

Certified Artificial Intelligence (AI) Practitioner (Exam AIP-110)

Certified Artificial Intelligence (AI) Practitioner (Exam AIP-110)

Part Number: CNX0008

Course Edition: 1.0

Acknowledgements

PROJECT TEAM

Authors	Technical Consultants	Media Designer	Content Editor
Jason Nufryk	Raymond Ptucha, Ph.D.	Brian Sullivan	Peter Bauer
Brian Wilson	Andrea Cogliati, Ph.D.		

CertNexus wishes to thank Chris Mawata and Andrew Palladino for their instructional and technical expertise during the creation of this course.

Notices

DISCLAIMER

While CertNexus, Inc. takes care to ensure the accuracy and quality of these materials, we cannot guarantee their accuracy, and all materials are provided without any warranty whatsoever, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. The name used in the data files for this course is that of a fictitious company. Any resemblance to current or future companies is purely coincidental. We do not believe we have used anyone's name in creating this course, but if we have, please notify us and we will change the name in the next revision of the course. CertNexus is an independent provider of integrated training solutions for individuals, businesses, educational institutions, and government agencies. The use of screenshots, photographs of another entity's products, or another entity's product name or service in this book is for editorial purposes only. No such use should be construed to imply sponsorship or endorsement of the book by nor any affiliation of such entity with CertNexus. This courseware may contain links to sites on the Internet that are owned and operated by third parties (the "External Sites"). CertNexus is not responsible for the availability of, or the content located on or through, any External Site. Please contact CertNexus if you have any concerns regarding such links or External Sites.

TRADEMARK NOTICES

CertNexus and the CertNexus logo are trademarks of CertNexus, Inc. and its affiliates.

All other product and service names used may be common law or registered trademarks of their respective proprietors.

Copyright © 2020 CertNexus, Inc. All rights reserved. Screenshots used for illustrative purposes are the property of the software proprietor. This publication, or any part thereof, may not be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, storage in an information retrieval system, or otherwise, without express written permission of CertNexus, 3535 Winton Place, Rochester, NY 14623, 1-800-326-8724 in the United States and Canada, 1-585-350-7000 in all other countries. CertNexus' World Wide Web site is located at www.certnexus.com.

This book conveys no rights in the software or other products about which it was written; all use or licensing of such software or other products is the responsibility of the user according to terms and conditions of the owner. Do not make illegal copies of books or software. If you believe that this book, related materials, or any other CertNexus materials are being reproduced or transmitted without permission, please call 1-800-326-8724 in the United States and Canada, 1-585-350-7000 in all other countries.

Certified Artificial Intelligence (AI) Practitioner (Exam AIP-110)

Lesson 1: Solving Business Problems Using AI and ML.... 1

Topic A: Identify AI and ML Solutions for Business Problems.....	2
Topic B: Follow a Machine Learning Workflow.....	10
Topic C: Formulate a Machine Learning Problem.....	15
Topic D: Select Appropriate Tools.....	25

Lesson 2: Collecting and Refining the Dataset.....37

Topic A: Collect the Dataset.....	38
Topic B: Analyze the Dataset to Gain Insights.....	52
Topic C: Use Visualizations to Analyze Data.....	71
Topic D: Prepare Data.....	85

Lesson 3: Setting Up and Training a Model..... 97

Topic A: Set Up a Machine Learning Model.....	98
Topic B: Train the Model.....	108

Lesson 4: Finalizing a Model.....	131
Topic A: Translate Results into Business Actions.....	132
Topic B: Incorporate a Model into a Long-Term Business Solution.....	137
Lesson 5: Building Linear Regression Models.....	143
Topic A: Build Regression Models Using Linear Algebra.....	144
Topic B: Build Regularized Regression Models Using Linear Algebra....	168
Topic C: Build Iterative Linear Regression Models.....	181
Lesson 6: Building Classification Models.....	193
Topic A: Train Binary Classification Models.....	194
Topic B: Train Multi-Class Classification Models.....	216
Topic C: Evaluate Classification Models.....	225
Topic D: Tune Classification Models.....	242
Lesson 7: Building Clustering Models.....	253
Topic A: Build k-Means Clustering Models.....	254
Topic B: Build Hierarchical Clustering Models.....	276
Lesson 8: Building Decision Trees and Random Forests.....	291
Topic A: Build Decision Tree Models.....	292
Topic B: Build Random Forest Models.....	313
Lesson 9: Building Support–Vector Machines.....	333
Topic A: Build SVM Models for Classification.....	334
Topic B: Build SVM Models for Regression.....	352
Lesson 10: Building Artificial Neural Networks.....	363
Topic A: Build Multi-Layer Perceptrons (MLP).....	364

Topic B: Build Convolutional Neural Networks (CNN).....	383
Topic C: Build Recurrent Neural Networks (RNN).....	405
Lesson 11: Promoting Data Privacy and Ethical Practices.....	423
Topic A: Protect Data Privacy.....	424
Topic B: Promote Ethical Practices.....	434
Topic C: Establish Data Privacy and Ethics Policies.....	439
Appendix A: Mapping Course Content to CertNexus® Certified Artificial Intelligence (AI) Practitioner (Exam AIP-100).....	449
Solutions.....	451
Glossary.....	467
Index.....	481

Do Not Duplicate Or Distribute

About This Course

Artificial intelligence (AI) and machine learning (ML) have become an essential part of the toolset for many organizations. When used effectively, these tools provide actionable insights that drive critical decisions and enable organizations to create exciting, new, and innovative products and services. This course shows you how to apply various approaches and algorithms to solve business problems through AI and ML, follow a methodical workflow to develop sound solutions, use open source, off-the-shelf tools to develop, test, and deploy those solutions, and ensure that they protect the privacy of users.

Course Description

Target Student

The skills covered in this course converge on three areas—software development, applied math and statistics, and business analysis. Target students for this course may be strong in one or two of these areas and looking to round out their skills in the other areas so they can apply artificial intelligence (AI) systems, particularly machine learning models, to business problems.

So the target student may be a programmer looking to develop additional skills to apply machine learning algorithms to business problems, or a data analyst who already has strong skills in applying math and statistics to business problems, but is looking to develop technology skills related to machine learning.

A typical student in this course should have several years of experience with computing technology, including some aptitude in computer programming.

This course is also designed to assist students in preparing for the CertNexus® Certified Artificial Intelligence (AI) Practitioner (Exam AIP-110) certification.

Course Prerequisites

To ensure your success in this course, you should have at least a high-level understanding of fundamental AI concepts, including, but not limited to: machine learning, supervised learning, unsupervised learning, artificial neural networks, computer vision, and natural language processing. You can obtain this level of knowledge by taking the CertNexus *AIBIZ™ (Exam AIZ-110)* course.

You should also have experience working with databases and high-level programming languages such as Python, Java, or C/C++. You can obtain this level of skills and knowledge by taking the following Logical Operations courses:

- *Database Design: A Modern Approach*
- *Python® Programming: Introduction*
- *Python® Programming: Advanced*

Course Objectives

In this course, you will implement AI techniques in order to solve business problems.

You will:

- Specify a general approach to solve a given business problem that uses applied AI and ML.
- Collect and refine a dataset to prepare it for training and testing.
- Train and tune a machine learning model.
- Finalize a machine learning model and present the results to the appropriate audience.
- Build linear regression models.
- Build classification models.
- Build clustering models.
- Build decision trees and random forests.
- Build support-vector machines (SVMs).
- Build artificial neural networks (ANNs).
- Promote data privacy and ethical practices within AI and ML projects.

The CHOICE Home Screen

Logon and access information for your CHOICE environment will be provided with your class experience. The CHOICE platform is your entry point to the CHOICE learning experience, of which this course manual is only one part.

On the CHOICE Home screen, you can access the CHOICE Course screens for your specific courses. Visit the CHOICE Course screen both during and after class to make use of the world of support and instructional resources that make up the CHOICE experience.

Each CHOICE Course screen will give you access to the following resources:

- **Classroom:** A link to your training provider's classroom environment.
- **eBook:** An interactive electronic version of the printed book for your course.
- **Files:** Any course files available to download.
- **Checklists:** Step-by-step procedures and general guidelines you can use as a reference during and after class.
- **Spotlights:** Brief animated videos that enhance and extend the classroom learning experience.
- **Assessment:** A course assessment for your self-assessment of the course content.
- Social media resources that enable you to collaborate with others in the learning community using professional communications sites such as LinkedIn or microblogging tools such as Twitter.

Depending on the nature of your course and the components chosen by your learning provider, the CHOICE Course screen may also include access to elements such as:

- LogicalLABs, a virtual technical environment for your course.
- Various partner resources related to the courseware.
- Related certifications or credentials.
- A link to your training provider's website.
- Notices from the CHOICE administrator.
- Newsletters and other communications from your learning provider.
- Mentoring services.

Visit your CHOICE Home screen often to connect, communicate, and extend your learning experience!

How To Use This Book

As You Learn

This book is divided into lessons and topics, covering a subject or a set of related subjects. In most cases, lessons are arranged in order of increasing proficiency.

The results-oriented topics include relevant and supporting information you need to master the content. Each topic has various types of activities designed to enable you to solidify your understanding of the informational material presented in the course. Information is provided for reference and reflection to facilitate understanding and practice.

Data files for various activities as well as other supporting files for the course are available by download from the CHOICE Course screen. In addition to sample data for the course exercises, the course files may contain media components to enhance your learning and additional reference materials for use both during and after the course.

Checklists of procedures and guidelines can be used during class and as after-class references when you're back on the job and need to refresh your understanding.

At the back of the book, you will find a glossary of the definitions of the terms and concepts used throughout the course. You will also find an index to assist in locating information within the instructional components of the book. In many electronic versions of the book, you can click links on key words in the content to move to the associated glossary definition, and on page references in the index to move to that term in the content. To return to the previous location in the document after clicking a link, use the appropriate functionality in your PDF viewing software.

As You Review

Any method of instruction is only as effective as the time and effort you, the student, are willing to invest in it. In addition, some of the information that you learn in class may not be important to you immediately, but it may become important later. For this reason, we encourage you to spend some time reviewing the content of the course after your time in the classroom.

As a Reference

The organization and layout of this book make it an easy-to-use resource for future reference. Taking advantage of the glossary, index, and table of contents, you can use this book as a first source of definitions, background information, and summaries.

Course Icons

Watch throughout the material for the following visual cues.

Icon	Description
	A Note provides additional information, guidance, or hints about a topic or task.
	A Caution note makes you aware of places where you need to be particularly careful with your actions, settings, or decisions so that you can be sure to get the desired results of an activity or task.
	Spotlight notes show you where an associated Spotlight is particularly relevant to the content. Access Spotlights from your CHOICE Course screen.
	Checklists provide job aids you can use after class as a reference to perform skills back on the job. Access checklists from your CHOICE Course screen.
	Social notes remind you to check your CHOICE Course screen for opportunities to interact with the CHOICE community using social media.

1

Solving Business Problems Using AI and ML

Lesson Time: 2 hours, 30 minutes

Lesson Introduction

Deep learning, machine learning (ML), and other forms of artificial intelligence (AI) are on the rise in organizations that are using these technologies to inform business decisions and guide operations—often with notable results. However, it can be challenging to identify which business problems are most amenable to these technologies.

Lesson Objectives

In this lesson, you will:

- Identify appropriate applications of AI and ML within a given business situation.
- Identify and describe steps in an organized process to plan, build, and apply a machine learning model.
- Formulate a machine learning approach to solve a specific business problem.
- Select appropriate tools to solve a given machine learning problem.

TOPIC A

Identify AI and ML Solutions for Business Problems

AI and ML are not magic—though they sometimes seem to be. To identify appropriate applications for these technologies, it helps to understand not only what they are good at doing, but also their limitations.

The Data Hierarchy—Making Data Useful

Business solutions based on AI and ML commonly focus on making data *useful*. To effectively use data, all businesses must somehow be able to translate raw data into actionable intelligence that benefits the organization. This process is sometimes explained according to the Data, Information, Knowledge (DIK) hierarchy.

- **Data** is often not very useful in its raw form. In its raw form, data often has little context, and may not be of much use to the organization until it is combined with other data and organized in a meaningful way.
Typically, many different data points must be aggregated, organized, and interpreted to be of use to the organization, transforming data into information.
- **Information** is data that is useful to inform business decisions. Information has some context and has been aggregated, organized, and interpreted to be meaningful, though it may not yet provide a complete picture that points to action that must be taken in response.
When information is combined with experience and insights that inform how the information should be dealt with, it becomes knowledge—knowing what to do based on the data provided.
- **Knowledge** is actionable intelligence. For example, it may reveal that a certain pump is about to fail, and should be replaced. Or it might reveal that a patient may not respond well to a particular medication, and should be taken off that medication as soon as possible.



Note: A fourth level of the DIK hierarchy, wisdom, is often added, making this the DIKW hierarchy. Wisdom builds upon knowledge through experience over time, informing subsequent decisions.

The process of transforming raw data into actionable intelligence can be time-consuming, error-prone, and expensive. As organizations produce larger amounts of data, this process becomes even more challenging.

Big Data

Modern business operations often result in **big data**, massive quantities of data that cannot easily be translated into actionable intelligence using traditional methods and technologies. New sources of business data, such as the Internet of Things (IoT), web and mobile apps, and social media platforms, may dwarf the amounts of data coming from traditional business data sources, such as databases, business transactions, and so forth.

This data scales to become *big* due to at least three major factors, which are sometimes described by three Vs—volume, variety, and velocity.

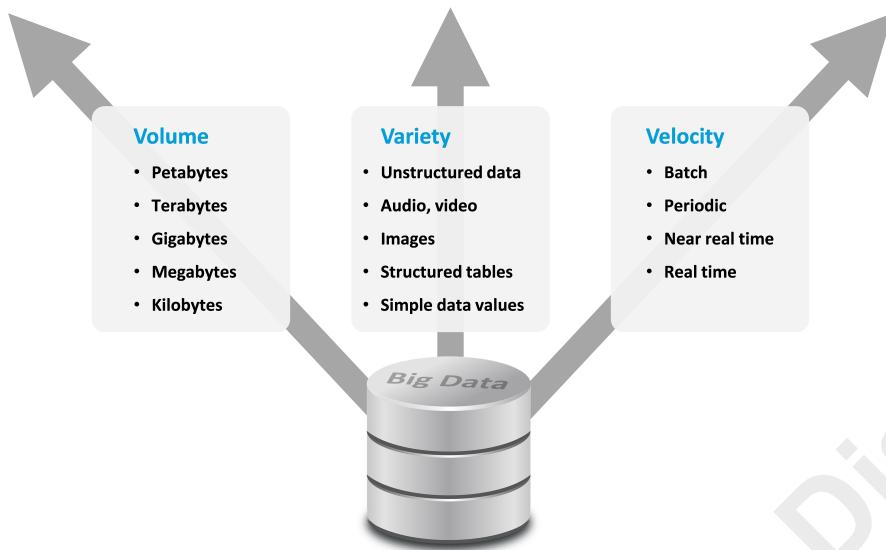


Figure 1–1: The three Vs of big data.

Factors of Big Data	Description
Volume	The sheer number of bytes of data being captured is one of the dimensions of big data. Organizations have taken advantage of the scalability offered by cloud storage, and have developed applications that accumulate massive amounts of data storage.
Variety	In the past, much of the data captured for use in data analysis came from structured data sources, like databases and transaction records. More commonly, complex and unstructured types of data are captured from sources such as ecommerce websites, social media sites, and IoT sensors. A wide variety of data types are captured, such as images, voice recordings and other audio, video, web logs, and social media posts.
Velocity	The speed at which data processing must occur. With data sources such as IoT sensors, data must often be collected and processed in real time or near real time.



Note: Veracity (the accuracy or reliability of data) is often included as a fourth V of big data's challenges. This refers to the fact that many sources of big data may not be completely reliable.

Guidelines for Working with Big Data

Follow these guidelines when working with big data in AI/ML projects.



Note: All of the Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Handle Large Datasets in Machine Learning

On projects with massive datasets, it can be challenging to store, move, and process the data. Strategies for dealing with large datasets focus on both sides of the problem—not only reducing the

amount of data you work with, but possibly also increasing the memory, storage, and computing capabilities of the systems you use to make it possible to work with more data.

- **Work with less data when possible.** A good first step is to determine if you really need to work with all of the data. Machine learning projects often involve numerous iterations, with a lot of experimentation. Often you can work with a small sample of the data pulled at random from the larger set. This will speed up your early work in the project. As you progress toward a final model, you can increase the size of the sample to refine the solution. Some machine learning algorithms support various measures that enable you to determine the minimal size of a sample set that will produce good results.
- **Use a more efficient data format.** Text-oriented data formats such as common text files or comma-separated values (CSV) are not particularly efficient. By using more efficient data formats, you may be able to use less storage, memory, and network bandwidth. Reducing precision and using different data types can reduce memory and storage requirements—for example, storing integers in 1 or 2 bytes, rather than 4 bytes. The choice of file format also affects efficiency. For example, file formats such as NetCDF (network Common Data Form) and Hierarchical Data Format (HDF) are generally more efficient than text files and are often used to store and transfer scientific data. GRIB (GRIdded Binary or General Regularly-distributed Information in Binary form) is another concise data format. It has commonly been used to store and transfer weather data.
- **Don't load all data at one time.** Depending on your machine learning environment and the algorithms you are using, it may be possible to stream or progressively load data into memory as needed, reducing the amount of memory required to process a large dataset. One possibility is using relational databases, which provide querying capabilities that enable you to extract and work with batches of data. Some machine learning tools can be configured to work directly from an SQL-based database. Machine learning algorithms that use an iterative approach to cycle through data tend to work well with this approach. However, other algorithms may not be suitable for this approach, since they require access to the entire dataset.
- **Reconfigure your system memory.** When you can't reduce the amount of data enough to efficiently process it with the systems you have, you may need to reconfigure or replace your systems. Before you consider adding hardware, first check to see whether the machine learning tools you use are configured to use all of the memory actually installed in your system. You may be able to change settings to use more memory. However, if your machine learning tools are already using all of the memory available on your system, then you may also need to consider installing more RAM.
- **Use a different system with more memory and processing power.** Your own computers may have hardware limitations that cannot be overcome simply through reconfiguration of software or memory upgrades. If that is the case, you might consider provisioning a different system with greater capacity. While buying and setting up a new system is one option, you should also consider renting storage and compute time on cloud services such as Amazon Web Services, Microsoft Azure, or Google Cloud Platform. In some situations, it may be best to use a platform designed for big data, such as Apache Hadoop with the Mahout machine learning library, or Apache Spark with the MLlib library.

Data Mining

Because of the amounts of data involved, transforming big data into actionable intelligence requires a systematic and scientific approach to ensure the process is done as efficiently as possible. **Data mining** (also called *knowledge discovery*) responds to this need by using statistical analysis methods to find meaningful patterns in data that reveal such things as **clusters** (groupings of similar records that belong together), **dependencies** (records that depend on each other for related information), and **anomalies** (records that don't seem to belong because they don't follow expected patterns). Data mining processes can be automated to some extent, leveraging the power of computers to transform big data into actionable intelligence.

Although data mining is a widely used expression, unfortunately it is often used to represent many different concepts. Even when used consistently, it may be misleading. For example, although the

goal of "silver mining" is to extract silver, the goal of "data mining" is not to extract data. Rather, the goal of data mining is to extract *knowledge*.

For example, a business might use data mining techniques on their massive stores of data to reveal knowledge such as:

- How certain environmental factors affect the failure rates of various types of machinery used in the business, providing knowledge that can improve predictive maintenance schedules.
- Factors that reveal a customer might be about to switch to a different service provider, indicating that it might be a good time to offer customer loyalty incentives, special offers, and so forth.
- How various attributes of customer data correlate with loan defaults.
- How customers' history of buying (how recent, how frequent, and purchase amounts) correlates with the types of offers they respond to.

Artificial intelligence and **machine learning** often excel at deriving meaningful information from big data. Although data mining can be performed without the use of these technologies, they provide new ways to perform data mining tasks, enabling it to be performed much faster, more efficiently, and in some cases, at lower costs than other methods.

Examples of Applied AI and ML in Business

Through data mining, natural language processing, **computer vision**, and numerous other applications, AI and ML have countless applications in business. Some examples are provided here.

Business Use	Examples
Inform business decisions	<ul style="list-style-type: none"> • Predict where and when office and call center staff, crews, fleet vehicles, and equipment are most likely to be needed to maximize utilization and productivity. • Identify product inventory that can be reduced to lower inventory carry costs.
Improve the Utilization of Staff and Equipment	<ul style="list-style-type: none"> • Process and interpret incoming help desk calls and accurately route them to the right person for problem resolution. • Make the most effective use of marketing and sales by focusing contact based on a customer's interests, browsing, and buying history.
Improve Sales and Customer Experience	<ul style="list-style-type: none"> • Recommend additional/other products based on those the customer has already shown interest in. • Provide "smart" software tools to guide staff on calls with customers to enable them to provide better service and support.
Automated Inspection	<ul style="list-style-type: none"> • Scan cast metal parts automatically as they emerge from an injection mold, and eject flawed parts into a rework bin. • Use flying drones to scan properties for forest fires, missing persons, wildlife counts, and so forth. • Inspect glass bottles in a bottling plant for cracks, and missing, misplaced, or incorrect labels, as they pass down an assembly line.
Finance	<ul style="list-style-type: none"> • Process loan applications, contracts, and other documents, providing recommendations such as loan approval, contract terms that should be reviewed, and so forth. • Price financial portfolios in overly aggressive markets by discovering latent features from the data in portfolios. • Identify risk within a portfolio based on current market trends.

Business Use	Examples
Security	<ul style="list-style-type: none"> Detect and prevent intrusions on computer networks. Sift through transactional data (call detail records, point of sale transactions, trading activity, and so forth) to identify specific activities that may be fraudulent. Use realtime security camera analysis to track the movement of people within a building to perform surveillance, identifying unusual traffic patterns or behavior, such as violent behavior, vandalism, and unaccompanied suspicious packages. Detect illegal garbage dumping or poaching on protected properties.
Identification or Classification Tasks	<ul style="list-style-type: none"> Scan IDs, license cards, paper money, and other physical artifacts to identify those that may be forged or tampered with. Scan medical diagnostic images to identify problems earlier and more consistently. Sort produce into various product grades based on multiple criteria, including color, shape, size, and so forth. Use facial recognition to unlock a smartphone, or scan security cameras in public areas for missing persons or intruders. Use machine learning to locate weeds among crops, so spraying equipment can target them, significantly reducing the herbicides used.
Process Guidance and Navigation	<ul style="list-style-type: none"> Provide visual overlays in a surgeon's field of view during a medical procedure, highlighting anatomy, the location of underlying bones, nerves, blood vessels, organs, and so forth. Use robotic delivery carts to transport mail, packages, and materials throughout a lab facility, using computer vision to avoid obstacles. Scan live imaging during surgery to guide and assist surgeons. Follow a ball or hockey puck during a broadcast sports game to provide better tracking with cameras. Guide farm equipment through the optimal route when harvesting crops.
Visual Interaction	<ul style="list-style-type: none"> Support augmented reality apps that enable users to point their mobile device camera at an object or location and view overlays with product information, pricing, coupons, guided tour narration, and so forth. Provide a virtual "try it out" feature for online shopping, where customer can see what they will look like when wearing the product, see what the product will look like in their home, and so forth. Deliver visual presentations to customers, tracking their eye movement, facial expressions, and other behavior to adjust the presentation accordingly. Frequent customers can be identified using facial recognition to refer to their past shopping history and preferences.

Guidelines to Select Appropriate Business Applications for AI and ML

Follow these guidelines when planning business solutions based on artificial intelligence and machine learning.

Identify Business Problems that Are Appropriate for AI/ML

When identifying whether a business problem is a good candidate for an AI or ML solution, consider the problem to be solved, the available data, and stakeholders' expectations. Consider using an AI or ML solution when:

- **You currently have, or can obtain sufficient data.** AI/ML technologies are data-driven. You can't use them without the right data to solve the problem. In general, the more data you have, the better. Because machine learning is driven by randomness in the data, more data samples generally lead to a better result. A single error or misleading data point is much less significant in a dataset of ten million records than it is in a dataset of ten records.
- **The problem is sufficiently self-contained or isolated from outside influence.** In other words, you can be relatively sure that the data you feed to the learning algorithm includes more or less all there is to the problem. Note that changes in business processes over time may result in new patterns of data that "break" an established machine learning solution, so it no longer functions at all or as well as it once did. Take this into account when you design solutions based on AI/ML.
- **It is acceptable for the machine to provide an answer without your being completely clear how it got that answer.** AI/ML can be very proficient in making predictions and identifying patterns in data, such as predicting when a machine part is likely to wear out, or finding other homes for sale that are similar to a house that a customer likes. Since the machine learned by example rather than being directly programmed, it may not always be clear how the machine came up with its results or recommendations, even though the answers may be quite dependable.
- **It is acceptable if the solution is not always 100% right.** AI/ML solutions are probabilistic, based on statistical processing of datasets. While they may even exceed humans in their ability to find order within a sea of data, the results they produce may not be 100% accurate, 100% of the time. If the situation allows for reasonably good answers that are mostly (but not always) right, then an AI/ML solution may be appropriate. For example, using AI, a car might provide automatic braking that stops the car safely in some situations that a human driver might miss. On the other hand, the AI might not engage the braking feature in every situation. If the solution provides greater overall safety without introducing new problems (such as braking when it shouldn't and causing an accident), then it might be considered acceptable, even though it is not 100% accurate, 100% of the time.
- **It is a learning problem, rather than just an automation problem.** AI/ML can help automate business processes, but not all automation problems require learning capabilities.

