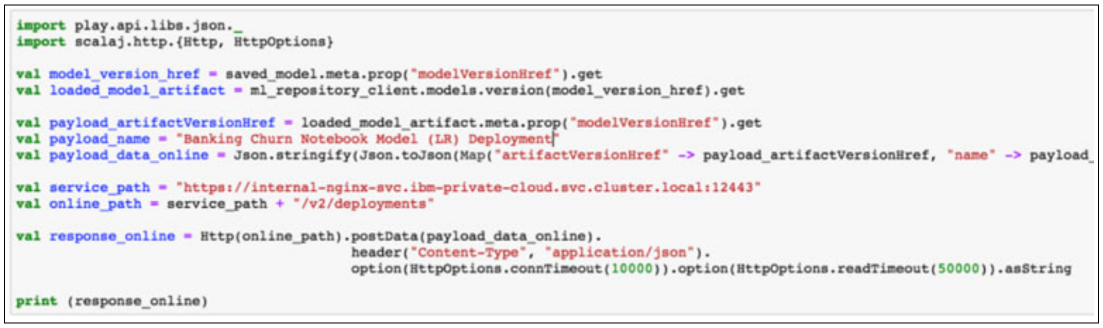


```
meta_service_base_url = http://9.30.245.125:12501
client = com.ibm.analytics.ngp.repository.MLRepositoryClient@a2509f51
modelArtifact = com.ibm.analytics.ngp.repository.SparkPipelineModelArtifact@68089f71
savedModel = com.ibm.analytics.ngp.repository.MLRepositoryClient$ModelAdapter$$anon$6
```

Figure 6-21 Output that shows successfully saved and published churn model

You can also verify that the churn model is successfully published by reviewing the Models tab of the Model Management page.

10. Deploy the churn model by entering the code that is shown in Figure 6-22.

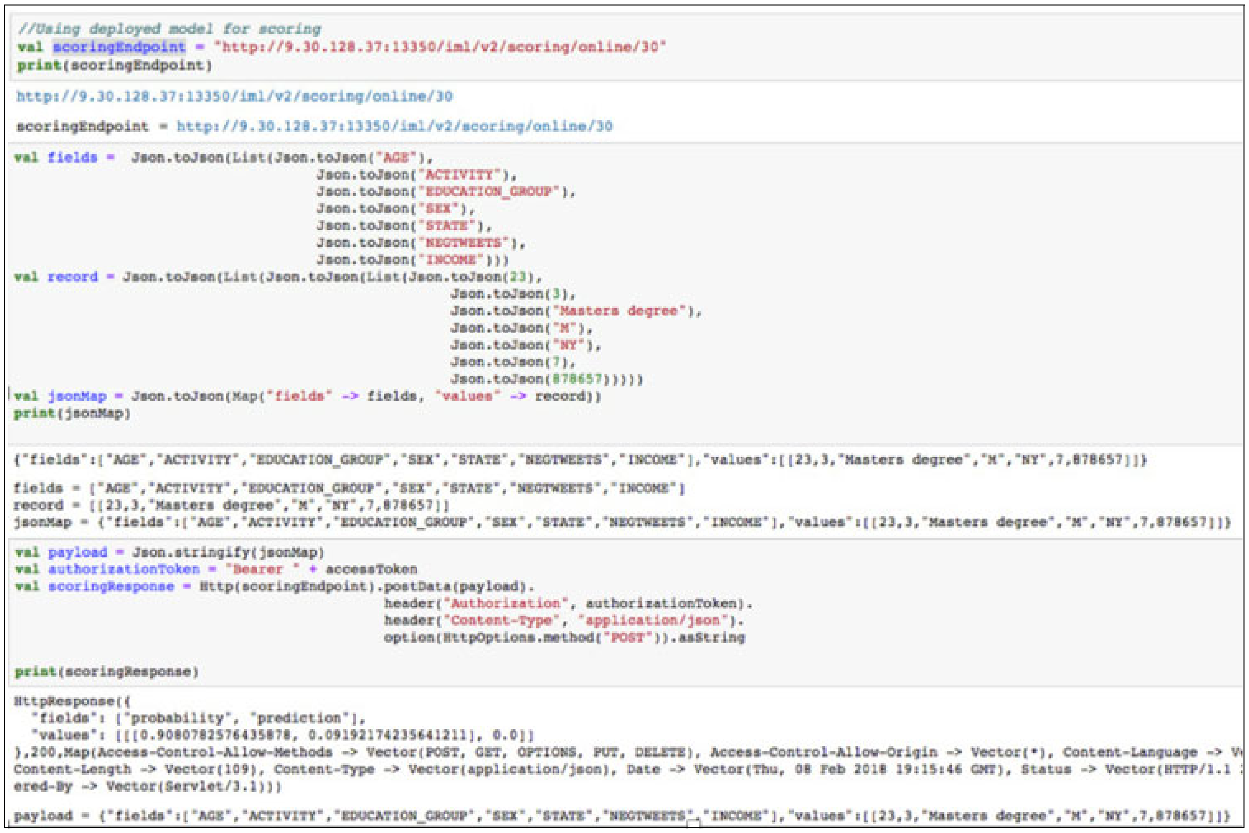


```
import play.api.libs.json._
import scalaj.http.{Http, HttpOptions}

val model_version_href = saved_model.meta.prop("modelVersionHref").get
val loaded_model_artifact = ml_repository_client.models.version(model_version_href).get

val payload_artifactVersionHref = loaded_model_artifact.meta.prop("modelVersionHref").get
val payload_name = "Banking Churn Notebook Model (LR) Deployment"
val payload_data_online = Json.stringify(Json.toJson(Map("artifactVersionHref" -> payload_artifactVersionHref, "name" -> payload_

val service_path = "https://internal-nginx-svc.ibm-private-cloud.svc.cluster.local:12443"
val online_path = service_path + "/v2/deployments"

val response_online = Http(online_path).postData(payload_data_online).
                                        header("Content-Type", "application/json").
                                        option(HttpOptions.connTimeout(10000)).option(HttpOptions.readTimeout(50000)).asString

print (response_online)
```

Figure 6-22 Deploying churn model

The output from running the code indicates that the churn model was deployed quickly and successfully.

11. Run scoring and predictions to test the churn model. Issue REST calls from an application that was developed in Python, PHP, Java, Ruby on rails, or another language. An example for a Scala implementation is shown in Figure 6-23.



```
//Using deployed model for scoring
val scoringEndpoint = "http://9.30.128.37:13350/iml/v2/scoring/online/30"
print(scoringEndpoint)

http://9.30.128.37:13350/iml/v2/scoring/online/30

scoringEndpoint = http://9.30.128.37:13350/iml/v2/scoring/online/30

val fields =  Json.toJson(List(Json.toJson("AGE"),
                               Json.toJson("ACTIVITY"),
                               Json.toJson("EDUCATION_GROUP"),
                               Json.toJson("SEX"),
                               Json.toJson("STATE"),
                               Json.toJson("NEGTWEETS"),
                               Json.toJson("INCOME")))
val record = Json.toJson(List(Json.toJson(List(Json.toJson(23),
                                               Json.toJson(3),
                                               Json.toJson("Masters degree"),
                                               Json.toJson("M"),
                                               Json.toJson("NY"),
                                               Json.toJson(7),
                                               Json.toJson(878657)))))
val jsonMap = Json.toJson(Map("fields" -> fields, "values" -> record))
print(jsonMap)

{"fields":["AGE","ACTIVITY","EDUCATION_GROUP","SEX","STATE","NEGTWEETS","INCOME"],"values":[[23,3,"Masters degree","M","NY",7,878657]]}

fields = ["AGE","ACTIVITY","EDUCATION_GROUP","SEX","STATE","NEGTWEETS","INCOME"]
record = [[23,3,"Masters degree","M","NY",7,878657]]
jsonMap = {"fields":["AGE","ACTIVITY","EDUCATION_GROUP","SEX","STATE","NEGTWEETS","INCOME"],"values":[[23,3,"Masters degree","M","NY",7,878657]]}

val payload = Json.stringify(jsonMap)
val authorizationToken = "Bearer " + accessToken
val scoringResponse = Http(scoringEndpoint).postData(payload).
                                    header("Authorization", authorizationToken).
                                    header("Content-Type", "application/json").
                                    option(HttpOptions.method("POST")).asString

print(scoringResponse)

HttpResponse({
  "fields": ["probability", "prediction"],
  "values": [[[0.9080782576435878, 0.09192174235641211], 0.0]]
},200,Map(Access-Control-Allow-Methods -> Vector(POST, GET, OPTIONS, PUT, DELETE), Access-Control-Allow-Origin -> Vector(*), Content-Language -> V
Content-Length -> Vector(109), Content-Type -> Vector(application/json), Date -> Vector(Thu, 08 Feb 2018 19:15:46 GMT), Status -> Vector(HTTP/1.1
ered-By -> Vector(Servlet/3.1)))

payload = {"fields":["AGE","ACTIVITY","EDUCATION_GROUP","SEX","STATE","NEGTWEETS","INCOME"],"values":[[23,3,"Masters degree","M","NY",7,878657]]}
```

Figure 6-23 Scala implementation example

6.1.3 Monitoring and reevaluating the churn model

The data is not stale and it changes over time. As a result, the performance of the model can degrade, which erodes the accuracy of the predictions. You can prevent this issue from occurring by monitoring the performance of the churn model and scheduling it for automatic reevaluation.

To set the churn model for automatic retraining, determine a performance threshold and an evaluation frequency. Also, specify the data source for retraining the model.

After the reevaluation is scheduled to run once or at a regular interval, Machine Learning for Z/OS collects and displays the model performance information on the dashboard for you to monitor.

6.2 Investment advisory: Helping clients make the right decisions

In investment firms, investment advisors strive to give the right recommendations that help clients make the right decisions based on the client's unique situation. Given the changing dynamics in financial markets and the inherent risks in investment, making sound recommendations require time-sensitive intelligence that is extracted from raw information about an investment, which varies from the financial performance of target industries to the inclinations of the client.

By using Machine Learning for z/OS, you can analyze clients and financial markets, no matter where the data is stored. By using this analysis, you can quickly extract insights with which you can turn into actionable recommendations.

Consider the industry affinity of your clients as an example. You want to predict and identify the industries in which your clients are interested in investing. Thus, together with your data scientists, you analyze the client data and predict the client's industry affinity by developing a machine learning model programmatically in Python with PySpark.

6.2.1 Analyze historical client affinity data

A sample data set about the clients is stored in a dashDB. This sample client data contains information about 10,000 clients, and includes the following variables:

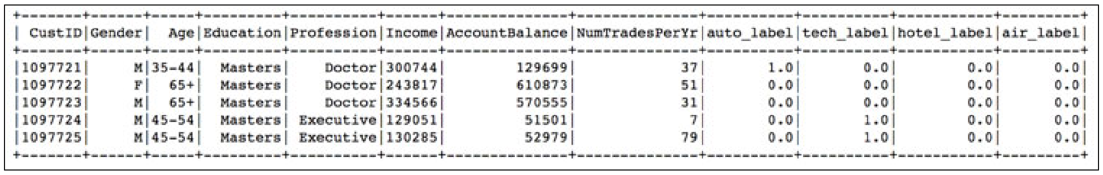
- Client ID (numeric; 1000000 - 1100000)
- Gender (nominal; M = Male F = Female)
- Age group (ordinal; 18 - 24, 25 - 34, 45 - 54, 55 - 64, 65+)
- Education (ordinal; from high school to doctorate)
- Profession (nominal; doctor, nurse, performer, and so on)
- Income (numeric; 20,000 - $58,000)
- Account balance (numeric; $3,000 - $283,000)
- Number of trades per year (numeric; 0 - 100)

The client data also contains labels for the auto, technology, hotel, and airline industries where "1" means affinity and "0" means no affinity.

To analyze the data, you first must load the data from dashDB in a notebook. This process can be done by adding a remote data set as described in Chapter 5, "Model development and deployment: A retail example" on page 93, or by specifying credentials for the database and retrieving data after the connection is set up. The following code shows how to specify credentials for a dashDB connection:

```
dashDBLocal = {
   'jdbcurl': 'yourURL'
   'user': 'yourUsername'
   'password': 'yourPassword'

}
```

After the connection is set up, you can load and preview the client data. The top five rows are shown in Figure 6-24 (each row being a client).



```
+-------+------+-----+---------+----------+------+--------------+--------------+----------+----------+-----------+---------+
| CustID|Gender|  Age|Education|Profession|Income|AccountBalance|NumTradesPerYr|auto_label|tech_label|hotel_label|air_label|
+-------+------+-----+---------+----------+------+--------------+--------------+----------+----------+-----------+---------+
|1097721|     M|35-44|  Masters|    Doctor|300744|        129699|            37|       1.0|       0.0|        0.0|      0.0|
|1097722|     F|  65+|  Masters|    Doctor|243817|        610873|            51|       0.0|       0.0|        0.0|      0.0|
|1097723|     M|  65+|  Masters|    Doctor|334566|        570555|            31|       0.0|       0.0|        0.0|      0.0|
|1097724|     M|45-54|  Masters| Executive|129051|         51501|             7|       0.0|       1.0|        0.0|      0.0|
|1097725|     M|45-54|  Masters| Executive|130285|         52979|            79|       0.0|       1.0|        0.0|      0.0|
+-------+------+-----+---------+----------+------+--------------+--------------+----------+----------+-----------+---------+
```

Figure 6-24 Client data preview

Before reviewing machine learning models, you might want to get more knowledge of the industry affinity of your customers. You can expect different customers have affinity to different industry, and such differences might be explained to some extent by the demographics.

Consider the automotive industry as an example. By running the Brunel code that is shown in Example 6-2, you can create several stacked bar plots to show the distribution of demographic variables between customers having affinity for the automotive industry (in red) and those who do not (in blue), as shown in Figure 6-25 on page 135.

Example 6-2 Brunel code to create bar plots

```
%%brunel data('autoStocks')
   x(TradesPerYearGroup:linear) y(total:linear)  sum(total) stack bar
   tooltip(#all)  axes(x:'Trades Per Year', y) color(Auto) legends(none) |

   x(AgeGroup) y(total:linear) sum(total) stack bar tooltip(#all) axes(x:'Age
   Group', y) color(Auto) |

   x(Gender) y(total:linear) sum(total)  stack bar tooltip(#all) axes(x:'Gender',
   y:' ') color(Auto) legends(none)|

   x(IncomeGroup:linear) y(total:linear) sum(total)  stack bar tooltip(#all)
   axes(x:'Annual Income', y) color(Auto) legends(none)
:: width=1000, height=800
```

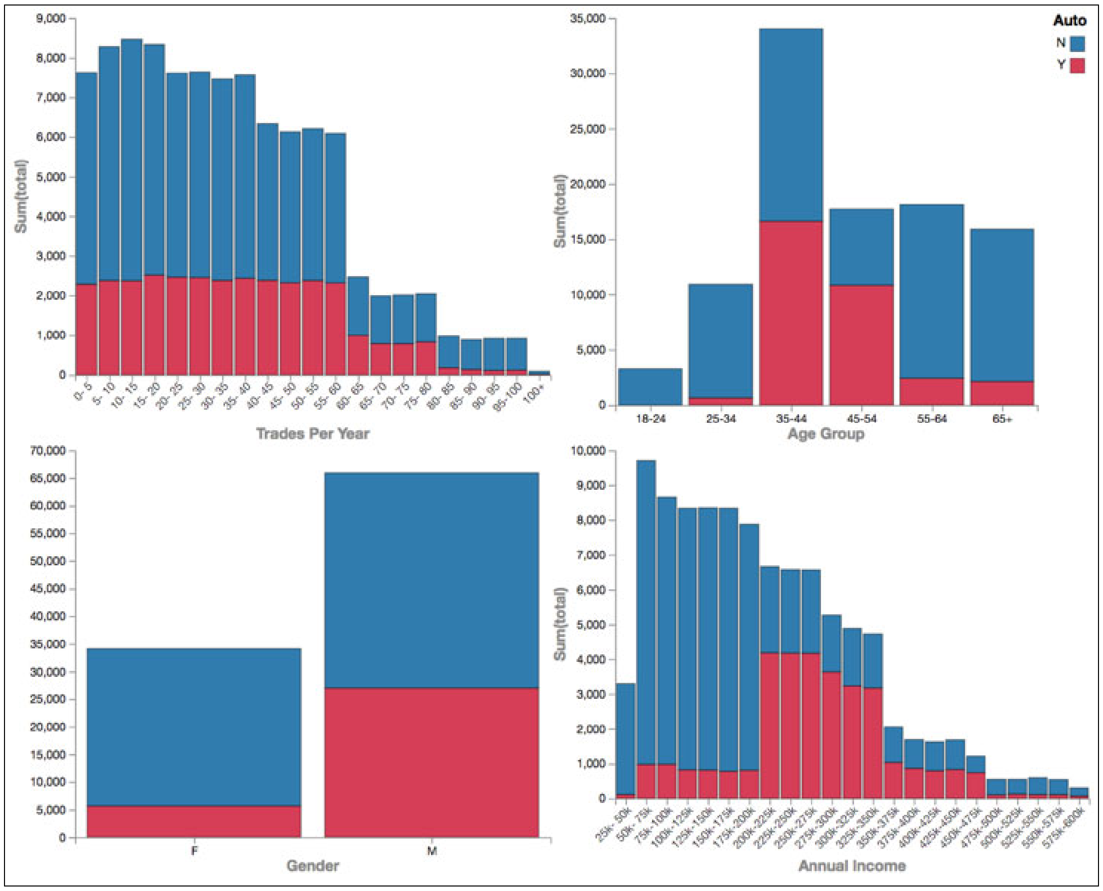


Figure 6-25   Distribution of demographic variables bar graphs

You observe that among all the age groups, middle-aged clients are much more likely to show affinity for automotive industry, most of whom are male, and have a high annual income $200,000 - $475,000.

So, the question becomes: How are customers who have affinity for the automotive industry different from those having affinity for another industry? To answer this question, you create a similar set of plots for technology industry by using the following code:

```
%%brunel data('techStocks')
   x(TradesPerYearGroup:linear) y(total:linear)  sum(total) stack bar
   tooltip(#all)  axes(x:'Trades Per Year', y) color(Tech) legends(none) |

   x(AgeGroup) y(total:linear) sum(total) stack bar tooltip(#all)  axes(x:'Age
   Group', y) color(Tech)|

   x(Gender) y(total:linear) sum(total)  stack bar tooltip(#all) axes(x:'Gender',
   y:' ') color(Tech) legends(none) |

   x(IncomeGroup:linear) y(total:linear) sum(total)  stack bar tooltip(#all)
   axes(x:'Annual Income', y) color(Tech) legends(none)

:: width=1000, height=800
```

The resulting bar graphs are show in Figure 6-26.

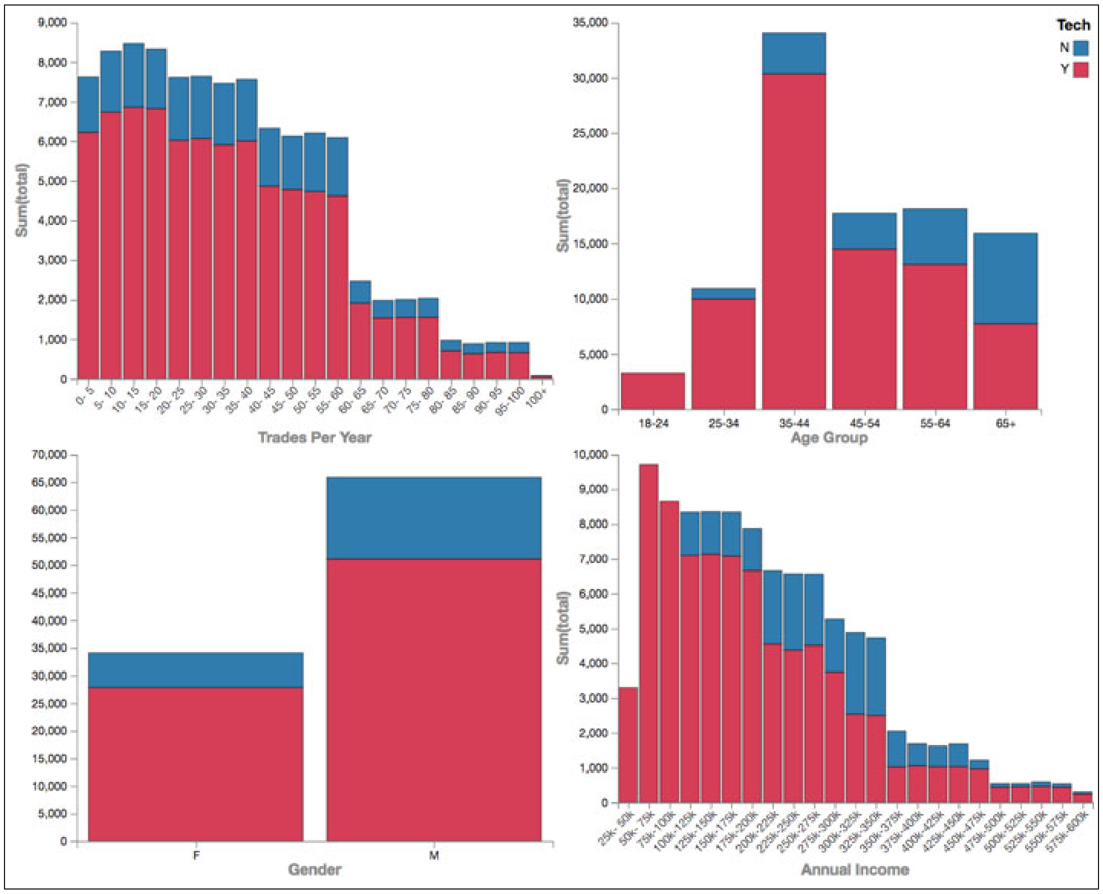


Figure 6-26 Technology industry bar graphs

Because the sum of the red and blue bars that are shown in Figure 6-26 represents all of the customers in your data, you see the same shape of the stacked bars, but the proportion of colors varies.