Identify Challenges of AI and ML in Business

As there is with any technology, there will be challenges associated with implementing AI/ML solutions within your organization, such as:

- **Collecting and storing essential data**—Your organization may already have all the data needed to meet the requirements you've established for an AI/ML project, but that data might not be on the right systems, in the right location, or in the right structure to be useful. Or a solution may require data that you have not collected yet. Data might be fragmented across multiple locations, and may need to be aggregated and processed before it can be used, introducing the need for new storage, networks/data pipelines, and processing systems.
- **Changing data landscape**—ML algorithms learn from the data provided to them. If the structure of the data or relationships and patterns within the data change over time, the ML algorithms need to re-learn the new patterns of data so they will continue to perform well.
- **Making informed decisions related to implementation costs**—New systems for collecting, aggregating, and processing data may introduce significant costs. Having a clear understanding of requirements (what problem you are trying to solve) will help to contain costs.
- **Using data legally and ethically**—AI/ML projects are data-driven, and that may mean increased collection and retention of data, which may contain sensitive information about customers, employees, and others. With this increased focus on data comes an increased possibility for errors in handling data in ways that are legal, ethical, and free of bias.

- **Selecting and implementing technology**—You may need to acquire numerous new tools and technologies to implement AI/ML solutions in your organization, including tools for data collection and processing, development and testing of solutions, presentation, and reporting.
- **Identifying staff with specialized skills**—Learning AI/ML algorithms and using them successfully is extremely challenging for those who do not have an advanced math background. Organizations that successfully implement AI/ML solutions typically have at least one data scientist on staff. A more challenging path involves utilizing technical professionals who understand the basics of AI/ML, perhaps consulting with data scientists on a contract basis, and using common off-the-shelf tools and machine learning algorithms.

Do Not Duplicate or Distribute

ACTIVITY 1–1

Identifying Appropriate Business Applications for AI and ML

Scenario

As an AI practitioner, you'll need to identify when AI/ML is appropriate to solve a given problem, and when it is not. In this activity, you'll consider some of the types of decisions you'll have to make.

1. You have found a dataset to use for your machine learning project, but someone on the project team doesn't think it would work well as the basis for a machine learning project.

What might lead her to such a conclusion from looking at the data?

2. You find another source of data that you could use for the project. It includes a historic database of more than three million transactions. Some of the specific data you need are not present, but it might be possible to infer what you need from the data that exists. Furthermore, some columns of data are missing as many as 5% of their data values. In this case, the statistician does not seem to think that the incomplete data is a problem, and she thinks that the dataset will actually work quite well for machine learning.

Why is this situation acceptable for a machine learning project?

3. You're working with a major online clothing retailer to enhance their online Search feature, which customers use to find articles of clothing they want to buy.

What types of improvements might be added to the Search feature through machine learning technology?

TOPIC B

Follow a Machine Learning Workflow

You have identified a project that is appropriate for machine learning, and now you need to take appropriate steps to plan, build, and apply a machine learning model.

Machine Learning Model

One of the significant products of a machine learning project is the creation of a machine learning **model**, a mathematical representation of the process or system that you need to analyze or automate in some way. For example, you might create a machine learning model to represent a problem scenario in medicine, sales, agriculture, or some other field, and then apply the model to gain insights, provide guidance, or automate the process in some way.

Following are some examples.

What the Model Represents	How the Model Might Be Applied
Attributes that influence how a person might react to a particular medical treatment	A system that helps doctors select an appropriate course of treatment
Attributes that influence product sales	A system that recommends appropriate product pricing that will help to meet sales objectives
Attributes that influence the overall health and growth of plants	A system that monitors crops and guides the application of fertilizer, soil amendments, irrigation, and so forth

Machine Learning Workflow

The process through which you create and use a machine learning model is called the **machine learning workflow**. The workflow comprises various tasks.

- Formulate the problem.
- Collect the dataset.
- Understand the dataset.
- Clean and prepare the data.
- Select, train, and tune the model.
- Apply the model and present the results.

Often, these phases overlap or may be done iteratively. For example, as you collect the dataset and get a sense of what patterns it might reveal, you may revisit and clarify how you've defined the problem. While every machine learning problem may offer its own unique challenges, being guided by a general workflow such as the one shown here will help you develop a common approach to solving any machine learning problem.



Note: The sequence of the workflow is not cast in stone. For example, you might begin by collecting and analyzing a dataset. Insights you gain from doing that might inspire you with ideas for problems you can solve by applying machine learning to the dataset. Then you might formulate the problem.

Data Science Skillset

Certain aspects of AI/ML projects tend to make them different from traditional IT or software development projects. In business, they tend to depend deeply on the quantity, quality, and type of data that is available to inform a particular business problem. Often, the impetus for a project is speculation that data already coming in to an organization might be useful in solving specific business problems or improving results.

Exploration and research may be needed to answer questions such as:

- What data is available?
- How does the data relate to the problems the organization wants to solve?
- What ML or AI method might be most appropriate to use?
- How should success be measured?

As you answer these questions, you might determine that a more traditional approach would be more efficient or cost-effective than using AI/ML, or you might confirm the benefits of using AI/ML on the project. Work in the exploration and research stage requires one to operate like a *data scientist*.

Traditional IT Skillsets

Once the research phase is completed, the method for solving the problem has been identified, and it is time to deploy the model within a long-term solution, then more traditional IT or software development skillsets may be needed. At this point in the project, it becomes necessary to answer questions such as:

- Should a data pipeline be constructed to support the model, and if it should, then how?
- How should the solution be developed and deployed to perform adequately at scale and allow for growth and future improvements?
- How should the system be maintained to keep it running optimally?

Of course, in many organizations the lines between traditional job roles (such as the merging of development and operations in a *DevOps* approach) have blurred. In some cases, there may be proficient software developers who also happen to be skilled data scientists. But even in organizations where people wear many different hats, it is important to recognize the shift in skillsets that occurs at different points in an AI/ML project, and make sure that people with the right skillsets are involved at the right time.

Some problems may require hardware you don't have, and you may have to set up new systems based on hardware configurations that are new to you, such as GPUs or computer clusters. Setting up, configuring, and managing these systems requires knowledge and skills beyond those of the traditional IT skillset. In some cases, you may find it makes more sense to provision the systems you need through cloud services, which introduces yet another skillset into the mix.

Concept Drift

Machine learning models are trained using historical data, enabling the finished model to make predictions about the future. While it is often said that "history repeats itself," this may not always hold true in machine learning. Over time, it might be more appropriate to say "the future is not what it used to be." Machine learning models may become less effective in working with new datasets due to *concept drift*.

For example, a model may predict the behavior of online shoppers, such as how they respond to advertising, coupons, or sales posted on a specific website or mobile app. However, over time, the general behavior of shoppers may change. For example, websites or apps that used to generate a lot of traffic may decline in popularity, so advertising on those sites doesn't produce the same results over time. So the effectiveness of the model may decay as the concept it is trying to predict changes with the passage of time.

You may not always be able to predict when a model will be subject to concept drift, but the possibility should always be considered when planning how models will be maintained in projects where a machine learning model is integrated into a long-term software solution.

Transfer Learning

In some cases, you may devote extensive compute time and resources to develop and refine models such as neural network models used for computer vision and natural language processing tasks. You may have to develop a new model to solve a similar problem. Or you may have to update an old model, perhaps one that is no longer functioning optimally due to concept drift.

Rather than starting over from scratch, you may begin on the new task by building on the previous model. Using a previous machine learning model as the basis for a new model is called **transfer learning**. This approach can speed up the **training** process and improve the performance of your new model.

Guidelines for Following the Machine Learning Workflow

Refer to these guidelines when developing a machine learning model.

Follow the Machine Learning Workflow

To develop a machine learning model, perform these tasks:

- **Formulate the problem**—Identify the question or problem that you're trying to address. For example, your goal might be to do a better job predicting when a particular machine part will fail, so it can be used as long as possible, but replaced before it fails.
- **Collect the dataset**—When you know what you're trying to accomplish, you will have some direction on the type of data you'll need. The data may already exist, or you may need to collect it. Different portions of the data you need may come from different sources, and you may have to aggregate it into a single dataset to make it useful.
- **Understand the dataset**—Once you have the data, you must get familiar with its structure and content, so you can determine what type of outcomes (regression, classification, etc.) you require, and whether the data will support those outcomes.
- **Clean and prepare the data**—The data may contain errors or missing values. Multiple pieces of data may be contained in a single column and may need to be extracted into separate columns. Anomalies in the recorded data may skew the results. To support your modeling, you may need to perform operations such as normalizing or standardizing the data, adding new columns, or removing unnecessary columns. As you develop and tune the model later, you may need to perform additional data cleaning and preparation tasks.
- **Select, train, and tune the model**—Once you have prepared a dataset, you can start developing an initial model for experimentation and testing. You will choose a machine learning algorithm to produce the outcome you're looking for. You may work iteratively, writing code to produce output, analyzing the output, and experimenting with various changes until you produce the outcome that you are looking for, at the quality of performance that you require. You may use various techniques to validate the model's performance, such as testing how well the model works on datasets other than the one you used to train it. You may find it necessary to change various settings or parameters to adjust how the model functions.
- **Apply the model and present the results**—When you have sufficiently tuned and tested the model, you can put it into use on data you actually need to analyze. Visualization tools, such as histograms, line graphs, and scatterplots, are often useful for presenting results, and can be very useful during the process of developing the model as well.

ACTIVITY 1–2

Planning the Machine Learning Workflow

Scenario

AI practitioners must learn a wide range of tools, principles, and procedures that enable them to use AI and machine learning technologies to solve problems. Of course, the challenges of the job extend beyond just working with technology, to working with the people and processes that you must interact with in the organizations that you support.

For example, the first step in a machine learning workflow is to express the problem to be solved as a machine learning problem. But it may not always be immediately clear how the business problems expressed by your clients translate into an AI or machine learning problem. You might be presented with a challenge such as "we need to reduce our cost to manufacture Product X." It may not be immediately obvious how machine learning can address a problem like this.

Even after you have clearly formulated a problem in terms of a machine learning solution, once you collect and examine the data that is available to you, you may find you don't have enough data (or the right data) to solve the problem.

Brainstorming problems like these with other project stakeholders, experts, and consultants is often a good place to start, and as you gain experience, you will find it easier to come up with solutions. In this activity, you will work with the class to brainstorm strategies you might use to deal with the problems described here.

-
1. How might you translate a statement like "We need to reduce our cost to manufacture Product X" into a problem that you can solve through AI/ML?

2. You have identified a business problem that you want to solve, but upon collecting and examining the data that is available to you, you think you may not have the data needed to solve the problem.

How should you proceed?

TOPIC C

Formulate a Machine Learning Problem

Before you start implementing a machine learning solution, you must identify the problem or problems you're trying to solve.

Problem Formulation

Take time to truly understand the problem you're trying to solve. Doing this step well is critical because it will guide your selection and preparation of input data and selection of machine learning algorithm, and it will ultimately enable you to determine when you have attained a model that meets your requirements.

Tom M. Mitchell, in his 1997 book *Machine Learning*, formally defined machine learning as follows: "A computer program is said to learn from experience (E) with respect to some class of tasks (T) and performance measure (P) if its performance at tasks in T, as measured by P, improves with experience E."

Since these are the essential elements of machine learning, ensure that you frame the problem to account for these elements. The following table outlines a general approach you might use.

Item	Description
Frame the problem.	<p>Write down a description of the problem you intend to solve, written clearly as though you planned to hand it off to someone else for further development. The process of having to put it into words will help you think through and clarify your intentions. Follow a pattern such as the following:</p> <ul style="list-style-type: none"> • Task (T): What the algorithm should accomplish—for example, "Predict the sale price for a house." • Experience (E): What dataset the algorithm uses to learn—for example, "Records of real estate transactions for King County for the year 2019." • Performance (P): How you will evaluate the performance of the model. How you measure this depends on the type of task the algorithm performs.

Item	Description
Identify why the problem must be solved.	<p>Aspects of this include:</p> <ul style="list-style-type: none"> • Rationale—Identifying why you need to solve this problem will help you evaluate the result, make appropriate compromises as you produce the solution, and determine when you have accomplished what you set out to do. • Benefits—Identify how the organization will benefit from the solution. This information will be useful if you need to get buy-in from stakeholders, or sell others on the need for additional resources or time to get the job done. It will also be useful to refer to when you try to determine how you will evaluate your success. • Lifetime and utilization—Identify who will use the solution and how long it will be used. This will help you determine how "polished" the solution must be, how you will need to support it, and help you consider deployment requirements if there will be other users. <p>Ensuring you are clear about these things will help to ensure you solve the right problem, in a way that will actually produce the benefits you were seeking in the first place.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  Note: It is easier to explain the justification for solving a problem when you can show how it aligns with the company's primary business objectives, which may be called <i>key performance indicators</i>. </div>
Provide background information that will help to solve the problem.	<p>As a reminder to yourself if you're developing the solution, or to provide guidance to others who will be working with you or for you, as you think through the problem, document such things as:</p> <ul style="list-style-type: none"> • Assumptions: A list of assumptions about the problem, such as: <ul style="list-style-type: none"> • Sources of data that are acceptable or unacceptable to use. • Special operating requirements, such as software environments you must use. • Business context, such as project time frames, who will use the solution, and so forth. • Reference Problems: A list of similar problems you have solved through applied machine learning. This will inform the approach you take toward implementing a solution.
Determine whether the problem is appropriate for AI/ML.	Once you have clearly defined the problem, you can explore whether it is more appropriate to be solved as an AI/ML problem or as a traditional programming problem.

ACTIVITY 1–3

Framing a Machine Learning Problem

Scenario

The CapitalR Real Estate company provides services to various independent real estate agencies throughout North America, in exchange for an annual membership fee. They have contracted with you to help develop a machine learning tool that will help agents price homes appropriately. Many factors determine the price at which a home will sell. If a home is overpriced, it might remain on the market for a long time. Even if the price is dropped, customers may ignore homes that have gone "stagnant" because they were initially overpriced. In the long run, the home may sell for a much lower price than it would have sold for if its initial offering price had been more reasonable.

On the other hand, a home that sells quickly may have been underpriced. The owner (and the salesperson, who is paid based on a percentage of the sale price) may suspect they could have gotten more for the home had they priced it higher.

For both the seller and the real estate salesperson, there are significant incentives to find the "right" price for a home so it will sell quickly enough, while maximizing the yield for the seller and the salesperson.

However, the real estate market fluctuates and is based on variables such as the local economy, time of year, public perceptions, and numerous other factors that change over time. Some customers may require a quick sale, while others are content to wait a long time for a buyer, if it means getting a better price for the home.

Considering all the data points that can influence the effectiveness of the prediction, it can be challenging to find the optimum price. Your task is to determine whether a software tool based on machine learning might provide a good solution.

1. **What sort of task should the model perform?**
2. **What sort of experience (training dataset) would you need to provide so the model could learn how to price a home?**
3. **Once you've created a prototype machine learning model, how might you evaluate the model's performance (that is, its ability to identify an optimum sales price)?**

4. Over time, after the real estate company has started using the tool, how might you evaluate whether the new tool has benefited the business?

5. Is a machine learning solution appropriate for this problem?

Differences Between Traditional Programming and Machine Learning

Programmers may use machine learning methods to implement software solutions. However, the process of developing a solution based on machine learning is different from traditional programming. Programmers will need to adjust their problem-solving approach when looking to implement machine learning solutions in their software.

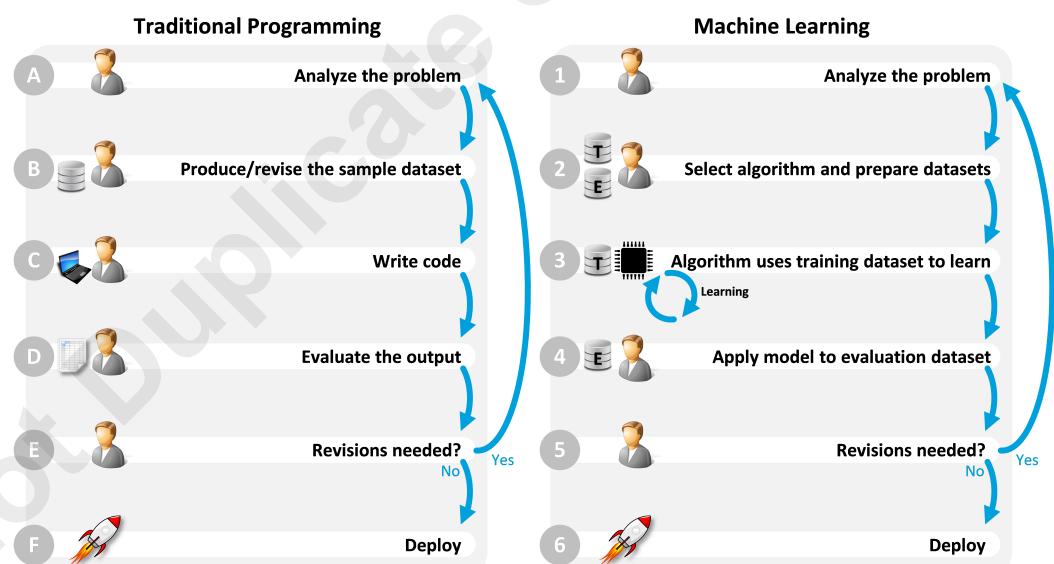


Figure 1-2: Traditional programming compared to supervised machine learning.

Both traditional programming and machine learning involve **algorithms**, a set of rules that define how data should be processed.

In *traditional programming*, the analyst examines the problem, draws conclusions about how it should be solved, and develops an algorithm specifically written to solve the problem. For example, an analyst might look at historical trends in the stock market, draw conclusions about certain patterns exhibited by those trends, and then write code to implement a specific algorithm to apply these patterns to current stock market trends in order to make predictions.

In a *machine learning* scenario, the analyst determines the type of task that needs to be performed (for example, predicting a future stock price based on current trends), then the analyst assembles an example dataset and considers what type of learning algorithm would be appropriate to perform the required task on the dataset. The analyst sets the machine in motion with an appropriate algorithm, which the machine uses to process the dataset. After the machine produces a solution, the analyst evaluates how well that solution works on a different dataset, making adjustments and having the machine retrain itself until acceptable results are achieved.

Learning algorithms typically use statistical approaches to reveal patterns within the dataset that can later be used on new datasets to make predictions. Since the dataset itself plays such a prominent role in training the machine to solve the problem, machine learning is often described as being data-driven.

Differences Between Supervised and Unsupervised Learning

People learn some tasks on their own. They learn other tasks under the supervision of an instructor. For example, the first time you washed and waxed a car, you may have learned a good waxing technique without the supervision of an instructor, through a process of trial and error. You might have started out applying wax in a large area before going back to wipe it off. When you returned to wipe off the wax, you may have found that the wax hardened so much that it was hard to remove the excess. So you experimented, applying wax in a smaller area, then going back more quickly to remove the excess. After some experimentation, making observations, and repeating the task many times, you may figure out the optimum time to let the wax harden before you wipe it off. You figure this out on your own, without the supervision of an instructor.

On the other hand, you might learn other tasks under the supervision of an instructor. For example, a grammar teacher might have taught you to recognize the difference between a subject and a verb in a sentence. The teacher provided you with example sentences and asked you to underline the subject and circle the verb. Each time, the teacher then provided you with the correct answer, so you could evaluate your performance and learn from it.

Similarly, some types of machine learning tasks are unsupervised, while others are supervised.

In supervised machine learning, you train the machine by providing a dataset for which you already have a set of correct answers, called *labels*. The machine learns by analyzing patterns within each record, and comparing them to the correct answers within the label set. In this way, the model learns by example what patterns result in a particular answer. This approach is typically used where a labeled dataset is available, and the model must make predictions based on patterns in the data or must correctly place items into an appropriate category.

Other types of machine learning tasks are unsupervised. A set of answers (labels) is not provided. Instead, the machine learns by looking for distinct patterns within the data and making observations. This approach is typically used where the goal is to reveal patterns or organization within the data, where a label set (correct answers) cannot be provided. For example, the machine might figure out how to organize related or similar items into clusters, or reveal underlying patterns or structure within the data.

Semi-Supervised Learning

Semi-supervised learning is a combination of both *supervised* and *unsupervised learning*: you typically have many unlabeled examples and some labeled examples. This is particularly useful in cases where labeling all of the data would be time consuming, cost-prohibitive, or otherwise infeasible. Labeling only some of the data may not be ideal, but it can enhance the performance of the model over standard unsupervised learning. Ultimately, the objective is to strike a balance between the overhead involved in labeling data and the actual predictive power of the model.

Randomness in Machine Learning

Machine learning is based on the mathematical fields of statistics and probability. Although statistics and probability are often mentioned together and sometimes used interchangeably, in fact they focus on slightly different matters. Statistics analyzes randomness within past events, while probability builds upon patterns identified by statistics in order to predict future events. Machine learning is heavily data-driven, and there are typically various aspects of randomness to that data—which data points are sampled, the order in which they are sampled, the samples that are used for training and testing a model, and so forth.

There is not only randomness in the data, but there may be randomness among different machine learning algorithms as well. Two different algorithms may produce similar outcomes—for example, classifying wine samples based on the wine's chemical characteristics. But they may produce slightly different results simply because the algorithms follow different steps to obtain their result. They may not only produce slightly different results, but they may have different performance characteristics as well. For example, one algorithm may take a few seconds to run, while another takes minutes. One algorithm may perform best on small datasets, while another tends to work best on large datasets.

The models produced in machine learning are often described using the term ***stochastic***. With stochastic modeling, individual data samples are inherently random and can't be perfectly predicted. But taken together, the entire set of data can be shown to follow a general pattern. By analyzing the general patterns established by the entire set, on the average you can make reasonably good predictions about individual data samples.



Note: By contrast, with deterministic models such as you might develop using traditional programming methods, randomness is external to the model. Given the same set of inputs, a deterministic model will always produce the same output, regardless of the number of times you run it.

Uncertainty

The random nature of machine learning models leads to some uncertainty, which must be managed to produce good models. To reduce uncertainty in machine learning models, you must use probability to your advantage. If different sets of data and different algorithms will produce uncertainty in the results, then getting more variety in those aspects will help you factor out uncertainty in your models. Making sure the data is clean and correct will help to make patterns in the data more apparent.

For example, you might employ strategies such as:

- Running the same algorithm many times on several different samples
- Running several different algorithms, testing the resulting models, and picking the one that produces the best results
- Running several different algorithms, allowing each one to cast its "vote" for the answer, and taking the "wisdom of the crowd" as the answer
- Selecting the right data (data that is relevant to the problem)
- Reducing noise (data values that are incorrect or misleading, making it harder for machine learning algorithms to find the important patterns in the data)

Random Number Generation

The programming code (algorithms) used in machine learning also involves some aspect of randomness. For example, some algorithms are based on a random number sequence generated by the computer. However, the random number sequences typically generated by a computer are not truly random. They are produced by a deterministic math function that returns the same sequence of numbers, which are mixed up to appear random. To provide an element of randomness to the output, computers use a seed value such as the current time (when the program runs) to seed the

random number generator. This makes the result appear to be random. But in fact, given the same seed value each time, the random number generator would always produce the same sequence of numbers. The function is sometimes called *pseudorandom* because it produces the same sequence of numbers each time it runs.

You can provide your own seed value to the random number generator, and if you provide the same value each time, it will always produce the same sequence of "random" numbers each time. This is often purposely done in machine learning (and is considered a best practice) so the algorithm will produce consistent results.

Machine Learning Outcomes

When planning your approach to a machine learning problem, begin with the end in mind. Identify the outcome you're looking for.

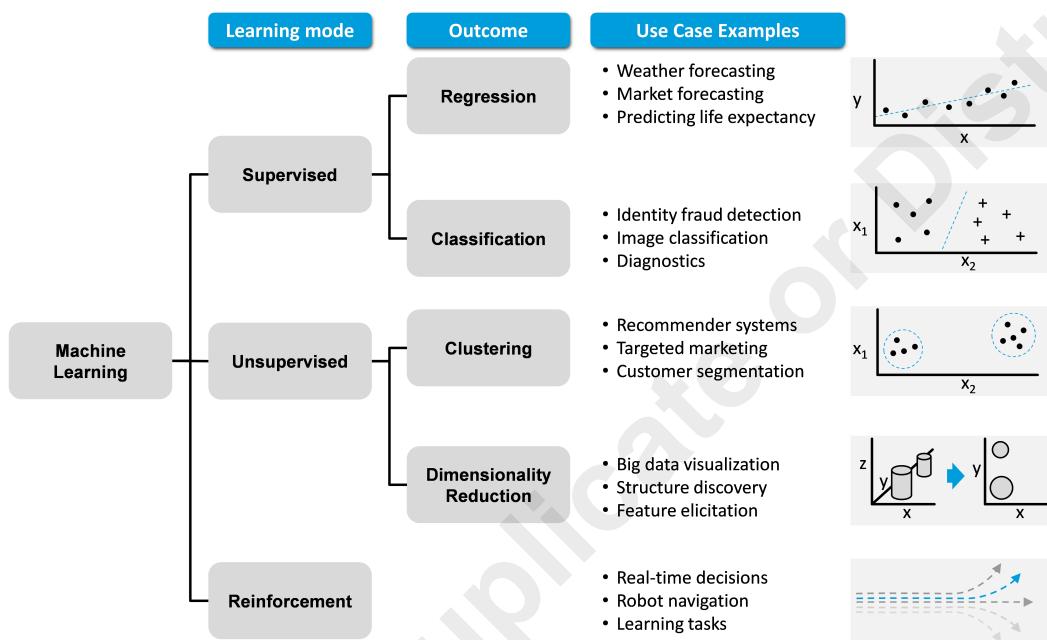


Figure 1–3: Types of machine learning algorithms, with examples of how they are used.

Common outcomes are described here.

Outcome	Description
Regression	<p><i>Given one or more input variables, predict the resulting outcome variable.</i></p> <p>A regression might be used to make a prediction. For example, based on patterns inherent in already measured data, a regression algorithm might predict the closing price of the Dow Jones Industrial Average on a given day. Or a regression might be used to determine if a value follows the expected pattern. For example, given the number of years of experience an employee has in a particular field, return a value that predicts the employee's salary.</p>

Outcome	Description
Classification	<p><i>Given a new instance of data, identify the class it belongs in.</i></p> <p>The goal may be to simply place the data instance into a pre-defined class or category. For example, given various attributes of an email message, return a value of 1 (spam) or 0 (not spam). Or the goal may be to estimate the probability that the observation belongs to various classes—75% likelihood of spam, 25% likelihood of not spam.</p>
Clustering	<p><i>Without any prior knowledge of a dataset's structure, identify components that belong grouped together.</i></p> <p>This outcome helps to reveal the composition and structure of a given set of data. Each cluster is characterized by a contained set of data points, and a cluster centroid. The cluster centroid is basically the mean (average) of all of the data points that the cluster contains, across all features.</p>
Dimensionality Reduction	<p><i>Given a machine learning model containing multiple input variables, eliminate the variables that are not required to produce the intended outcome.</i></p> <p>This outcome can be useful in a number of situations. For example, it can reduce the amount of storage and processing power required to process a dataset, and it can make datasets less confusing to the viewer when presented in charts.</p>

	<p>Note: It is quite common to mix multiple learning modes and outcomes in the course of conducting a particular machine learning workflow. For example, you might start with an unsupervised algorithm to select the minimal set of required features (columns) in your training dataset. Then you might employ a supervised approach, such as regression or classification, to produce the machine learning model.</p>
---	---

Guidelines for Formulating a Machine Learning Outcome

Follow these guidelines to formulate a business problem as a machine learning problem.

Formulate a Machine Learning Problem

Not all business problems can be solved through ML, and for situations where an ML solution could be used, it may not always be appropriate to do so. In those cases in which an ML solution would be appropriate, understanding the underlying business drivers will help you more clearly define and articulate the technical solution. Perform the following steps to identify business problems that can be solved through ML.

- **Describe the problem in plain language.** Having to actually write down the problem in simple, direct words requires you to think through what you really need to accomplish.
- **Identify the ideal outcome.** Regardless of the model you use, or the technical solution you come up with, identify what *business* outcome you are trying to achieve.
- **Define your success metrics.** These should still be focused on business, addressing the ideal outcome you identified earlier. How will you determine if the machine learning model fails to live up to business requirements? How will you know it has succeeded? The metrics you identify should be specific, measurable, attainable, realistic, and should specify timeframes, if appropriate.
- **Define the ideal output.** Again, having to write down a description of the desired output will help you clarify what you are really looking for. The output should be quantifiable, something that a machine learning algorithm can actually predict. For example, it might not be able to quantify a customer's degree of happiness, but it might be able to predict with some accuracy the likelihood that a customer will give the social media site a "thumbs up" or recommend a product to another customer.

- **Identify where the data will come from.** Machine learning is heavily dependent on data. In order to train a predictive model, you must be able to obtain historic data representing the outputs you want to be able to predict. Although some manipulation of existing data is possible, if you simply can't obtain the right data, you may need to reformulate your machine learning problem to use data that you *can* obtain.
- **Determine when and how the inputs and output will be used.** Practical considerations of when input data is available and when output data must be produced may have an impact on the design of your solution. For example, input data that your solution depends on may be released in batches, once a month. If a solution based on month-old data will not suffice, then you'll need to reconsider the solution.
- **Identify ways the problem might be solved without ML.** Brainstorming ways to solve the problem without using ML will help you focus on the business logic and may lead you to simpler, more elegant solutions, even if you do end up using an ML solution.
- **Formulate the problem as an ML problem.** From a business perspective, you have already identified the type of outcome you need to produce. Now consider how to express that outcome in terms of various types of outcomes a machine learning model might produce (regression, classification, clustering, and so forth). While you may eventually end up using an advanced machine learning model, simpler models are easier to think through and implement, so initially try to see if you can find a simple way to cast the problem, such as "predict the optimum selling price for a particular house" (regression) or "recommend houses that a particular buyer might want to purchase" (classification).

ACTIVITY 1–4

Selecting a Machine Learning Outcome

Scenario

IOT Company is a global manufacturing company that produces eco-friendly heating and cooling systems for large buildings, such as hotels, apartment buildings, retail stores, office buildings, and factories. Many of the products they manufacture include cast metal parts. These parts are cast at IOT Company's foundry, where they use robotic inspection cameras to scan parts coming out of molds to identify cracks, voids, and other defects that render the parts unusable. In this activity, you will consider which of the following machine learning outcomes might be used in this manufacturing scenario.

- Regression
- Classification
- Clustering
- Dimensionality reduction

1. Based on scans of parts coming out of the mold, machine learning algorithms identify which parts are acceptable for use, and which parts are defective.

What type of outcome would you look for in this example?

2. Over time, parts coming out of a mold tend to show increasingly more defects, to the point where eventually a threshold is passed. At that point, parts produced in the mold can't be used, and the mold must be cleaned. Casting faulty parts wastes production time, so the manufacturing team would like to predict when the threshold is about to be reached, so the mold can be cleaned before faulty parts are produced. A machine learning algorithm can be used to examine the quality of parts produced and predict when the threshold is about to be reached.

What type of outcome are you looking for in this example?

TOPIC D

Select Appropriate Tools

A wide range of tools are available to enable you to develop machine learning solutions.

Open Source AI Tools

Following are examples of open source tools commonly used in machine learning and other AI tasks.

Tool	Description
Programming Languages	A variety of programming languages are available for machine learning. The Python and R languages are frequently used, and a large number of support tools and software libraries have been developed in these languages to provide extensive support for manipulation of datasets, statistical analysis, machine learning algorithms, visualization, and other capabilities related to AI/ML. Prevalent languages used for general software development—such as Java, C++, C#, and so forth—can also be used for AI/ML. New languages continue to be developed and are finding significant use in AI/ML. Even very old languages traditionally used in the field of artificial intelligence, such as Lisp and Prolog, are still in use today.
Weka	While programming languages provide a very flexible environment for developing custom solutions based on machine learning, an alternative approach for some machine learning tasks is to use visual data mining and machine learning tools that do not necessarily require you to write code. Examples of open source tools in this category include Weka, Orange, KNIME, and SIMON. Note that some of the tools in this category do require some programming; however, programming is accomplished through a visual interface so you don't have to write code directly.
NumPy	This is a library for the Python programming language that enables you to create and perform mathematical calculations on large, multi-dimensional arrays (matrices). It also supports advanced mathematical functions for operating on these arrays, particularly in the domain of linear algebra. NumPy is a foundational library for machine learning using Python because it supports much more efficient computations than standard Python arrays, and NumPy arrays can store many different types of data.
SciPy	SciPy is another component of the Python machine learning stack. It builds on NumPy by offering more powerful mathematical functions, particularly those used in the field of scientific computing. It also includes some of the other machine learning components discussed in this table, including pandas and Matplotlib.
pandas	Part of SciPy, the pandas library supports data structures and data analysis functions for Python programming. In machine learning, the pandas DataFrame object type is particularly useful, as it enables you to load data into a structure similar to a spreadsheet (i.e., with rows and columns). You can also use various pandas functions to describe and manipulate data in a DataFrame.

Tool	Description
Matplotlib	Also part of SciPy, Matplotlib includes various methods for plotting data on graphs. Plotting data visually can provide you with new and useful perspectives about your data, which may influence your data workflow (and the training process itself). Matplotlib supports many different kinds of visualization techniques.
Seaborn	Seaborn extends the functionality of Matplotlib by incorporating more types of plots, as well as more sophisticated versions of Matplotlib plots. Seaborn also makes data visualization more attractive. For example, one common application of Seaborn is creating heatmaps to represent values in a matrix using colors and shading.
scikit-learn	Whereas SciPy and its modules are foundational to machine learning, scikit-learn actually implements machine learning algorithms. It provides support for fundamental supervised and unsupervised machine learning algorithms, including linear and logistic regression, decision trees, clustering, and so forth.
NLTK	The Natural Language Toolkit (NLTK) is a suite of Python libraries that support <i>natural language processing (NLP)</i> . NLTK is most commonly used as a teaching tool for linguistic theory and practical application. It provides functionality for classification, tokenization, stemming, and other language-based operations.
TensorFlow	This library supports deep learning algorithms, enabling you to quickly set up, train, and deploy artificial neural networks with large datasets. TensorFlow supports a wide variety of programming languages, particularly Python and C. TensorFlow can leverage the power of GPUs to increase training performance. A companion library called TensorBoard provides visualization tools for TensorFlow neural networks.
PyTorch	 <p>Note: The "Tensor" in "TensorFlow" refers to a type of data structure used in linear algebra and commonly used in machine learning. For now, you might think of a tensor as a special sort of multi-dimensional array.</p>
Keras	Keras is a library that acts as a high-level interface for experimenting with artificial neural networks (ANNs) used in <i>deep learning</i> . It can run on top of several different neural network technologies, including TensorFlow. Keras is particularly useful for beginners to ANNs, as well as for rapid prototyping, because it abstracts some of the discipline's more complex components.
Apache Spark MLlib	Apache Spark is a framework for cluster computing, a technique in which multiple computers work together as if they were one system. Spark includes MLlib, a machine learning library that leverages the power of cluster computing to improve the performance of machine learning techniques like classification, regression, dimensionality reduction, feature extraction, and many more. Spark MLlib also supports a DataFrame object provided by Spark SQL, similar to pandas.

Tool	Description
Jupyter Notebook	Jupyter Notebook is a web application that enables users to create, view, and share interactive notebooks—files that include live, executable code, as well as explanatory markup text. Program code in a notebook is often separated into multiple blocks, and the user can execute each block independently, sequentially, or all at once. While not specifically designed for machine learning, Jupyter Notebook is often used to teach machine learning principles through a hands-on, step-by-step instructional process. In addition to Python, Jupyter Notebook also supports code written in R and Julia.
Google Colaboratory	Google provides a free cloud service based on Jupyter Notebook that provides free GPU access. There are limits, of course, such as the amount of data and duration of sessions, but this service enables users to practice working with GPUs and develop deep learning applications using libraries such as PyTorch, TensorFlow, Keras, and OpenCV.
Anaconda	Anaconda is a cross-platform open source distribution of many Python and R data science libraries, including NumPy, SciPy, pandas, Matplotlib, Seaborn, scikit-learn, NLTK, TensorFlow, and PyTorch. It also includes a graphical user interface (GUI) called Anaconda Navigator for managing packages and running processes. Because it includes so many of the libraries that make up open source machine learning, Anaconda is often the preferred method of setting up a machine learning environment rather than installing each component individually.



Note: Anaconda is used in this course's activities.

Proprietary AI Tools

Following are examples of proprietary tools commonly used in machine learning and other AI tasks.

Tool	Description
IBM Watson	<p>Based on the system that defeated the reigning champions at <i>Jeopardy!</i>, IBM has since sold Watson as a software solution for companies that are interested in incorporating the power of AI in their applications. Watson is not just one service, but a whole suite. Example products in this suite include:</p> <ul style="list-style-type: none"> • Watson OpenScale—an enterprise-grade AI project management platform. • Watson Assistant—a virtual assistant like Apple's Siri or Amazon's Alexa. • Watson Machine Learning—a platform for training a variety of machine learning models. • Natural Language Understanding—a text analysis service. • Natural Language Classifier—a language classification service. • Speech to Text and Text to Speech—services for intelligently converting between both text and speech. • Visual Recognition—an image and video classification service.

Tool	Description
AWS AI	<p>Amazon Web Services (AWS) is a cloud computing provider that offers many different services. AWS provides these services to customers on demand, meaning that AWS makes its cloud computing infrastructure and resources available to customers when needed. This is a popular option for organizations, as they can offload some or all of their IT operations—including AI operations—to a third party. Examples of services that are included in the AI category of AWS include:</p> <ul style="list-style-type: none"> • Amazon SageMaker—a managed machine learning and deep learning service that supports frameworks like PyTorch and TensorFlow. • Amazon Forecast—a machine learning service for predictive modeling. • Amazon Comprehend—a text analysis service. • Amazon Lex—a service for building chatbots. • Amazon Polly—an intelligent text-to-speech converter. • Amazon Transcribe—an intelligent speech-to-text converter. • Amazon Rekognition—a service for analyzing images and video. • AWS DeepRacer—an autonomous toy race car and accompanying cloud environment for training the race car using reinforcement learning. This is primarily a teaching tool for those new to machine learning.
Microsoft Azure AI	<p>Like AWS, Microsoft Azure is an on-demand cloud computing service. While AWS has held the market share in this industry for several years, as of 2019, Azure is starting to rival AWS. Some services that are part of the Azure AI platform include:</p> <ul style="list-style-type: none"> • Azure Machine Learning—a managed machine learning and deep learning service that supports frameworks like scikit-learn, PyTorch, and TensorFlow. • Azure Databricks—a service that integrates with Apache Spark for cluster computing, while supporting many of the open source machine learning libraries discussed earlier. • Cognitive Services—a collection of NLP and computer vision services for understanding and processing speech and text, as well as identifying and captioning images and video. • Azure Search—a service that leverages NLP and computer vision to provide AI-powered searching. • Azure Bot Service—a service for building chatbots.

Tool	Description
Google Cloud AI	<p>Google Cloud is the third of the three major on-demand cloud computing providers. Although it doesn't have the same market share as AWS or Microsoft Azure, Google offers a strong AI portfolio from its cloud services. Examples of some of those services include:</p> <ul style="list-style-type: none"> • AI Hub—a central repository for privately sharing AI content within an organization. • AI Platform—a collection of tools that help machine learning practitioners design, build, and deploy AI projects. • Cloud AutoML—a beginner-friendly managed service for training machine learning models. Google typically offers AutoML versions of its various AI services in addition to standard APIs that include pre-trained models. • Cloud Natural Language—a collection of NLP services for understanding and processing speech and text. • Cloud Text-to-Speech and Cloud Speech-to-Text—conversion services that leverage machine learning. • Vision AI—a service for object detection and classification in images. • Video Intelligence—a service for object detection and classification in video.
MATLAB	MATLAB (matrix laboratory), developed by MathWorks, is a numerical computing environment and programming language that supports matrix manipulation, visualization, implementation of algorithms, and the creation of user interfaces. Programs developed in MATLAB can interact with programs written in other languages such as Python, C++, C#, Java, and Fortran.
Mathematica	Mathematica, developed by Wolfram Research, provides a technical computing environment and programming language that supports data science , machine learning, neural networks, image processing, and numerous other computing functions. The program includes a processing backend or kernel which users program through tools provided in the user interface, using the Wolfram Language.
Power BI	Power BI, developed by Microsoft, is a business analytics service that provides data warehouse capabilities including data preparation, data discovery, visualization, and interactive dashboards, which users can access through a local user interface called the Power BI Desktop.

Additional Google AI Tools

Google also offers AI tools outside of its cloud services platform, including:

- ML Kit—a collection of machine learning tools for mobile app developers provided on Google's Firebase platform.
- DeepDream—a program that uses neural networks to transform images, making them more "dream-like."
- Dataset Search—a website that enables you to search for datasets to use in machine learning projects.

New Tools and Technologies

New tools and technologies are constantly being added to the AI/ML toolbox. New capabilities such as quantum computing, distributed artificial intelligence, hyper-heuristics, and federated

learning have the potential to massively accelerate certain types of machine learning solutions, particularly those that simulate nature.

For the AI practitioner, new tools and technologies can be a two-edged sword. They provide additional tools to prepare datasets, solve problems, and present results. And though they may make some tasks easier, they may increase your hardware requirements and add complexity to projects.

To obtain consistent results and ensure that code will compile and run correctly, you must ensure that everyone on the team is using the same versions of software. If one person uses functions from a particular software library, other members of the team will need to ensure they have the same library installed on their development machine.

Hardware Requirements

Parallelization enables you to scale up the performance of your machine learning environment by dividing up tasks among multiple processors. This involves setting up a hardware configuration with more processors and memory, providing the right software and configuration to support them, and using machine learning algorithms that can divide computing tasks into multiple sub-tasks that can be delegated to multiple processors running in parallel.

Parallelization can significantly reduce the time needed to run a training algorithm—for example, from several weeks to just a few hours in some cases. This saves you time, making it more practical to experiment with multiple models, and enabling you to retrain models on fresh data more frequently. The addition of processing power through parallelization also makes it more practical to accurately fine-tune your models, enabling you to attain performance levels that would otherwise take too long to train.

In some cases, you can scale up adequately by adding more processors to a single computer. When scaling by relatively small amounts, a single computer may actually be faster than using multiple computers since it avoids delays introduced by networking. Of course, for massive scaling, multiple machines are needed.

GPUs vs. CPUs

The kind of processors you add is important. For deep learning tasks, **GPUs** (graphics processing units, the computer chips typically used as the core component in graphics adapters) are often better suited than **CPUs** (central processing units, typically used as the main processor in personal computers and servers). CPUs are optimized for processing small amounts of memory quickly. On the other hand, GPUs are optimized for processing large amounts of memory at one time, which is exactly the kind of task involved in processing video. This is why graphics adapters use GPUs. As with video processing, many deep learning tasks involve **matrix operations**—essentially performing operations on all data elements with a large data structure at one time, so processors optimized for processing large amounts of memory at one time can significantly improve performance.

Comparing CPUs and GPUs is a bit like comparing a sports car to a big delivery truck. The sports car can carry only a small load, but it is agile and can stop, turn around, and pick up passengers quickly. Big delivery trucks are not nearly as agile, but can handle a much larger load. Likewise, CPUs are agile, but they are designed to work with small amounts of memory at one time. They compensate for their small memory bandwidth by being able to quickly load and unload data from memory. GPUs, on the other hand, are like big delivery trucks when it comes to processing data. They have much larger memory bandwidth than CPUs, which makes them well suited for deep learning tasks, which often involve processing large batches of memory.

Of course, there is some latency in the time it takes for these large batches of memory to be accessed by the GPU. In deep learning, this is handled by using multiple GPUs in parallel. It's like having a line of big trucks loading packages at a warehouse. If there are always enough trucks parked at the loading dock, packages can be continually transported from the warehouse to the truck, without having to wait for another empty truck to be backed up to the loading dock.

GPU Platforms

Vendors such as Nvidia and AMD provide GPUs that can be used for deep learning, but when setting up a system, it's important to note that not every GPU will work for a particular task. The GPUs must be specifically supported by the algorithms you are using. Currently the combination of CUDA (Compute Unified Device Architecture) and Nvidia GPUs dominates deep learning. CUDA works with all Nvidia GPUs from the G8x series onwards, including GeForce, Quadro, and the Tesla line, and CUDA is compatible with most standard operating systems. CUDA provides good support across Nvidia's own GPUs, but GPUs by other vendors (such as AMD) are not supported by Nvidia's standard.

While AMD does not yet have the extensive support for deep learning that is available for Nvidia GPUs, AMD has developed a new high performance computing platform, called ROCm (**Radeon Open Compute platform**). This is intended to provide a common, open-source environment that can interface not only with AMD GPUs, but also Nvidia (through CUDA). This approach unifies Nvidia and AMD GPUs under a common programming language, though many programming frameworks have not yet been adapted to support it.

Cloud Platforms

Rather than purchasing GPU cards and setting up your own hardware, you can provision similar capabilities using a cloud hosting service such as Amazon AWS, Microsoft Azure, or Google TPU.

Amazon Web Services and Microsoft Azure enable you to create virtual machine instances of GPUs, which you can easily scale up and scale down as needed. However, the costs of using these services can add up quickly, so for experimentation, prototyping, and climbing the learning curve, you might actually find it more cost-effective to set up some cheap GPUs on your own local hardware, and then move your projects to cloud services only when you need to scale massively.

Google provides a product called Tensor Processing Unit (TPU), which may be more cost-efficient for some projects. TPU is essentially a set of specialized GPUs that have been packaged together specifically to support fast matrix operations. TPUs were designed with a sophisticated parallelization infrastructure. A single cloud TPU is equivalent to 4 GPUs, with a significant speed benefit over GPUs and potentially lower cost.

However, because TPUs are a more customized (or packaged) product than raw GPUs, they may not be quite as versatile or flexible. For tasks where TPUs can be used, they provide a powerful, cost-effective option. When more flexibility is needed, you can use cloud-hosted GPUs instead. By provisioning these resources from the cloud, you can freely switch between them depending on your needs. Of course, they are used a bit differently, so switching between them means learning how to use both, and taking the time to set up your projects to work with one or the other. Furthermore, developing a smooth workflow between GPUs and TPUs can be challenging, so factor this in when considering a solution that involves both.

Guidelines for Configuring a Machine Learning Toolset

Follow these guidelines when setting up a machine learning toolset.

Assemble a Machine Learning Toolset

When you set up a toolset (or "tool stack") for ML development, a wide variety of software configurations are possible. Following is a summary of the sorts of tools you might need to collect and set up when configuring a machine learning toolset.

- **Platform**—Start with the base environment in which you will perform the machine learning workflow from start to finish. This could be an integrated development environment with a programming language such as Python or R. You could use a notebook environment such as Jupyter Notebook or Google Colaboratory, which can host a programming language such as Python or R. Or you might use an application specifically geared toward data analysis with support for machine learning, such as Weka or MATLAB.

- **Core Machine Learning Capabilities**—Some platforms may come with support for machine learning built-in, but some general platforms such as programming languages may require that you add core machine learning support through broad software frameworks or libraries such as scikit-learn in Python, JSAT in Java, or the Accord framework in .NET.
- **Libraries, Modules, or Plug-ins**—Enhance the core machine learning capabilities you have added to the base platform by installing software libraries, modules, or plug-ins that support specific capabilities you require, such as the ability to import data in specific file formats, manipulate data, use specific machine learning algorithms, generate specific types of charts and visualizations, and so forth.
- **Integrated Peripheral Tools**—Some tasks you need to perform may be supported by separate desktop, network, or cloud-based applications. For example, you may need to pull data down from cloud services to feed the machine learning workflow. When setting up your toolset, you may need to find out how to share data between tools. In some cases, you may be able to accomplish this through settings in a dialog box (typing in the URL where the remote data can be found, for example). In other cases, you may need to write code to perform such tasks.



Note: Some products, like the Anaconda Python distribution, already include many of the components you'll need, which can save you considerable time setting up and integrating a development environment.

Select a GPU Platform

When deciding where to deploy a GPU platform, consider the following points.

- **For experimentation, prototyping, and learning**—Use a small number of local GPUs since the costs of cloud-based GPUs can add up quickly.
- **Straightforward matrix operations in the cloud**—For typical deep learning matrix operations in supported frameworks such as TensorFlow, consider using TPUs to keep your costs down. Example tasks include training object recognition or transformer models.
- **Other matrix operations in the cloud**—Use cloud-hosted GPUs like those offered by Amazon AWS or Microsoft Azure.



Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Install Anaconda.

ACTIVITY 1–5

Selecting a Machine Learning Toolset

Scenario

Before you start developing machine learning models, you must have a system to work on. While there are ways to work completely on systems hosted in the cloud, another option is to install software on your own computers. You have many options when setting up your own machine learning software stack, so at some point you'll need to determine what software you intend to use.

1. You need to load a data file in Python so you can manipulate the columns and rows of data it contains.

Which of these tools would enable you to do this?

- Matplotlib
- TensorFlow
- pandas
- Seaborn

2. Which software library implements machine learning algorithms?

- scikit-learn
- Seaborn
- Matplotlib
- SciPy

3. What capabilities does Anaconda provide for a machine learning project?

- The ability to process natural language
- The ability to load and process data organized in columns and rows
- The ability to plot charts
- All of these capabilities

4. Explore machine learning tools on GitHub.

- a) In your web browser, open the website at github.com

GitHub is a popular hosting site for software developers. This site hosts various machine learning tools, which you can download.

- b) In the search box on the GitHub website, enter the search term ***machine learning***, and examine the results.

Thousands of projects are listed, including curated lists, e-books, data examples, code, and software tools. Sites like GitHub can be a great resource for your continued learning, although browsing through the listings can be time consuming.

- c) In the search box, enter the search term ***scikit-learn***, and examine the results.

Thousands of projects related to scikit-learn are returned, including the repository for the actual scikit-learn software library for Python, which is named scikit-learn/scikit-learn.

- d) Select the **scikit-learn/scikit-learn** link.

The file repository for the project is shown. From this page you *could* download the project files directly. However, depending on the system you are using, there may be better ways to perform the installation.

- e) Scroll down to see the **Read Me** section.

The screenshot shows the GitHub README page for the scikit-learn project. At the top, it displays build status from Azure Pipelines (succeeded), build (passing), codecov (97%), circleCI (passing), Python 3.5, and PyPI package (0.21.3). Below this, the project name "scikit-learn" is displayed. The page contains the following sections:

- scikit-learn**: A brief description stating it's a Python module for machine learning built on top of SciPy and is distributed under the 3-Clause BSD license.
- Installation**: A section detailing dependencies and installation methods using pip or conda.
- Dependencies**: A list of required packages:
 - Python (>= 3.5)
 - NumPy (>= 1.11.0)
 - SciPy (>= 0.17.0)
 - joblib (>= 0.11)
- Scikit-learn 0.20 was the last version to support Python 2.7 and Python 3.4.** It notes that scikit-learn 0.21 and later require Python 3.5 or newer.
- Scikit-learn plotting capabilities**: Functions start with "plot_" and classes end with "Display". They require Matplotlib (>= 1.5.1). For running examples, Matplotlib >= 1.5.1 is required. A few examples require scikit-image >= 0.12.3, and a few require pandas >= 0.18.0.
- User installation**: Instructions for using pip or conda.

At the bottom, it says "The documentation includes more detailed [installation instructions](#)".

- Tools hosted on GitHub typically provide a Read Me file that explains the general purpose of the tool. For example, scikit-learn is a Python module that supports machine learning.
- Typically, the Read Me file also identifies where you can find the project's website, documentation, important notes, and so forth.
- The Read Me file may also identify how to perform installation on different platforms. For example, you can use the Python `pip` command to add scikit-learn to a Python setup, or you could use the Anaconda `conda` command to install scikit-learn on a computer that has Anaconda installed.

5. Examine a machine learning software bundle.

- a) In the web browser, open the website at www.anaconda.com

- b) Select the **Download** link and examine the description of Anaconda.