The high proportion of red indicates that the technology industry is popular among all age groups (younger people are more keen on it than older people), both genders (females are a little more likely than males), and customers with any income level (relatively lower income group and relatively higher income group show higher probability than middle income group).

If you compare the two sets of plots, it is evident that clients with an affinity for the auto industry have distinctively different demographics from those with an affinity for the technology industry. This difference in distribution indicates why demographics can be good predictors and how they can be used to predict affinity.

6.2.2 Defining a pipeline for an affinity model

Machine learning in Spark uses specific workflow, or pipeline, where various stages (transform features, specify machine learning algorithms, and so on) are piped together for the data to be passed into.

You can change the components when you define the pipeline because it is a collection of rules or actions that has no effect on the data yet. After the pipeline is defined, data can be passed into the pipeline by using the `fit()` method, and it is then that the time when the computation starts.

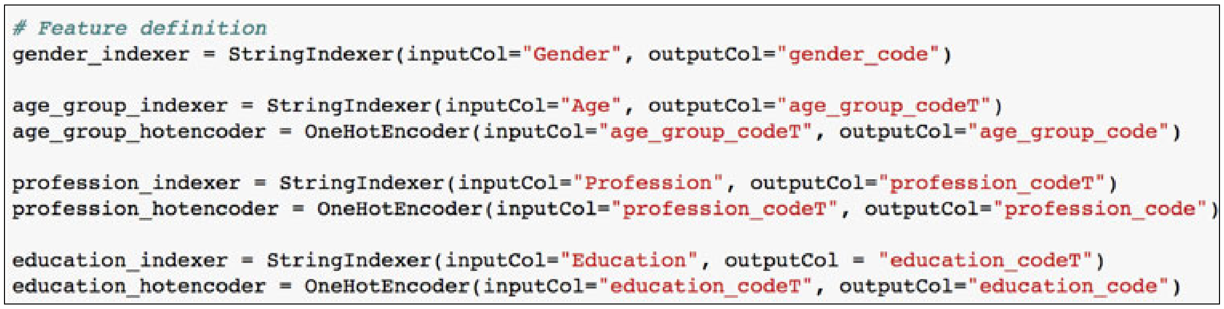
To define a pipeline, define the components of the pipeline, stage by stage, and then, combine the components to make a complete pipeline.

In this example, you can prepare the pipeline by using the following process:

1. Transform categorical features in your client data to numeric representations.

   The computation of machine learning algorithms is, in essence, mathematical operations. Therefore, categorical variables must be represented as numbers. To handle categorical features, you can use the `StringIndexer()` in Scala to turn categorical variables to a frequency-based rank number. Then, `OneHotEncoder()` is used to map the numbers to one hot vector. These transformations add new columns to the data frame (the original categorical features still exit).

   The cell that is shown in Figure 6-27 applies string indexer and one hot encoder to categorical variables in this data set. In this case, the binary categorical feature Gender does not require a one hot mapping. Age Group, Profession, and Education as multi-value categorical features must be transformed by using StringIndexer and OneHotEncoder.



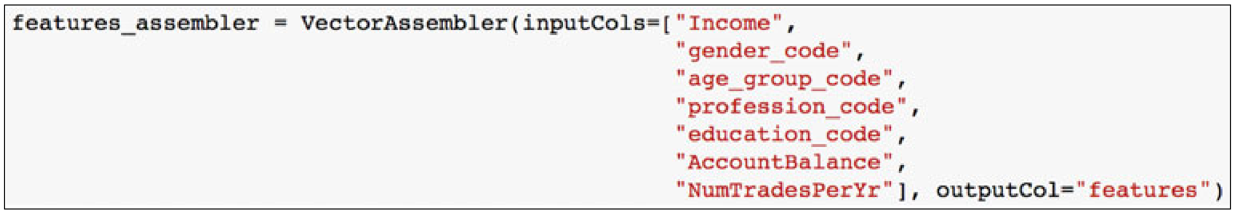
```
# Feature definition
gender_indexer = StringIndexer(inputCol="Gender", outputCol="gender_code")

age_group_indexer = StringIndexer(inputCol="Age", outputCol="age_group_codeT")
age_group_hotencoder = OneHotEncoder(inputCol="age_group_codeT", outputCol="age_group_code")

profession_indexer = StringIndexer(inputCol="Profession", outputCol="profession_codeT")
profession_hotencoder = OneHotEncoder(inputCol="profession_codeT", outputCol="profession_code")

education_indexer = StringIndexer(inputCol="Education", outputCol = "education_codeT")
education_hotencoder = OneHotEncoder(inputCol="education_codeT", outputCol="education_code")
```

Figure 6-27 Cell that applies string indexer and one hot encoder

   Running the code generates the numeric representations you need of the categorical variables (that is, gender, age, profession, and education).

2. Select the feature columns to be used in the affinity model.

   Not all columns are feasible (because the categorical features are still in the data frame) or wanted. The code that is shown in Figure 6-28 uses Scala's `VectorAssembler()` to select the features to be used.



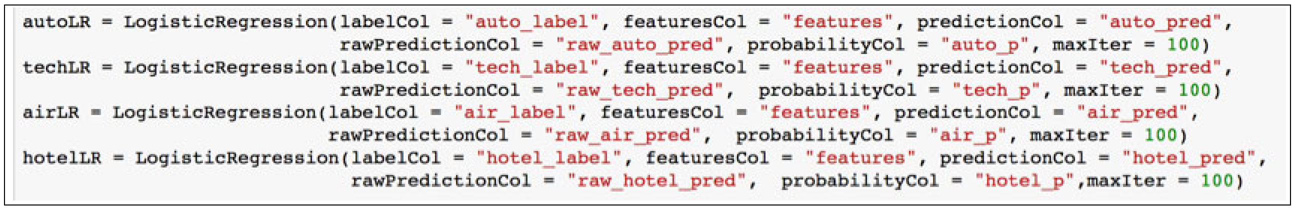
```
features_assembler = VectorAssembler(inputCols=["Income",
                                                "gender_code",
                                                "age_group_code",
                                                "profession_code",
                                                "education_code",
                                                "AccountBalance",
                                                "NumTradesPerYr"], outputCol="features")
```

Figure 6-28 Using Scala's VectorAssembler()

   Here, you select the numeric variables (income, account balance, and number of trades per year) and the transformed categorical variables (gender_code, age_group_code, profession_code, and education_code).

3. Specify machine learning algorithms.

A simple logistic regression is used here to estimate whether a client prefers one industry or not. The probability estimate that is produced by the model indicates the likelihood that a customer has affinity for the industry. You must specify the label column, the feature column (returned by the vector assembler), and the column where the predictions are stored by using the code that is shown in Figure 6-29.



```
autoLR = LogisticRegression(labelCol = "auto_label", featuresCol = "features", predictionCol = "auto_pred",
                            rawPredictionCol = "raw_auto_pred", probabilityCol = "auto_p", maxIter = 100)
techLR = LogisticRegression(labelCol = "tech_label", featuresCol = "features", predictionCol = "tech_pred",
                            rawPredictionCol = "raw_tech_pred",  probabilityCol = "tech_p", maxIter = 100)
airLR = LogisticRegression(labelCol = "air_label", featuresCol = "features", predictionCol = "air_pred",
                           rawPredictionCol = "raw_air_pred",  probabilityCol = "air_p", maxIter = 100)
hotelLR = LogisticRegression(labelCol = "hotel_label", featuresCol = "features", predictionCol = "hotel_pred",
                             rawPredictionCol = "raw_hotel_pred",  probabilityCol = "hotel_p",maxIter = 100)
```

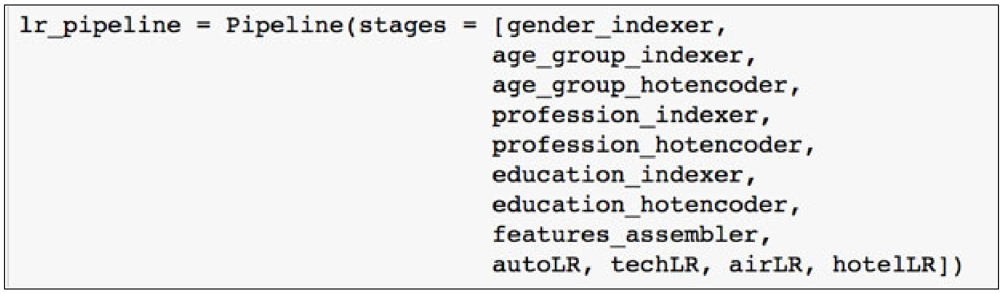
Figure 6-29 Column specification code

4. Create a complete pipeline by combining the components of feature transformation, feature selection, and algorithm definition in the order in which they were prepared.

   Usually, the order of the stages cannot be changed. For example, feature assembler cannot be put before feature transformation because feature assembler is calling columns that are built by the feature transformation stage.

   However, the order of steps within a stage can be changed. For example, transforming gender to numeric representation first and then age group is equivalent to transforming age group first, then gender.

   The code that is shown in Figure 6-30 generates the pipeline that you need, including the string indexers and one hot encoder, followed by feature assembler and machine learning algorithms.

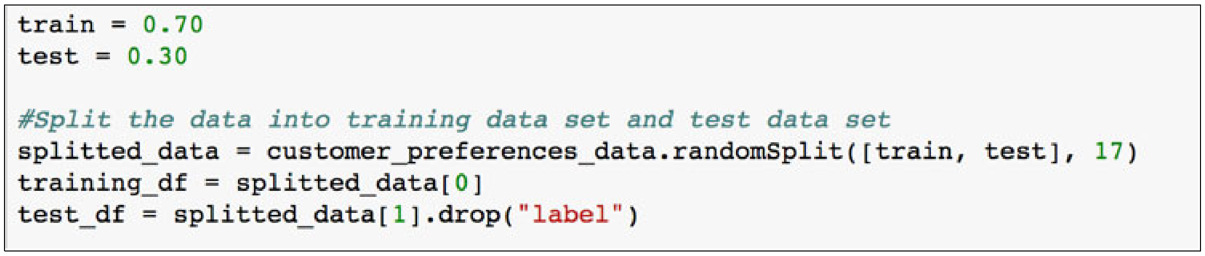


```
lr_pipeline = Pipeline(stages = [gender_indexer,
                                 age_group_indexer,
                                 age_group_hotencoder,
                                 profession_indexer,
                                 profession_hotencoder,
                                 education_indexer,
                                 education_hotencoder,
                                 features_assembler,
                                 autoLR, techLR, airLR, hotelLR])
```

Figure 6-30 Code to generate pipeline

6.2.3 Training the affinity model

As a common rule in machine learning, the historical client data must be split before the affinity model is trained. The data is split into at least into two sets (a training set and a test set) by entering the code that is shown in Figure 6-31.

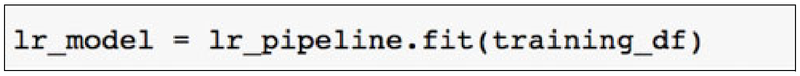


```
train = 0.70
test = 0.30

#Split the data into training data set and test data set
splitted_data = customer_preferences_data.randomSplit([train, test], 17)
training_df = splitted_data[0]
test_df = splitted_data[1].drop("label")
```

Figure 6-31 Creating train and test sets

A training set is used to train the model; a test set is used to test the model's performance. The test set separated from the training set so that the model is tested by using new data that the model has not seen before. Such a test is a good imitation of the real-world situation in which the model is to be applied on new data for prediction. The ratio of training set size and test set size can be adjusted.

After preparing the pipeline and the data, you can pass the training data set into the pipeline to train the affinity model by running the code that is shown in Figure 6-32.



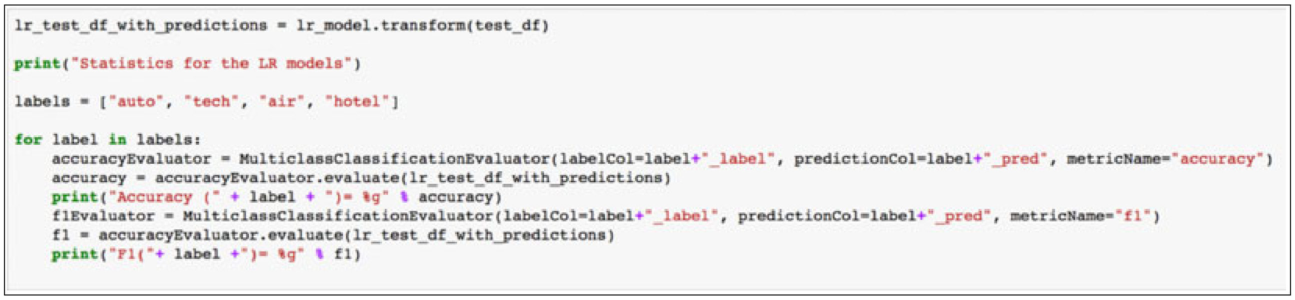
```
lr_model = lr_pipeline.fit(training_df)
```

Figure 6-32 Passing training data set into the pipeline

Running the code trains the affinity model. This model now is ready for evaluation.

6.2.4 Evaluating the affinity model

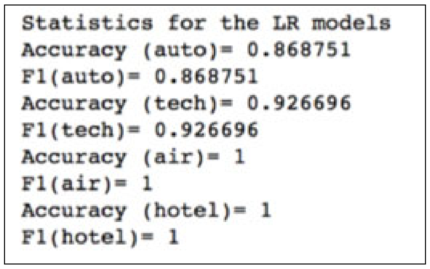
After the affinity model is trained, you might want to validate it with the test data set that was held separately (Figure 6-33). The test results are used to evaluate the predictions that are made by the model. In this case, four machine learning models are used (each for a particular industry), for which the accuracy can be computed one by one in a for loop.



```
lr_test_df_with_predictions = lr_model.transform(test_df)

print("Statistics for the LR models")

labels = ["auto", "tech", "air", "hotel"]

for label in labels:
    accuracyEvaluator = MulticlassClassificationEvaluator(labelCol=label+"_label", predictionCol=label+"_pred", metricName="accuracy")
    accuracy = accuracyEvaluator.evaluate(lr_test_df_with_predictions)
    print("Accuracy (" + label + ")= %g" % accuracy)
    f1Evaluator = MulticlassClassificationEvaluator(labelCol=label+"_label", predictionCol=label+"_pred", metricName="f1")
    f1 = accuracyEvaluator.evaluate(lr_test_df_with_predictions)
    print("F1("+ label +")= %g" % f1)
```

Figure 6-33 Validating the affinity model

Running the code generates the evaluation results of four affinity model, each for a particular industry, as shown in Figure 6-34.



```
Statistics for the LR models
Accuracy (auto)= 0.868751
F1(auto)= 0.868751
Accuracy (tech)= 0.926696
F1(tech)= 0.926696
Accuracy (air)= 1
F1(air)= 1
Accuracy (hotel)= 1
F1(hotel)= 1
```

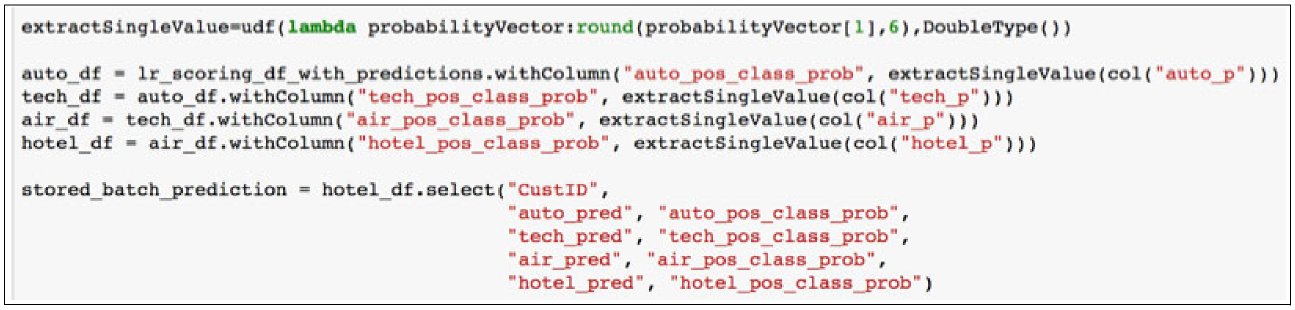
Figure 6-34 Evaluation results

6.2.5 Saving the predictions of the affinity model

The predicted probability includes numbers with many digits, which can take up storage space. You can round up the numbers if you do not plan to use them for further calculations.

In addition, the prediction output includes all of the columns that are used to build the model, including the same columns in your original sample data. Many columns are redundant *after* the output is produced. The only information you want to keep often is the predictions.

The prediction columns that are required are selected from all columns, of which the numbers are rounded up, as shown in Figure 6-35.



```
extractSingleValue=udf(lambda probabilityVector:round(probabilityVector[1],6),DoubleType())

auto_df = lr_scoring_df_with_predictions.withColumn("auto_pos_class_prob", extractSingleValue(col("auto_p")))
tech_df = auto_df.withColumn("tech_pos_class_prob", extractSingleValue(col("tech_p")))
air_df = tech_df.withColumn("air_pos_class_prob", extractSingleValue(col("air_p")))
hotel_df = air_df.withColumn("hotel_pos_class_prob", extractSingleValue(col("hotel_p")))

stored_batch_prediction = hotel_df.select("CustID",
                                          "auto_pred", "auto_pos_class_prob",
                                          "tech_pred", "tech_pos_class_prob",
                                          "air_pred", "air_pos_class_prob",
                                          "hotel_pred", "hotel_pos_class_prob")
```

Figure 6-35 Selecting prediction columns

To write the prediction into the remote database, specify the credentials, such as user name, password, current schema, and driver for authentication, and the address and the table name (see Example 6-3). With all these settings configured, you can run the code to write your predictions to the remote database.

Example 6-3 Code to specify credentials

```
properties = {
   'user': 'yourUsername'
   'password': 'yourPassword'
   'currentSchema': 'yourCurrentSchema'
   'driver': 'yourDriver'
}

stored_batch_prediction.write.jdbc(url='yourURL, table='yourTableName', mode =
'overwrite', properties = properties)
```

6.3 Loan approval: Analyzing credit risk and minimizing loan defaults

Loans are a major source of income for banks, and banks approve loans regularly. Good loans bring tremendous profits, but bad loans can mean significant financial losses.

Loan default or delinquency is one of the most common types of bad loans. Loan default occurs when a borrower fails to make payments when the payments are due. To minimize loans in default, banks must carefully analyze each application and accurately determine the creditworthiness of an applicant.

Typically, banks use the combination of business rules, corporate policies, and expert recommendations to evaluate and decide whether to approve or reject a loan application. The business rules are explicit statements that are induced from certitudes, facts, laws, and regulations. Although this practice might be sufficient when an application is straightforward and complete with information of proven creditworthiness, it does not work well in cases where explicit rules do not apply, and credit data is incomplete.

Machine Learning for z/OS can help solve this problem. Instead of relying exclusively on explicit rules, machine learning uses a range of algorithms to induce and unveil implicit patterns that are hidden in a vast amount of data. These implicit patterns can provide insights that you can use to optimize your loan decision-making and expose the hidden credit risk in an application, while rooting out applications with the potential to default and thus minimizing the possibility of financial loss.