The screenshot shows the Anaconda Distribution website. At the top, it says "Anaconda Distribution" and "The World's Most Popular Python/R Data Science Platform". There is a "Download" button. Below the heading, there is a list of features and a grid of icons for various libraries like Jupyter, Spyder, NumPy, SciPy, Numba, pandas, DASK, Bokeh, Holoviews, Matplotlib, Levenshtein, H2O.ai, TensorFlow, and CONDA. At the bottom, there are links for Windows, macOS, and Linux. The main content area is titled "Anaconda 2019.10 for Windows Installer" and shows two options: "Python 3.7 version" and "Python 2.7 version", each with a "Download" button and file sizes (462 MB and 410 MB for Python 3.7; 413 MB and 356 MB for Python 2.7).

Anaconda is a software bundle that supports both the Python and R programming languages. It includes many of the most common machine learning libraries. Starting with a distribution that has been pre-configured for machine learning can save you significant time researching and installing software libraries.

- c) Close the web browser.

Summary

In this lesson, you learned how to integrate AI and machine learning into business processes. You considered ways that these technologies can be applied to solve business problems, and you started to explore the machine learning workflow—the overall process through which a model is produced, from concept to implementation. You also examined various sources for obtaining tools used in AI and machine learning.

What sorts of business problems do you expect to solve using AI and machine learning solutions?

Have you already selected software tools that you intend to use in your machine learning stack? If so, what tools are you using?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

2

Collecting and Refining the Dataset

Lesson Time: 4 hours

Lesson Introduction

Once you have identified the problem you're trying to solve, you'll have to find datasets you can use and determine how the data must be prepared so it functions well with your machine learning model.

Lesson Objectives

In this lesson, you will:

- Collect and prepare a dataset to use for training and testing a machine learning model.
- Analyze a dataset to gain insights.
- Use visualizations such as histograms, scatterplots, and maps to analyze data.
- Clean data in preparation for use in machine learning.

TOPIC A

Collect the Dataset

There are many factors to consider when you collect and prepare data that you will use to train and test a machine learning model.

Machine Learning Datasets

The datasets for machine learning projects come from a wide range of sources. Depending on the business requirements driving a machine learning project, you may only need to collect data once, or you may need to collect it on a recurring basis.

For example, a particular business problem may require that you obtain a historic dataset to perform some analysis that you publish in a report. You might need to use the dataset once, to obtain the answer to a particular question. You might use a dataset produced by your own organization—for example, manufacturing production records, business databases, surveys, or marketing records. Or you might use data that you have obtained from an external source, such as medical research data, census reports, or public records that you have downloaded from a government website.

On the other hand, the situation may require you to create an automated machine learning solution that collects a new batch of data on a regular, perhaps frequent basis. For example, your machine learning solution might operate on data from IoT sensors, sales transaction records, system logs, or other datasets that are continually added to. The data might originate from a combination of different sources, which may provide data in different formats, such as JSON, CSV, XML, database/SQl, and so forth.

Many machine learning algorithms have been designed to work iteratively, with an entire dataset, so it is common for machine learning datasets to be contained in flat (non-relational or denormalized) data sources, so time-consuming join operations don't have to be performed.



Note: In situations where you only need to download and prepare the data once, it may be sufficient to manually download and prepare the data, but in situations where the dataset must be collected and prepared on a recurrent basis, it is beneficial to develop scripts to automate the processes that download, convert, and combine the various data components coming in from various sources.

Structure of Data

Data used in machine learning workflows may be structured or unstructured. **Structured data** is in a format that facilitates searching, filtering, or extracting data—such as a spreadsheet or database, in which data categories are separated and/or labeled. Specific chunks of data (such as height, age, first name, last name) can be easily retrieved for any record using programming code or a querying language such as SQL (structured query language).

Unstructured data, on the other hand, is not as easy to query. Examples include such things as images, video, or audio files, data posted on social media sites, the content of email documents, and so forth. Information in such formats is not necessarily recorded in neat, pre-defined containers like a spreadsheet or database. Nonetheless, unstructured data is often a very important source of information in machine learning, and often represents a much larger portion of a business's data than structured data.

Some data may be considered **semi-structured**. For example, while the content of email data might be unstructured, the email documents themselves do contain some structure. For example, fields associated with the sender, recipient, send date, and so forth provide structured data that can be directly searched, filtered, and extracted.

Some data sources (such as XML and JSON documents) may be treated as either structured or unstructured in a machine learning workflow, depending on the consistency of their structure and content. In some situations, such as logging output from a server or application, these documents might have a very consistent structure, while in other cases, such as human-authored documents, they may be very inconsistent.

Terms Describing Portions of Data

Data sources such as databases, spreadsheets, and file formats such as CSV (comma-separated values) typically organize data within columns and rows. Columns may be called fields and rows may be called records, and the data stored within the intersection of a column and row may be called a value. In machine learning, there are additional names for these entities, and how they are used depends on context.

You may think of an **attribute** as a column that will act as input for the model. Attributes are those columns that contain the independent variables a prediction model evaluates to make its predictions. For example, in a dataset you're using to predict the price a house will sell for, attributes the model uses to predict the price might include the number of bedrooms and bathrooms. These columns are said to be among the attributes the model uses. The total number of different attributes it uses are counted to identify the model's **dimensions**.

Sometimes you may refer to the individual data value held in an attribute. This is known as a feature. For example, a particular house may have a feature of bedrooms = 5.

In practice, the words *attribute* and *feature* are often used interchangeably.

Data Quality Issues

Researchers have demonstrated that even very simple machine learning algorithms can perform as well as much more advanced and complex algorithms—if they are provided with large amounts of good data.

	Note: For more information, see https://dl.acm.org/citation.cfm?id=1073017 .
To produce a good machine learning model, it is essential to start with a good training dataset. When you select and prepare datasets for machine learning, there are numerous issues you'll have to consider regarding the quality of the dataset.	
Quality Issues	Description
Sufficient Quantity	<p>Generally, the more informative data you have, the better the model it will produce. Furthermore, the more complicated the task, the more data you will need to produce a good model.</p> <p>While having a large dataset can help to minimize the influence of a few bad data points, it is important when preparing your dataset to try to minimize any problems that would adversely influence the outcome.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  Note: One rule of thumb calls for having at least 10 times as many records as the number of input variables that the model is using. For example, if the model is making a prediction based on three columns of data as input, then at least 30 records would be needed to produce a good model. Of course, more than that would be even better. </div>

Quality Issues	Description
Relevant Features	<p>While it is critical to have sufficient quantity, you should also strive for sufficient quality. Make sure the dataset includes attributes (columns) that are relevant to the task you're trying to accomplish.</p> <p>For example, suppose you're developing a model to estimate real estate prices. The price may generally be higher for homes with more <i>bedrooms</i>, but the house's <i>location</i> may be an even stronger indicator of price than the number of bedrooms. If your data doesn't include one or more columns with data representing the house's location, you may be missing out on an important feature for predicting the price.</p> <p>On the other hand, suppose you have multiple columns representing location, such as zip code, latitude, and longitude. Since the combination of latitude and longitude is more precise in identifying location than zip code, it may produce a better prediction than zip code. Therefore you might consider zip code redundant and drop it from the training dataset.</p> <div data-bbox="674 788 754 916" style="border: 1px solid #ccc; width: 50px; height: 50px; margin-right: 10px;"></div> <p>Note: Data reduction (eliminating data that doesn't provide a positive contribution to the prediction) is an important step in preparing machine learning data, to avoid unnecessary processing and ensure the best results.</p>
Representative	<p>For each of the attributes in your dataset, the dataset should include a diverse set of data values that adequately represent those attributes during training. A good data sample will represent the entire range of possibilities you might encounter, and will not be subject to selection bias.</p>
Errors, Outliers, and Noise	<p>Specific values within a dataset may be inaccurate or faulty in some way, requiring cleanup or remediation before they are used, so they don't skew or mislead the training process. For example, a raw dataset may contain:</p> <ul style="list-style-type: none"> • Errors—Data may contain incorrect or missing values, which may influence how the model is trained. • Outliers—These are values outside the normal distribution, deviating significantly from the rest of the values in the dataset. They may be caused by errors in measurement or execution. • Noise—Some data values, columns, or records may not be necessary to make a good prediction. Even worse, they may actually hinder the process, leading a machine learning algorithm to miss important patterns in the data. Such data components are called noise because they make it difficult for the algorithm to "hear" patterns revealed by the data that is actually relevant.

Data Sources

Various terms are used to refer to the various types of repositories where data is collected. While these terms are intended to represent different concepts, they are unfortunately sometimes used interchangeably.

Data Repository	Description
Data Lake	<ul style="list-style-type: none"> Purpose: Machine learning, big data analytics, predictive analytics, and data discovery. Data may be used at any time or never at all. A specific purpose for keeping the data may not yet exist, but it is kept for possible future needs. Source: Structured and unstructured data from a variety of sources such as sensors, websites, business apps, mobile apps, server logs, and so forth. Structure: Widely varying. Data is typically kept in its original forms, which may include non-traditional data types such as web server logs, sensor data, social network activity, text, and images. Consuming and storing such data can be expensive and difficult.
Operational Data Store	<ul style="list-style-type: none"> Purpose: Collects, aggregates, and prepares data for use in operations. May feed into a data warehouse. Source: Transactional data captured from multiple enterprise applications and other sources. Structure: Data has been structured for fast and easy access, but may require additional preparation before it can be transferred to a data warehouse.
Data Warehouse	<ul style="list-style-type: none"> Purpose: Business intelligence, batch reporting, data visualization. Source: Relational data captured from multiple, relational sources including applications, transactional systems, operations databases. Structure: Data has been structured for fast and easy access.
Data Mart	<ul style="list-style-type: none"> Purpose: Data used by a particular department or business function to support analytics. Source: A subsection of the data warehouse, housing data specifically intended to support a particular department or business function. Structure: Data has been structured for fast and easy access.
Relational Database	<ul style="list-style-type: none"> Purpose: Data supporting applications, typically for transaction processing and record keeping. Source: Data captured from a single source, such as a transactional system. Structure: Organized, consistent, and normalized. Data has been structured for fast and easy access.

Open Datasets

Assembling massive datasets for use in AI and ML can be a laborious task, but it is often worth the effort because of the insights that can be gained through using them. Researchers have long recognized the value these datasets provide for research, and many feel that sharing such data across the research community ultimately benefits everyone.

To make datasets freely available to anyone to use and republish as they wish (without restrictions from copyright, patents, or other control mechanisms), researchers commonly share their datasets through open source data sharing sites. Hundreds of such sites are accessible over the web.

Following are some examples.

Item	Description
University of California Irvine Machine Learning Repository	The University of California Irvine School of Information and Computer Science hosts this repository that includes hundreds of datasets, many of which are cleaned and ready to be used. Datasets are classified by the type of machine learning problem. https://archive.ics.uci.edu/ml/index.php
Kaggle Datasets	Google hosts this online repository and community of data scientists and machine learners. Each dataset functions as a community site where you can discuss data, find and share public code, create your own projects in notebooks, and read and post blog articles. In Kaggle competitions, which regularly attract more than a thousand teams and individual competitors, companies post problems and machine learners compete to build the best algorithm. Researchers have published papers in peer-reviewed journals based on their performance in Kaggle competitions. https://www.kaggle.com/datasets
Amazon Datasets	Amazon Web Services (AWS) hosts a registry of open datasets covering many different fields such as satellite imagery, web crawler data, public transportation, bird migrations, and so forth. A search feature is provided to help you find the dataset you are looking for. The directory provides detailed dataset descriptions and project examples. This site is particularly convenient if you are using AWS. Data transfer will be very quick if you are using AWS for machine learning, since they are hosted on AWS storage services. https://registry.opendata.aws/
Microsoft Research Open Data	Microsoft Research, the research subsidiary of Microsoft, hosts this repository that provides a collection of free datasets covering fields such as natural language processing, computer vision, and domain specific sciences. This site is particularly convenient if you are using Microsoft's cloud services. You can download or copy datasets directly to an Azure cloud-based Data Science Virtual Machine. https://msropendata.com/
Open Media Library (OpenML)	OpenML is an online experiment database for machine learning, which hosts tens of thousands of open source datasets. Resources are categorized as datasets (rows of data in table form), tasks (a dataset, together with an example machine learning task to perform, such as classification or clustering and an evaluation method), flows (a particular machine learning algorithm from a particular library or framework such as Weka, mlr, or scikit-learn), and runs (a flow applied to a particular task). https://www.openml.org/
Government Repositories	Numerous governments share their datasets through sites such as the EU Open Data Portal (https://data.europa.eu/euodp/data/dataset), Data.gov (United States), Data.gov.uk (United Kingdom), Data.gov.in (India), and Open.canada.ca (Canada).

A variety of classic datasets (such as the Time Series, Bayesian Data Analysis, Iris Flower Dataset, MNIST Databases, and Bupa Liver Data) have been used extensively to demonstrate and practice

statistical analysis and machine learning tasks. Many of these datasets are available for download through repositories such as those listed above.



Note: Google's Dataset Search tool (<https://toolbox.google.com/datasetsearch>) enables you to find datasets available on various sites across the web.

Guidelines for Selecting a Machine Learning Dataset

Follow these guidelines when selecting the dataset you will use for a machine learning project.



Note: All of the Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Select a Machine Learning Dataset

The data you use for a machine learning project may come from a variety of different sources. Follow these guidelines when selecting the data you will use.

- **Identify what you want to predict, so you can decide what data you need to collect.** When planning to start collecting data that you will use for AI or machine learning, consider the business problems you want to solve and the outcomes you therefore need to produce (classification, clustering, regression, etc.). For example, if it's a supervised machine learning problem, you'll need to have some data that includes "correct answers" for the purpose of training the model. If it will cluster or group the data subjects, you'll need to collect data describing the various qualities that will be used to group the data subjects.
- **Consider all data sources within your organization.** Don't overlook existing data sources you may already have, such as paper records, records stored in document files (such as Excel workbooks), or data collected in software-as-a-service applications such as customer relationship management systems. With some effort, you can import data sources such as these and transform them for use in machine learning.
- **Consider using third-party data when you don't have your own data.** Some business problems can only be solved using data specific to your organization. But other problems may be generic and can be sufficiently represented using generic data. If you don't have your own data, consider using data accumulated by others, such as open source data provided by researchers, government sources, or other companies willing to share their data.
- **Establish repeatable mechanisms for collecting data.** Data fragmentation, when the data you need originates from many different sources, can make it hard to obtain the datasets needed for AI and ML, particularly when new data needs to be obtained on a regular basis. Work with the various data owners to put systems into place to optimize the standardization, collection, and aggregation of data that will be used. Automation of the data pipeline can save time and reduce the possibility of human error.
- **Collect "right data," not necessarily "big data."** While machine learning tends to produce better results when it is trained using a fairly large dataset, it is also important to have the right data—which represents an unbiased sample, contains data values in forms that machine learning algorithms can process, and so forth. In fact, overly large datasets can bog down performance, and can reduce effectiveness in some situations. Getting the data right requires not only technical know-how (experience and good judgment in regard to the data needed to support the machine learning algorithms being used), but also in regard to business knowledge (knowing what data points adequately represent the business processes being modeled).

ACTIVITY 2-1

Examining the Structure of a Machine Learning Dataset

Data File

/home/student/CAIP/Workflow/Workflow-Housing.ipynb

Scenario

You are continuing your work on the machine learning tool that CapitalR Real Estate company will use to predict an appropriate sale price for houses. You have proposed a model that will base the price on various attributes such as the size of the home, the number of bathrooms and bedrooms, location, and so forth.

The house's price will be the *output* (the dependent variable). The model will determine (*predict*) the price through multiple *inputs* (independent variables). This seems like an appropriate task for a *regression* model.

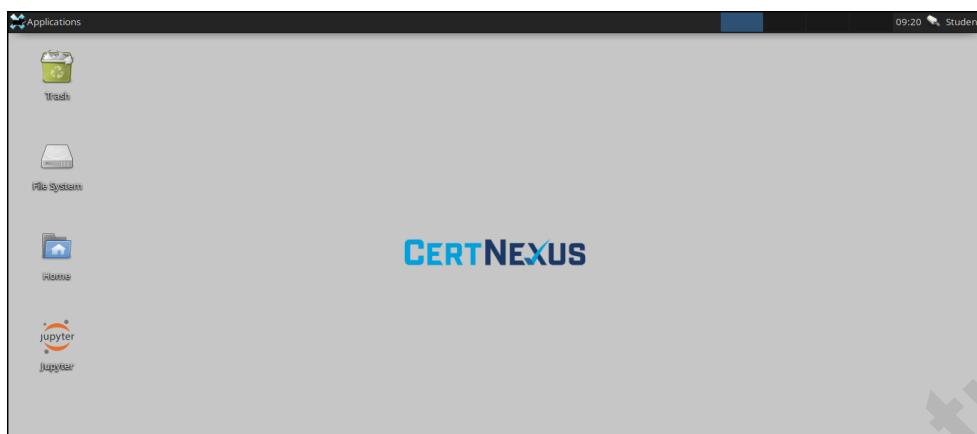
You have found a dataset you can use to train the machine learning model. It is a CSV file containing more than 20,000 real estate transactions conducted in King County, Washington. Before you start writing code to load this dataset in Python for analysis and manipulation, you will preview the contents of the data file.

	Note: You can obtain a copy of this public domain dataset at https://www.kaggle.com/harlfoxem/housesalesprediction .
	Note: The system on the VM is configured to log the user in automatically. If you are prompted at any time to log in, the account is named student and the password is Pa22w0rd .
	Note: Activities may vary slightly if the software vendor has issued digital updates. Your instructor will notify you of any changes.

1. Start the VM and launch Jupyter Notebook.

- a) Start the Oracle VM VirtualBox application.
- b) In the Oracle VM VirtualBox Manager, from the list on the left, select **CAIP** and select **Machine→Start→Normal Start**.

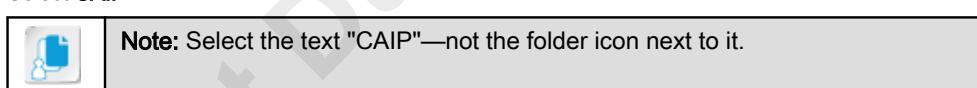
- c) Wait as the virtual machine finishes loading and the operating system starts.



- d) On the desktop, double-click the **Jupyter** icon to launch the Jupyter Notebook server and open a web client.



- The server launches first in a terminal window. Then the web browser is launched to provide the user interface for Jupyter Notebook.
 - The web client shows a listing of directories on the computer.
 - You can use this listing to navigate to folders that contain notebooks you want to open.
- e) Select **CAIP**.



The **/CAIP** directory contains various subdirectories for different notebooks and datasets.

f) Select Workflow.

The Workflow subdirectory contains a subdirectory named **housing_data** and a notebook file named **Workflow-Housing.ipynb**.



2. Examine the dataset.

a) Select **housing_data**.

The data subdirectory contains one file: **kc_house_data.csv**.

b) Select **kc_house_data.csv**.

```

1 id,date,price,bedrooms,bathrooms,sqft_living,sqft_lot,floors,waterfront,view,condition,grade,sqft_above,sqft_basement,yr_built,yr_renovated,zipcode,lat,long,sqft_living15,sqft_lot15
2 "7129300520","20141013T000000",221900,3,1,1180,5650,"1",0,0,3,7,1180,0,1955,0,"98178",47.5112,-122,257,1340,5650
3 "6414100192","20141209T000000",538000,3,2.25,2570,7242,"2",0,0,3,7,2170,400,1951,1991,"98125",47.721,-122,319,1690,7639
4 "5631500400","20150225T000000",180000,2,1,770,10000,"1",0,0,3,6,770,0,1933,0,"98028",47.7379,-122,233,2720,8062
5 "2487200875","20141209T000000",604000,4,3,1960,5000,"1",0,0,5,7,1050,910,1965,0,"98136",47.5208,-122,393,1360,5000
6 "1954460510","20150218T000000",510000,3,2,1680,8080,"1",0,0,3,8,1680,0,1987,0,"98074",47.6168,-122,045,1800,7503
7 "7237550310","20140512T000000",1,225e+006,4,4,5,5420,101930,"1",0,0,3,11,3890,1530,2001,0,"98053",47.6561,-

```

- The contents of the file are shown. This is the data you will use to train and test the machine learning model.
- The data is stored in CSV format. Values are separated by commas. Text values, which may include spaces, are enclosed within double quotes.

c) Examine the column labels in the first row.

They include:

- id**—Unique identifier for each house sold.
- date**—Date of the house's most recent sale.
- price**—Price the house most recently sold for.
- bedrooms**—Number of bedrooms in the house.
- bathrooms**—Number of bathrooms. A room with a toilet but no shower is counted as 0.5.
- sqft_living**—Square footage of the house's interior living space.
- sqft_lot**—Square footage of the lot on which the house is located.
- floors**—Number of floor levels in the house.
- waterfront**—Whether the property borders on or contains a body of water. (0 = not waterfront, 1 = waterfront)
- view**—An index from 0 to 4 representing the subjective quality of the view from the property. The higher the number, the better the view.
- condition**—An index from 1 to 5 representing the subjective condition of the property. The higher the number, the better the condition.

- **grade**—An index from 0 to 14 representing the quality of the building's construction and design. The higher the number, the better the grade.
 - **sqft_above**—The square footage of the interior housing space that is above ground level.
 - **sqft_basement**—The square footage of the interior housing space that is below ground level.
 - **yr_built**—The year the house was initially built.
 - **yr_renovated**—The year of the house's last renovation.
 - **zipcode**—What zipcode area the house is located within.
 - **lat**—Latitude of the house's location.
 - **long**—Longitude of the house's location.
 - **sqft_living15**—The square footage of interior housing living space for the nearest 15 neighbors.
 - **sqft_lot15**—The square footage of the land lots of the nearest 15 neighbors.
- d) Close the `kc_house_data.csv` browser tab.

Extract, Transform, and Load (ETL)

Even when the original data sources are clean, problems may be introduced when data from different sources are combined. For example, different databases may use different schemas to organize data. Columns may be named differently, and data values may be represented by different data types or encoding schemes. Records may overlap or contain duplicate data. You should resolve all of these problems in your data before attempting to apply machine learning algorithms.

The process used to combine data from multiple sources is often called *extract, transform, load (ETL)*.

- **Extract**—Pulls all or some of the data from various sources, which may have similar or dissimilar data structures
- **Transform**—Converts data into a proper and uniform storage format or structure
- **Load**—Inserts data into the destination where it will be stored for later modeling

Machine Learning Pipeline

In some cases, it may be tempting to perform ETL tasks by hand—using an Excel worksheet, for example, or an interactive data mining application. The rationale for doing this may be that you just plan to perform the task once. Often, however, machine learning projects intended as a quick experiment take on a life of their own, and you find you need to repeat ETL tasks that you've performed before. Machine learning often involves workflows that must be repeated over and over again, as new data is accumulated. For this reason, consider scripting your workflow from the start to make it easily repeatable and essentially self-documenting.

A data ***pipeline*** automates various phases of the process from the initial accumulation and cleanup of data from multiple sources, to the data's incorporation into machine learning models, to deployment where results such as predictions and classifications are provided, to the ultimate retirement and safe disposal of data.

Automation makes the workflow repeatable and enables you to optimize it for simplicity, speed, portability, and reusability.



Note: Even if you don't develop scripts to automate your ETL tasks, consider taking notes so you can remember exactly what you've done to the data, particularly if you might be called on to explain the results. If you automate ETL tasks through a script, the script can demonstrate the exact steps you took.

ML Software Environments

The software environments used for machine learning projects are often assembled from hundreds of different components, including a host platform (such as Linux or Windows), a programming environment (such as Python or R), online data repositories and cloud services, and software

libraries used to accomplish different sorts of tasks (such as pandas, scikit-learn, and Matplotlib). These software libraries, in particular, may have a lot of interdependencies that may affect whether your projects will run, and the results they produce. Carefully document the tools and versions your model uses, so you (or anyone who inherits your project) can reconstruct the correct environment later, if necessary.

Guidelines for Loading a Dataset

Follow these guidelines when loading datasets into a Python machine learning project.

Use Python to Load a Dataset

The pandas `DataFrame` object defines a two-dimensional labeled data structure that may contain columns of different types, similar to a spreadsheet or SQL table. It is commonly used to import structured datafiles (such as JSON or CSV) for use in machine learning or data analysis projects. The following example shows how to load a dataset from a CSV file into a pandas dataframe:

```
import pandas as pd  
my_dataframe = pd.read_csv(data_raw_file)  
num_records = len(data_raw)
```

Tasks performed by this example:

- Import the pandas library so dataframes can be used.
- To read a CSV file into a dataframe, call pandas' `read_csv()` method, passing in the combined path and file name of the CSV file.
- You can determine the number of records (rows) that were loaded by calling the `len()` function to obtain the length of the dataframe.

ACTIVITY 2-2

Loading the Dataset

Data File

/home/student/CAIP/Workflow/Workflow-Housing.ipynb

Before You Begin

You have started the Debian virtual machine in Oracle VirtualBox, and have launched Jupyter Notebook. The **CAIP/Workflow/housing_data** folder is displayed.

Scenario

You have obtained a dataset in the form of a CSV file containing tens of thousands of real estate transactions made in King County, Washington. Using Python and Jupyter Notebook, you will load this dataset into a pandas dataframe object, which you can then use to analyze and manipulate the data.

1. Run code to import various Python software libraries commonly used in machine learning.

- a) In the breadcrumb navigation area, select **Workflow** to navigate back to that directory.



Note: Alternatively, you can select the .. folder label to navigate up one directory.

- b) Select the **Workflow-Housing.ipynb** notebook label.
The **Workflow-Housing** notebook is opened in Jupyter Notebook.

- c) Examine the Python code beneath the heading **Import software libraries**.

Scroll, if necessary, to see the entire code listing.



Note: If line numbers are not showing along the left side of the code listing, then select **View→Toggle Line Numbers** to add them. While line numbers are not required to run the code, they will make it easier to discuss the code in class.

Import software libraries

```
In [ ]: 1 import sys          # Read system parameters
2 import os           # Interact with the operating system
3 import numpy as np # Work with multi-dimensional arrays and matrices
4 import pandas as pd# Manipulate and analyze data
5 import matplotlib # Create 2D charts
6 import scipy as sp  # Perform scientific computing and advanced mathematics
7 import sklearn      # Perform data mining and analysis
8 import seaborn as sb # Perform data visualization
9
10 # Summarize software libraries used
11 print('Libraries used in this project:')
12 print('- NumPy {}'.format(np.__version__))
13 print('- Pandas {}'.format(pd.__version__))
14 print('- Matplotlib {}'.format(matplotlib.__version__))
15 print('- SciPy {}'.format(sp.__version__))
16 print('- Scikit-learn {}'.format(sklearn.__version__))
17 print('- Python {}\n'.format(sys.version))
```

- The **import** statements in lines 1 through 8 enable various software libraries to be used in this program.
- Lines 11 through 17 will print the names of the different libraries, along with the version number of each library installed on the computer.

- d) Click within the code listing.

Import software libraries

```
In [ ]: 1 import sys          # Read system parameters
2 import os           # Interact with the operating system
3 import numpy as np # Work with multi-dimensional arrays and matrices
4 import pandas as pd# Manipulate and analyze data
5 import matplotlib # Create 2D charts
6 import scipy as sp  # Perform scientific computing and advanced mathematics
7 import sklearn      # Perform data mining and analysis
8 import seaborn as sb # Perform data visualization
9
10 # Summarize software libraries used
11 print('Libraries used in this project:')
12 print('- NumPy {}'.format(np.__version__))
13 print('- Pandas {}'.format(pd.__version__))
14 print('- Matplotlib {}'.format(matplotlib.__version__))
15 print('- SciPy {}'.format(sp.__version__))
16 print('- Scikit-learn {}'.format(sklearn.__version__))
17 print('- Python {}\n'.format(sys.version))
```

A border is added around the cell that contains the code listing, showing that the cell that contains the code is now selected.

- e) Select the **Run** button to run the code in the selected cell.



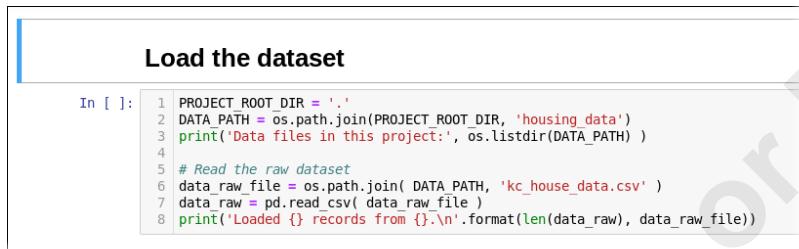
- f) Observe the output from the code you just ran.

```
Libraries used in this project:
- NumPy 1.16.2
- Pandas 0.24.2
- Matplotlib 3.0.3
- SciPy 1.2.1
- Scikit-learn 0.20.3
- Python 3.7.3 (default, Mar 27 2019, 22:11:17)
[GCC 7.3.0]
```

- Libraries used in the project are listed, along with their version numbers.
- It is important to document which tools and versions you are using, so you can reconstruct the correct environment later, if necessary. Default configurations and settings may also change over time, possibly affecting your results.

2. Load the dataset.

- a) Scroll down to view the cell titled **Load the dataset**, and examine the code listing beneath it.



```
In [ ]: 1 PROJECT_ROOT_DIR = '.'
2 DATA_PATH = os.path.join(PROJECT_ROOT_DIR, 'housing_data')
3 print('Data files in this project:', os.listdir(DATA_PATH) )
4
5 # Read the raw dataset
6 data_raw_file = os.path.join( DATA_PATH, 'kc_house_data.csv' )
7 data_raw = pd.read_csv( data_raw_file )
8 print('Loaded {} records from {}'.format(len(data_raw), data_raw_file))
```

- Lines 1 and 2 identify the location of the directory that contains the dataset.
 - Line 3 prints a list of files in the data directory.
 - Line 6 constructs a text string that includes the full path and file name of the datafile.
 - Line 7 reads data from the datafile into a pandas dataframe named `data_raw`. Once this statement executes, the dataset will be in memory, where it can be read directly from the `data_raw` variable.
 - Line 8 gets the length of (number of rows/records in) `data_raw`, and displays that number in a message.
- b) Click within the code listing, and select **Run**.
- c) Observe the output.

```
Data files in this project: ['kc_house_data.csv']
Loaded 21613 records from ./housing_data/kc_house_data.csv.
```

- 21,613 records have been loaded from the `kc_house_data.csv` file.
- These records are now loaded in the `data_raw` dataframe object, from which they can be displayed or manipulated through Python code.

TOPIC B

Analyze the Dataset to Gain Insights

You have formulated a machine learning problem, and have identified a potential dataset to use. Now you will analyze the dataset to develop ideas on how to make the best use of the information it contains as you prepare to create your initial machine learning model.

Dataset Structure

A good place to start when analyzing a dataset is to get familiar with the content and format of the various columns that the dataset contains, as well as the data type of each column.

Familiarizing yourself with the data structure will enable you to start thinking about what you can do with the data, as well as what sort of preparation you must do to prepare it for machine learning. For example, machine learning algorithms typically require (or work best with) *numeric* data. Suppose your dataset contains geographic locations in a combined latitude and longitude *string* value, such as "(43.214470, -77.938797)". To enable a machine learning algorithm to make best use of this data, you could split the string into separate numeric columns, with a `lat` value of 43.214470 and a `long` value of -77.938797.

Furthermore, your dataset might originate from multiple sources or locales, with values encoded differently. For example, one data source might represent money in Euros while another source represents it in U.S. dollars. Different character sets might be used. Different date formats might be used. To properly process data values for machine learning, you'll need to make the data values consistent with each other.



Note: UTF-8 is the most commonly supported character encoding in Python 3. If you plan to use Python 3 for machine learning, consider using UTF-8 as your common character encoding.

If you have the dataset loaded in a Python dataframe, you can call the dataframe's `info()` function to list all of the dataset's attributes and the data type of each attribute. If you would also like to examine samples of various data values within the dataset, you can call the dataframe's `head()` function to view the first five rows of data, or you can provide the function with a number value to see more rows, as in `head(10)`.

Guidelines for Exploring the Structure of a Dataset

Follow these guidelines when you are exploring the structure of a dataset loaded in a pandas dataframe.

Use Python to Summarize Data

pandas provides various functions you can call to get information about the data loaded within a dataframe. The following examples assume you have already created a dataframe named `my_dataframe`, which is loaded with data.

- `my_dataframe.info()`—Returns the number of rows and columns in the dataframe and how much memory it is currently using. For each column of data, the column name is shown along with the data type and number of rows that contain a value within that column.
- `my_dataframe.head()`—Returns a table displaying the first five rows of data in the dataframe, with each row numbered and each column labeled by name. If you provide a number between the parentheses, the function will return that number of rows.
- `my_dataframe.describe()`—Returns a table showing descriptive statistics for values in each column, representing central tendency (mean), dispersion (min, max, values at each quarter), and standard deviation.

- `pandas.option_context()`—This pandas class method enables you to change various settings used by pandas. Specify the settings you want to change by providing the name of the setting and the value. For example, `pandas.option_context('float_format', '{:.2f}'.format)` returns a context in which floating point values are output with two decimal places. Use this method in a `with` statement to set a context for subsequent pandas function calls.



Note: These guidelines provide a quick summary of various objects used in course activities. Complete documentation covering the objects mentioned in these guidelines may be found at <https://pandas.pydata.org>.

ACTIVITY 2–3

Exploring the General Structure of the Dataset

Data File

/home/student/CAIP/Workflow/Workflow-Housing.ipynb

Before You Begin

If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the **CAIP/Workflow/Workflow-Housing.ipynb** notebook. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled "Get acquainted with the dataset". Select **Cell→Run All Above**.

Scenario

Now that you have loaded the real estate dataset, you can use Python and various software libraries installed in your Python environment to get acquainted with the data. It's helpful to know what kind of information is present in each column and the data types stored in those columns. This will help you to start thinking about which columns will be useful to train your machine learning model, and whether you need to perform any sort of "cleanup" tasks before you set up the model.

1. Examine the data types recorded in the dataset.

- Scroll down to view the cell titled **Get acquainted with the dataset**, and examine the code listing beneath it.

Get acquainted with the dataset

In []: 1 print(data_raw.info()) # View features and data types

Line 1 will output information about the various data types included in the dataset.

- Select the cell that contains the code listing, and select **Run**.

- c) Observe the information about the data types used in this dataset.

Get acquainted with the dataset

```
In [3]: 1 print(data_raw.info())    # View features and data types
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
id            21613 non-null int64
date          21613 non-null object
price         21613 non-null float64
bedrooms      21613 non-null int64
bathrooms     21613 non-null float64
sqft_living   21613 non-null int64
sqft_lot       21613 non-null int64
floors        21613 non-null float64
waterfront    21613 non-null int64
view          21613 non-null int64
condition     21613 non-null int64
grade          21613 non-null int64
sqft_above     21613 non-null int64
sqft_basement 21613 non-null int64
yr_built       21613 non-null int64
yr_renovated   21613 non-null int64
zipcode        21613 non-null int64
lat            21613 non-null float64
long           21613 non-null float64
sqft_living15  21613 non-null int64
sqft_lot15     21613 non-null int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
None
```

- 21,613 records ("entries" regarding a particular house) are in the dataset.
 - Each column in the dataset is listed, along with its data type and the number of records that include a data value.
 - Five columns contain floating point number values: price, bathrooms, floors, lat, and long.
 - Fifteen columns contain integer number values: id, bedrooms, sqft_living, sqft_lot, waterfront, view, condition, grade, sqft_above, sqft_basement, yr_built, yr_renovated, zipcode, sqft_living15, and sqft_lot15.
 - One column (date) contains a date value (reported as an "object" value).
 - There are no missing data values. Each column contains 21,613 entries.
- d) Scroll down to view the cell titled **Show example records**, and examine the code listing beneath it.

Show example records

```
In [ ]: 1 # View first ten records
2 print(data_raw.head(10))
```

This call to the dataframe's head() function will display the first ten rows of data.

- e) Select the cell that contains the code listing, and select Run.

- f) Examine the first ten records in the dataset.

	<code>id</code>	<code>date</code>	<code>price</code>	<code>bedrooms</code>	<code>bathrooms</code>	<code>sqft_living</code>	\		
0	7129300520	20141013T000000	221900.0	3	1.00	1180			
1	6414100192	20141209T000000	538000.0	3	2.25	2570			
2	5631500400	20150225T000000	180000.0	2	1.00	770			
3	2487200875	20141209T000000	604000.0	4	3.00	1960			
4	1954400510	20150218T000000	510000.0	3	2.00	1680			
5	7237550310	20140512T000000	1225000.0	4	4.50	5420			
6	1321400060	20140627T000000	257500.0	3	2.25	1715			
7	2008000270	20150115T000000	291850.0	3	1.50	1060			
8	2414600126	20150415T000000	229500.0	3	1.00	1780			
9	3793500160	20150312T000000	323000.0	3	2.50	1890			
	<code>sqft_lot</code>	<code>floors</code>	<code>waterfront</code>	<code>view</code>	...	<code>grade</code>	<code>sqft_above</code>	<code>sqft_basement</code>	\
0	5650	1.0	0	0	...	7	1180	0	
1	7242	2.0	0	0	...	7	2170	400	
2	10000	1.0	0	0	...	6	770	0	
3	5000	1.0	0	0	...	7	1050	910	
4	8080	1.0	0	0	...	8	1680	0	
5	101930	1.0	0	0	...	11	3890	1530	
6	6819	2.0	0	0	...	7	1715	0	
7	9711	1.0	0	0	...	7	1060	0	
8	7470	1.0	0	0	...	7	1050	730	
9	6560	2.0	0	0	...	7	1890	0	
	<code>yr_built</code>	<code>yr_renovated</code>	<code>zipcode</code>	<code>lat</code>	<code>long</code>	<code>sqft_living15</code>	\		
0	1955	0	98178	47.5112	-122.257	1340			
1	1951	1991	98125	47.7210	-122.319	1690			
2	1933	0	98028	47.7379	-122.233	2720			
3	1965	0	98136	47.5208	-122.393	1360			
4	1987	0	98074	47.6168	-122.045	1800			
5	2001	0	98053	47.6561	-122.005	4760			
6	1995	0	98063	47.3097	-122.327	2238			
7	1963	0	98198	47.4095	-122.315	1650			
8	1960	0	98146	47.5123	-122.337	1780			
9	2003	0	98038	47.3684	-122.031	2390			
	<code>sqft_lot15</code>								
0	5650								
1	7639								
2	8062								
3	5000								
4	7503								
5	101930								
6	6819								
7	9711								
8	8113								
9	7570								

[10 rows x 21 columns]

- Because all of the columns can't fit within the page width, they wrap around to display in four chunks. But each listing includes row numbers to help you associate them with each other.
- Even in this small sample, you can see significant variation in prices, number of bedrooms and bathrooms, living space, and so forth.

2. Which attributes do you think might have an influence on price?

Normal Distribution

Normal distribution is a common starting point for discussion of probability and statistics. When you take large samples of data and plot the values in a chart where the x-axis shows the value measured and the y-axis shows the number of samples counted with a particular value, when normal probability is at play, you typically see a bell-shaped curve, such as the one shown here.

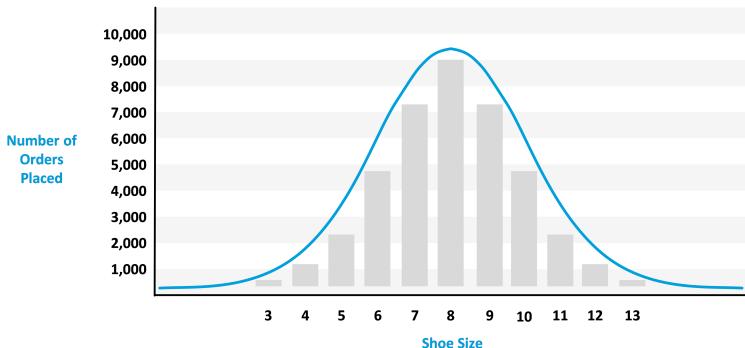


Figure 2-1: Normal distribution.

The normal distribution (also called a *Gaussian curve*) is:

- Bell-shaped
- Symmetrical
- Centered
- Unimodal (only one hump)

Non-Normal Distributions

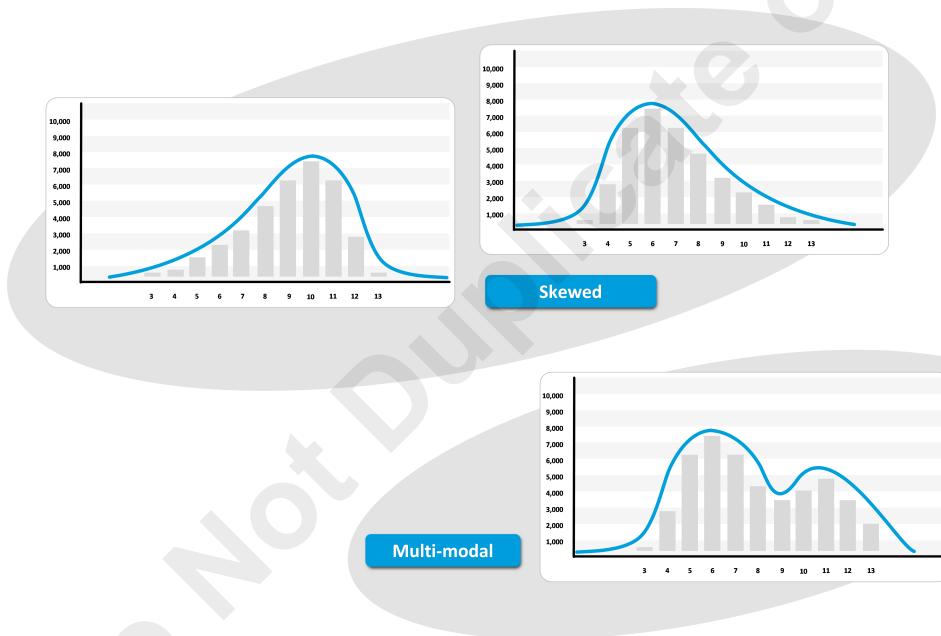


Figure 2-2: Non-normal distributions.

Of course, not every distribution is normal. There are various ways a distribution might depart from the normal curve. For example, a distribution might *skew*, with a high density of values distributed toward the lower or higher end of the x-axis, perhaps with a long tail on the other end.

While a normal distribution is *unimodal*, having only one hump in the middle, a distribution might be *bimodal* or *multimodal*—not having just one hump, but two or more.

Using probability to make predictions is foundational to machine learning, so many machine learning tasks require data samples that are normally distributed. You must be careful to avoid

applying statistical algorithms to data that is not distributed in a normal curve, since it could lead to wrong conclusions. You may have to perform certain operations on the data to "normalize" it before you can use it effectively.

Descriptive Statistical Analysis

By examining your datasets through the lens of descriptive statistics, you can evaluate data in line with the way machine learning algorithms deal with data. Descriptive statistical analysis involves various measures or descriptions you can use to summarize patterns and relationships in data, using *numbers* produced by mathematical calculations, as well as *visualizations* such as graphs or tables that help to reveal significant information in those numbers.

By looking at data the way a machine learning model "looks at" data, you can gain insights that will help you improve the dataset's usefulness in training a machine learning model. Ultimately, this will lead to better performance in the machine learning models that you create.

Concepts described through descriptive statistics include:

- Central Tendency
- Variability
- Variance
- Standard Deviation
- Modality
- Skewness
- Kurtosis

Central Tendency

Some statistical measures are concerned with what values are typical for the data. In a perfect normal distribution, the most common value is perfectly located the same distance from the right and left of the bell-shaped curve. In this case, the mean and median would be the same.

However, the distribution may not be perfect, so it's useful to have various ways to measure the center. Collectively, these are called *measures of central tendency*.

- **Mean**—This simple average of all numbers in the set is sometimes also called the *arithmetic mean*. This value is calculated by adding together all numbers in the set, and then dividing by the total count of numbers.
In the set [12, 28, 41, 51, 68], the mean is 40.
- **Median**—The number value located in the middle of a sorted set.
In the set [12, 28, 41, 51, 68], the median is 41.
- **Mode**—The number that occurs most often within a set of numbers.
In the set [12, 14, 16, 16, 16, 17], the mode is 16.

When to Use Different Measures of Central Tendency

While you might expect these three measures to produce a similar result within a normal distribution (some value near the center of the distribution), these different measures may identify very different values as the center of a non-normal distribution. The type of measure that will be most useful depends on the data within the distribution, as shown in the following table.

Measure	Most Useful For Describing
Mean	Continuous data in a symmetric distribution with no outliers
Median	Distributions that are skewed or that have outliers

Measure	Most Useful For Describing
Mode	Qualitative and categorical data—where data values are selected from a limited set such as ["for sale", "sale pending", "sold"] or ["public", "private"]

Variability

Variability is the extent to which data varies across all values in the dataset. Other words used to describe this aspect include dispersion, scatter, or spread. Comprehending the variability of a dataset includes a sense of whether the distribution is stretched out widely among a range of different values or condensed among a group of very similar values. Common measures of statistical dispersion include variance, standard deviation, and interquartile range.

Range Measures

Range is the difference between the smallest and largest values in the data. For example, suppose you are analyzing the prices of houses that have sold in a particular region. If the least expensive home sold for \$100,000, and the most expensive home sold for \$7,500,000, then the range would be \$7,400,000—the difference between the least expensive and most expensive home. Identifying the minimum and maximum values gives you a sense of where the data is spread.

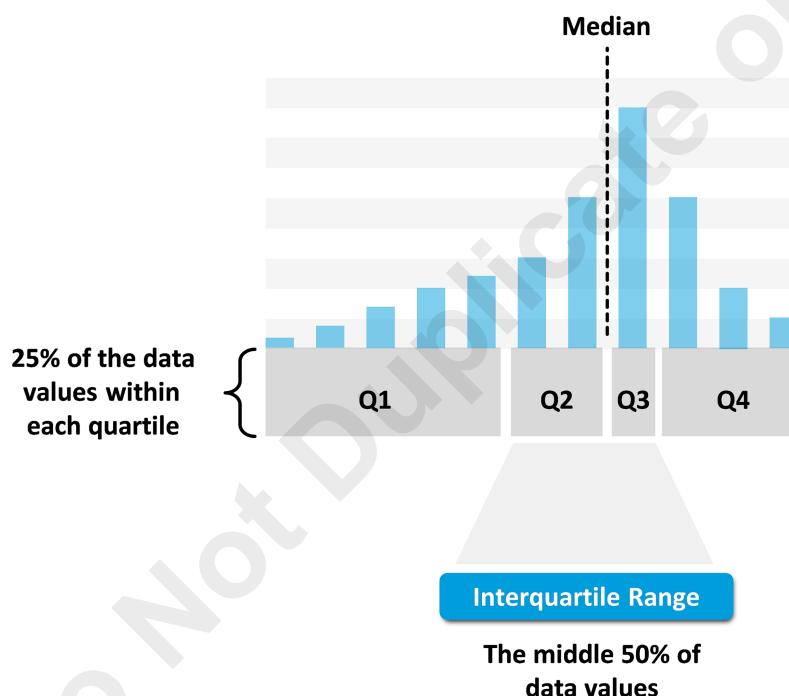


Figure 2–3: Interquartile range.

In conjunction with the minimum, maximum, and range, the central tendency measures (mean, median, and mode) start to give you a good sense of the distribution of values. Another way of measuring range, **interquartile range (IQR)**, helps to show where the majority of values lie. IQR divides the range into four subsets, where each subset contains the same number of values. The two inner sets are the interquartile range, where the middle half of data values lie.

To manually identify the four subsets, start by dividing the dataset in two at the median. Then identify the median for each of these two parts, and split them there to create the four subsets. The interquartile range occurs where 50% of the data values are located—in other words, in Q2 and Q3, as shown in the figure.

In a Python dataframe, you can call the dataframe's `describe()` function to list all of the dataset's attributes and identify which data values land on the quartile boundaries at 25%, 50%, and 75%.

Variance and Standard Deviation

When comparing the variability of data in different datasets, you might need to compare datasets containing different numbers of samples or datasets with a different central tendency. But when focusing on variability, it is necessary to have a way to measure it independent of these other measures. To be most useful, then, a measure of variability:

- *Should* represent the amount of variability (dispersion, scatter, or spread) in the data
- *Should not* represent:
 - The number of samples in the data
 - The central tendency of the data

Both **variance** and **standard deviation** (which is based on variance) meet these requirements.

Calculation of Variance

Variance is a measure of how far each value in the dataset is from the mean.

The formula for *variance in a population* is expressed as:

$$\sigma^2 = \frac{\sum(X - \mu)^2}{N}$$

Where:

- σ^2 represents variance of a population.
- Σ is the summation operator, meaning "add the following numbers."
- X is the quantity measured in the data, such as house prices or the number of bedrooms.
- μ is the mean, the average of all of the values in the population.
- N is the number of values included in the set.

Variance is calculated for a column of data by:

1. Finding the mean (average) of all values in the set.
2. For each number in the set, subtracting the mean and then squaring the difference.
3. Adding up all of the values you calculated in Step 2, and dividing by the number of values in the set to get the average.

In Step 2, the process of squaring the difference provides two benefits. It gives more weight to values farther away from the mean. It also ensures that numbers less than the mean (which produce a negative value when subtracted) won't cancel out numbers greater than the mean (positive differences). Values that are less than the mean or greater than the mean will both have a positive influence on the measure of variance.

Variance in a Sample Set

Often you must work with a **sample set**, a smaller group than the population. In such cases, you should use a slightly different formula. You should subtract 1 from the total number of values in the

sample when performing the calculation. This is called **Bessel's Correction**, and it is done to compensate for the lack of information that would be provided by the larger population.

The formula for *variance in a sample set* is expressed as:

$$s^2 = \frac{\sum(X - \bar{X})^2}{n - 1}$$

Where:

- s^2 represents variance of a sample.
- Σ is the summation operator, meaning "add the following numbers."
- X is the quantity measured in the data, such as house prices or the number of bedrooms.
- \bar{X} is the mean, the average of all of the values in the sample.
- n is the number of values included in the sample set.



Note: The n is shown in lowercase here to signify that the number of items refers to the count in a *sample* set, not the number of items in the *entire population*.

Calculation of Standard Deviation

You may wonder why variance is commonly shown as sigma *squared* (σ^2). The σ symbol represents *standard deviation*, a different measure of variability closely related to variance. In fact, standard deviation is simply the square root of variance. These two measures of variability are used for different purposes.

Variance is calculated by squaring the differences between each value and the mean. Because of this squaring operation, the resulting measure of variability may end up being much larger than the actual values in the dataset. While the resulting measure is quite useful for math operations, it is not intuitive for people trying to get a sense of the average amount of deviation in relation to the actual data values.

Standard deviation reverses the squaring operation to express the measure of variability in the same scale as the actual data values. This is helpful, since other descriptive measures such as the mean, median, mode, min, and max, are all in the same scale as the data itself. So standard deviation is often used for explaining or reporting purposes. It is easy for people to understand the amount of deviation in relationship to the scale of the actual values.

The formula for *standard deviation for an entire population* is expressed as:

$$\sigma = \sqrt{\frac{\sum(X - \mu)^2}{N}}$$

Where:

- σ represents the standard deviation of a population.
- Σ is the summation operator, meaning "add the following numbers."
- X is the quantity measured in the data, such as house prices or the number of bedrooms.
- μ is the mean, the average of all of the values in the set.
- N is the number of values included in the set.

If you have already obtained the variance, you can obtain the standard deviation simply by getting the square root of the variance. Or you can calculate standard deviation by:

1. Finding the mean (average) of all values in the set.
2. For each number in the set, subtracting the mean and then squaring the difference.
3. Adding up all of the values you calculated in Step 2, and dividing by the number of values in the set to get the average.
4. Taking the square root of the Step 3 result.