Next, we consider an example in which you have historical data about past loans and loan approvals. You want to use the historical data as a base to build a loan approval model that can help predict the risk of default before approving a new loan. The data set is stored in a Db2 for z/OS subsystem. You want to build a model in Scala by using the integrated Notebook of Machine Learning for z/OS.

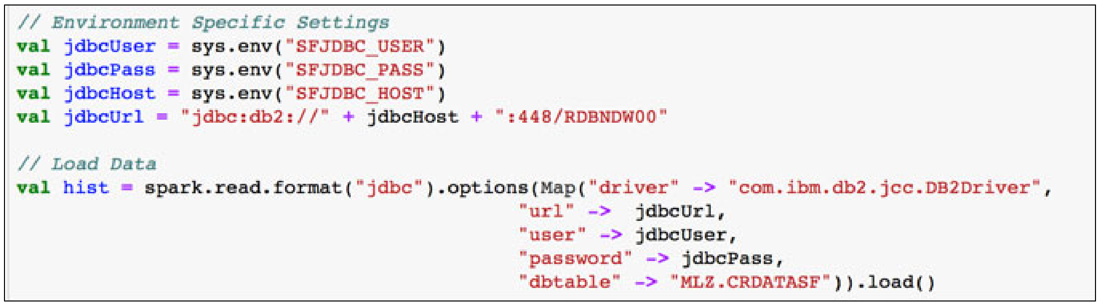
6.3.1 Analyzing historical loan approval data

Suppose that the loan approval historical data shows paid and unpaid loans and their borrowers. In this example, the sample data set includes the following variables:

- Age (numeric)
- Annual income (numeric)
- Number of credit cards (numeric; 0 - 6)
- Number of car loans (numeric; 0 - 3)
- Education level (ordinal; with eight levels from elementary school to doctoral degree)

The outcome variable is a binary label that indicates whether the loan was reimbursed (represented by 1) or not (represented by 0).

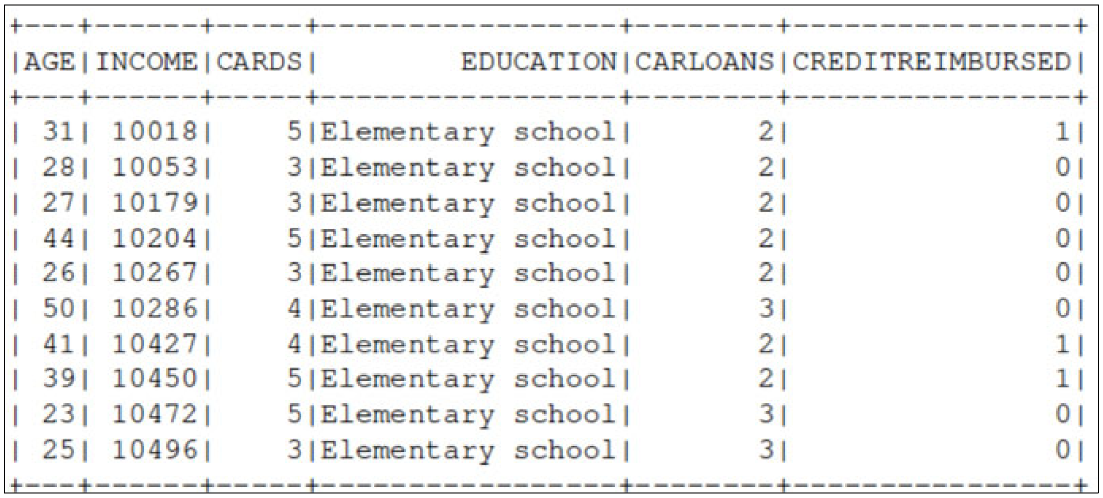
The first step in any process such as this process is to load the data (Figure 6-36). The data table is stored in a IBM DB2® database. In addition to adding a remote data set by using the UI, you can specify the credentials in the notebook and set up a connection by using the credentials to fetch data.



```
// Environment Specific Settings
val jdbcUser = sys.env("SFJDBC_USER")
val jdbcPass = sys.env("SFJDBC_PASS")
val jdbcHost = sys.env("SFJDBC_HOST")
val jdbcUrl = "jdbc:db2://" + jdbcHost + ":448/RDBNDW00"

// Load Data
val hist = spark.read.format("jdbc").options(Map("driver" -> "com.ibm.db2.jcc.DB2Driver",
                                                 "url" ->  jdbcUrl,
                                                 "user" -> jdbcUser,
                                                 "password" -> jdbcPass,
                                                 "dbtable" -> "MLZ.CRDATASF")).load()
```

Figure 6-36   Loading the data

Running the cell connects to your Db2 database and loads the historical data. You can preview the data set by printing the first several rows (Figure 6-37).



```
+---+------+-----+----------------+--------+----------------+
|AGE|INCOME|CARDS|       EDUCATION|CARLOANS|CREDITREIMBURSED|
+---+------+-----+----------------+--------+----------------+
| 31| 10018|    5|Elementary school|       2|               1|
| 28| 10053|    3|Elementary school|       2|               0|
| 27| 10179|    3|Elementary school|       2|               0|
| 44| 10204|    5|Elementary school|       2|               0|
| 26| 10267|    3|Elementary school|       2|               0|
| 50| 10286|    4|Elementary school|       3|               0|
| 41| 10427|    4|Elementary school|       2|               1|
| 39| 10450|    5|Elementary school|       2|               1|
| 23| 10472|    5|Elementary school|       3|               0|
| 25| 10496|    3|Elementary school|       3|               0|
+---+------+-----+----------------+--------+----------------+
```

Figure 6-37 Previewing the data set

You can use the data visualization library, such as Brunel, in the Notebook editor to further examine the historical data.

The cell that is shown in Example 6-4 generates a review of the data through Brunel (see Figure 6-38 on page 143). This review provides an insight into the relationship between the outcome, a variable indicating whether the loan was repaid in full, and the available data points that can help predict the results.

Example 6-4 Code that provides data review

```
%%brunel data('hist') x(EDUCATION) y(CREDITREIMBURSED) mean(CREDITREIMBURSED) bar
tooltip(#all) |
         data('hist') x(CARDS) y(CREDITREIMBURSED) mean(CREDITREIMBURSED) bar
tooltip(#all) |
         data('hist') x(CARLOANS) y(CREDITREIMBURSED) mean(CREDITREIMBURSED) bar
tooltip(#all)
:: width=800, height=500
```

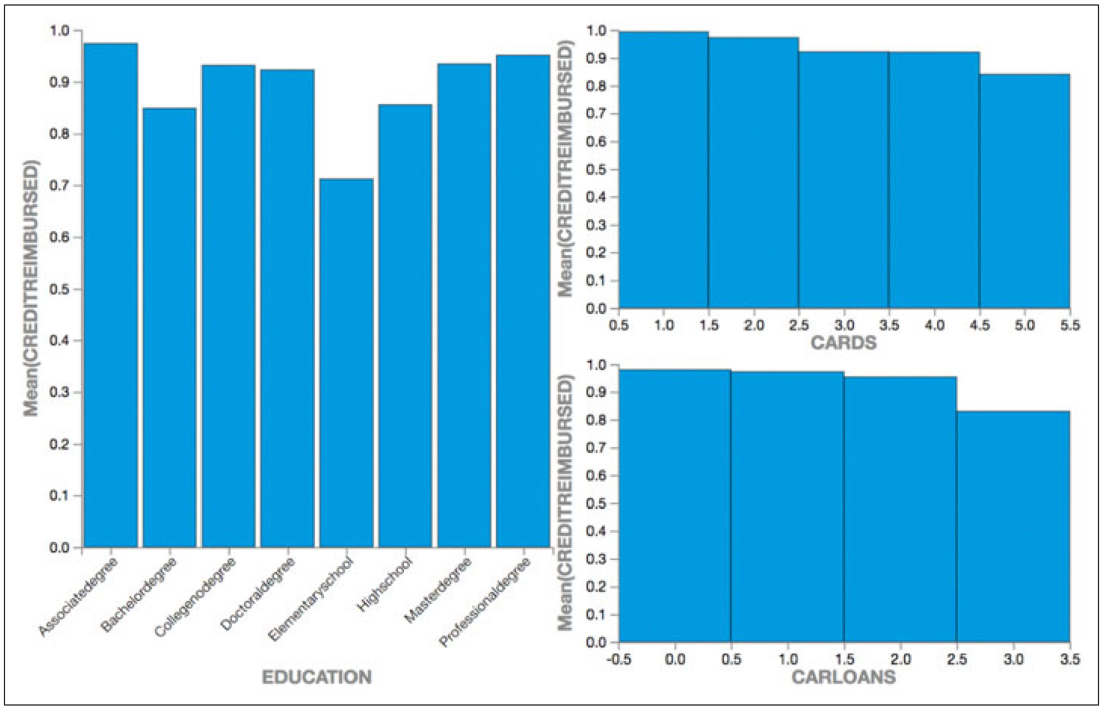


Figure 6-38 Data review results

The set of bar charts that is shown in Figure 6-38 shows the proportion of borrowers that repaid the loans, varied by education level (the upper plot), number of car loans (the lower left plot), and number of credit cards (the lower right plot).

The bottom two plots suggest a negative effect of owning more credit cards, owning more car loans, and perhaps having lower education level on the probability that the client fully repaid the loan.

Based on the visualizations, these three factors are likely to be predictive; therefore, you might want to include them in your machine learning model.

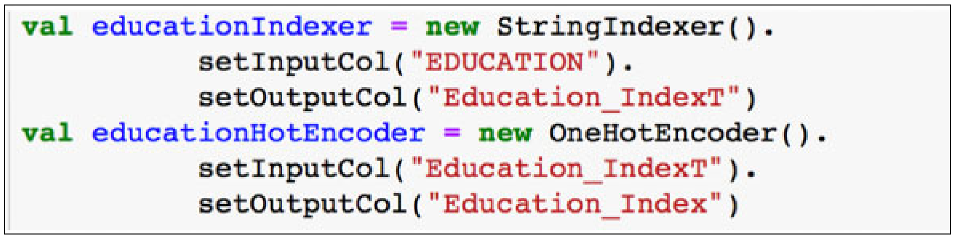
6.3.2 Defining a pipeline for the loan approval model

In 6.2, “Investment advisory: Helping clients make the right decisions” on page 133, the standard procedure to create a pipeline is explained. However, the procedure can be customized according to your need. In this section, a modified procedure is described to include cross-validation into the pipeline.

In addition to the standard parameters that the model automatically learns from the input data, you can manually assign some high-level variables or hyperparameters to cross-validate and improve the performance of the model, the effect of which is usually assessed by cross-validation method.

Complete the following steps to define the pipeline:

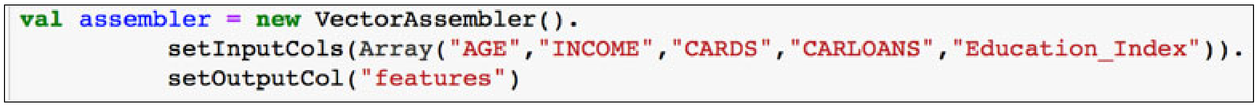
1. Transform categorical features in your data to numeric representations. The historical loan approval data contains the categorical features of age, income, credit cards, car loans, and education. The first four categories already are numeric. Education is the only category that must be managed by entering the code in a cell that is shown in Figure 6-39 on page 144.



```
val educationIndexer = new StringIndexer().
          setInputCol("EDUCATION").
          setOutputCol("Education_IndexT")
val educationHotEncoder = new OneHotEncoder().
          setInputCol("Education_IndexT").
          setOutputCol("Education_Index")
```

Figure 6-39   Code to manage Education category

2. Select and assemble the features to be used in the model through the vector assembler by entering the code that is shown in Figure 6-40.

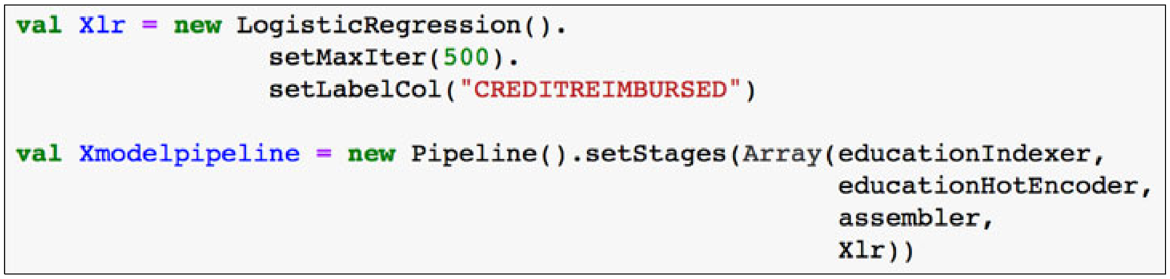


```
val assembler = new VectorAssembler().
         setInputCols(Array("AGE","INCOME","CARDS","CARLOANS","Education_Index")).
         setOutputCol("features")
```

Figure 6-40   Vector assembler code

Unlike in the investment advisor use case where you created a pipeline with stages from feature transformation to model setting, model setting in this case is not included in the pipeline because the models are compared with different hyperparameter settings.

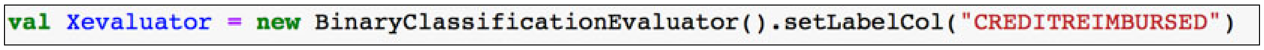
3. Specify machine learning algorithms to be included in the pipeline. As shown in Figure 6-41, logistic regression is used to manage binary categorical outcome CREDITREIMBURSED.



```
val Xlr = new LogisticRegression().
                setMaxIter(500).
                setLabelCol("CREDITREIMBURSED")

val Xmodelpipeline = new Pipeline().setStages(Array(educationIndexer,
                                                    educationHotEncoder,
                                                    assembler,
                                                    Xlr))
```

Figure 6-41   Code to manage binary categorical outcome CREDITREIMBURSED

4. Specify evaluation metrics for identifying the hyperparameters that help improve the performance of the model. During cross-validation, the performance of the model is recorded so that the best set of hyperparameters can be identified based on the metrics. The code that is shown in Figure 6-42 uses BinaryClassificationEvaluator as the evaluator to measure model performance.



```
val Xevaluator = new BinaryClassificationEvaluator().setLabelCol("CREDITREIMBURSED")
```

Figure 6-42   Use of BinaryClassificationEvaluator as the evaluator to measure model performance

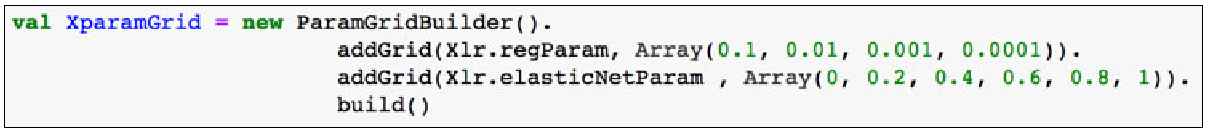
5. Specify sets of hyperparameters to cross-validate the performance of your model.

   A classic logistic regression does not contain hyperparameters. However, for machine learning algorithms that involve linear combination of predictors, you can add high-level hyperparameters to penalize the use of too many predictors. This technique known as *regularization*, which helps prevent overfitting of the model.

L1 norm regularization (LASSO) and L2 norm regularization (Ridge) are among the most frequently used methods of regularization. The use of LASSO and Ridge hyperparameters together is called Elastic Net.

In Spark machine learning library, the elastic net hyperparameter (elasticNetParam) is in the form of proportion of LASSO hyperparameter in the sum of regularization hyperparameters of both methods. A 0 means Ridge only and 1 means LASSO only. Any number 0 - 1 indicates an Elastic Net, or a combination of LASSO and Ridge.

Enter the code that is shown in Figure 6-43 to specify the hyperparameters that you want to use.

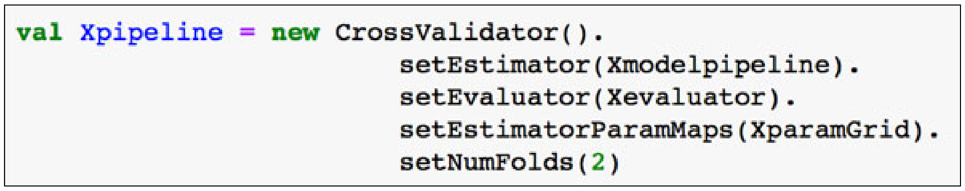


```
val XparamGrid = new ParamGridBuilder().
                    addGrid(Xlr.regParam, Array(0.1, 0.01, 0.001, 0.0001)).
                    addGrid(Xlr.elasticNetParam , Array(0, 0.2, 0.4, 0.6, 0.8, 1)).
                    build()
```

Figure 6-43 Code to specify the hyperparameters

Here, grid search is used to find the best combination of the two hyperparameters. Four numbers in regParam and six ratios in elasticNetParam are tested. The number of hyperparameter set is 4 x 6 = 24.

6. Create a complete pipeline by combing the components of feature transformation, feature selection, algorithm definition, evaluator definition, and hyperparameter specification in the correct order. You can use the code that is shown in Figure 6-44 to create the pipeline.



```
val Xpipeline = new CrossValidator().
                    setEstimator(Xmodelpipeline).
                    setEvaluator(Xevaluator).
                    setEstimatorParamMaps(XparamGrid).
                    setNumFolds(2)
```

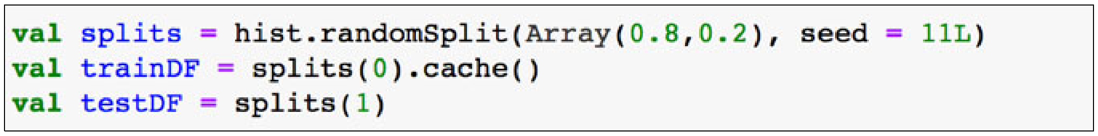
Figure 6-44 Code to create a pipeline

The setNumFolds variable indicates the “k” in k-fold cross validation, which refers to the number of times that the training data set is further split for model training and evaluation. The setNumFolds (2) is 2-fold, meaning that the training data set is further sliced into two halves and that the model is trained and evaluated twice.

6.3.3 Training the loan approval model

The training set is used to build the model and the test set is used to evaluate the model performance. This test imitates the real world scoring problem where you have some known data (the training set) to train and select the best model and apply the model on unknown future data (the test set).

You can split the historical loan approval data into two subsets, one for training and the other for testing, as shown in Figure 6-45.

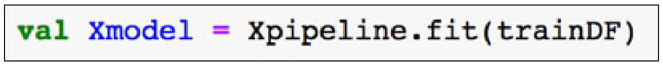


```
val splits = hist.randomSplit(Array(0.8,0.2), seed = 11L)
val trainDF = splits(0).cache()
val testDF = splits(1)
```

Figure 6-45 Splitting data into two subsets

Running the code splits the historical data into a training set (80%) and a test set (20%).

By using the completed pipeline, you can train the loan approval model by passing the training data into the pipeline, as shown in Figure 6-46.



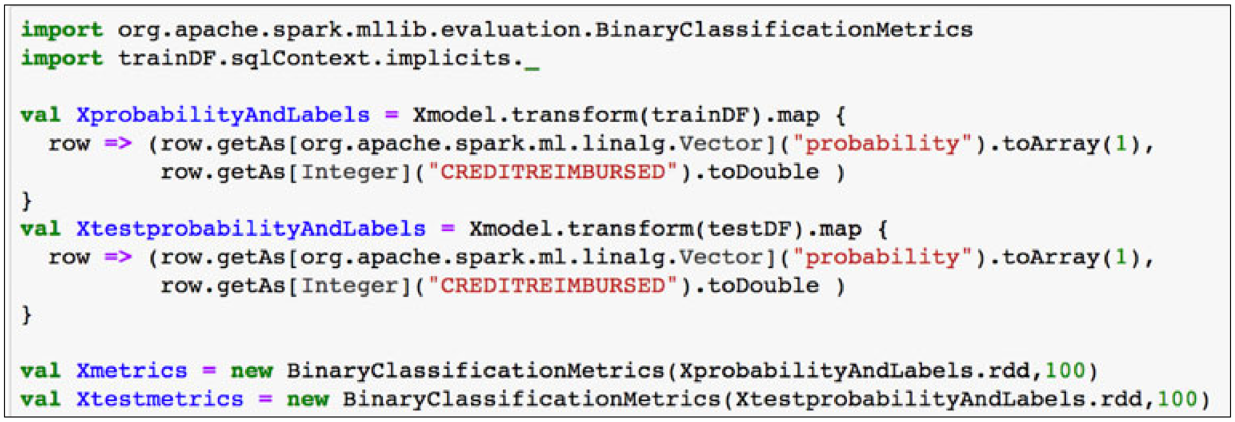
```
val Xmodel = Xpipeline.fit(trainDF)
```

Figure 6-46   Passing the training data into the pipeline

It can take much longer if the training includes hyperparameter optimization. In this case, a total of 4 x 6 x 2 = 48 models are built to generate a ranking of hyperparameter sets by using the current settings.