When you are working with a sample, you must adjust the formula for standard deviation like you adjust the variance formula. Thus the formula for *standard deviation for a sample* is expressed as:

$$s = \sqrt{\frac{\sum(X - \bar{X})^2}{n - 1}}$$

Where:

- s represents standard deviation of a sample.
- Σ is the summation operator, meaning "add the following numbers."
- x is the quantity measured in the data, such as house prices or the number of bedrooms.
- \bar{x} is the mean, the average of all of the values in the sample set.
- n is the number of values included in the sample set.

Skewness

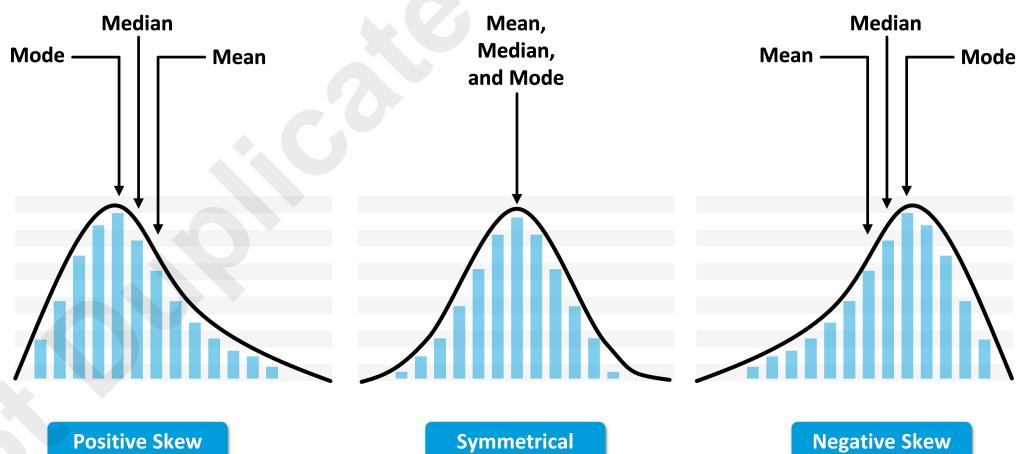


Figure 2-4: Three types of skewness.

Skewness describes the extent to which a distribution's symmetry differs from a normal distribution (zero skew, or symmetrical), either to the left (positive skew) or the right (negative skew).



Note: Positive skews tend to occur more frequently than negative skews.

There are various numerical measures of skewness, including Pearson's First Coefficient of Skewness, which is based on the mode, and Pearson's Second Coefficient of Skewness, which is based on the median. While the first method is sufficient for large datasets, for small datasets, the second method may be better because mode may not be the best measure of central tendency when there are a small number of values in the set.

Calculation of Skewness Measures

The formulas for Pearson's skewness measures are shown here.

$$Sk_1 = \frac{\bar{X} - Mo}{\sigma}$$

$$Sk_2 = \frac{3(\bar{X} - Md)}{\sigma}$$

Where:

- \bar{X} is the mean.
- σ is the standard deviation.
- Mo is the mode.
- Md is the median.

While the two formulas produce different values, you interpret the result the same way.

- The sign of the result (positive or negative) identifies the direction of skewness. A positive value means the peak is shifted to the left, and a negative value means the peak is shifted to the right. Zero means no skewness at all.
- The size of the result (how much it differs from zero) represents the degree of skewness from a normal distribution. Values farther away from zero represent more skewness.

Kurtosis

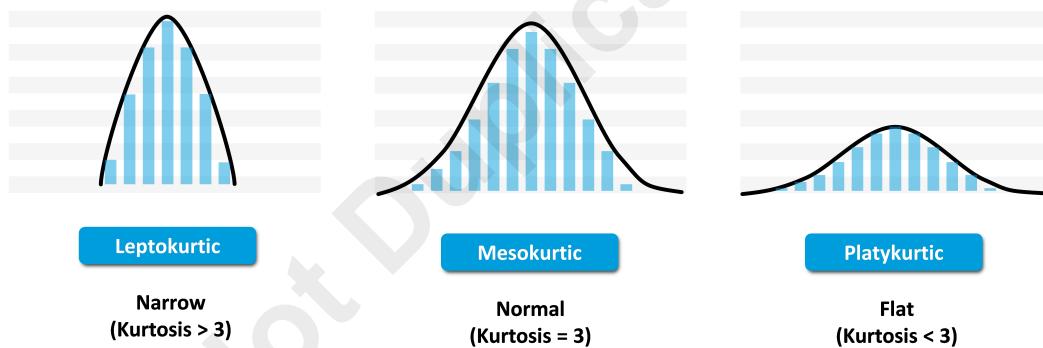


Figure 2-5: Three types of kurtosis.

In a normal bell-shaped distribution, there are tails on the left and right sides. The **kurtosis coefficient** is a measure of the shape of these tails. It measures the combined weight of the tails relative to the center of the distribution. A normal bell-shaped distribution would have a kurtosis coefficient of 3. A distribution with the classic bell shape is described as **mesokurtic**.

A distribution may be bunched to the center, dropping off quickly at the sides, resulting in very small tails. A kurtosis measurement greater than 3 would result from a small-tailed distribution, described as **leptokurtic**.

On the other hand, a distribution may be flat with extended tails, described as **platykurtic**. This sort of distribution would have a kurtosis coefficient less than 3.

Calculation of Kurtosis

Kurtosis for a sample set is defined as:

$$a_4 = \sum \frac{(X_i - \bar{X})^4}{ns^4}$$

Where:

- a_4 represents the measure of kurtosis.
- X_i is the i^{th} X value in the dataset.
- \bar{X} is the mean.
- n is the sample size.
- s is the sample standard deviation.



Note: In most machine learning situations, you can use software functions to perform this and other calculations described in this course. However, understanding how the calculation is performed will help you understand what it is measuring, so it is provided here for your reference.

For a normal (mesokurtic) distribution, this formula would produce a kurtosis coefficient of 3. Values less than 3 are platykurtic, and values greater than 3 are leptokurtic. However, to make it simpler to identify distributions that are not normal, 3 is often subtracted from the kurtosis result, producing a value called *excess kurtosis*, which shows zero for a normal distribution. Platykurtic distributions would then be negative, and leptokurtic distributions would be positive.

Distributions that are leptokurtic, with long skinny tails, may include outliers—data values that misrepresent the general patterns of data. When you are training a machine learning model, the kurtosis measurement may reveal problems in your training data that you need to address before you fit the model.



Note: If you use applications or programming functions to calculate the kurtosis coefficient, you need to find out whether the result shows *kurtosis* or *excess kurtosis*, so you can make the right judgement based on the result. Read the documentation for the function to determine what is actually calculated, as functions named *kurtosis* may actually return the value of *excess kurtosis*.

Statistical Moments

Various measures covered so far describe the characteristics of a distribution. Collectively, these measures are sometimes referred to as statistical **moments**.

- **First Moment**—The *mean*, which identifies where numbers typically lie.
- **Second Moment**—The numerator of the *variance*, which describes the spread or scale of the distribution of the normals.
- **Third Moment**—The *skewness*, which describes the symmetry of the distribution.
- **Fourth Moment**—The *kurtosis*, which describes the flatness or peakedness of the distribution.

Correlation Coefficient

Many of the algorithms used for machine learning look for correlations between data values. For example, if you were developing a model to predict the price a house would sell for, you might observe that there is a correlation between the number of bedrooms in a home and the price the home sells for. On the average, more bedrooms might mean a higher-priced home. Of course, the

correlation may not be perfect. Some 5-bedroom houses might sell for a lower price than some 3-bedroom houses. But in general, if you find that higher-priced houses tend to have more bedrooms, you could say that there is a positive correlation between bedrooms and price.

The **Pearson correlation coefficient (PCC)** provides a measure of the linear correlation between two variables commonly called x and y . It produces a value between +1 and -1 that shows the strength of their dependence on each other.

- **1** signifies the strongest possible positive correlation between x and y . As x increases, y increases. If you plot a set of data on a chart with dots showing where each value of x (on the horizontal axis) would produce a value of y (on the vertical axis), the values would all land on a straight line heading from the lower-left to the upper-right.
- **0** signifies no correlation between x and y .
- **-1** signifies the strongest possible negative correlation between x and y . As one increases, the other decreases. An x - y plot would show all values landing on a straight line heading from the upper-left to the lower-right.

 **Note:** Pearson's correlation coefficient goes by several other names. It is also called Pearson's r , r value, Pearson product-moment correlation coefficient (PPMCC), and the bivariate correlation.

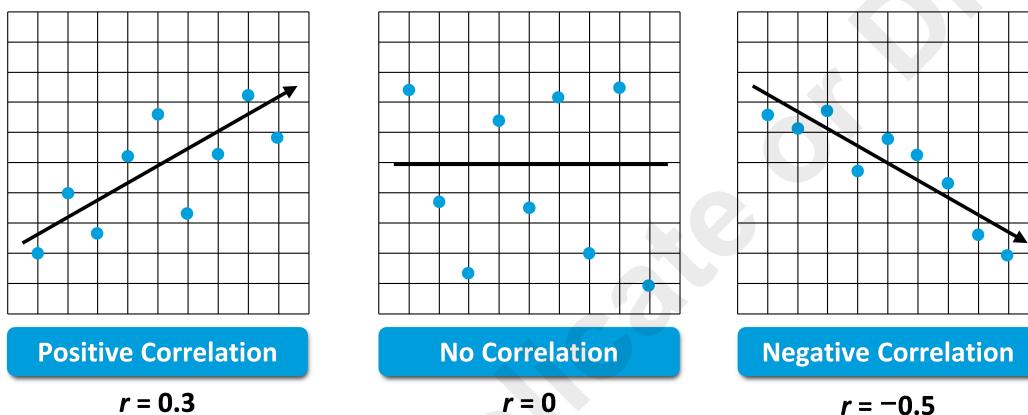


Figure 2-6: Correlations shown in an x - y plot.

The farther the correlation coefficient is from zero, the stronger its correlation, signifying that x and y are dependent variables, and that one may be a good predictor of the other.

Calculation of Pearson's Correlation Coefficient

Pearson's correlation coefficient is defined as:

$$r = \frac{n \sum xy - \sum x \sum y}{\sqrt{[n \sum x^2 - (\sum x)^2][n \sum y^2 - (\sum y)^2]}}$$

Where:

- r represents Pearson's correlation coefficient.
- n is the number of pairs.
- Σ is the summation operator.
- x and y are the possibly dependent values.

Guidelines for Analyzing a Dataset

Follow these guidelines when performing your initial analysis of a machine learning dataset.

Identify Noise within a Dataset

In machine learning, noise takes on a number of forms that cause algorithms to misidentify patterns and get bogged down in unnecessary processing, ultimately resulting in an ineffective model. When looking for noise in tables of training data, look in:

- **Specific data items**—A particular data value may be missing, or it may contain an extreme or atypical value (outlier) that does not follow the general pattern established by other data items in the same column. *Anomalous data items may need to be modified to adhere to the general pattern, or the record that contains them may need to be dropped from the training dataset.*
- **Columns**—Some of the columns you have initially selected for use in a model may have little or no correlation to the target, or may be redundant since they are dependent on other columns that are already included in the model. *Unnecessary columns may need to be dropped from the training dataset.*
- **Rows**—In some cases, the individual data items within a particular row/record may each be fine, but together they may represent an anomalous combination, one that defies the overall pattern established by data in other records. Unusual or anomalous data combinations may mislead the algorithm during training. *Anomalous rows may need to be dropped from the training dataset.*



Note: Odd combinations of data values within records may be hard to reveal by looking at individual statistical measures, and may be discovered through machine learning methods such as anomaly detection.

Reveal Outliers in a Dataset

The primary goal of some models, such as those detecting fraud or network intrusions, is to reveal outliers. For other models, however, the presence of outliers in the training dataset can skew and mislead your initial analysis, hamper training, increase training time, and diminish the model's effectiveness. To identify outliers in a training dataset:

- **Use statistics measures to reveal extreme values.** For example, calculate the standard deviation for every value in the dataset, and filter them to reveal values with a standard deviation from the mean of more than 2 or 3, which are the outliers.
- **Use visualization tools to reveal extreme values.** For example, use visualizations such as a histogram or box-and-whisker plot for distributions extending out in long, skinny (*leptokurtic*) tails, which may represent outliers in the data. In a scatterplot, look for values far away from any clusters.
- **Use machine learning algorithms to reveal outliers.** For example, the *k*-means clustering algorithm can identify natural clusters of data based on multiple values within a dataset. Identify those values with the greatest distance from any cluster as outliers.



Note: Confirm suspected outliers by removing them from the dataset, retraining the model, and testing its performance to see if their removal improves performance of the model.

Use Python to Analyze Correlations

The pandas library provides various functions you can call to get information about the data loaded within a dataframe. The following examples assume you have already created a dataframe named `my_dataframe`, which is loaded with data.

- `my_dataframe.corr()`—Returns a table showing the correlation coefficient (a floating point number) for each pair of columns in the dataframe. You can specify the type of correlation to be performed, with Pearson standard correlation being the default type.
- `my_dataframe.sort_values()`—Sorts the values along either axis. You can specify a list of names to sort by and specify whether the sort should be in ascending or descending order. The function returns a dataframe with the values sorted. You can assign the return value back to the original dataframe if you want the original to contain the sorted results.



Note: These guidelines provide a quick summary of various objects used in course activities. Complete documentation covering the objects mentioned in these guidelines may be found at <https://pandas.pydata.org>.

Do Not Duplicate Or Distribute

ACTIVITY 2–4

Analyzing a Dataset Using Statistical Measures

Data File

/home/student/CAIP/Workflow/Workflow-Housing.ipynb

Before You Begin

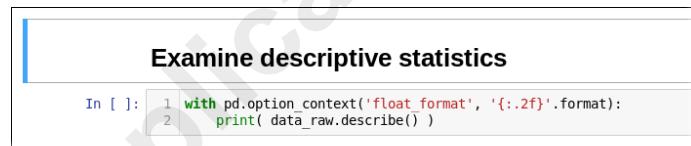
If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the **CAIP/Workflow/Workflow-Housing.ipynb** notebook. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled "Examine descriptive statistics." Select **Cell→Run All Above**.

Scenario

You have explored the general structure of the dataset and have gotten familiar with the various columns of data, including the data type of each column. Now you will examine various statistical measures as you continue considering which features may be useful to predict a house's optimum price.

1. View descriptive statistics for the dataset.

- a) Scroll down to view the cell titled **Examine descriptive statistics**, and examine the code listing beneath it.



```
In [ ]: 1 with pd.option_context('float_format', '{:.2f}'.format):  
        2     print(data_raw.describe())
```

These two lines of code work together to output a statistical description of the data contained within `data_raw`.

- Line 2 outputs a statistical description of the data contained within the `data_raw` dataframe.
 - Line 1 sets the context for the statement in line two, specifying that floating point numbers should be displayed with two decimal places.
- b) Select the cell that contains the code listing, and select **Run**.

- c) Examine the statistics describing the dataset.

```

      id      price    bedrooms   bathrooms    sqft_living    sqft_lot \
count  21613.00  21613.00  21613.00  21613.00  21613.00  21613.00
mean  4580301528.86  540688.14  3.37  2.11  2079.98  15106.97
std   2876565571.31  367127.20  0.93  0.77  918.44  41420.51
min   1000102.00  75000.00  0.00  0.00  290.00  520.00
25%  2123049194.00  321950.00  3.00  1.75  1427.00  5040.00
50%  3904930410.00  450000.00  3.00  2.25  1910.00  7618.00
75%  7308900445.00  645000.00  4.00  2.50  2550.00  10688.00
max  9900000198.00  7700000.00  33.00  8.00  13540.00  1651359.00

      floors    waterfront    view    condition    grade    sqft_above \
count  21613.00  21613.00  21613.00  21613.00  21613.00  21613.00
mean  1.49  0.01  0.23  3.41  7.66  1788.39
std   0.54  0.09  0.77  0.65  1.18  828.09
min   1.00  0.00  0.00  1.00  1.00  290.00
25%  1.00  0.00  0.00  3.00  7.00  1190.00
50%  1.58  0.00  0.00  3.00  7.00  1560.00
75%  2.00  0.00  0.00  4.00  8.00  2210.00
max  3.50  1.00  4.00  5.00  13.00  9410.00

      sqft_basement    yr_built    yr_renovated    zipcode    lat    long \
count  21613.00  21613.00  21613.00  21613.00  21613.00  21613.00
mean  291.51  1971.01  84.40  98077.94  47.56  -122.21
std   442.58  29.37  401.68  53.51  0.14  0.14
min   0.00  1900.00  0.00  98001.00  47.16  -122.52
25%  0.00  1951.00  0.00  98033.00  47.47  -122.33
50%  0.00  1975.00  0.00  98065.00  47.57  -122.23
75%  560.00  1997.00  0.00  98118.00  47.68  -122.12
max  4820.00  2015.00  2015.00  98199.00  47.78  -121.31

      sqft_living15    sqft_lot15
count  21613.00  21613.00
mean  1986.55  12768.46
std   685.39  27304.18
min   399.00  651.00
25%  1499.00  5100.00
50%  1840.00  7620.00
75%  2360.00  10083.00
max  6210.00  871200.00

```

The average (mean) home in this dataset has a price of \$540,088.14, 3.37 bedrooms, 2.11 bathrooms, 2,079 square feet of living space, and 1.49 floors.

2. Summarize the mode for non-continuous or categorical data values.

- a) Scroll down to view the cell titled **Summarize the most common values**, and examine the code listing beneath it.

Summarize the most common values

```

In [ ]: 1 # Summarize most common values for features with non-continuous or categorical values
2 features_to_summarize = ['view','waterfront','grade','zipcode','bedrooms','bathrooms','floors']
3 data_raw[features_to_summarize].mode()

```

This code displays the mode for selected features.

- Line 1 specifies which features should be summarized.
- Line 2 outputs the mode (most common data value) for each of the specified features in the dataframe.

- b) Select the cell that contains the code listing, and select **Run**.
c) Examine the mode values displayed for the various features.

	view	waterfront	grade	zipcode	bedrooms	bathrooms	floors
0	0	0	7	98103	3	2.5	1.0

The typical house:

- Does not have a "view" and is not on the waterfront.
- Has a grade of 7.
- Has a zipcode of 98103.
- Has 3 bedrooms, 2.5 bathrooms, and 1 floor level.

3. Look for values that seem to correlate with price.

- a) Scroll down to view the cell titled **Show correlations with price**, and examine the code listing beneath it.

Show correlations with price

```
In [ ]: 1 # Look for correlations with price
          2 print('Pearson correlations with price')
          3 corr_matrix = data_raw.corr()
          4 corr_matrix['price'].sort_values(ascending=False)
```

- Line 2 uses the pandas software library to find the standard (Pearson) correlation coefficient between each feature in the dataset.
 - Line 3 outputs the correlations for the price feature, sorting to show the results in order by how closely they correlate with price. Items that correlate with price most closely will have the highest values, and will appear at the top of the list.
- b) Select the cell that contains the code listing, and select **Run**.
 c) Examine how the various features correlate with **price**.

```
Pearson correlations with price
price           1.000000
sqft_living     0.702035
grade           0.667434
sqft_above      0.605567
sqft_living15   0.585379
bathrooms       0.525138
view            0.397293
sqft_basement   0.323816
bedrooms        0.308350
lat              0.307003
waterfront       0.266369
floors           0.256794
yr_renovated    0.126434
sqft_lot         0.089661
sqft_lot15      0.082447
yr_built         0.054012
condition        0.036362
long             0.021626
id               -0.016762
zipcode          -0.053203
Name: price, dtype: float64
```

- The feature that has the strongest correlation with **price** is **sqft_living**.
- Interestingly, the reported **condition** of the house does not have a strong correlation with **price**, nor does lot size (**sqft_lot15** and **sqft_lot**) or **waterfront**.
- You can disregard the perfect 1.000000 standard correlation value for **price**. (Of course, it correlates perfectly with itself.)

TOPIC C

Use Visualizations to Analyze Data

The numbers produced by various statistical measures can provide you with significant insights, but visual tools such as charts and maps may reveal concepts about your data that are difficult to see using numbers alone.

Visualizations

Data visualization involves representing data in charts that clearly depict patterns, trends, and correlations that might go unnoticed when the data is shown as a list of numbers, text, and other data values. Data scientists have long understood the value of good visualization tools to provide insights and to help communicate data outcomes, and have used charts for centuries, long before computers or machine learning came into use.

Visualization tools can be useful at various points throughout a machine learning project, such as when you:

- Initially explore data
- Train and evaluate the model
- Communicate results to stakeholders

Visualization tools can help you and others engage with data quickly and meaningfully, in ways that are often not possible when data are represented only as columns and rows of data values.

Visualization is often accomplished through plotting software, such as a Matplotlib and Seaborn—software libraries used with Python. By working in an interactive development environment (such as Python and Jupyter Notebook), the practitioner can quickly create and manipulate charts to view datasets from different perspectives to gain insights into patterns that might be worth exploring. Charts can also help to expose data outliers, noise, and unexpected outcomes.

Histogram

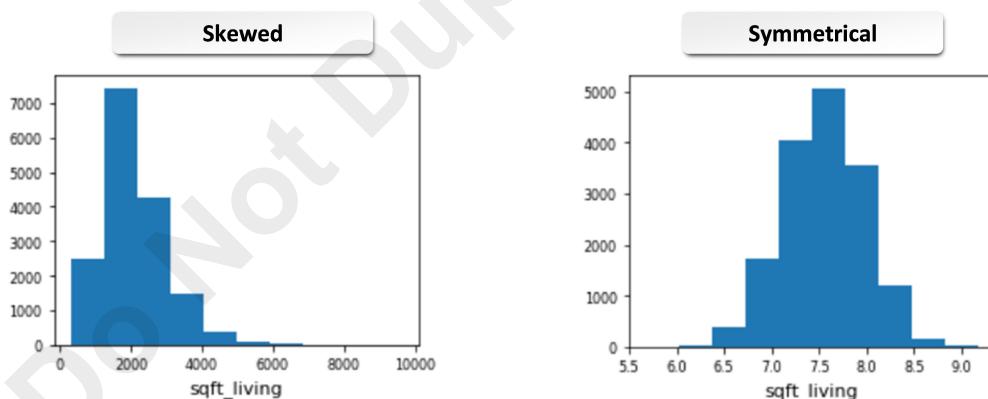


Figure 2-7: Two histograms, showing skewed and symmetrical distributions.

A **histogram** uses a column chart to show the distribution of data over a continuous interval or discrete bins/periods. Each bar in a histogram represents the calculated frequency of data values at each interval or bin. Histograms provide a visual summary that can be quickly interpreted to

understand where values are concentrated, where the extremes are located, the general skewness of the distribution, and whether there is multimodality.

Box Plot

Box plots (or box-and-whisker plots) provide another way of viewing a distribution of data along a line that shows the entire range of values. This type of visualization focuses on showing where data is distributed in relation to the quartiles. A box plot can tell you where you have outliers, how symmetrical the distribution is, how tightly data are grouped in the distribution, and how the data is skewed.

Various items comprise a box plot:

- A horizontal axis representing the range of values in the distribution.
- A box enclosing the interquartile range (IQR, the combined range of values within Q2 and Q3, or the middle 50%).
- A vertical line within the box, showing the median, which is the dividing line between Q2 and Q3.
- Vertical lines on either side of the box called whiskers, which show the location of the **minimum** ($Q1 - 1.5 \times IQR$) and **maximum** ($Q3 + 1.5 \times IQR$).
- Outliers, data points that fall outside the minimum and maximum, plotted as individual points on the horizontal axis.

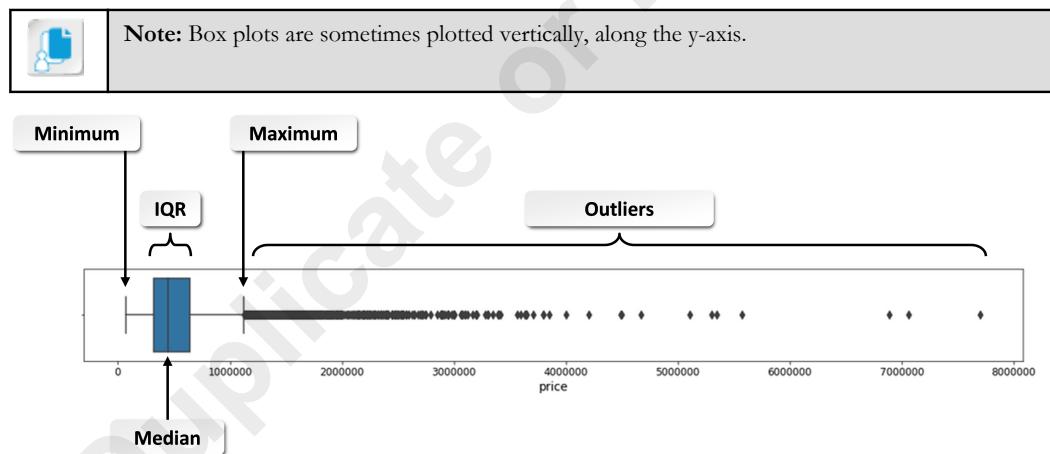


Figure 2–8: A box plot showing the distribution of price values in a dataset.

Scatterplot

A **scatterplot**, also known as a scatter graph, scatter chart, scattergram, point graph, or x–y plot, draws each data point at an intersection of the x- and y-axis. This can be useful for showing the relationship of x and y data pairs within a dataset. For example, if a relationship with the two values is such that the x value increases proportionally as the y value increases, then the data points will tend to form a line in the visualization.

Different types of correlation can be identified through patterns displayed on a scatterplot, such as:

- Positive correlation—The values increase together, forming a line from southwest to northeast.
- Negative correlation—One value decreases as the other increases, forming a line from northwest to southeast.
- No correlation—The points don't form a line and may appear randomly scattered.
- Exponential—The points form a hockey stick shape.

The strength of correlation can be observed by how tightly the points are packed within the plot, and points far away from clusters may be identified as outliers. A trend line or curve may be superimposed over the scatterplot to show the line of best fit, depicting a function that would predict any y value if provided with an x value.