6.3.4 Evaluating and testing the loan approval model

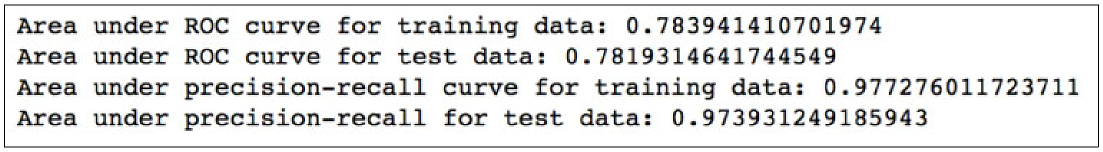
By using the best set of hyperparameters that emerges in the cross-validation process, the final model can be trained on the training data set and applied to the test data set for scoring and evaluation, as shown in Figure 6-47.



```
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import trainDF.sqlContext.implicits._

val XprobabilityAndLabels = Xmodel.transform(trainDF).map {
  row => (row.getAs[org.apache.spark.ml.linalg.Vector]("probability").toArray(1),
          row.getAs[Integer]("CREDITREIMBURSED").toDouble )
}
val XtestprobabilityAndLabels = Xmodel.transform(testDF).map {
  row => (row.getAs[org.apache.spark.ml.linalg.Vector]("probability").toArray(1),
          row.getAs[Integer]("CREDITREIMBURSED").toDouble )
}

val Xmetrics = new BinaryClassificationMetrics(XprobabilityAndLabels.rdd,100)
val Xtestmetrics = new BinaryClassificationMetrics(XtestprobabilityAndLabels.rdd,100)
```

Figure 6-47   Final model training

Running the code generates the evaluation results that are shown in Figure 6-48.

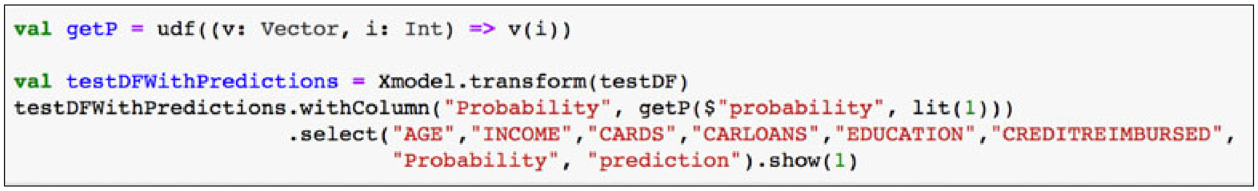


```
Area under ROC curve for training data: 0.783941410701974
Area under ROC curve for test data: 0.7819314641744549
Area under precision-recall curve for training data: 0.977276011723711
Area under precision-recall for test data: 0.973931249185943
```

Figure 6-48   Evaluation results

The results show the area under ROC curve and the PR curve of your best-performing model on the training and test data sets. Although the area under ROC curve is less than 0.80 (not an ideal number), the area under PR curve is as high as 0.97.

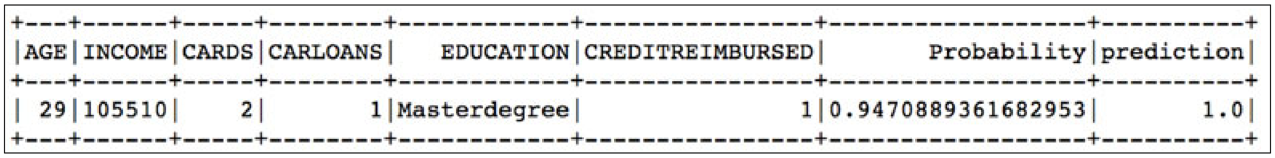
By applying the model on test data, you can get the predicted probability and predicted outcome for each loan application in the test set, as shown in Figure 6-49 on page 147.



```
val getP = udf((v: Vector, i: Int) => v(i))

val testDFWithPredictions = Xmodel.transform(testDF)
testDFWithPredictions.withColumn("Probability", getP($"probability", lit(1)))
                         .select("AGE","INCOME","CARDS","CARLOANS","EDUCATION","CREDITREIMBURSED",
                                 "Probability", "prediction").show(1)
```

Figure 6-49 Applying model on test data

Running the code generates predicted probability and outcome results, as shown in Figure 6-50.



```
+---+------+-----+--------+------------+----------------+------------------+----------+
|AGE|INCOME|CARDS|CARLOANS|   EDUCATION|CREDITREIMBURSED|       Probability|prediction|
+---+------+-----+--------+------------+----------------+------------------+----------+
| 29|105510|    2|       1|Masterdegree|               1|0.9470889361682953|       1.0|
+---+------+-----+--------+------------+----------------+------------------+----------+
```

Figure 6-50 Predicted probability and outcome results

In this example, a 29-year old loan borrower with an annual income of $105,510, two credit cards, one car loan, and a master's degree is predicted to have a 94.7% probability to pay the loan in full. The corresponding outcome is 1, which means the loan will be fully repaid. The prediction is consistent with the CREDITREIMBURSED value in the historical data.

6.4 Fraud detection: Rooting out frauds in government benefit programs

Fraud is age-old and pandemic across industry sectors and government agencies. As businesses and governments move their operations online, fraud crimes become even more rampant and sophisticated, and fraud losses are increasingly crippling.

Take government benefit programs as an example. The programs are put in place to serve people in need. But, persistent fraudulent enrollments and claims disrupt the social services and deplete the limited resources, which makes it almost impossible for the right people to receive the right benefits.

Fighting frauds in government agencies is an uphill battle. Although the perpetrators use evolving technologies and change their tactics at will, the process that is used to uncover those offenses remains mostly manual, slow, and drawn out.

Effective fraud detection requires the computational capability to quickly analyze a vast amount of data, accurately identify hidden patterns of fraudulent behaviors, and swiftly turn them into intelligence that enables organizations to make real-time decisions. Machine Learning for z/OS can deliver that industry-leading capability.

Suppose that the State of California runs a Supplemental Nutrition Assistance Program (SNAP), intended for families in dire need of food help. Rampant fraudulent enrollments threaten to derail that objective. The state decides to use Machine Learning for z/OS to detect, expose, and deny fraudulent SNAP applications.

Assume that as a data scientist, you and your team developed a SNAP anti-fraud model. The model was trained with historical SNAP data and fitted with random forest classification. The model was developed in R in the RStudio Desktop and exported as a PMML model.

You want to use the Machine Learning for z/OS PMML capability to import the model and then deploy it to score all incoming SNAP applications and root out the fraudulent applications.

6.4.1 Overview of historical SNAP data and the SNAP model

The historical SNAP data contains the following information about more than 16,000 applicants:

- Gender (nominal; F = female M = male)
- Education (ordinal; from high school to doctorate)
- Income (numeric; 0 - 60000)
- Marital status (nominal; single, married, divorced)
- Employment status (nominal; disabled, medical leave, retired, unemployed, and employed)
- County (nominal; 58 counties in California)

The outcome variable is a label showing whether an application is fraudulent (1) or normal (0).

A simple random forest model was trained on 80% of the data by using randomForest package in Rstudio, which consists of 100 trees. Each tree in a random forest model is slightly different, which gives its own prediction. The final prediction is a combination of the predictions from all 100 trees.

The model performance on the test set (20% of the data) is satisfactory, achieving area under ROC curve of 0.897 and area under PR curve of 0.806.

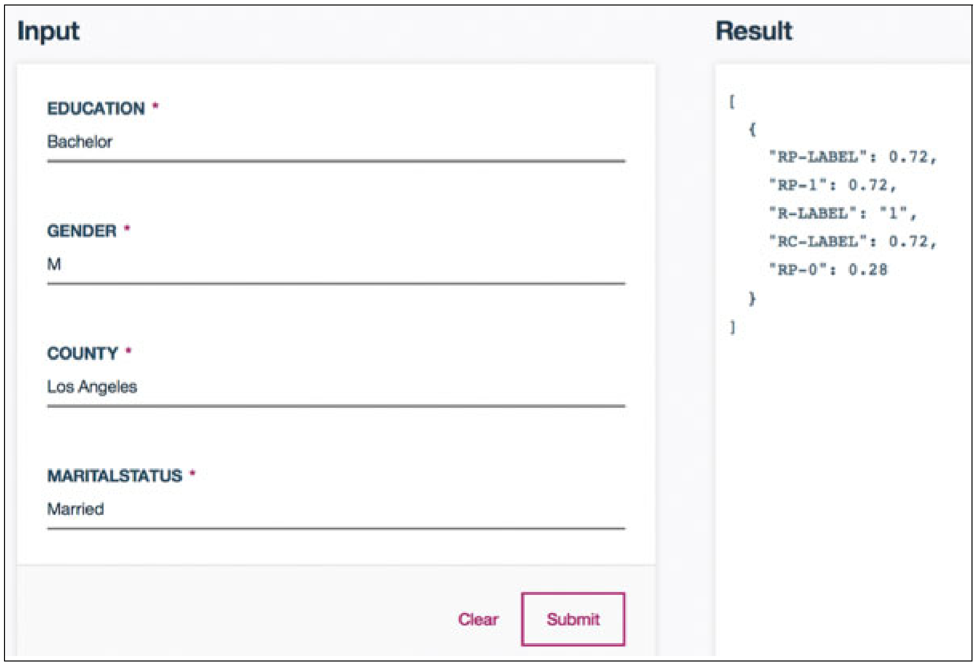
6.4.2 Importing, deploying, and testing the SNAP model

PMML models can be imported to MLz and are ready for immediate use by using the following process:

1. Import the SNAP model:
   a. On the Models tab of the Model Management page, click **Add model** and then specify a name for the imported model.
   b. Click **Browse** to select and upload the SNAP model file on your local machine.
   c. Click **Add model** to confirm importing the SNAP model. Now, you can verify that the new model is on the Models list.
2. Deploy the SNAP model.

   In the Models list, find your model and click **ACTIONS** menu, where you can see an option to deploy the model. The deployed PMML SNAP model can be handled and used in the same way as the models created natively in Machine Learning for z/OS.
3. Test the SNAP model.

   After the model is deployed, you can find your model on the Deployments tab. If you click the **ACTIONS** menu of the model, you can select **Test API** to start the Input and Result page, as shown in Figure 6-51 on page 149.



EDUCATION *
Bachelor

GENDER *
M

COUNTY *
Los Angeles

MARITALSTATUS *
Married

```
[
  {
    "RP-LABEL": 0.72,
    "RP-1": 0.72,
    "R-LABEL": "1",
    "RC-LABEL": 0.72,
    "RP-0": 0.28
  }
]
```

Figure 6-51 Input and Result page

You can enter a value in all the input fields and then click **Submit** to generate the corresponding results.

The example shows the scoring results of the SNAP application from a married male applicant in the county of Los Angeles. The applicant has a bachelor's degree and is employed with an annual income of $50,000. The SNAP model predicts that the application has a 72% probability to be fraudulent.

After the model is integrated into production, it scores a new incoming SNAP application with all required information in the correct data format. Given the set of results, you can choose what you want to do:

- Use predicted label R-LABEL to decide what action to take for fraudulent applications (it assigns 1 = fraudulent to the input data when predicted probability of fraud is larger than 0.5).
- Examine closely the probabilities (RP-1 and RP-0) to assign a fraud label by using customized threshold; for example, 0.8, which is based on which action you decide to take.

Important: Compress the PMML file for the SNAP model before you upload the model into Machine Learning for z/OS.

6.5 ITOA: Detecting system anomalies and resolving issues before they arise

Machine learning for z/OS and its predictive capability can greatly benefit IT operations. Enterprises worldwide use the IBM Z hardware and software, such as Db2 for z/OS, to support their mission critical processes. These Z solutions can collect and save extensive data of system operation and performance. You can use this historical data to build machine learning models for monitoring system operations and predict issues before they arise.

Suppose that you want to use the Machine Learning for z/OS ITOA Health Tree application to build a health tree model that is fitted with a linear regression/Pearson algorithm. By using the health tree model, you can monitor the operational health of your Db2 for z/OS, establish a key performance indicator (KPI) baseline, identify anomalies, and pinpoint the root causes. System administrators can then proactively resolve the issues before they disrupt the operation and performance of the subsystem.

6.5.1 Preparing historical health tree data

The Machine Learning for z/OS ITOA Health Tree application can use Db2 statistics and accounting traces in comma-separated value (CSV) format as input data. Complete the following steps to prepare the historical health tree data:

1. Collect the statistics and accounting trace records of your Db2 subsystem. The Db2 instrumentation facility component (IFC) provides a trace facility to record data. Each trace class captures information about several Db2 events that are identified by instrumentation facility component identifiers (IFCIDs). Db2 traces include several data types, such as statistics, accounting, audit, performance, monitor, and global data.

   The statistics trace reports the usage information of Db2 system and database services, while the statistics trace records general information about the subsystem and storage. The Db2 system general KPIs include data about CPU time, data set open and close, logging, locking, DDF, and latch. The Db2 storage KPIs contain data about real available storage, DBM1 real storage usage, active threads, and so on.

   The accounting trace records transaction-level data at the completion of a transaction. The data enables you to conduct Db2 capacity planning and tune application programs. The accounting KPIs contain specific accounting data, including CLASS2_ELAPSED, CLASS2_CPU_TOTAL, CLASS2_CPU_AGENT, LASS3_LOCK, CLASS3_LATCH, and CLASS3_ACCEL.

2. Convert the statistics and accounting trace records into CSV data. Use the IBM OMEGAMON® Spreadsheet Input-Data Generator utility (or a similar product) to generate the required CSV files from the SMF Type 100,101,102 historical records. The CSV data can then be imported into the train and test a health tree model. Complete the following steps:

   a. Generate statistics file data set STFILDD1 by issuing the following STATISTIC command:

      ```
      STATISTIC
      FILE
      DDNAME(STFILDD1)
      EXEC
      ```

b. Generate accounting file data set ACFILDD1 with DATATYPE set to GENERAL by issuing the following ACCOUNTING command:

```
ACCOUNTING
FILE
DATATYPE(GENERAL)
DDNAME(ACFILDD1)
EXEC
```

c. Create two statistics CSV data sets by using the input that is available in the Performance Database (PDB) in the TKO2SAMP library and your own input from field selection lists in the TKANSAMF library.

   Use the CSVGEN utility (as shown in the following examples) and specify SGEN and SSTG as the record types and FPEPSGEN and FPEPSSTG as the corresponding field selection lists:

```
//CSVGEN   EXEC PGM=FPEPCSV,
//   PARM='SGEN N Y Y , . CANDLE.V540.TKO2SAMP',
//   COND=(4,LT)
//STEPLIB  DD  DSN=CANDLE.V540.TKANMOD,DISP=SHR
//FLDSEL   DD  DSN=CANDLE.V540.TKANSAMF(FPEPSGEN),DISP=OLD

//CSVSTG   EXEC PGM=FPEPCSV,
//   PARM='SSTG N Y Y , . CANDLE.V540.TKO2SAMP',
//   COND=(4,LT)
//STEPLIB  DD  DSN=CANDLE.V540.TKANMOD,DISP=SHR
//FLDSEL   DD  DSN=CANDLE.V540.TKANSAMF(FPEPSSTG),DISP=OLD
```

d. Create an accounting CSV data set by also using the input that is available in the performance database (PDB) in the TKO2SAMP library and your own input from field selection lists in the TKANSAMF library.

   Use the CSVGEN utility (as shown in the following example) and specify AFGE as the record type and FPEPAFGE as the corresponding field selection list:

```
//FPEPCSV  EXEC PGM=FPEPCSV,
//   PARM='AFGE N Y Y , . CANDLE.V540.TKO2SAMP'
//STEPLIB  DD  DSN=CANDLE.V540.TKANMOD,DISP=SHR
//FLDSEL   DD  DSN=CANDLE.V540.TKANSAMF(FPEPAFGE),DISP=OLD
```

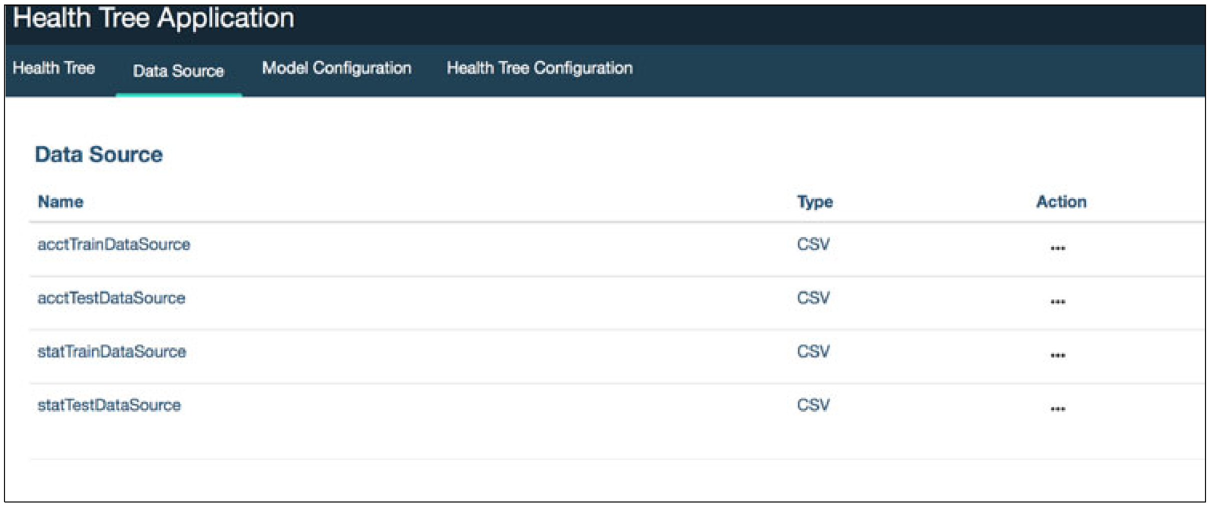
e. Copy the new statistics and accounting CSV data sets to your z/OS UNIX files by using the OCOPY command, as shown in the following example:

```
//CPYTSUS EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//IN DD DISP=SHR,DSN=MLZSAMP.ACCT.CSVGEN
//OUT DD PATH='/tmp/mlzsamp.acct.csvgen',
// PATHDISP=(KEEP,DELETE),
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIROTH)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
OCOPY INDD(IN) OUTDD(OUT)
```

6.5.2 Building the health tree model

After the CSV files are created, you are ready to train and build a health tree model in the Health Tree application. Complete the following steps:

1. From the sidebar of the Machine Learning for z/OS UI, select and browse to **Pattern Repository**.
2. On the ITOA tile, click **APPLICATION** to open the Health Tree application.
3. On the Data Source tab, define data sources in the Health Tree application to train and score the health tree model, as shown Figure 6-52.

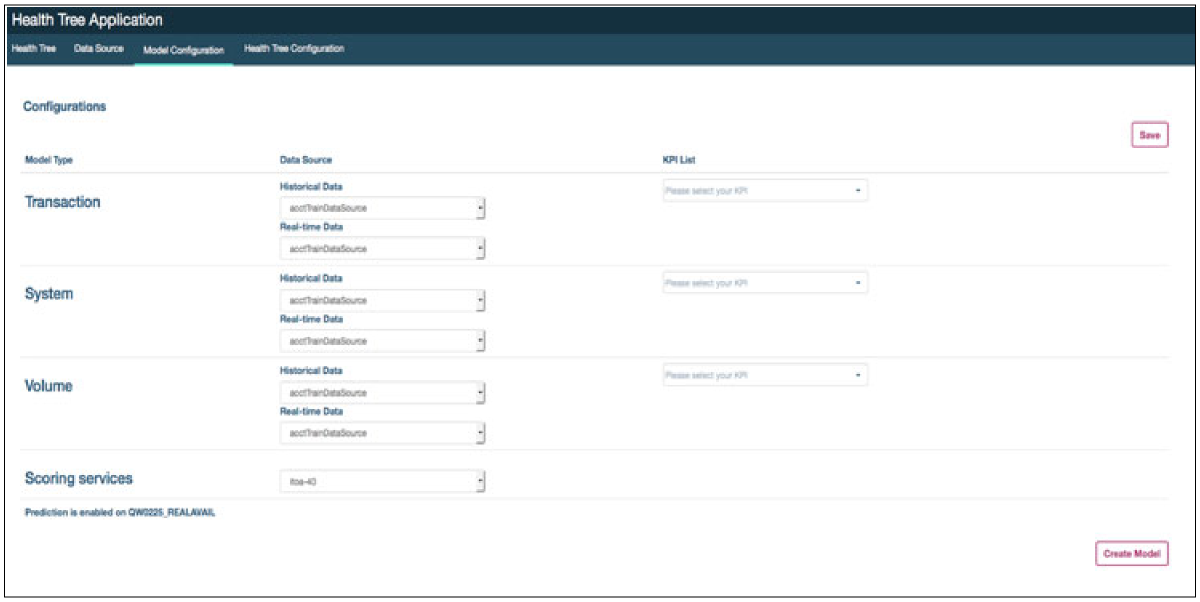


Data Source

Name	Type	Action
acctTrainDataSource	CSV	•••
acctTestDataSource	CSV	•••
statTrainDataSource	CSV	•••
statTestDataSource	CSV	•••

Figure 6-52 Defining data sources

4. On the Model Configuration tab, specify the training data set and then, click **Create Model**, as shown in Figure 6-53.



Configurations

Model Type	Data Source	KPI List
Transaction	Historical Data: acctTrainDataSource; Real-time Data: acctTrainDataSource	Please select your KPI
System	Historical Data: acctTrainDataSource; Real-time Data: acctTrainDataSource	Please select your KPI
Volume	Historical Data: acctTrainDataSource; Real-time Data: acctTrainDataSource	Please select your KPI
Scoring services	itoa-40	

Prediction is enabled on QW0225_REALAVAIL

Figure 6-53 Specifying the training data set

Alternatively, you can create a health tree model and visualize the health tree in the integrated Notebook editor. View a sample health tree notebook by clicking **NOTEBOOK** on the ITOA tile on the Pattern Repository page.

6.5.3 Building the health tree based on the health tree model

After the health tree model is created, you are ready to configure and build the health tree by using the model. Complete the following steps:

1. On the Health Tree Configuration tab, specify the fields to configure the tree.
2. On the Health Tree tab, click **Start** to display the tree. The tree data is imported into a data frame with which the Health Tree application uses to draw the tree, as shown in Figure 6-54.

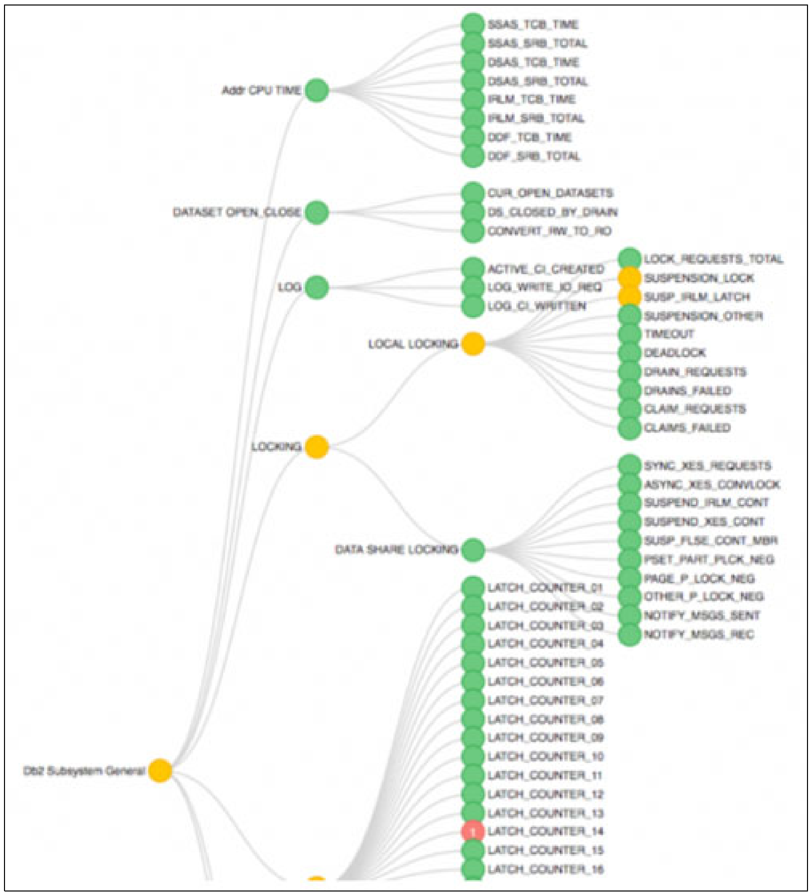


Figure 6-54 Importing tree data

If necessary, click **Pause** to pause the display of the health tree. Click **Play** to resume the application.

6.5.4 Monitoring the health of your Db2 subsystem

As shown in Figure 6-55, the health tree of your Db2 subsystem is split into three trees that are accessible on three tabs: Db2 Subsystem General, Db2 Transaction, and Db2 Storage. Toggle the tabs to view the trees and monitor the health of your Db2 subsystem.

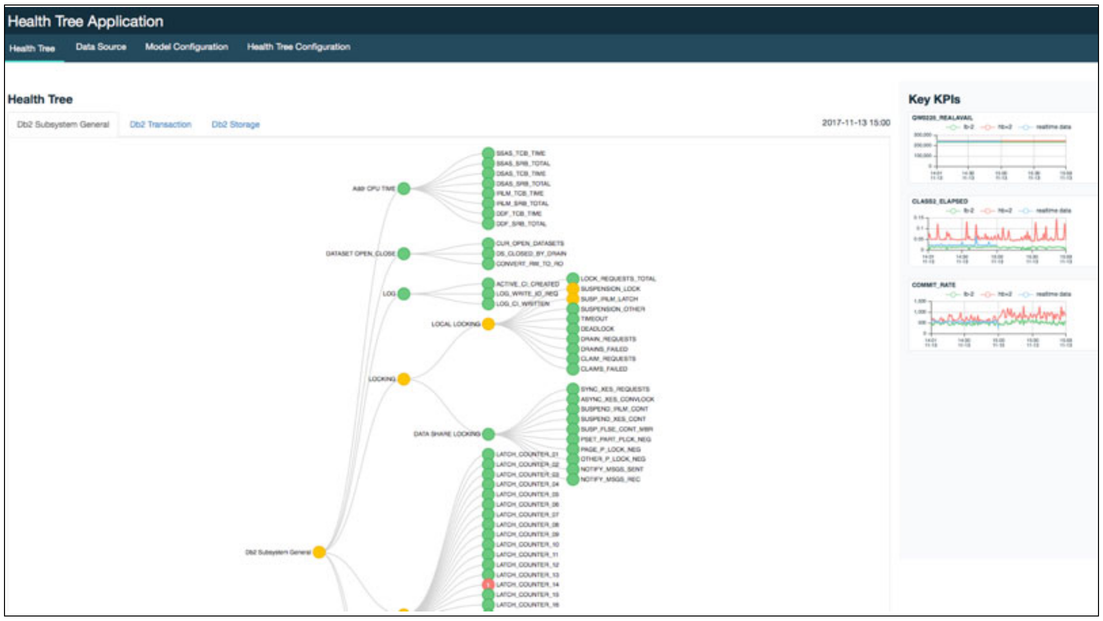


Figure 6-55 Three heath trees

For each tree, a leaf node represents an individual KPI. A top-level node typically indicates a function area, but in some cases, it can also be an individual KPI.

A node can appear in the color of green, yellow, or red, similar to how a traffic light changes color. Green indicates normal status, yellow represents a warning, and red denotes an anomaly. When a node is in red, a number in the middle of the node shows the duration of the abnormal status.

The dynamic baselines of three key KPIs are displayed to the right side of each tree. The green line is downward, and the red line is upward while the blue line represents the real value of the KPI.

At the bottom of the tree, a graphic timeline is displayed, as shown in Figure 6-56.

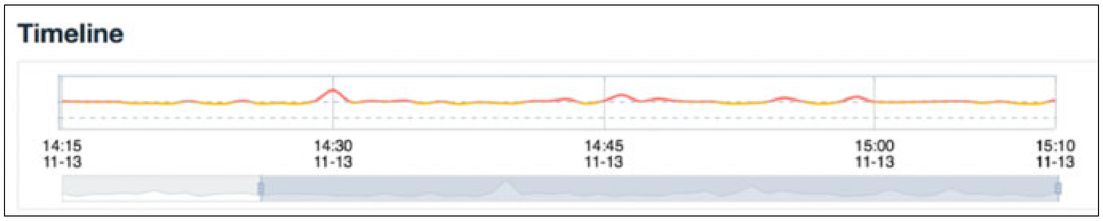


Figure 6-56 Timeline display

The timeline displays the health status of your Db2 system over a 24-hour period. You can move your cursor over the timeline and pick any point in time to see more information about the system health status.

A

Machine learning basics

To organizations, the fundamental objective of machine learning is to help solve a business problem or explore a business opportunity. This process can be accomplished through building, evaluating, deployment, and scoring a machine learning model.

A machine learning model describes the relationship between the outcome variable and multiple predictors, which is mainly expressed by the algorithm, hyper-parameters, and parameters.

The algorithm must be chosen manually. The hyper-parameters are optional (default values are provided) but can influence the performance of the final machine learning model. The parameters (also called *weight* or *loading* in some algorithms) are automatically derived, or as the name says, “learned”, from the training data. With a reliable machine learning model, predictions can be made to drive business insights and strategies in a more efficient and effective way.

After you understand what a machine learning model is doing, you know how to convert your business problem into a machine learning problem so that you can use the power of machine learning.

As the conversation about machine learning moves beyond a data scientist’s office, misconceptions start to emerge. One of the common misunderstandings is that machine learning is all about algorithms, which are explicitly related to machine learning models. Although algorithms and models are inherently essential to machine learning, they are only a part of it. In fact, machine learning involves a full workflow that includes the following stages:

- Data collection
- Data cleaning
- Data exploration
- Feature preprocessing
- Model building
- Model evaluation
- Model deployment

This appendix provides some machine learning basic information by briefly introducing each of these stages (except for model deployment) from a data scientist's point of view. Although model deployment is integral to the complete machine learning workflow, it is a stage in which data scientists are relatively less involved.

This appendix includes the following topics:

- "Data collection" on page 157
- "Data cleaning" on page 157
- "Data exploration" on page 159
- "Feature preprocessing" on page 163
- "Model building" on page 165
- "Model evaluation" on page 175

Data collection

Unlike an expert system where all the rules are specified by humans, machine learning models learn patterns from the training data and automatically form “rules” that are based on these patterns.

Before building a machine learning model, you must collect the data that is used to train the model. The data must be related to the problem that you want the model to solve. Examples of data sources include public data sets, private data sets (on-premises or on cloud), and self-built data set. In some sense, data collection defines the upper bound of the performance of your machine learning model, or how good your model performance can be.

Consider data collection for a rice price model as an example. Assume the price range (low, medium, or high) of a 5-pound bag of rice can be perfectly predicted by using its quality, variety, and place of origin. “Perfectly” refers to when you have the data for these three variables, you can train a machine learning model to predict the rice price 100% correct.

If you do not know how the rice price is derived and you want to predict the rice price and attempt to collect data that you think is relevant to the answer, you can collect the following information:

- The wanted variables; for example, the quality of the rice.
- The variables that are related to the wanted variables ones; for example, the color of the rice, which is related to (but not the same) as the variety of the rice.
- The variables that are unrelated to the wanted variable; for example, the seller.

Compared to the wanted predictors from which the price is derived, you might notice that some of the wanted predictors are not collected at all. In this case, the place of origin is not collected. This piece of information is lost in the data set that is collected by the data scientists.

Assume excluding the place of origin and the use of only the quality and variety reduces the accuracy to 80%. By using a data set with no variable providing information about the place of origin, you never get an accuracy rate over 80% no matter what great machine learning model is used.

In real-world cases, no one knows the wanted predictors. Losing a piece of important information (you do not know which information is important) limits the performance of the machine learning model.

Data cleaning

You might think model building is the most time-consuming stage in the machine learning workflow. However, it is data cleaning or cleansing that takes most of your time because the data is rarely clean and ready for use.

More often than not, you must cleanse and prepare the data before you can use it. It might be counter-intuitive, but data scientists often spend up to 60% of their time on data cleaning for a machine learning project.

Why is data cleaning tedious work? For example, if you want to perform text analysis on tweets that are collected from Twitter, the text data contain abbreviations, spelling errors, grammatical errors, and other problems. Humans can easily detect these problems and understand the content without any issue, but machines cannot do it automatically and therefore, these problems prevent machine learning algorithms from figuring out the proper patterns.

Another example is relational data tables, which are a common structure for enterprise data. Most of the machine learning algorithms are essentially matrix operations that require the input data to be in one matrix. You must join the tables (sometimes aggregate the tables) to form a matrix as expected by your machine learning algorithm.

By cleaning the data, you prepare and organize the data set into a form that can be imported into the machine learning algorithms. Next, we describe some typical issues that can occur in your data sets.

Missing values

A missing value is a common problem in most of the data sets. The reason why values might be missing varies. For example, some fields might not be required on the electronic form when the data is collected, or the data entry worker might accidentally skip some cells or rows.

Generally, the process for resolving missing values includes:

1. Identify the reason that causes missing values. Sometimes, the values are missing at random but other times a pattern exists.
2. Take the corresponding remedy to impute the missing values.

This problem must be addressed because most of the machine learning algorithms inherently cannot deal with missing values.

Duplications

The data set might contain unwanted duplicated records. Although a small portion of duplication does not necessarily affect your machine learning algorithms, duplication biases the algorithms and makes them favor the duplicates more than the non-duplicates.

Multiple data tables

At times, you must combine information from various data tables. For example, by combining a customer profile table, an order history table, and a merchandise information table, you want to analyze the spending preferences of the customer and train a machine learning model to predict the probability of future purchases.

Data tables can be joined by using the key variables or identifiers. Typical key variables include customer ID, merchandise ID, and order numbers. At times, you must aggregate tables before or after you prepare the joint table to have a data table in a proper format as input to machine learning algorithms.

Incorrect data

Some information in a data set might be entered by machines. Examples include the receipt number that is automatically generated for a transaction, the product number that is scanned from the bar code, and the time stamp when a request is processed. Other information in the data set might be entered by humans, which is more error-prone.

You must set up several screening criteria to detect incorrect data. These criteria might be constraints on the uniqueness of a variable (for example, in a latest customer profile table the customer ID should be unique), the length of a variable (for example, the values of a phone number should not exceed a specific number of digits), or the range of a variable (for example, the number of pets a user registered should not be a negative number).

Coding and recoding

Coding refers to transforming nominal variables into numbers. For example, a binary gender variable with two levels (male or female) can be coded by using 0 and 1, where 0 represents one gender and 1 represents the other gender.

Whether this step is required depends on the machine learning function and the programming language. For example, linear regression in R supports factor type of data where the coding is done behind the scenes. Linear regression in python requires a numeric matrix input where coding must be done manually when the data is prepared.

At times, the data is already coded (and usually includes a codebook where the meaning of codes or the coding process is explained) but not necessarily in a way you want. Recoding is applied in this case.

Data exploration

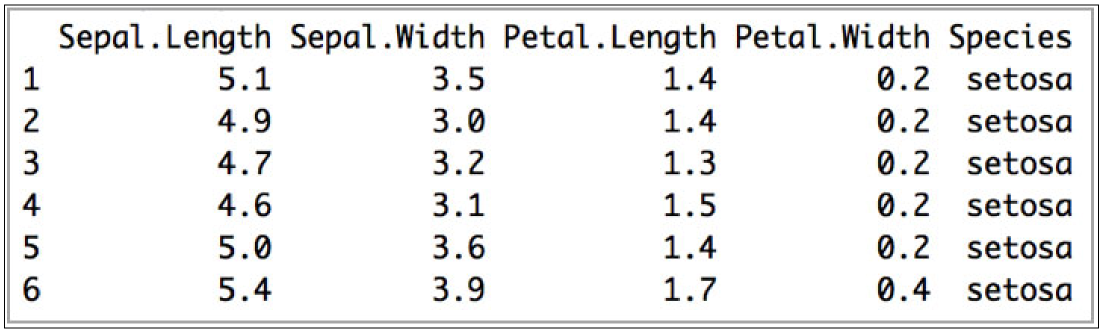
Understanding the data is the starting point of using it. Data preview, summary tables, and visualization plots are the most used data exploration methods. In this example, we use the Iris data set in R to describe these methods.

Data preview (head and tail)

When a data set contains thousands of data points, where each data point is an observation with values in many variables, it is impossible to examine the data points individually. To get a rough idea about the data points and variables in the data set, you can take a closer look at the first (the head) and last (the tail) several rows.

The last several rows cannot provide more clarification, except for the data set that is ordered by some variable; for example, time stamps. Having the head part and the tail part sometimes gives you a sense of the range of ordered variables.

The first six rows of the Iris data set are shown in Figure A-1.



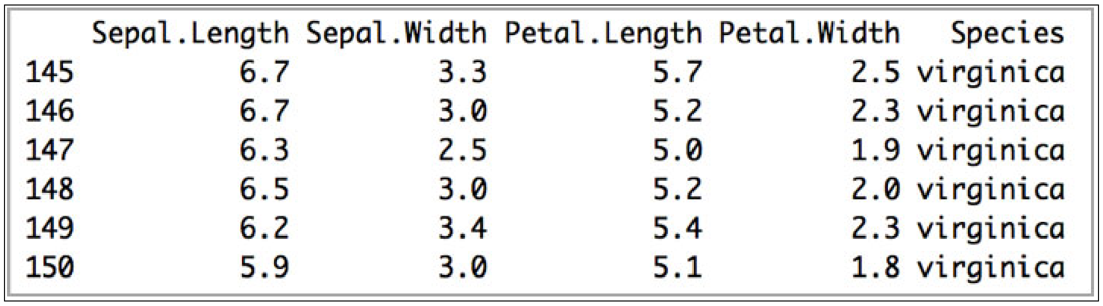
```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Figure A-1 First six rows of iris data set

From the head part of the data set, you see the following variables that are contained in the data set and the potential data type, including:

- Sepal length in some scale (numeric)
- Sepal width in some scale (numeric)
- Petal length in some scale (numeric)
- Petal width in some scale (numeric)
- Species of the flower (categorical)

The last six rows of the same data set are shown in Figure A-2.



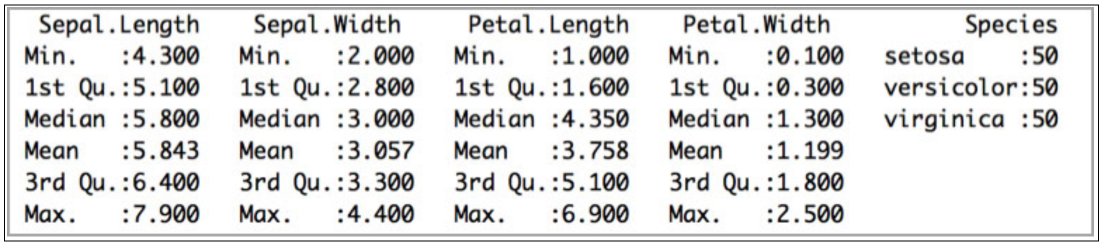
```
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
145          6.7         3.3          5.7         2.5 virginica
146          6.7         3.0          5.2         2.3 virginica
147          6.3         2.5          5.0         1.9 virginica
148          6.5         3.0          5.2         2.0 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
```

Figure A-2 Last six rows of iris data set

In this example, the tail part does not provide much more information.

Summary table

A summary table provides descriptive statistics of each variable so that you can view the data set at high level. A summary table for the Iris data set is shown in Figure A-3.



```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

Figure A-3 Example summary table for iris data set

This summary table describes the range and distribution of numeric variables (from Sepal Length to Petal Width), and the number of instances of each species.

Visualization

Visualization helps you understand the data intuitively. In this section, we visualize the Iris data set in bar, histogram, and scatter plots.

Bar plot

A bar plot often uses the length (or height) of the bars to indicate value and shows an intuitive comparison of values through the comparison of bar length. If you want to know whether and how the predictors differ across three species, you can apply bar plot on the mean value of the predictors, as shown in Figure A-4.