Note: Scatterplots are useful to analyze or depict relationships between numerical data pairs. However, it is important to remember that correlation is not causation. Furthermore, the apparent relationship between data pairs may actually depend on a third, unnoticed variable that is influencing the results.

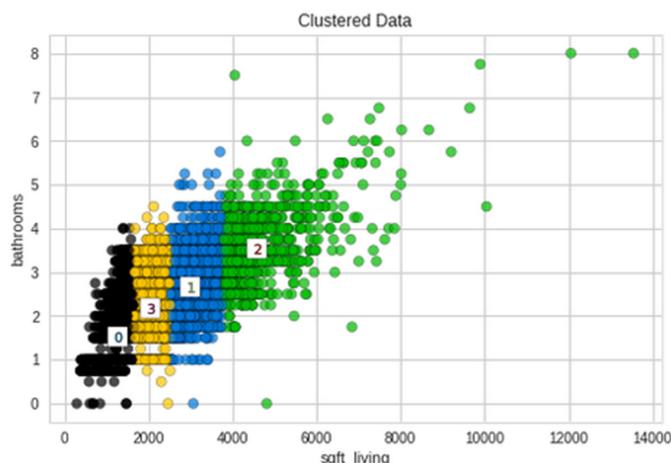


Figure 2-9: A scatterplot, color-coded to show four clusters of x-y data.

Geographical Maps

Some types of data may describe a particular geographic location, and may therefore be appropriate to analyze using geographical maps. Numerous tools are available to plot data points in a map. The dataset must include some data items that can be used to identify coordinates or locations within the map, such as latitude and longitude or zip codes. Although maps are typically thought of in regard to large areas such as a city or country, they may also be used to visualize data points covering a small area or assembly, such as a single building, the anatomy of a person or animal, a piece of equipment (such as the location of structural faults observed within a particular airplane), and so forth.

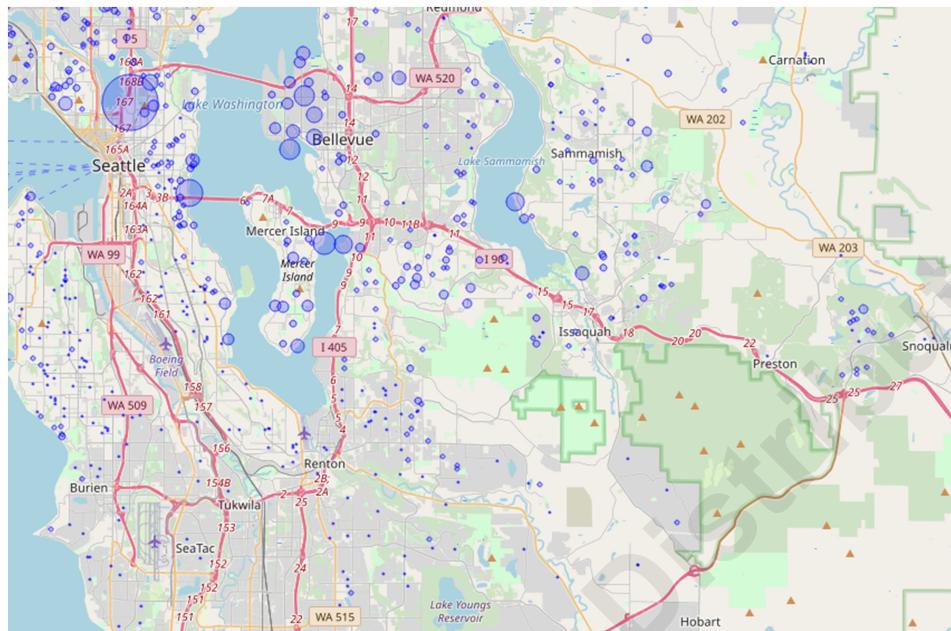


Figure 2-10: Geographical map using larger markers to show more expensive homes.

Heat Maps

A **heat map** plots regions of the chart with an intensity or shade of color based on data values in that location of the chart. This color coding makes it easier for a viewer to identify patterns of values by their color and location within the grid. For example, a heat map might plot the locations of houses on a map. Parts of the map where the density of houses is high might be highlighted with a particular shade or color. Where the density of houses is low, a different shade or color might be shown. In various parts of the map, the blend of the two shades or colors will reflect how sparse or dense the houses are in that location.

While heat maps can be used to present a grid that represents physical locations (such as houses located within a city), they can also be used to show conceptual divisions. For example, a heat map can be used to enhance a correlation matrix. The correlation matrix is a grid that shows how the data items in every column correlate with the data items in every other column. Each cell in the grid shows the correlation coefficient between two columns. Pairs with the highest correlation might be shown in one shade, while pairs with the lowest correlation are shown in another shade. By scanning the intensity of the shades shown, you can quickly identify column pairs with the highest or lowest correlation.

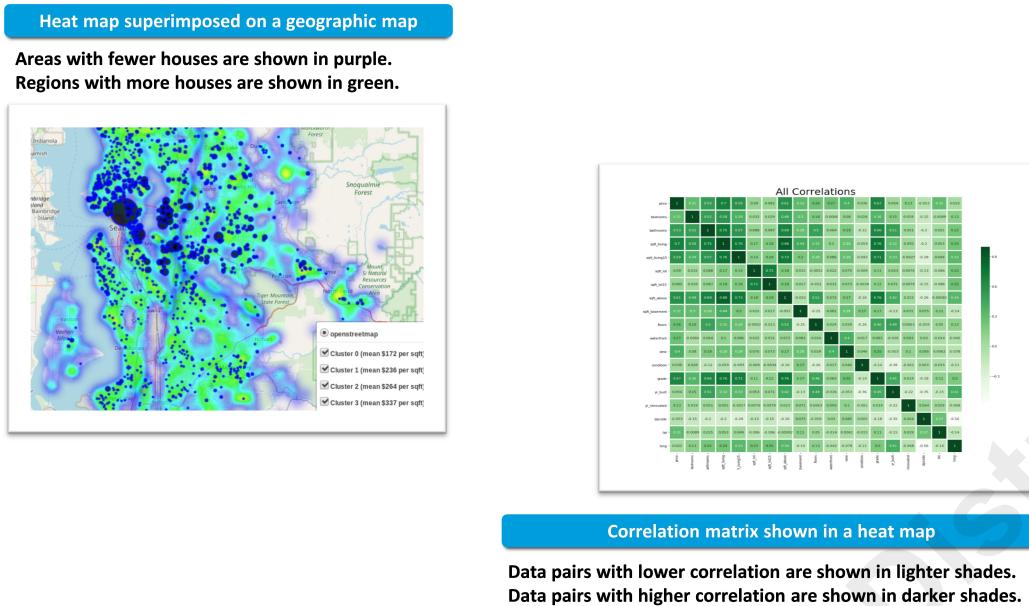


Figure 2-11: Heat maps using color to represent data values.

Guidelines for Using Visualizations to Analyze Data

Follow these guidelines when you generate visualizations from datasets in Python.

Use Python to Generate Visualizations

The Matplotlib library provides various objects and functions you can call in Python to create charts. Various other software libraries, such as pandas and Seaborn, build upon the functions provided by Matplotlib to enable you to create a wide variety of specialized charts. Following are some of the objects and functions you can use to generate visualizations using Python, Matplotlib, and pandas.

- `matplotlib.pyplot.rc('axes', labelsize = 14)`—This function, shown here with two parameters, enables you to change various settings in Matplotlib. The first parameter specifies the category of the setting you're specifying ('`axes`' in this example), and it is followed by a name-value pair making the actual configuration assignment (setting the default size of axis labels to 14). You can use the `rc()` function to set a wide range of different chart properties, such as the type size and font used to label an axis, thickness of lines used in plots, and so forth.
- Note:** The name "rc" pays homage to the longstanding convention in Unix-like operating systems to use files ending in `.rc` to hold "run commands," configuration instructions that would run when a program was initially launched. In Matplotlib, you can just think of the `rc()` function as a way to modify the default configuration of a chart.

 - `dataframe.hist(figsize=(20,15))`—The pandas dataframe `hist()` function creates a separate histogram for each column data in the dataframe. It calls on the Matplotlib library to perform the actual plotting. You can pass in various parameters to format the chart, such as the `figsize` name-value parameter shown here, which is setting the chart size.
 - `matplotlib.pyplot.figure()`—Matplotlib's `figure()` function creates a new figure. When running in Jupyter Notebook, the image is displayed inline, within the code cell's output.
 - `seaborn.heatmap()`—Seaborn's `heatmap()` function plots a correlation matrix—a grid showing the correlation coefficients between every pairwise combination of values. Matplotlib is used to generate the visualization. The correlation coefficient is shown at the intersection of each column, and color coding shows the strength of the correlation. You must pass in various parameters to specify the data to be used and the formatting options for the heatmap.
- Licensed For Use Only By: Abdulwahab Alweban dev.abdulwahab@gmail.com Jan 8 202
- Lesson 2: Collecting and Refining the Dataset / Topic C

- `matplotlib.pyplot.title()`—Matplotlib's `title()` function can be used to set the x-axis, y-axis, or general title of a chart. You can provide parameters to set the alignment, padding, font formatting, and label of the title.
- `matplotlib.pyplot.subplot()`—Matplotlib enables you to combine multiple plots into a single visualization. This function adds a subplot.



Note: These guidelines provide a quick summary of various objects used in course activities. Complete documentation covering the objects mentioned in these guidelines may be found at <https://matplotlib.org>, <https://pandas.pydata.org>, and <https://seaborn.pydata.org>.

ACTIVITY 2–5

Analyzing a Dataset Using Visualizations

Data File

/home/student/CAIP/Workflow/Workflow-Housing.ipynb

Before You Begin

If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the **CAIP/Workflow/Workflow-Housing.ipynb** notebook. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled "Analyze cross correlations." Select **Cell→Run All Above**.

Scenario

You have used various statistical measures to examine values in the dataset. Now you will use visualizations (various types of charts) to gain additional insights regarding the data.

1. Analyze cross correlations.

- Scroll down to view the cell titled **Analyze cross correlations**, and examine the code listing beneath it.

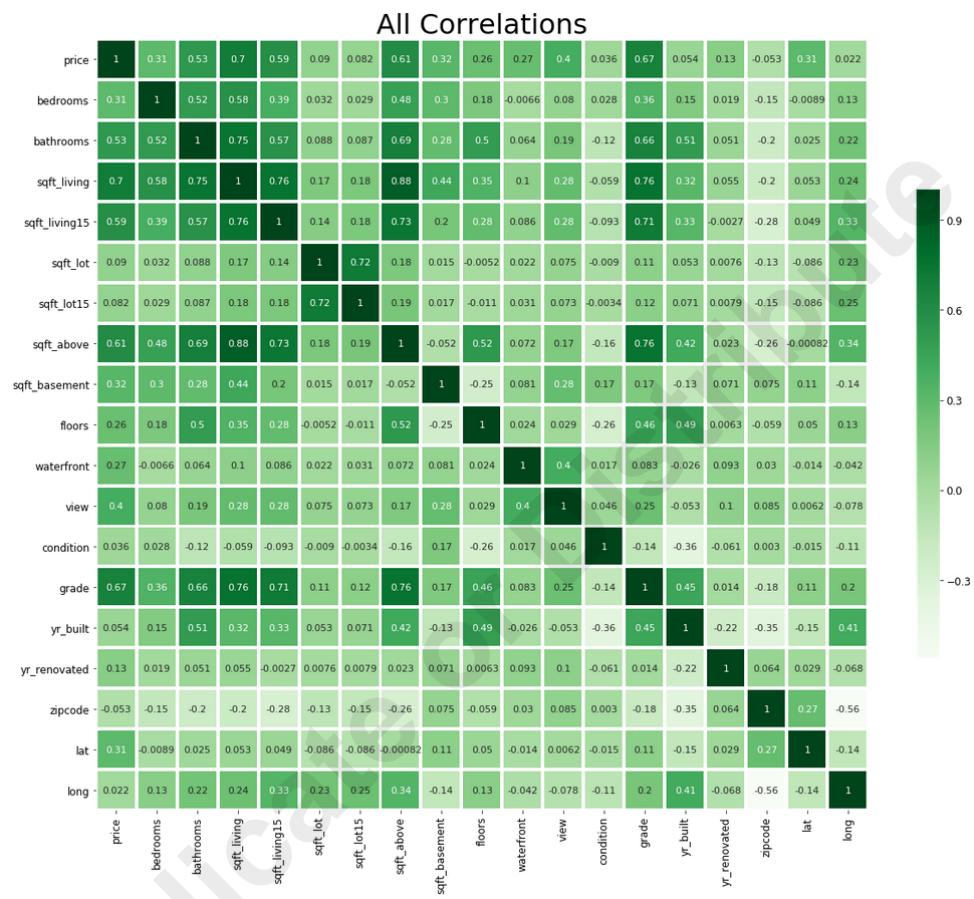
Analyze cross correlations

```
In [ ]:
 1 # Use Matplotlib for visualization
 2 %matplotlib inline
 3 import matplotlib as mpl
 4 import matplotlib.pyplot as plt
 5
 6 # Specify size and title for the visualization
 7 f, axes = plt.subplots(figsize=(20, 20))
 8 plt.title('All Correlations', fontsize=32)
 9
10 # For the purpose of visualization, we'll use a different order for the features.
11 # We'll start with price, to make it easier to compare all other features with it.
12 features = ['price','bedrooms','bathrooms',
13             'sqft_living','sqft_living15','sqft_lot','sqft_lot15','sqft_above','sqft_basement',
14             'floors','waterfront',
15             'view','condition','grade',
16             'yr_built','yr_renovated',
17             'zipcode','lat','long']
18
19 # Use Seaborn library to plot the correlation matrix as a heatmap
20 sb.heatmap(data_raw[features].corr(),
21             linewidths = 3.0,
22             square = True,
23             cmap = 'Greens',
24             linecolor='w',
25             annot=True,
26             annot_kws={'size':11},
27             cbar_kws={'shrink': .5});
```

- Lines 7 and 8 prepare to plot the visualization, specifying its size and title.
- The statement in lines 12 through 17 identifies the columns that will be cross correlated and the order in which they will appear.
- The statement in lines 20 through 27 calls the Seaborn library's `heatmap()` function, passing in the correlation values to be displayed in the heatmap.

- Select the cell that contains the code listing, and select **Run**.

- c) Examine how the various features correlate with each other.



- This heatmap visualization shows correlations between different features in the dataset as numeric values, but enhances them with color coding that helps you quickly see which values correlate the most.
- Each feature is shown on the x-axis and y-axis.
- At each intersection, the correlation coefficient is shown for the combination of features represented on the two axes.
- Darker tones highlight values with a high correlation coefficient. The darkest values appear diagonally where features intersect with themselves.

2. Show how the various features are distributed.

- a) Scroll down to view the cell titled **Use histograms to visualize the distribution of various features**, and examine the code listing beneath it.

Use histograms to visualize the distribution of various features

```
In [ ]: 1 mpl.rcParams['axes', labelsize=14)
2 mpl.rcParams['xtick', labelsize=12)
3 mpl.rcParams['ytick', labelsize=12)
4
5 data_raw.hist(figsize=(20,15));
6 plt.figure();
```

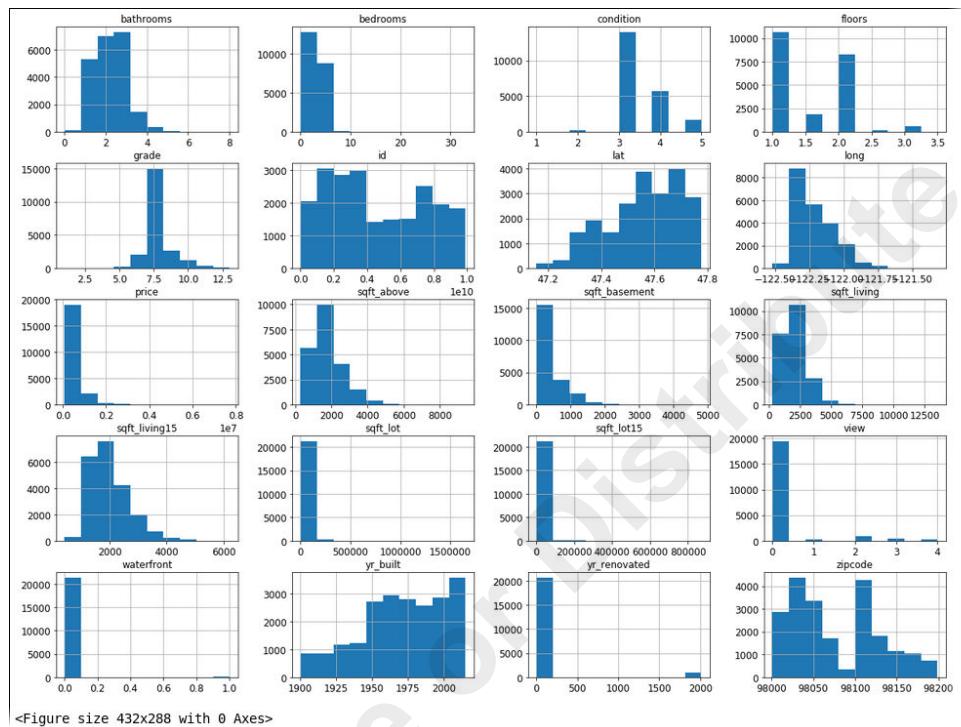
- Lines 1 through 3 specify the font size to be used for the histograms' axis titles and tick mark labels.
- Line 5 calls the pandas dataframe object's `hist()` method to create a histogram for each column of numeric data in the dataset.
- Line 6 plots the entire figure using the settings configured in lines 1 through 3.

- b) Select the cell that contains the code listing, and select **Run**.



Note: It may take several seconds for processing of the code to finish.

- c) Examine the distributions shown in the figures.



- Some features, such as `bathrooms`, `bedrooms`, and `sqft_living` are fairly close to a normal distribution. Others, such as `zipcode` are not.
- The `yr_built` (year built) values likely follow the general pattern for house building over the years. More houses have been built in recent years, and there was a decline in building during the Great Depression years (late 1930s).
- The central tendency can be observed for several values, such as `condition` (3), `bathrooms` (2.5), `grade` (7), and `floors` (1).
- The x-axis of the `bedrooms` histogram extends out to more than 30. Charts typically adjust the x- and y-axes to show the range of values being plotted, so there may be some outlier data values in the `bedroom` column that you should investigate. Something similar is happening with the `price` histogram.

3. Use a map visualization to gain insights regarding the relationship between price and location.

- a) Scroll down to view the cell titled **Visualize with a geographic map to gain insights regarding location**, and observe the code listing beneath it.

Visualize with a geographic map to gain insights regarding location

```

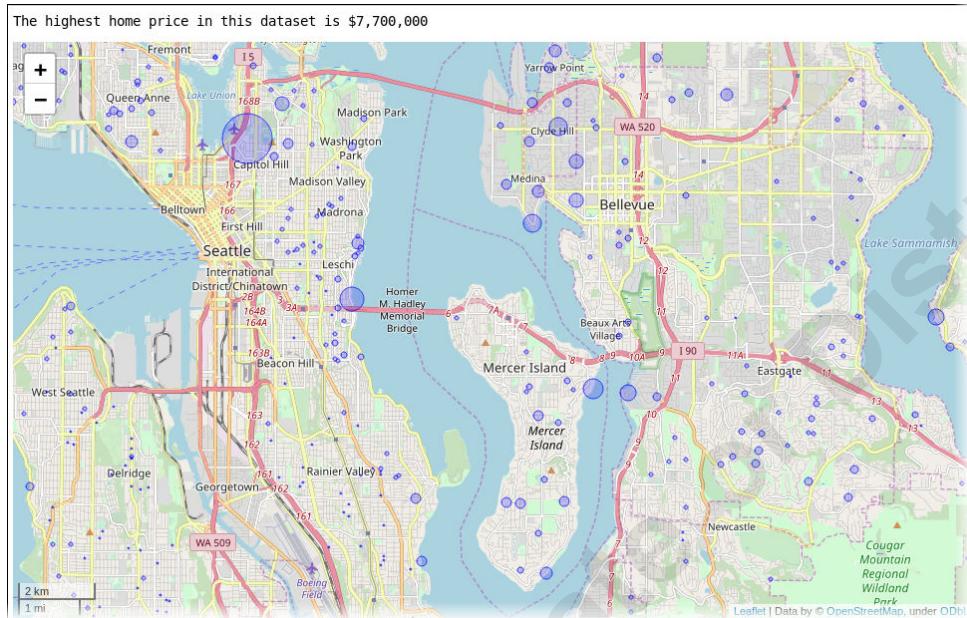
1 # To avoid overwhelming the visualization tool, we'll only plot every Nth house.
2 n_homes = 20
3 data_raw_subset = data_raw.sort_values(by =['price'], ascending = False)[::n_homes]
4
5 # Output highest house price
6 max_price = data_raw_subset.loc[data_raw_subset['price'].idxmax()]['price']
7 print(f'The highest home price in this dataset is ${max_price:.0f}')
8
9 # Descriptions of the building grades used in King County

```

This is a large chunk of code. It will be easier to understand what the code does if you first run it see the map it produces.

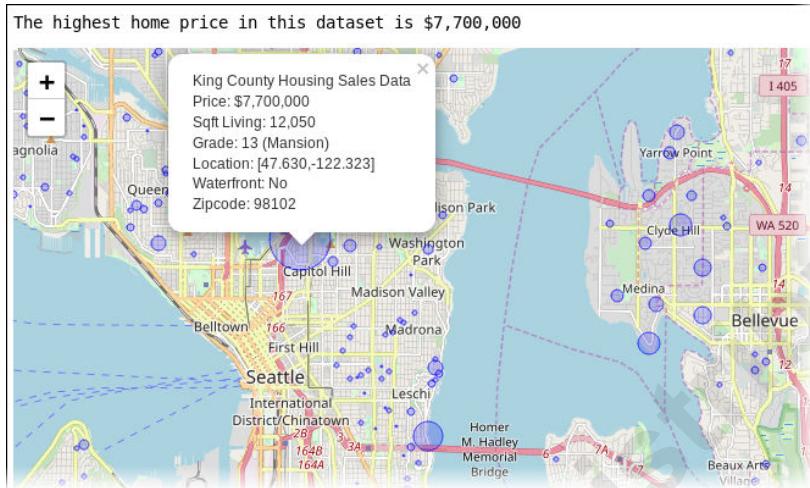
Licensed For Use Only By: Abdulwahab Alweban dev.abdulwahab@gmail.com Jan 8 202

- b) Select the cell that contains the code listing, and select **Run**.
Plotting the map may take several seconds to complete.
- c) In the upper-left corner of the map, select the **+** button to zoom in. After the map is redrawn, select the **+** button again to zoom in further.
The ability to zoom in and out can be useful as you analyze the map to look for patterns of pricing.
- d) Drag the map so Seattle and Bellevue are centered with Mercer Island in the middle, as shown.
Observe the map.



- Above the map is a message that shows the highest house price.
- The map shows houses in King County, Washington.
- Each dot on the map represents a house from the dataset. The size of the dot corresponds to the price of the house.

- e) Select the largest dot, which is located west of Washington Park and north of Capitol Hill.



- When you select the dot, popup text shows some of the relevant data for this house.
 - This is the \$7,700,000 house identified in the message above the map.
- f) Refer to the map as you explore the relationship between location and housing prices.



Note: Drag to adjust the location and zoom in and out as needed.

4. What patterns seem to exist regarding house prices and location?

5. Examine the code that produced the map visualization.

- a) Scroll up and view lines 2 through 7 of the code that produced the map.

Visualize with a geographic map to gain insights regarding location

```

1 # To avoid overwhelming the visualization tool, we'll only plot every Nth house.
2 n_homes = 20
3 data_raw_subset = data_raw.sort_values(by =['price'], ascending = False)[::n_homes]
4
5 # Output highest house price
6 max_price = data_raw_subset.loc[data_raw_subset['price'].idxmax()]['price']
7 print(f'The highest home price in this dataset is ${max_price:, .0f}')
8
9 # Descriptions of the building grades used in King County

```

- Lines 2 and 3 create a temporary copy of the dataset that sorts the homes by price, but filters the records to only include every 20th house in the dataset. While this sample should still be adequate to visually depict any price trends based on location, it also ensures that the mapping tool won't be overwhelmed with data.
- Lines 6 and 7 output the message identifying the highest house price.



Note: If you wanted to see more houses, you could reduce the number assigned in line 2. Bear in mind, however, that including more houses will increase the time needed to plot the map, and may bog down the entire Jupyter Notebook application.

- b) View lines 11 through 13 of the code that produced the map.

```

9 # Descriptions of the building grades used in King County
10 # Values obtained from http://www5.kingcounty.gov/sdc/FGCDocs/resbldg_extr_faq.htm
11 bldg_grades = ['Unknown', 'Cabin', 'Substandard', 'Poor', 'Low', 'Fair',
12                 'Low Average', 'Average', 'Good', 'Better',
13                 'Very Good', 'Excellent', 'Luxury', 'Mansion', 'Exceptional Properties']

```

- Lines 11 through 13 provide a lookup for the numeric building grades in the dataset. This information came from the King County website. In line 29, `bldg_grades` is used to generate the popup text.
- In the popup text, it is more meaningful to show the description than to just show the number. For example, the most expensive house's popup text showed that it was grade 13, which is described as a "Mansion".