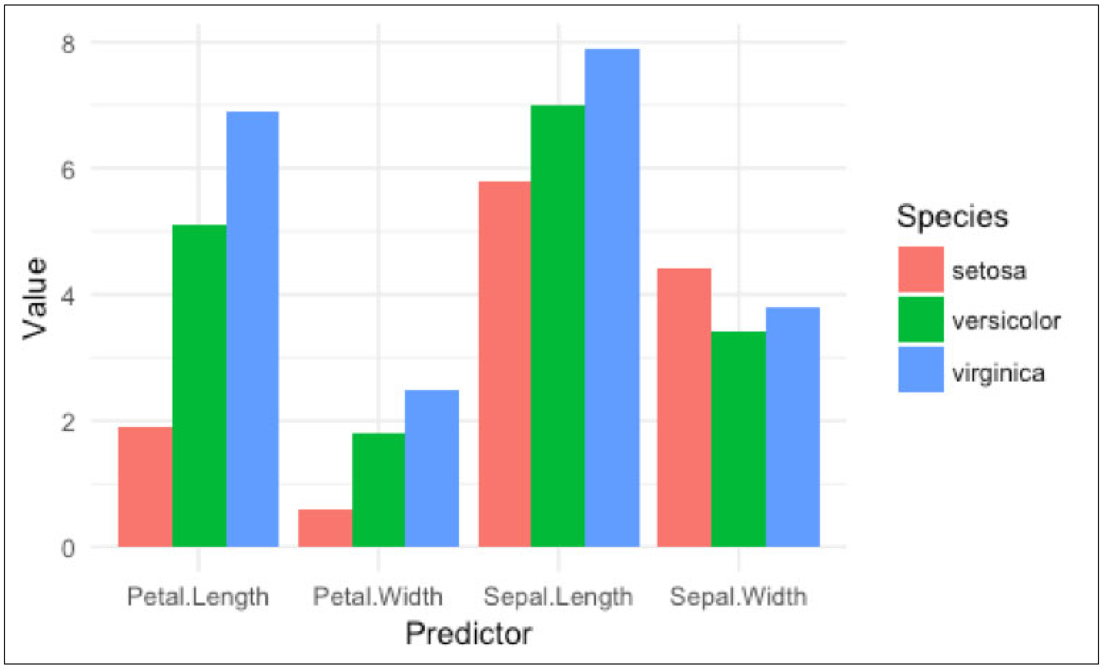


Figure A-4 Bar plot

The bars are grouped by predictor and colored by species, and the length of the bars represents the mean value. The bar plot shows a similar pattern for Petal Length, Petal Width, and Sepal Length, where the setosa iris has the lowest mean value, followed by versicolor iris, and the virginica iris has the highest mean value. Sepal width is different from the other three predictors, in which the setosa iris has the highest mean value.

Note: A more strict and quantitative examination of differences in mean value between groups requires statistical tests, such as the t-test that was used in AB tests.

Histogram plot

A histogram plot looks similar to a bar plot because it also consists of bars. Bar plots are used to display aggregated values, such as mean value. Histogram plots are used to display distribution, which is the count of observations that have the predictor value in a certain range. For example, you can apply histogram plot on the iris data set, as shown in Figure A-5.

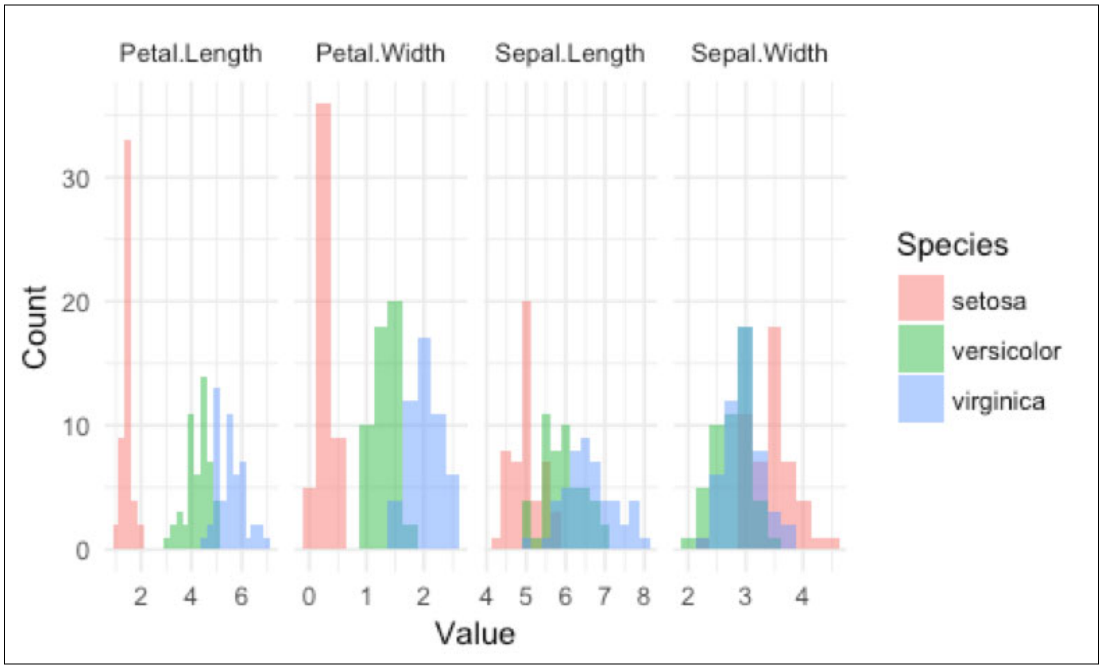


Figure A-5 Histogram plot

The plot that is shown in Figure A-5 shows the distribution of each species in each predictor. For Petal Length and Petal Width, the distributions of the species have little or none overlap. More specifically, the red setosa distribution does not overlap with the other two species, which means that by using only one predictor (either Petal Length or Petal Width), the setosa iris can be perfectly differentiated and predicted. The slight overlap between the green versicolor iris and the blue virginica iris indicates a strong capability of Petal Length and Petal Width to distinguish one species from the other.

Scatter plot

A scatter plot displays the data points in a specific feature space, and often includes lines or curves to indicate the relationship between two variables.

According to the bar plot and the distribution plot, you might wonder how Petal Length is related to Petal Width because you notice that they share a similar pattern in both plots. You can apply scatter plot on these two variables, as shown in Figure A-6 on page 163.

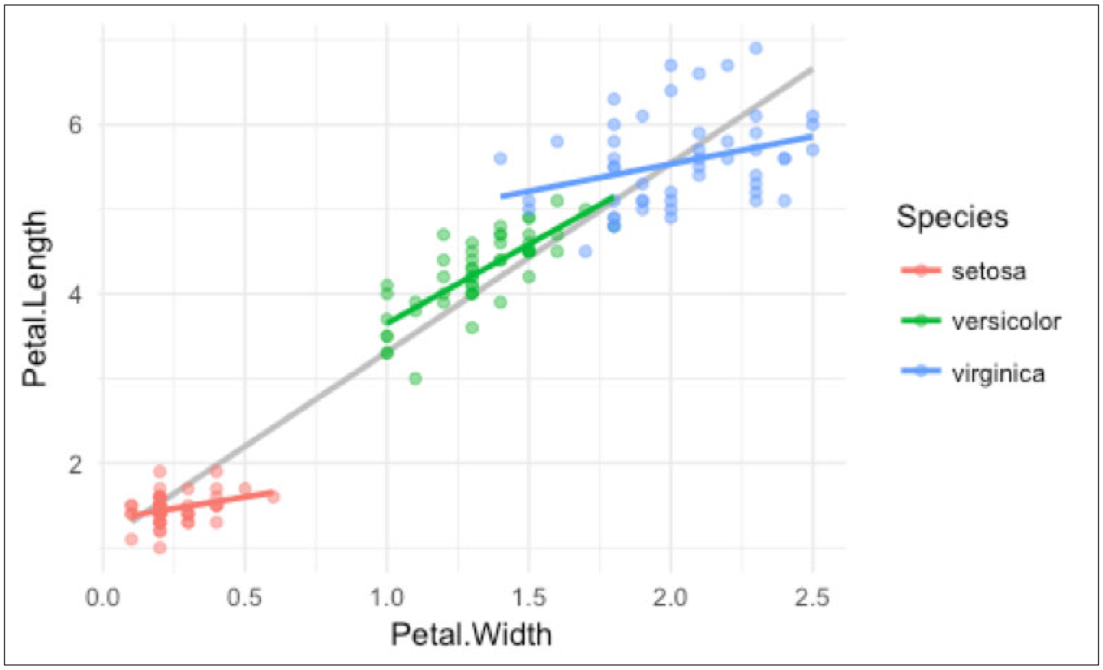


Figure A-6 Scatter plot

The scatter plot that is shown in Figure A-6 shows the data points on a 2-dimensional panel. The colored lines are linear regression lines within species, and the gray line is the linear regression line that is computed on all the data points across species. No matter within a species or across species, the linear relationship between Petal Length and Petal Width is suggested clearly by the plot.

Note: A more strict and quantitative examination of linear relationship requires the evaluation of linear model and significance of coefficient.

Feature preprocessing

Although the data is in a form that can be imported into machine learning algorithms any time, you might want to preprocess the features based on the insights you gain from the exploration or the experience you get from previous practices. This review results in removing existing features or creating features.

Feature selection

As the name implies, *feature selection* is the selection of features. The idea is to select a set of the most important features and remove the redundant features.

One reason is that not all available features in a data set are helpful. In fact, some features can be harmful to a model because they introduce noises into the data and make it difficult for machine learning algorithms to learn meaningful patterns.

In addition, many features can cause the so-called curse of dimensionality in some algorithms. This issue is particularly true for a fixed number of samples or data points.

As the number of features increases, the number of dimensions increases. Also, the feature space keeps growing and the distribution of data points becomes more sparse. Sparsity reduces the reliability and validity of the estimate and prediction.

Frequently used feature selection methods include the following examples:

- Checking the predictability of the features individually is the simplest method. Although this method is fast and easy, it ignores the combination of features that can yield a larger predicting power than the sum of those of individual features.
- Testing all of the possible feature combinations works well for small feature set, where the time it takes to go through all possibilities is affordable, as shown in the following examples:
  - Forward selection
  - Backward selection
- Some machine learning algorithms can function as feature selection tools, as shown in the following examples:
  - Lasso
  - Random Forest

Feature transformation

Feature transformation, or *feature engineering*, refers to the methods that derive new features from the existing features. The combination or transformation of several features is likely to clarify the patterns in the data set and derive new features and therefore helps improve the performance of the model.

For example, combining the amount of credit card debt and the amount of payment yields a new payment ratio feature, which is a stronger predictor for credit card default compared to the original two features. If the transaction amount has a large range and most of the transactions involves a small amount of money, transforming the transaction amount to log scale can be more helpful.

Frequently used feature transformation methods include the following examples:

- Principle component analysis (PCA) is a typical technique for dimension reduction, which projects (or compresses) the high dimensional data into a low dimensional representation in which the variance of data is kept as much as possible.
- Logarithm transformation changes a highly skewed distribution to a normal distribution (bell curve). This transformation is a common technique to deal with extremely skewed variables, such as number of followers of each social media user, where most people have relatively fewer followers while a few users have thousands of followers. Some algorithms assume that the variables are normally distributed, and violating this assumption negatively affects the performance of a model.

Model building

After you finalize the features, it is time to input the data into machine learning algorithms to train a model. A trained machine learning model consists of the following components:

- Algorithm: Defines the way that your model translates the input data
- Hyper-parameters: Specified manually before model training to further adjust the algorithm
- Parameters: Is learned automatically from the data

What happens in model building stage is the computation of the parameters given the algorithm, hyper-parameters, and the training data.

To select the proper machine learning algorithms, you must have a basic understanding of common machine learning algorithms.

Two main types of machine learning algorithms are available: unsupervised learning and supervised learning.

Unsupervised learning

Unsupervised learning refers to learning patterns on data without outcome variable, and clustering is the most frequently used type of unsupervised learning. However, the performance of an unsupervised machine learning model can hardly be measured or evaluated because of the lack of ground truth.

For example, customer segmentation is a scenario of unsupervised learning where subgroups are derived from the similarity or distance metric that is computed on many features (for example, age, gender, occupation, and annual income). In the customer segmentation case, no correct answer exists, and how good the segmentation depends on the business need.

K-means algorithm

In a K-means algorithm, “K” is the number of subgroups that is determined by data scientists in some way. For each subgroup, a point in the space represents the location of the centroid of this subgroup. The location of centroids can be initialized randomly and iterated and changed until the end of the algorithm.

The K-means algorithm requires computation of distance, and in some sense is equivalent to similarity between each data point and each centroid. The closest centroid becomes the subgroup to which this customer belongs.

In each iteration, the following process occurs:

1. The distance is computed.
2. The subgroup is assigned according to the newly computed distance for each data point.
3. The location of centroid is computed based on the result of assignment by taking the average of information of all data points in the corresponding subgroup.

The plot that is shown in Figure A-7 on page 166 shows the result of K-means algorithm applied on the Iris data set with three clusters (K = 3). In the computation of clusters, the label column Species is removed from the data set. The color of data points represents cluster assignment and the shape of data points represents species.

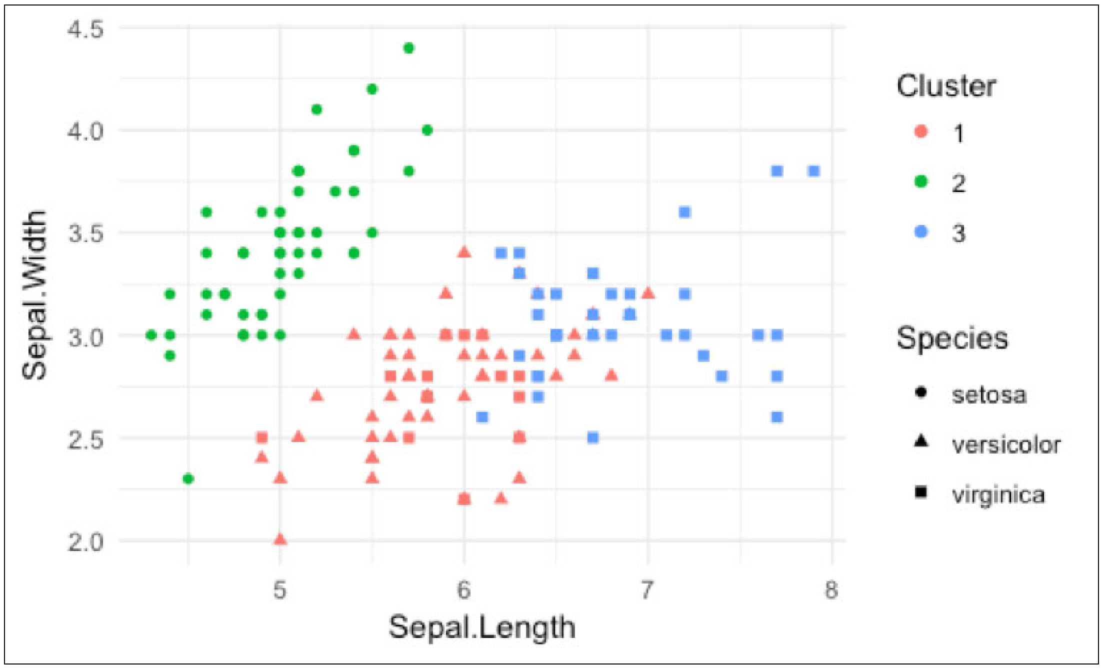


Figure A-7 Result of K-means algorithm

In this scatter plot, all setosa irises are clustered into one group (green and round points) and versicolor and virginica are mixed (triangle points and square points are colored in both red and blue). Therefore, it is not a good choice to apply this clustering algorithm on a large unlabeled data set and assign the species name to each cluster, except when you are concerned about only the differentiating setosa from the other two species.

Supervised learning

Supervised learning refers to learning patterns on data with outcome variable, essentially capturing the relationship between a set of predictors and one outcome variable. The performance can also be easily evaluated by comparing the predictions to the ground truth (that is, the outcome variable).

In supervised learning, model performance largely relies on the size of labeled data and the quality of label. For example, in the field of healthcare, sufficient data might not be available to train a machine learning model to screen a rare disease. Also, a human expert’s judgment whether an x-ray image shows a certain disease is not necessarily true.

Depending on the type of outcome variable, the supervised machine learning algorithms can be categorized into regression (continuous outcome variable; for example, revenue, temperature, and weight) or classification.

Regression is a categorical outcome variable, including binomial classification that has two-class categorical outcome variable, such as fraud or not, male or female. Multinomial classification has a multi-class categorical variable, such as more than two types of product, or more than two types of email label.

The supervised learning algorithms can also be categorized into one of the two modeling approaches: instance-based or model-based. Instance-based learning does not include a model and does not need training.

For each new unseen case, instance-based learning compares the case to all of the known samples, determines the most similar cases, and predicts the outcome of the unseen case by using the known outcome of the most similar cases. In contrast, model-based learning requires model training, where the model parameters are derived based on the known samples in some way.

Note: For model-based learning algorithms, loss function is used to evaluate the performance of a set of parameters and provide the guide to improve the model by deriving a better set of parameters to derive the model parameters. Loss function defines the difference between the actual outcome and the predicted outcome, and the model parameters are derived by minimizing such loss.

Supervised algorithms to be introduced include K-nearest neighbors, decision trees, Naïve Bayes classifier, linear regression, logistic regression, and neural networks.

K-nearest neighbors (K-NN)

Imagine that you want to buy a puppy from a breeder. The breeder posts pictures of all the available puppies but does not include the price information. How can you get an estimated price for the puppy you like?

Intuitively, you can start to search puppies of similar breed and color from similar breeders. You can also consider the prices and attributes of other puppies this breeder has available. Combining all of these pieces of information, you can get a rough estimate of the price for the puppy in which you are interested.

This example is the idea of K-nearest neighbors algorithm. K-NN predicts the class or value of a specific data point by picking and taking into consideration the K-nearest data points. Similar to the K-means algorithm, “k” is the number of nearest neighbors that are taken into consideration.

Assume that you set K to 3 and after the computation, the three observations that are listed in Table A-1 are ranked as the three most similar observations to the puppy you want to buy from the breeder. The final prediction of the price is the average of the prices in these three observations, which is $1,000.

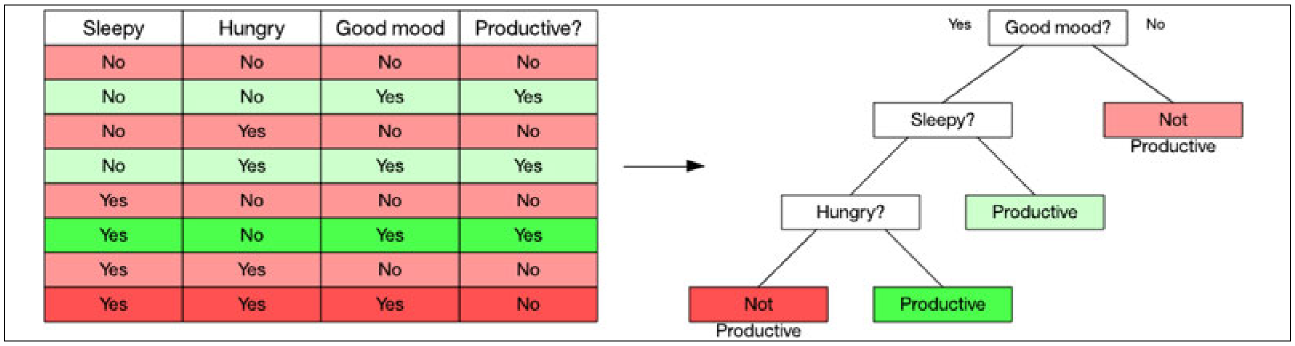
Table A-1 Three observations ranked

Rank	Breeder	Puppy gender	Puppy color	Price
1	A	Male	Fawn	$1,000
2	A	Male	Black and white	$1,200
3	B	Male	Fawn	$800

As one of the instance-based learning algorithm, no explicit model parameters are available to learn from the known data in k-NN algorithm. Instead, each unseen test case must be compared with all of the known data to get the ranks. Then, the computation cost is exploded as the number of test cases and the number of known cases increases.

Decision tree

A decision tree (see Figure A-8) can be considered as an automatically generated rule-based algorithm that is composed of a sequence of rules in a tree structure. The sequential rules mimic the process of human decision-making.



Sleepy	Hungry	Good mood	Productive?
No	No	No	No
No	No	Yes	Yes
No	Yes	No	No
No	Yes	Yes	Yes
Yes	No	No	No
Yes	No	Yes	Yes
Yes	Yes	No	No
Yes	Yes	Yes	No

Figure A-8 Decision tree

To build a decision tree, the algorithm starts from a pool of data points, recursively picking the feature and the split point on the feature that minimizes the loss or maximizes the gain. In some sense, this process is close to humans picking the most influential factor and forming the rule. Every split is a rule, such as “blood pressure > 16 kPa”, “humidity < 30%”, or “gender == female”.

In each step (splitting and growing branches), the decision tree essentially generates a rule to split the data points from a larger group into two smaller subgroups, where a data point in one group is more “similar” to members of its group than members of the other groups.

Given a new test case, the decision tree finds the most similar small group for it following the rules. The known data points in this final small space can be used to determine the outcome of the test case, which is similar to how K-NN algorithm determines the final prediction.

Decision trees can be used for classification and regression tasks. Classification trees and regression trees work in this way. The difference is how to find the best split point when building the tree and how to determine the outcome of the test case based on the final small group of known data points.

Classification trees usually minimize Gini impurity (measures the heterogeneity in a group of data points in terms of outcome label) or maximize information gain (measures the decrease of heterogeneity where the heterogeneity computation is different from Gini impurity) to find the best split point.

Regression trees use metrics, such as the sum of squares (measures how different the continuous outcome values are in a group of data points). Different metric certainly yields slightly different rules.

Regarding the prediction, most of the outcome labels in the final small group are the predicted label of the unseen test case in classification trees. In regression trees, the predicted value can be the average or median of outcome value of all data points in the final small group.

Despite the interpretability, single decision trees suffer from problems, such as overfitting (the sequence of rules performs well on the known data but can hardly be generalized) and instability (changing a small proportion of data can result in a totally different tree).

This technique asks the question: If a single decision tree is not satisfying, can we improve the performance by using more trees? Classical decision trees are deterministic, which means that given the same setting, the decision tree that is grown from the same data set always is the same no matter how many trials you make.

To improve the model performance by adding decision trees, the first challenge is to build a different tree each time. Two common approaches that are used are bootstrap aggregating (also known as bagging) and boosting.

- Random Forest (bagging example)

  Whenever the tree is built, random forest algorithm randomly samples the data points to be used from the known data to generate different decision trees. Some “lucky” data points might be included multiple times, while some “unlucky” data points might be excluded (that is, bootstrap).

  In addition, only a random proportion of features is used to build the tree. In this way, slightly different trees are built, and the predictions that are returned by all of the trees are equally considered to be aggregated for the final prediction (that is, of the same weight).

  For classification tasks, the output of each decision tree in a random forest is a label and the final predicted label is the majority label. For regression tasks, the output of each decision tree is a numeric value, and the final prediction can be the average or median of all these values.

  As each tree is built independently, random forest algorithm can be easily parallelized to reduce model training time.

- Gradient Boosted Decision Trees (GBDT) (boosting example)

  The classic boosting method builds decision trees on a different data set each time by changing the weight on data points. In classification tasks, more weight is placed on incorrectly predicted cases. In regression tasks, the change of weight depends on how far away the predicted value deviates from the actual values. The new data set that is used to build the next decision tree is the original data set times the weight that is calculated based on the prediction.

  Gradient boosting takes a slightly different approach by using the negative gradients of the loss, which usually are residuals in regression tasks as the new data set to build the next decision tree. It allows the output of each new decision tree to adjust the predictions of the previous trees, which leads to a better model performance.

  For example, if you want to predict the age of users (a continuous outcome variable), build the first tree on all the data, the way a single decision tree is built. Then, build the other trees on the residual of the actual value and the overall predicted value.

  If the first tree predicts the age of a user to be 60 but the actual age is 40, the residual -20 instead of the actual age 40 is used in the second data set to build the second tree. Then, if the second tree gives a prediction of -10, the overall prediction of their age is 60 + (-10) which is 50. The residual of the actual age 40 and this overall prediction 50, which is -10, replaces -20 in the third data set to build the third tree.

  Compared to random forest, the new data set for each tree in GBDT depends on the prediction of all the previous trees, which means the data sets must be generated sequentially. The result is much longer model training time.

Note: Although decision trees can theoretically handle categorical variables by generating rules, such as “weather == sunny”, this kind of rule is not implemented in many popular decision tree libraries, such as scikit-learn in python and randomForest in R. When these libraries are used to build the decision trees, categorical predictors still must be transformed into numeric format.

Naïve Bayes classifier

Naïve Bayes classifier is a simple but powerful machine learning algorithm, which is based on Bayes' theorem. Bayes' theorem explains that given the probability of event A happens (P(A), the prior) and the probability of event B happens (P(B), the evidence), how to convert the probability of event B happening given that event A happened (P(B|A), the likelihood) to the probability of event A happening given that event B happened (P(A|B), the posterior).

Consider the medical test of a disease as an example. Suppose that the test has 99% sensitivity (99% of real patients will be diagnosed as positive) and 99% specificity (99% of non-patients will be diagnosed as negative). A large-scale national research effort finds that 0.1% of the population might have the disease. When your doctor tells you that your test result is positive, what is the probability that you have the disease? You can use the Naïve Bayes classifier to find the answer as shown in the next example.

Given the following evidence:

```
P(test is positive | have disease) = 99%
P(test is negative | do not have disease) = 99%
P(have disease) = 0.1%
```

The following formulas and calculations are used:

```
P(test is positive) = P(test is positive | have disease) * P(have disease) +
P(test is positive | do not have disease) * P(do not have disease) = 99% * 0.1%
+ (1 - 99%) * (1 - 0.1%) = 1.094%
P(have disease | test is positive) = P(test is positive | have disease) *
P(have disease) / P(test is positive) = 99% * 0.1% / 1.094% = 9.05%
```

Similar logic can be applied to other scenarios, such as spam email classification. Users report some emails as spam, from where the conditional probability of a word showing up in the email given the email is spam can be computed (P(word | is spam)). For a new email, the naïve Bayes classifier computes the probability of this email being spam based on the words it contains (P(is spam | word)).

The training time for Naïve Bayes classifiers is relatively short compared to other machine learning algorithms. A simple Naïve Bayes model sometimes outperforms more complicated machine learning models.

Linear and logistic regression

Modeling the outcome variable as a function of linear combination of predictors is a common way to depict the relationship between outcome and predictors. It is easy to understand and interpret. Linear regression and logistic regression are two types of popular linear models. In these linear models, each predictor has a separate and additive effect.

Assume that you want to predict the height of a group of children. You collect data about weight and age, and then build a linear regression model with the formula "Height = Weight + Age + intercept." The model parameters (coefficient of weight, coefficient of age, and intercept) must be derived that can minimize the loss or the difference between predicted height and actual height. After the model is built, you can easily tell how weight and age influence height.

Linear regression does not work properly when the outcome variable is categorical. One of the reasons is that the range of prediction in a linear regression model, or so-called linear probability model in this case, on binary outcome exceeds the range that you want. In this situation, you can turn to logistic regression.

Although logistic regression has *regression* in its name, it can handle a binary outcome variable and is considered more as a classification algorithm. Logistic regression can be understood as a linear regression with a non-linear sigmoid transformation, which maps the outcome of linear regression to a range 0 - 1. Therefore, the predicted value of logistic regression always falls in the range 0 - 1.

The chart that is shown in Figure A-9 visualizes linear regression (blue line) and logistic regression (red curve) on the Iris data set. Sepal length is used as the only predictor to predict two species (setona and versicolor). It is clear that, according to the linear regression model in this plot, any iris with sepal length over 6.5 has a probability greater than 1 to be labeled as one species, while any iris with sepal length lower than 4.5 has a negative probability to be labeled as one species. Compared to the linear regression model, the logistic regression model always gives a meaningful prediction 0 - 1.

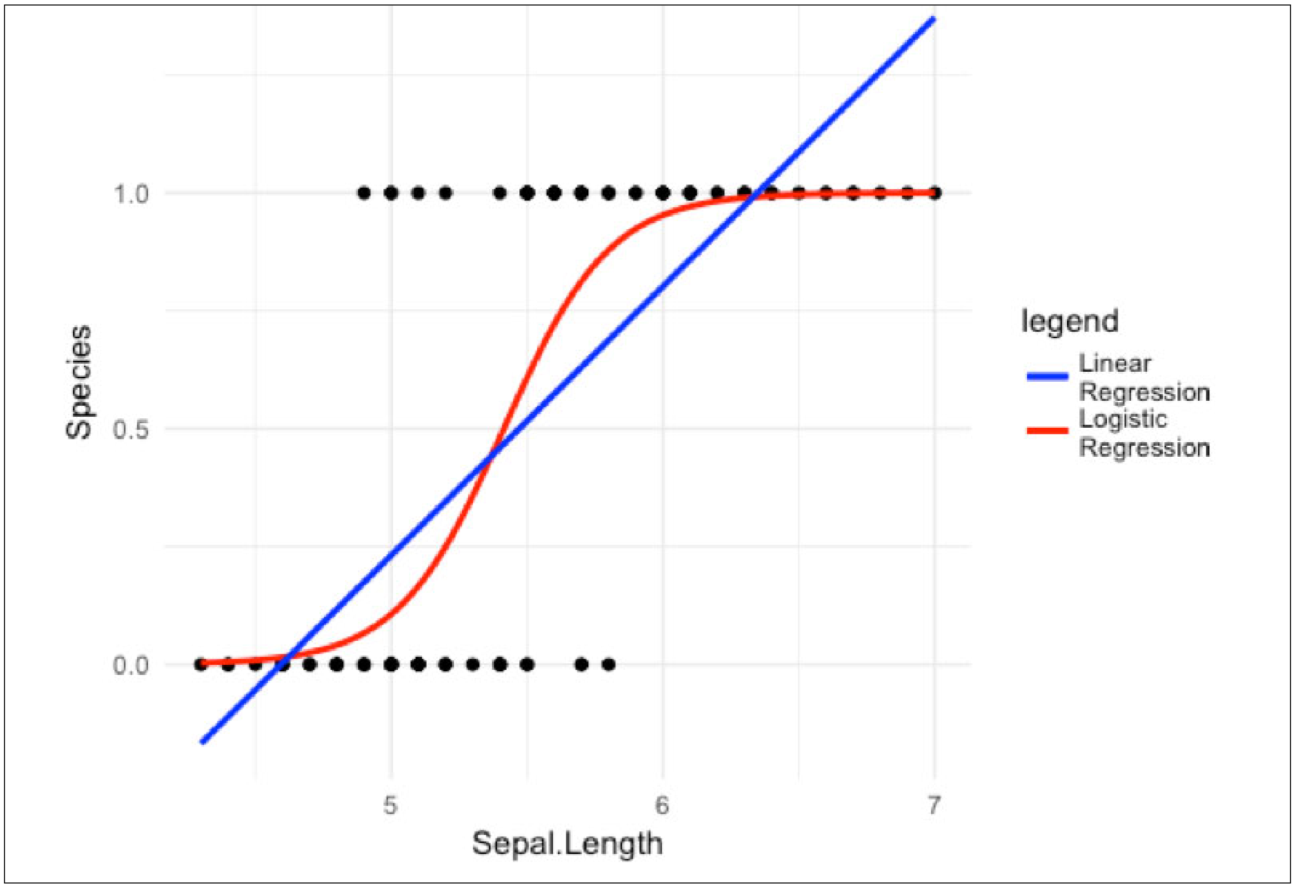


Figure A-9 Feed forward neural network example

For multi-class prediction problems, multiple logistic regression models can be used. For example, if the outcome variable is blood type with four unique values (A, B, AB, and O), four logistic regression models can be built with outcome variable A or Not A, B or Not B, AB or Not AB, and O or Not O, respectively, and combine the predictions to generate the final prediction.

Another slightly different situation is ordinal multi-class prediction where the outcome variable is categorical and ordinal. Take egg quality with Grade A - C as an example. Assume that Grade A is better than Grade B and Grade B is better than Grade C, and the relationship between the predictors and egg quality meets the assumptions of ordinal logistic regression. In this case, the use of ordinal logistic regression might be a better choice than building multiple logistic regression models.

Ordinal logistic regression takes the ordinal relationship within the outcome variable into consideration. It also captures the relationship between predictors and outcome variable in one model, which is much simpler than building multiple logistic regression models.

Note: The meaningful range of prediction is one of the reasons why logistic regression is a better binary classifier than linear regression. Other reasons involve mathematical proofs and statistical considerations that support the superiority of logistic regression over linear regression in binary classification tasks.

Neural networks

Neural networks are a layer-by-layer linear combination of variables with usually non-linear activation functions that are applied on each layer (that is, each linear combination). In practice, neural networks boost the performance of machine learning in many tasks to an extent that surprises and surpasses humans.

Neural networks require a huge amount of data to learn from and the training takes substantially longer time than most of the traditional machine learning algorithms. The massively growing data and the rapidly increasing computational capabilities bring neural networks to life.

Neural networks feature the following important characteristics:

- The layer-by-layer or hierarchical structure in which each layer contains many hidden units allows the model to learn feature combinations automatically in a bottom-up way. For example, the lower layers in a convolutional neural network (CNN) that is trained on animal images for animal classification tasks can contain detectors of lines and simple curves in certain positions. The higher layers are the combination of these simple detectors to identify unique and distinguishable patterns like the beak of birds. These patterns are useful in differentiating birds from cats or dogs.
- Non-linear activation functions that are applied on a linear combination of features make it theoretically possible for neural networks to approximate almost any non-linear relationship. Although linear models, such as linear regression and logistic regression, sometimes work well, the relationship between the predictors and the outcome variable in real world can rarely be expressed by a simple linear combination of variables, Most of the relationships are non-linear.

  Although human experts must spend a large amount of time figuring out the correct non-linear formula to depict the relationship appropriately, neural networks might achieve similar or even better performance by automatically learning from the training data.

The basic types of neural networks that are universally popular include feed forward neural networks (or fully connected neural networks), convolutional neural networks (CNN), and recurrent neural networks (RNN). How a feed forward neural network works is shown in Figure A-10.

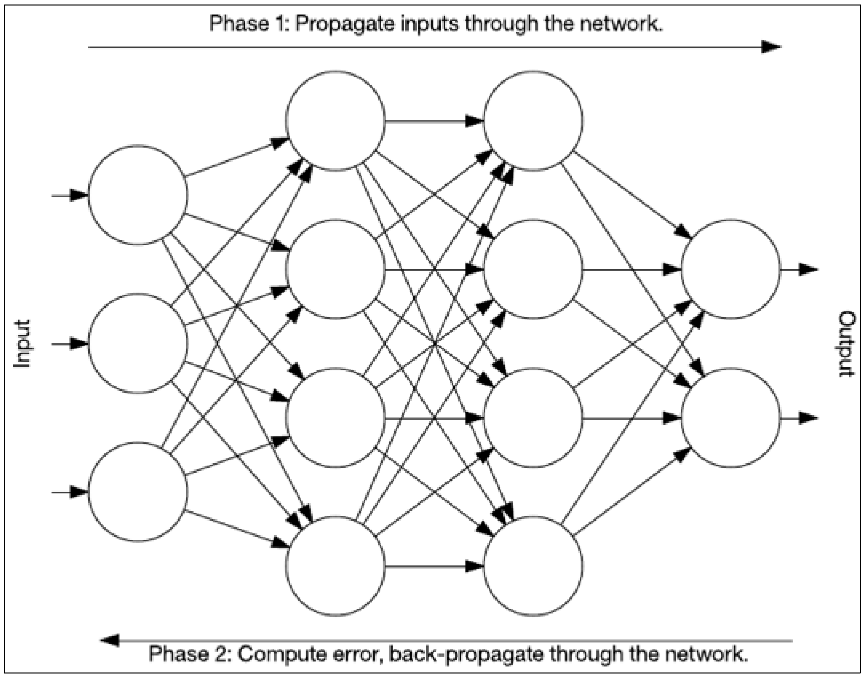

Phase 1: Propagate inputs through the network.
Input
Output
Phase 2: Compute error, back-propagate through the network.

Figure A-10 Feed forward neural network example

The data in the feed forward neural network is imported into the input layer, and the output of each node in the input layer flows into every node in the first hidden layer. After transformation, the output of each node in the first hidden layer flows into every node in the second hidden layer.

In the end, the four outputs from the second hidden layer are gathered in the final two nodes in the output layer. You might notice that in a feed forward neural network, the output of each node flows into all the nodes in the next layer.

Cross validation for hyper-parameter optimization

After you decide on the machine learning algorithms, hyper-parameters also must be determined. *Hyper-parameters* refer to the parameters that are assigned by the data scientists to define the model at a high level, such as the number of layers in a neural network, which are different from the parameters the model learned automatically from the training data. In the case of most of the machine learning algorithms, hyper-parameters can influence the model performance significantly (Figure A-11 on page 174).

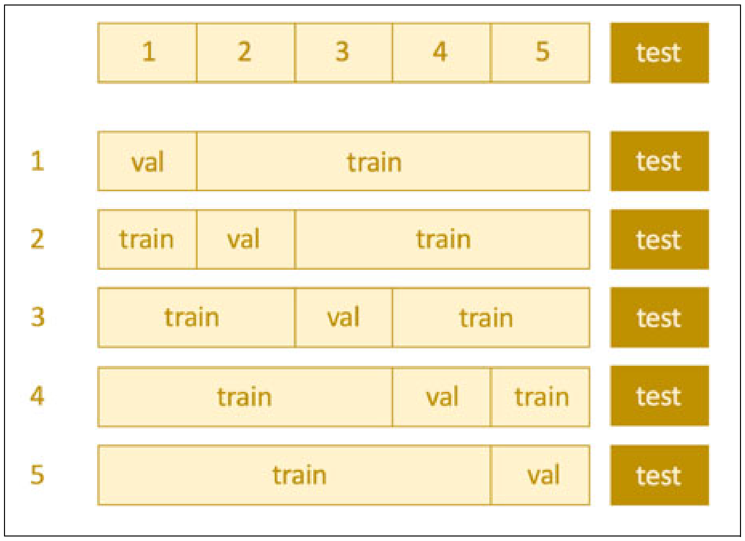


Figure A-11 Cross validation for hyper-parameter optimization

To optimize hyper-parameters, one of the frequently used techniques is the train/test split with k-fold cross-validation method ("k" is the number of folds). The process includes the following steps:

1. Split the entire data set into a training set and test set; for example, split 60% of the data into the training set and the remaining 40% into the test set that is held back and kept separate during the model training phase.
2. For several sets of hyper-parameters, cross-validation is performed by using only the training data set:
   a. Training data is further split into k parts. In 5-fold cross-validation, the training data set is split into five chunks.
   b. The model is trained and evaluated k times, each time by using one unique part as a validation set and the remaining four parts as training data set to build the model. In 5-fold cross-validation, the model is trained and evaluated five times. It is built by using different combination of data parts.
   c. Average of the metrics across k times evaluation is calculated to determine the final evaluation of the model under a particular model setting (a specific machine learning algorithm and a specific set of hyper-parameters). Other metrics, such as the standard deviation of loss, can also be derived to measure the stability of the model performance.
3. The best set of hyper-parameters is used (the criteria depend on the task and expectation) to train the final model on all of the data for cross validation, which is the full data set excluding the test set.
4. The model is evaluated by using the held-out test set.

Cross-validation is a method to evaluate the model performance more precisely by measuring its performance multiple times. Because it evaluates the model performance on the validation data set that is not included in data based on which the model is trained (within the cross-validation step), cross-validation assesses the generalization capability of hyper-parameters.

Model evaluation

After a trained machine learning model is available, evaluation is needed to understand how well or how worse the model's future predictions might be.

Loss function measures the difference between the prediction and the actual outcome value, which evaluates the performance of a machine learning model. In the training process, a best set of parameters is derived to minimize the loss and thus maximize the performance of the model.

Loss functions that are used in regression tasks are different from those used in classification tasks. For the numeric outcome variable, the difference between the prediction and the actual value quantifies the extent to which the prediction is close to truth. For example, when predicting temperature, a 1-degree deviation from the truth is better than 10-degree deviation. However, the same method cannot be applied to the categorical outcome variable.

In addition to loss, other methods are available to evaluate the model performance.

Regression

In regression tasks, mean squared error (MSE) is one of the most commonly used loss functions. It is the average of squared error, where the squared error is calculated as the square of the difference between the predicted outcome value and the actual outcome value.

Mean Absolute Error (MAE) is another popular loss function in regression tasks. It is calculated as the average of absolute difference between the predicted outcome value and the actual outcome value.

For linear regression, R-squared indicates the range of variance (0% - 100%, the higher the better) in the data can be explained by the model, which is calculated as the explained variance versus the total variance.

Classification

In classification tasks, the basic loss function is 0-1 loss, which assigns 1 to a misclassified case and 0 to a correctly classified case. A 0-1 loss is also used to compute the misclassification rate, which is the number of misclassified cases versus the number of all cases.

Because all of the classification algorithms produce a predicted probability of the test case being in a class, a more careful evaluation can be made by taking the probabilities into consideration. For example, in binary classification, a probability more deviated from 0.5 (which is the default threshold to differentiate predicted label) implies a more "confident" classification result.

Given two binary classification models with 0% misclassification rate, the model that is giving predicted probabilities, such as 90% and 10%, is preferred over the other model that is giving predicted probabilities, such as 40% and 60%. Loss functions, such as hinge loss and exponential loss, assign a larger value to a more deviated prediction.

Consider the following attributes related to classification models:

- Confusion matrix

  Confusion matrix is a popular and useful evaluation of a binary classification model where the outcome variable has only two unique labels. All situations where a prediction can be identified are listed in Table A-2.

Table A-2 Identifiable predictions

	Predicted negative	**Predicted positive**
Actual negative	True negative	False positive
Actual positive	False negative	True positive

  Consider the following points:

  - False Positive (FP) refers to the test cases that are negative but predicted as positive.
  - False Negative (FN) refers to the test cases that are positive but predicted as negative.
  - True Positive (TP) refers to the test cases that are positive and correctly predicted as positive. the True Positive Rate (TPR) is the ratio of number of true positive cases (TP) versus the number of all actual positive cases (TP + FN).
  - True Negative (TN) refers to the test cases that are negative and correctly predicted as negative. The True Negative Rate (TNR) is the ratio of number of true negative cases versus the number of all actual negative cases (TN + FP).

  Accuracy and precision metrics can be derived based on the numbers in a confusion matrix. Consider the following points:

  - Accuracy is the number of all correctly predicted cases (TP + TN) divided by the number of all test cases.
  - Precision is the ratio of number of true positive cases (TP) versus the number of all predicted positive cases (TP + FP).

- ROC curve and PR curve

  As the classification model returns probabilities, the threshold can be moved to generate different classification results. For example, in a model that is used to screen a dangerous disease, predicting an actual healthy person as patient (false positive) might result in one more further screening test. Predicting an actual patient as a healthy person (false negative) can cost the person their life.

  In this example, the threshold can be moved lower from 0.5 (by default) to 0.3 so that participants who get more than 30% predicted chance to have the disease are reported by the model as patient.

  Such tradeoff can be shown in receiver operating characteristic (ROC) curve and precision-recall (PR) curve, which is obtained by using the metrics that are computed in a confusion matrix. You can have multiple ROC curves on a plot and each curve on the plot represents a model, which consists of points that are computed under different thresholds.

  In an ROC curve plot, the x-axis is FPR and the y-axis is TPR. The ROC curve describes the change in TPR as FPR increases. The TPR increases monotonically because a higher false positive rate implies a lower threshold where more actual positive cases can be predicted as positive.

The plot that is shown in Figure A-12 is an example of a ROC curve.

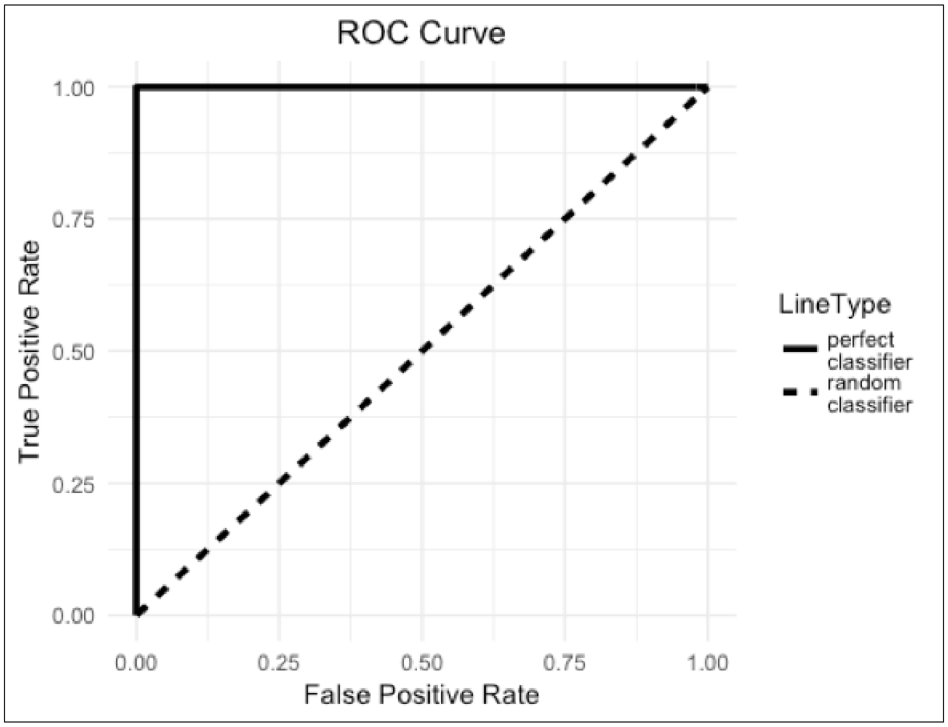


Figure A-12 ROC curve example

An ideal model includes a steeply increasing ROC curve at the beginning, which means that the model can achieve a high TPR with a low FPR. On the ROC curve, random guess is the diagonal line from the origin of the axes at the lower left to the point at the upper right. Any model with a ROC curve below or similar to the diagonal line is no better than the random guesses.

In a PR curve plot, the x-axis is Recall (TPR) and the y-axis is Precision. Typically, as Recall increases, Precision decreases. PR curve is sensitive to class imbalance (a small number of positive cases versus many negative cases) and is more useful than ROC curve in such situation.

The plot that is shown in Figure A-13 is an example of a PR curve.

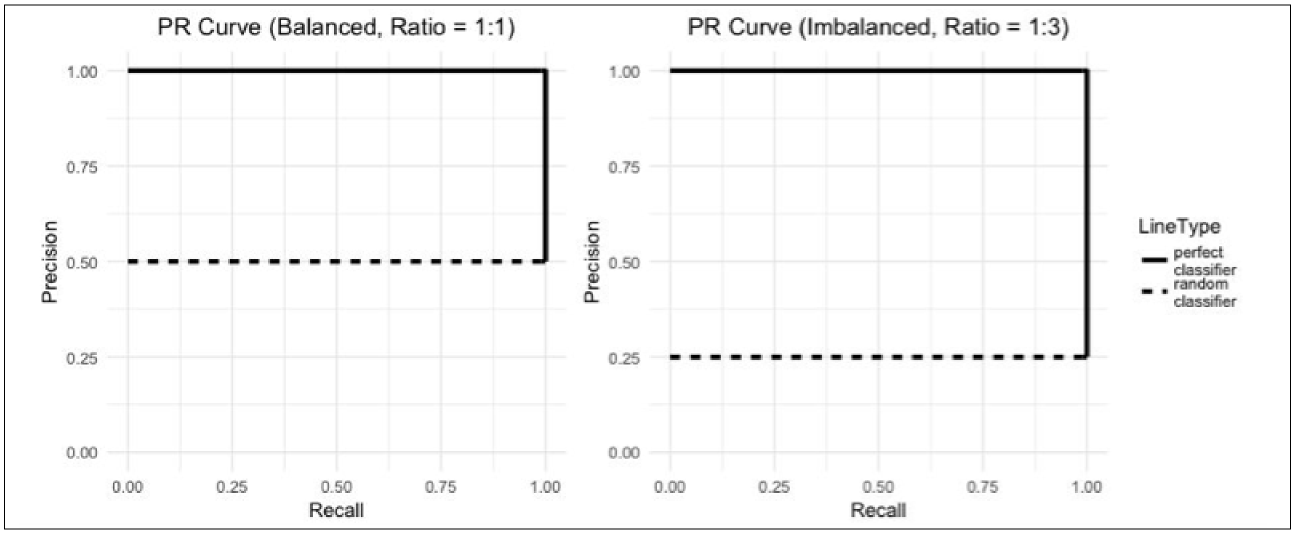


Figure A-13   PR curve example

An ideal model has steeply decreasing PR curve at the end, which means that the model can keep a high precision score regardless of the increase in recall. On a PR curve, random guess is a horizontal line whose position depends on the ratio of positive cases versus negative cases. Any model with a PR curve below or similar to the horizontal line is no better than random guesses.

In addition to visualizing the ROC and PR curves, the area under curve (AUC) can be used to evaluate the performance of a model and compared quantitatively.

The confusion matrix, ROC curve, and PR curve are all designed to evaluate performance of binary classification models and it is difficult to apply them on multi-class classification models.

Consider a 3-class classification model as an example. If a confusion matrix is used, the predictions and actual values form a 3x3 table where metrics-like false positive rate (FPR) cannot be computed by using the same definition. Correspondingly, the ROC curve and the PR curve cannot be made. These concepts can be extended to multi-class classification with some change in definition and interpretation.

B

R and RStudio

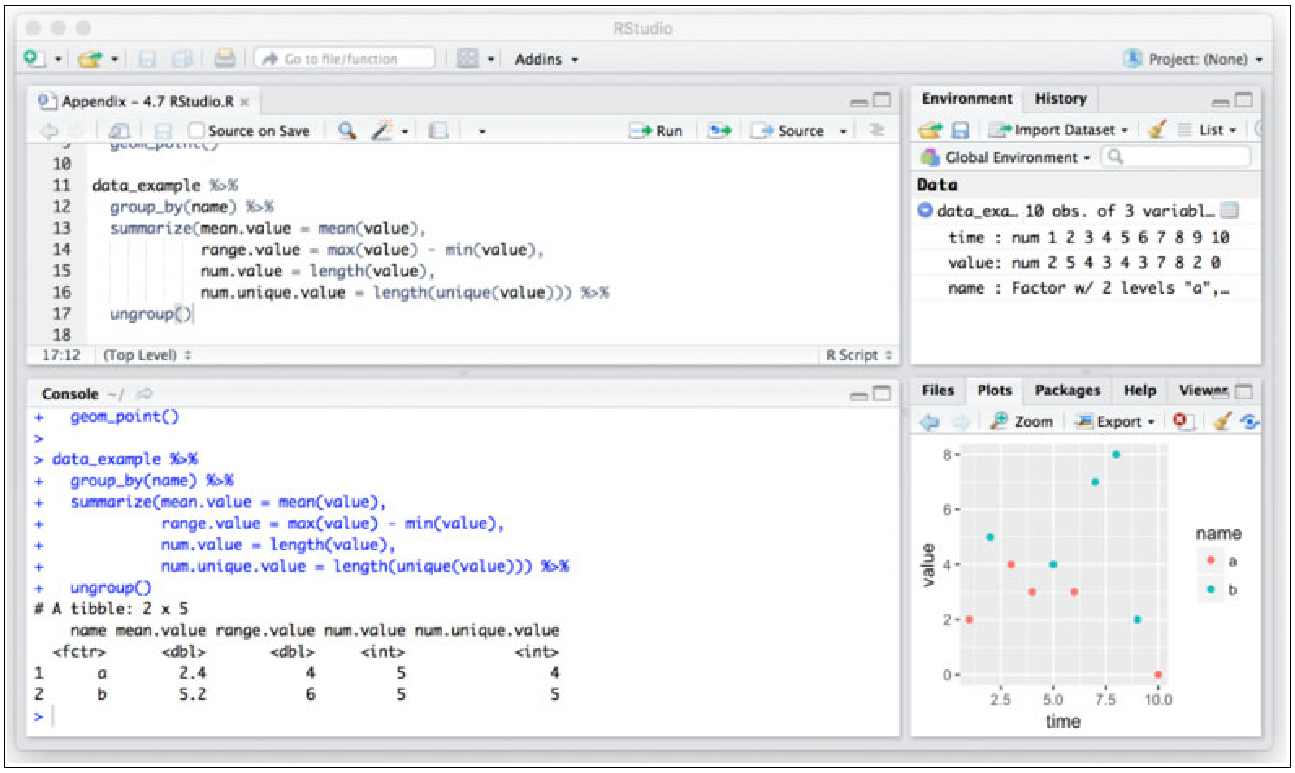
In this appendix, we introduce R and RStudio, highlight the benefits of using R for data science, and demonstrate the readability of code in R.

This appendix includes the following topics:

Overview

R is an open-source software environment that was developed for statistical computing and became popular among statisticians. It is also one of the most widely used languages for data analytics.

RStudio is an open-source integrated, development environment (IDE) for R. The RStudio client user interface (UI) is similar to that of MATLAB, as shown in Figure B-1.



```
data_example %>%
  group_by(name) %>%
  summarize(mean.value = mean(value),
            range.value = max(value) - min(value),
            num.value = length(value),
            num.unique.value = length(unique(value))) %>%
  ungroup()
```

```
data_exa… 10 obs. of 3 variabl…
  time : num 1 2 3 4 5 6 7 8 9 10
  value: num 2 5 4 3 4 3 7 8 2 0
  name : Factor w/ 2 levels "a",…
```

```
+   geom_point()
>
> data_example %>%
+   group_by(name) %>%
+   summarize(mean.value = mean(value),
+             range.value = max(value) - min(value),
+             num.value = length(value),
+             num.unique.value = length(unique(value))) %>%
+   ungroup()
# A tibble: 2 x 5
    name mean.value range.value num.value num.unique.value
  <fctr>      <dbl>       <dbl>     <int>            <int>
1      a        2.4           4         5                4
2      b        5.2           6         5                5
>
```

Figure B-1 RStudio UI default configuration

The default configuration of the RStudio UI includes the following components:

- The Script pane (upper left side) shows your scripts in R. You can select to run individual lines or the entire script.
- The Interactive console pane (lower left side) shows the code that you ran and the output that is generated by that code. It also allows you to enter and run code directly without a script. The code that is run is displayed in blue and the output is displayed in black.
- The Workspace pane (upper right side) includes the following tabs by default:
  - Environment tab: All variables are listed with related information.
  - History tab: Records all of the codes that are run.
- The pane in the lower right side features several tabs, including:
  - Plots: Displays plot information.
  - Help: Shows the help information for a specific package or function.

Why R?

By design, R is a language for statisticians and inherently supports concepts, such as matrix and data frame. Although being considered not computationally efficient in some cases (for example, for loop), R is a great option for data manipulation, exploration, and visualization.

One of the many R packages available, Tidyverse[1], or the so-called Hadley environment, consists of several frequently used packages for these purposes. This collection of R packages makes writing and reading R code much easier and faster. It also provides simpler solutions for data scientists to understand data and therefore, improve their productivity.

Tidyverse includes the following notable packages:

- dplyr[2]

  This package is the grammar of efficient and clean data manipulation that uses a set of verbs for most common operations that can be piped together. This package also works seamlessly with remote databases, including the following examples[3]:

  - MySQL and Maria DB
  - Postgres and Redshift
  - SQLite
  - Commercial databases that support ODBC (an open database connectivity protocol)
  - Google's BigQuery

- ggplot2[4]

  This package is a plotting system for R that implements a graphic scheme that is named Grammar of Graphics, where graphs are built by using semantic components in a layer-by-layer manner.

Readability of R

The code in R can be descriptive, and easy to write and read. The examples in this section are snippets of R code that demonstrate the readability of ggplot2 and dplyr packages.

Loading a built-in data set

The built-in airquality data set in R contains the daily air quality metrics in New York from May 1973 to September 1973. Metrics that are included in this data set are Ozone in parts per billion, solar radiation in Langleys, average wind speed in miles per hour, and maximum daily temperature in degrees Fahrenheit. You can load the data frame into the current environment by running the following code in R:

```
data('airquality')
```

Visualizing data quickly

To simplify the problem, we review the relationship between wind speed and Ozone. You might ask the following questions:

- How do the data points distribute in the space?
- Do the data points that are collected in a different month have different patterns?

[1] For more information, see https://www.tidyverse.org/.
[2] For more information, see http://dplyr.tidyverse.org/.
[3] For more information, see http://db.rstudio.com/dplyr/.
[4] For more information, see http://ggplot2.tidyverse.org/.

You can use ggplot2 to quickly plot the data. By running the following code, you can set the x-axis to wind speed and the y-axis to Ozone, plot a LOESS curve to show the local direction of change among the data points, and split the plot by month:

```
airquality %>%
   ggplot(aes(x = Wind, y = Ozone)) +
   geom_point() +
   geom_smooth(method = 'loess', se = False) +
   facet.grid(.~Month)
```

The code generates the plot that is shown in Figure B-2.

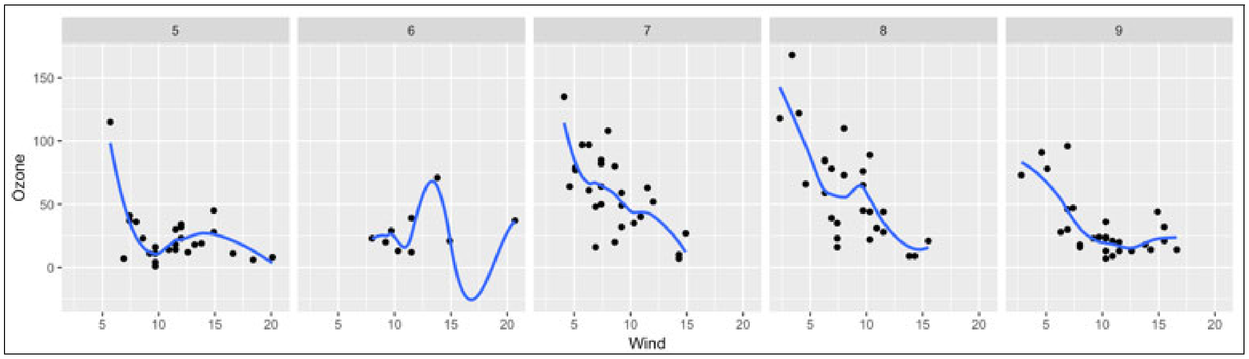


Figure B-2 Data plot

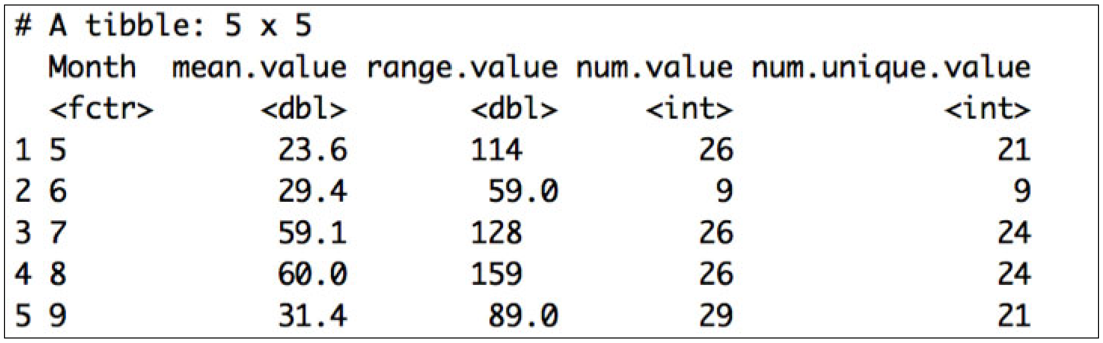
The plot shows that in July and August, the relationship between wind speed and Ozone is similar. It also shows that the pattern in May and September is similar to each other. The curve in June seems to be different from the curve in the other months, which might be caused by the missing values.

Exploring the outcome variable

To see how the Ozone varies by month, you might want to use several customized metrics. In this example, the mean of values, range of values, number of values, and number of unique values are included in the following code:

```
Airquality %>%
   group_by(Month) %>%
   summarize(mean.value = mean(Ozone),
            range.value = max(Ozone) - min(Ozone),
            num.value = length(Ozone),
            num.unique.value = length(unique(Ozone))) %>%
   ungroup()
```

Running the code generates the summary table that is shown in Figure B-3.



```
# A tibble: 5 x 5
  Month  mean.value range.value num.value num.unique.value
  <fctr>      <dbl>       <dbl>     <int>            <int>
1 5            23.6       114          26               21
2 6            29.4        59.0         9                9
3 7            59.1       128          26               24
4 8            60.0       159          26               24
5 9            31.4        89.0        29               21
```

Figure B-3 Summary table

As anticipated, Ozone in June includes many missing values (nine valid numbers compared to over 20 in other months). Insufficient samples in June leads to an unreliable estimate. In addition, the air quality in terms of Ozone is relatively low in July and August. The average value of Ozone in these months is almost double the average in the other months.

Redbooks Turning Data into Insight with IBM Machine Learning for z/OS

(0.2"spine)
0.17"<->0.473"
90<->249 pages

SG24-8421-00

ISBN 0738457132

Printed in U.S.A.