- c) View the remaining lines of code.

```

15 # Use Folium library to plot values on a map.
16 import folium
17
18 # Generate the base map, centering on King County.
19 base_map = folium.Map(location = [47.5300, -122.2000],
20                      control_scale = True,
21                      max_zoom = 20,
22                      zoom_start = 10,
23                      zoom_control = True)
24
25 # Plot homes by price.
26 for index, row in data_raw_subset.iterrows():
27
28     # Get the grade description for this row.
29     grade_desc = bldg_grades[row['grade']]
30     waterfront_desc = "Yes" if (row['waterfront'] == 1) else "No"
31
32     # Add popup text. Click each point to show details.
33     popup_text = '<br>'.join(['KingCountyHousingSalesData',
34                             'Price:$:.0f',
35                             'SqftLiving:',
36                             'Grade:{}',
37                             'Location:[.3f],{:.3f}]',
38                             'Waterfront:{}',
39                             'Zipcode:{}')
40
41     popup_text = popup_text.format(row['price'],
42                                    row['sqft_living'],
43                                    row['grade'], grade_desc,
44                                    row['lat'], row['long'],
45                                    waterfront_desc,
46                                    row['zipcode'])
47
48     # Add each home to the map, but show larger dots for higher prices.
49     scaling_value = (row['price'] / max_price) # 1.0 for highest price.
50     folium.CircleMarker([row['lat'], row['long']],
51                         radius = 25 * scaling_value,
52                         weight = 1,
53                         fill = True,
54                         fillColor = '#0000FF',
55                         fillOpacity = 0.7,
56                         color = '#0000FF',
57                         opacity = 0.7,
58                         popup = popup_text).add_to(base_map)
59
60 base_map

```

- Line 16 imports the Folium software library, which is used to generate the map.
- Lines 19 through 23 create a `Map` object, which will be referred to as `base_map`. This will generate the street map, but subsequent code will have to add markers for each house.
- Lines 26 through 58 provide a loop that iterates through each row (house) in the dataset, constructing the popup text (lines 29 through 46), and creating a scaled dot (lines 49 through 58) for each house.
- Up to this point, the map exists in memory, but is not yet plotted on the page. Line 60 outputs the map.

TOPIC D

Prepare Data

Before a dataset can be used with a machine learning model, there are typically various tasks you need to perform.

Data Preparation

Once you understand the dataset and have identified what it includes and what is missing, you may start the **data cleaning** process (also called *data cleansing* or *scrubbing*). This is the process of locating and removing errors and inconsistencies in data. It may involve tasks such as removing or handling incorrect or missing data, outliers, and so forth. You may also have to standardize column names to make them consistent, readable, and understandable. Data may include padding or embedded spaces that need to be removed.

Even after the data is clean, consistent, and well structured, there may be additional modifications you must make to prepare it for use in machine learning. For example, a single column might contain multiple values, such as house number, street name, city, state or province, and postal code all in a single value that you must extract into separate columns. The process of transforming columns of raw data in the original dataset into *features* that are useful to support machine learning is called **feature engineering**.

The entire process of data preparation can be tedious and may take a significant amount time on a machine learning project. Reflecting the challenge of the task, it is sometimes called **data wrangling** or **data munging**, particularly when it is performed manually or outside of formal, repeatable processes.

Software libraries such as pandas provide functions that enable you to automate the process of data preparation. This is especially valuable when you must repeat the cleanup process on other datasets or when new data is added over time.



Note: Whenever you perform operations on data, consider creating a backup copy so you can revert back to the original dataset should anything go wrong.

Data Types

Data items may be represented differently in the various sources from which you obtain data. Different database systems and data stores support different data types and might store values with different levels of precision. For example, a seemingly straightforward value such as a date might be stored as a string, a datetime object, or a number. Dates from one source might be represented as string values, while dates from another source might be represented as datetime objects. When you combine values from multiple sources, they must use a consistent data type, one that is compatible with the data store or database you're using, and in a format that will support the machine learning algorithms you're using to process the data.

Multiple data types may be embedded within a single data item. For example, address data might be stored in a column named Address. An address value might be expressed in the form 900 Main Street, Machias, ME 04654. A column of data like this might be useful for machine learning if it were broken out into multiple features including Street Address (900 Main Street), City (Machias), State (ME), and zip Code (04654).

Once these new features have been created based on data in the original Address feature, they will be more expressive and meaningful than the original feature, facilitating better pattern detection by a machine learning algorithm (such as patterns corresponding to a particular zip code, house number, street, or state).

	Note: To be useful for machine learning, categorical data like city, state, and zip code would require some additional preparation to convert them to numeric codes.
	Note: When using a strongly typed programming language, you may need to use typecasting to convert data from one type to another.

Operations You Can Perform on Different Types of Data

Statistical computations like those used in machine learning can't be indiscriminately applied to just any type of data. For example, it is not particularly apparent how or why you would calculate the *mean* value of a set of hair colors because there is no inherent order or sequence. It seems much more straightforward to calculate the most common (*mode*) hair color.

Following are different types of data, showing statistical measures that make sense to be performed on each type.

Type of Data	Description
Quantitative (or numerical) data	<p>Quantitative (or numerical) data hold number values. Various center and spread measures can logically be made on a set of quantitative data values.</p> <p>Valid measures for the quantitative dataset (18, 18, 29, 45, 59) are shown here.</p> <ul style="list-style-type: none"> • Mean - $18 + 18 + 29 + 45 + 59 / 5 = 33.8$ • Median - 29 • Mode - 18 • Range - $59 - 18 = 41$
Qualitative (or categorical) data	<p>Qualitative (or categorical) data hold a value from a set of values that are typically limited. For example, Eye Color might hold colors such as blue, brown, or hazel.</p> <p>Most center and spread measures are inappropriate for qualitative data, although mode can still be determined.</p> <p>Valid measures for the qualitative dataset (blue, hazel, brown, brown, brown) are shown here.</p> <ul style="list-style-type: none"> • Mode - brown
Ordinal data	<p>Ordinal data are not quantitative, but they can be ordered. For example, a Letter Grade feature might hold a student's grade, which can be A, B, C, D, or F. An order is represented (for example, an A is a better grade than a C).</p> <p>Valid measures for the class dataset (A, B, C, F, F) are shown here.</p> <ul style="list-style-type: none"> • Median - C • Mode - F

	Note: Not all numerical data is quantitative. For example, Social Security numbers have no logical sequence, and it would not make sense to perform mathematical operations on such a number.
---	--

Continuous vs. Discrete Variables

A *continuous variable* is one whose values are uncountable and can extend infinitely. Consider a person's age. You might constrain this variable to be between 0 and 129. So, for example, you could

describe someone as being 40 years old. However, you can get more specific and say they're 40 years and 3 months; or, 40 years, 3 months, and 6 days; or, 40 years, 3 months, 6 days, and 19 hours; and so on, continuing indefinitely. This is a continuous variable because it can take on all real values between a set of values.

Certain machine learning algorithms, especially decision-tree-based algorithms, cannot effectively work with continuous variables. In contrast, a **discrete variable** is one whose values are countable and limited, because there is a definite gap between each value in a range of values. A feature like "age in years" is a discrete variable because it is either a 0, 1, 2, 3, etc., and cannot be divided into more precise values.

Data Encoding

While machine learning algorithms are typically geared toward working with numerical data, many sources of enterprise data contain text. To be useful in machine learning, often these values must be transformed into numerical data. Numerical values contained within text strings should be extracted to new columns and formatted with an appropriate numerical data type. Categorical labels, such as ["fine", "medium", "coarse"] or ["France", "Belgium", "UK"], generally need to be converted to numbers.

There are various methods for converting category labels to numbers. Some examples are listed here.

Encoding Method	Description
Label encoding	<p>Label encoding translates a set of categorical labels (["small", "medium", "large"], for example) into numerical labels (such as [0, 1, 2]).</p> <p>Because machine learning algorithms may perceive an order or ranking to numbered categories, label encoding is most suitable for categories meant to imply a sequence or rank. For example, if you assigned the countries ["France", "Belgium", "UK"] to the numeric labels [0, 1, 2], a machine learning algorithm might perceive that the last country (UK, with a value of 2) is ranked the highest, which may not be what you intended.</p>
One-hot encoding	<p>If you don't want to imply a sequence or rank when you encode categorical labels, you can use one-hot encoding, which is sometimes called dummy coding.</p> <p>With this method, you create a dummy column for each class of a categorical attribute. For example, you might create three columns named <code>isFrance</code>, <code>isBelgium</code>, and <code>isUK</code>. The presence of each class is represented by 1 and its absence is represented by 0. This ensures that the machine learning algorithm will give no class (France, Belgium, or UK, in this case) more value than the others.</p>
Binary encoding	<p>Since one-hot encoding only needs to represent a 0 or 1, multiple one-hot encoding values may be combined within a single binary number, so only one feature is needed to represent all of the values.</p>
Frequency-based encoding	<p>With this approach you calculate how frequently each class occurs within the training set, and you use this number as the code for that class, essentially determining the weight of that class within the dataset, and using that value as its code.</p>
Target mean encoding	<p>Each class is encoded as a function of the mean of the target. This method can be used for machine learning problems in which the model is trained to perform a classification task.</p>

Encoding Method	Description
Hash encoding	Uses a hash encoding algorithm to map a particular text string to a number value. While the resulting hash value appears to be random, it is algorithmic, based on the characters in the text string, and will produce the same number value each time a particular text string is provided to it. This method can be a useful way to generate consistent codes from text values when there are hundreds of classes/categories.
	 Note: In some cases, a hashing function may generate the same numeric value from different text strings. So this approach should not be used when such a "collision" (different categories ending up with the same numeric value) would adversely affect the model.

 **Note:** scikit-learn provides various Python functions that you can use to automate data encoding processes, available through the `category_encoders` package.

Dimensionality Reduction

As you know, data fed into a machine learning model has a set of features that describe particular examples of the data. In some datasets, there are so many features that it becomes difficult to visualize the data. Likewise, some features may be redundant, which can complicate the training and evaluation process and have a negative impact on performance. **Dimensionality reduction** is the process of simplifying a dataset by eliminating redundant or irrelevant features.

Dimensionality reduction can help reduce the problem of tuning a model so closely to the training data that the model performs poorly on new data samples (a problem called "overfitting").

Dimensionality reduction can also decrease computation time and alleviate storage space issues.

Two categories of dimensionality reduction are feature selection and feature extraction.

In **feature selection**, you select a subset of the original features. This subset includes relevant and/or unique features, and excludes features deemed redundant or irrelevant to the problem. The model trains on this subset rather than the whole dataset. Feature selection is particularly useful in datasets that have a disproportionately large number of features compared to actual data examples.

In **feature extraction**, you derive new features from the original features. This is typically done by combining multiple correlated features into one. For example, if you're trying to predict storage drive failure, the feature "age" probably correlates highly with the features "reads" and "writes," so the three can be merged into one derivative feature that represents the drive's logical usage. Feature extraction is particularly useful in computer vision applications like image processing.

While the goal is to minimize the loss of useful data as much as possible, there is still a risk of this happening. Nevertheless, dimensionality reduction is almost always worth doing, especially with complicated, feature-rich datasets.

Principal Component Analysis (PCA)

One common method for reducing the dimensionality of a dataset is principal component analysis (PCA). Specifically, PCA performs a type of feature extraction by taking data that is in high dimensions and projecting that data into a space of equal or lower dimensions. It does this by selecting only the features that contribute to the greatest amount of variance in the dataset, while dropping the features that contribute very little to the variance.

Impute Missing Values

Some machine learning libraries implement algorithms that can handle some missing values on their own, but it is best if you identify missing data as part of your initial exploration of the dataset, and

decide exactly how you will deal with missing data. If you simply ignore missing data, the algorithm may cope with the missing data, but the resulting model may not perform as well. When preparing training data, you can drop records with missing values, but that may be a bad choice too, depending on how many records you have to delete and what other data they contain.

In some cases, the best approach may be to impute missing values as part of the initial cleanup process. **Imputation** means providing your best estimate to fill in the missing values. There are numerous strategies for doing this. Some examples are described here.

Imputation Method	Description
Mean imputation	Calculate the mean of all items that are not missing in that column. Use the result to fill in missing values. This approach is simple. It preserves the value of the mean and sample size, but may not be as good as other methods listed here.
Substitution	Use data from a new record that is not in the sample.
Hot Deck Imputation	Find records in the sample that have similar values on all other data items than the one that is missing, and copy the missing value from one of the similar records. If there is more than one similar record, randomly select the one you copy from.
Cold Deck Imputation	Similar to Hot Deck Imputation, but without the random variation. For example, may always choose the middle record to copy from.
Regression Imputation	Uses a prediction model to identify what the missing value should be, based on data in the record. Uses patterns established among other records that are not missing the value to determine what the value should be.
Stochastic Regression Imputation	Similar to Regression Imputation, but includes a randomizing value.



Note: The general concept of imputation is simple, but doing it well in practice can be challenging. Some of these methods may be prone to some bias, and in some cases may produce estimated values that produce worse results than you might obtain by simply deleting the record containing the missing value. When it is essential to provide the very best estimates, you may opt to use a multiple imputation approach, combining multiple methods to find the "best guess" and reduce bias.

Duplicates

Depending on how the data is collected or aggregated, some duplication of records may occur. Duplicates should be identified and removed so they are only represented once within the dataset.

Normalization and Standardization

Many machine learning scenarios involve multiple pieces of data, such as a person's age and weight. With enough samples and time, machine learning algorithms can eventually produce effective models, but advance preparation of the data can improve the efficiency of the training process.

For example, depending on how different attributes are represented, one attribute may have more influence than another. A dataset might include age values varying from 23 to 69 (years old), with weight values varying from 105 to 332 (pounds). In this scenario, the weight values may exert more influence than age values since the weights are generally larger numbers. **Normalization** and

standardization are both techniques for ensuring that different attributes are measured according to a common scale.

For example, you might *normalize* ages by dividing data values such that the youngest age is represented by 0 and the oldest age is represented by 1. If you similarly normalize the lowest weight to 0 and the highest weight to 1, then the two attributes are represented on the same scale.

Alternatively, you might *standardize* ages and weights by converting them to statistical measures. For example, when you plot data values in a frequency distribution, you can measure any value's ***z-score*** (also called *standard score*) as the number of standard deviations that the sample is above or below the mean of all values in the sample. By plotting a distribution curve for all ages and another one for weights, you can obtain the *z-score* for any age or weight. Typically values are standardized to have a mean value of zero and a standard deviation of 1. The *z-scores* for these values would be much more comparable than the raw data values, and would therefore be more suitable to use when training a machine learning algorithm.

Summarization

Freeform text documents may contain significant amounts of information that is useful, but not in a form that can be readily consumed by other processes. Automatic data summarization tools can process a body of text, and summarize the information it contains to prepare it for consumption by machine learning or a human audience. Summarization can be used to create an abstract, find the most informative sentences, or classify the type of information. Summarization tools can even be used to extract information that can be used to populate features used in a machine learning model.

Holdout Method

After you have done some basic data cleaning, and before you get started training and tuning the model, you may need to set aside a portion of your dataset. The ***holdout*** method involves splitting up the original dataset so that you have multiple sets. It is essentially a process of sampling the data.

- **Training Set:** Data you use to train (or *fit*) the model. In supervised machine learning, the learning algorithm operates on the training set, in many cases referring to an answer key (labels).
- **Validation Set:** The model doesn't learn from this dataset. You use it to evaluate how well the model can perform on a new dataset that it wasn't trained on. For example, based on the model's performance on the validation set, you may determine that you need to tune settings of the algorithm. You will be familiar with the data in the validation set, and you will have access to the correct answers (labels) for it.
- **Test Set:** This is another testing set. You should not use it to tune performance (like you would use a validation set), but rather only for testing the final fit of the model.

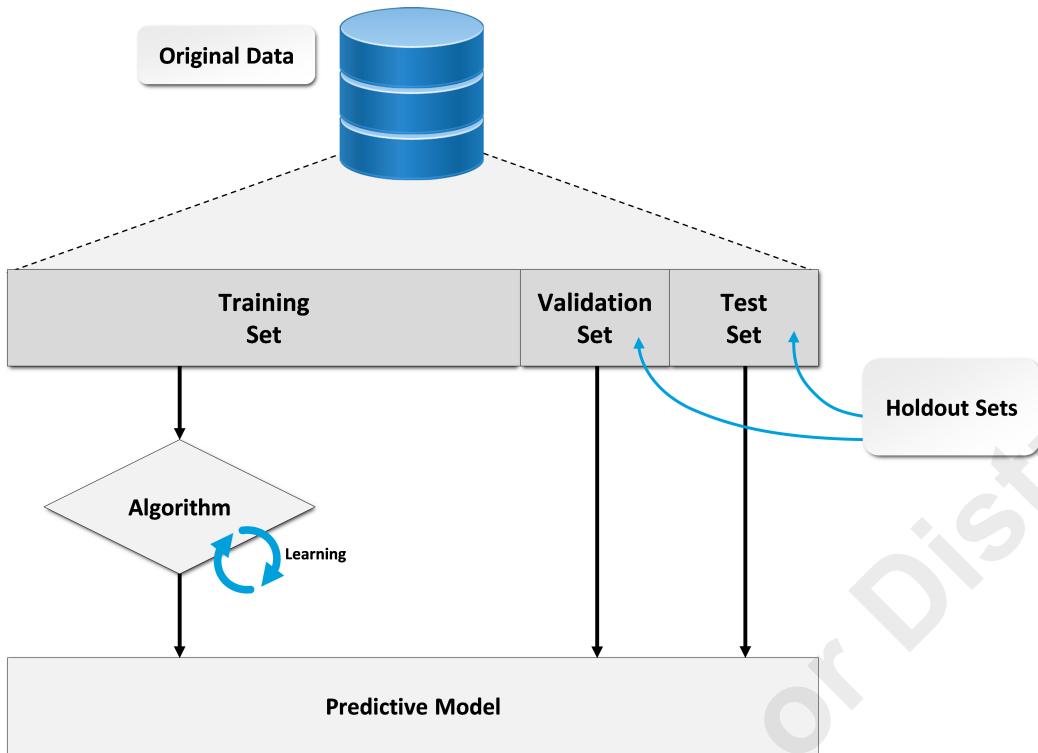


Figure 2–12: Using the holdout method to create a validation and testing set.

A validation set may not be needed in every situation. It's generally only used in situations where there are settings for the learning algorithm (hyperparameters) that you can adjust, since performance tuning is the primary reason for using a validation set. If you use a validation set, typical percentages for splitting the data are 60% training, 20% validation, and 20% testing. If you just use a training and testing set, typical splits are around 75% training and 25% testing. Either way, you can see that the majority of data is typically used for training, in order to produce a better model. The actual numbers you use depend on the situation, and you may use your judgment to adjust the split amounts to produce the best result.

Guidelines for Preparing Training and Testing Data

Follow these guidelines when you are preparing datasets for training and testing in Python.

Split the Training and Testing Data

The scikit-learn library provides the `sklearn.model_selection.train_test_split()` function, which you can call to split a dataset into random training and testing datasets.

- As the input dataset, you can provide a list, NumPy array, SciPy matrix, or pandas dataframe.
- If you also pass in a set of training labels, the label dataset will be split in the same proportions as the training and test dataset.
- You can specify the portion you want as a training set and test set by providing a floating point value between 0.0 and 1.0, where .25 would produce a dataset containing 25% of the records.
- If you don't specify portions, the test set will include 25% of the records, and the training set will include 75% of the records.
- You can also provide a random number seed value to ensure the same randomization is used each time you call the function.



Note: These guidelines provide a quick summary of various objects used in course activities. Complete documentation covering the objects mentioned in these guidelines may be found at <https://scikit-learn.org/>.

Do Not Duplicate or Distribute

ACTIVITY 2–6

Splitting the Training and Testing Datasets and Labels

Data File

/home/student/CAIP/Workflow/Workflow-Housing.ipynb

Before You Begin

If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the **CAIP/Workflow/Workflow-Housing.ipynb** notebook. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled "Split the data into training and testing sets and labels". Select **Cell→Run All Above**.

Scenario

The housing dataset is already in pretty good shape. You have already observed that all rows contain data, so no missing values need to be added. But you have identified some attributes that you won't use to train the model.

Features You Won't Use	Rationale
id and date	The id and date values just identify the record and the date the house was sold. They have no bearing on price.
sqft_above	The sqft_living and sqft_above features showed a high correlation with each other (0.88), so it may be redundant to use both of these features in the model. The sqft_living feature showed a 0.70 correlation with price, whereas sqft_above showed only a 0.61 correlation with price. So of these two features, sqft_living will be the more useful one to predict the price.
sqft_living15 and sqft_lot15	These features summarize the size of the house and lot for the 15 nearest properties. They don't directly describe the property itself. While this might provide some indication of the neighborhood, the house's own measurements (sqft_living and sqft_lot) show a stronger correlation with price than these measures of the house's neighbors.

Attributes that remain include sqft_living, grade, bathrooms, view, sqft_basement, bedrooms, lat, waterfront, floors, yr_renovated, sqft_lot, yr_built, condition, long, and zipcode. Later you can also drop any of these attributes if you decide not to use them.

Before you get any further into the project, you should split your testing data from your training data. You also need to separate the house prices from the dataset. Since price is the output the model should provide (the dependent variable), it should not be included in the training or testing dataset. However, you'll need to keep a copy of the labels so you can verify your results.

1. Split the data into training and testing sets and labels.

- a) Scroll down to view the cell titled **Split the data into training and testing sets and labels**, and examine the code listing beneath it.

Split the data into training and testing sets and labels

```
In [11]: from sklearn.model_selection import train_test_split
# Price is the dependent variable (value to be predicted), so it will be
# removed from the training data and put into a separate dataframe for labels.....
5
6 label_columns = ['price']
7
8 training_columns = ['sqft_living',
9     'grade',
10    'bathrooms',
11    'view',
12    'sqft_basement',
13    'bedrooms',
14    'lat',
15    'waterfront',
16    'floors',
17    'yr_renovated',
18    'sqft_lot',
19    'yr_built',
20    'condition',
21    'long',
22    'zipcode']
23
24 # Split independent and dependent variables.
25 data_train,data_test,data_train_labels,data_test_labels = train_test_split(data_raw[training_columns],
26                                         data_raw[label_columns],
27                                         random_state = 42)
28
29 # Compare the number of rows and columns in the original data to the training and testing sets
30 print(f'Original Set: {data_raw.shape}')
31 print('-----')
32 print(f'Training Features: {data_train.shape}')
33 print(f'Testing Features: {data_test.shape}')
34 print(f'Training Labels: {data_train_labels.shape}')
35 print(f'Testing Labels: {data_test_labels.shape}'')
```

- Line 1 imports the scikit-learn `train_test_split()` method, which will be used to split the raw dataset into separate training and testing datasets.
 - Line 6 identifies the column that you'll include in the dataframe that will contain the labels for machine learning. In this project, `price` is the *dependent* variable, which the model should learn to predict. So the `price` column will provide the labels that you'll use to train the model.
 - Lines 8 through 22 identify the columns that you'll include in the dataframe that contains the attributes (the *independent* variables) the model will use to determine the price. As you train and test the model, you may find that you need to combine, drop, or transform some of these to produce better results.
 - Lines 25 through 27 call the `train_test_split()` method to split various columns from the original dataset into four separate datasets:
 - `data_train` contains the independent variables that will be used to train the model.
 - `data_test` contains the independent variables that will be used to test the model after it has been trained.
 - `data_train_labels` contains the dependent variable (`price`) for each row used to train the model.
 - `data_test_labels` contains the dependent variable (`price`) for each row used to test the model.
 - The seed value provided in line 27 (42 in this example, provided as the `random_state` parameter) is used by the random number generator in the `train_test_split()` method to randomly shuffle the records.
- b) Examine the code that summarizes the split datasets.
- Lines 30 through 35 compare the number of rows and columns in the original data to those in the training and testing sets.
 - By default, the `train_test_split()` method will include 75% of the rows in the training set and 25% of the rows in the testing set. Since this code example doesn't specify percentages for the two sets, the default proportions will be used.

2. Run the code and view the results.

- Select the cell that contains the code listing, and select **Run**.
- Examine the output.

The datasets have been split as shown.

```
original Set: (21613, 21)
-----
Training Features: (16209, 15)
Testing Features: (5404, 15)
Training Labels: (16209, 1)
Testing Labels: (5404, 1)
```

- The training features and labels include 16,209 of the original 21,613 records, whereas the testing features and labels include only 5,404 records.
- The features datasets include 15 of the original 21 columns, whereas the labels include only 1 column (for price).

Summary

In this lesson, you learned how to collect and refine a machine learning dataset. You learned about various data sources, and issues that arise when aggregating, collecting, and cleaning data. You learned how to use various statistical measures and visualizations to analyze the dataset, and tasks you might need to perform to prepare a dataset for use in machine learning.

In your machine learning projects, what data sources will you use, and what sorts of tasks will be particularly challenging to collect, aggregate, and prepare that data for machine learning?

Will you primarily use existing data for your machine learning projects, or will you have to put new systems in place to collect the data you will use?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

3

Setting Up and Training a Model

Lesson Time: 2 hours, 10 minutes

Lesson Introduction

By the time you have identified the business problems you need to solve and have collected, prepared, and analyzed the dataset, you will have a pretty good idea of how you might implement an appropriate model. At this point, you can start setting up and training a model, experimenting with it, and tuning it to produce the type of outcome you're looking for, at a performance level that meets your requirements.

Lesson Objectives

In this lesson, you will:

- Select and implement an appropriate algorithm to solve a given business problem.
- Set up and train a machine learning model as needed to meet business requirements.

TOPIC A

Set Up a Machine Learning Model

To set up a machine learning model in an environment like Python, you must determine the algorithm that will produce the results you're after, and then use it to create a model based on your training data.

Design of Experiments

Experimentation is an important part of the AI practitioner's job. The kinds of problems addressed through machine learning are often too complex to be solved through the analysis methods typically used in traditional computer programming. Fortunately, you typically don't have to get it right the first time. With machine learning, you may produce a solution through a process of systematic experimentation.

Following the *Design of Experiments* approach (DOE, DOX, or *Experimental Design*—an approach used by data analysts, medical researchers, and others), you begin with a hypothesis, and then you systematically change the variables that *you can control* to see their impact on the variables you *can't directly control*.

When you perform such experiments, the variables you can directly change are called *independent variables*. They may also be called input variables or predictor variables. The variables that you don't control, which might change indirectly as a result, are called *dependent variables*. They may also be called output variables or response variables.

For example, through this type of experimentation, you might:

- *Determine which combination of independent variables will produce the best model for your needs.* You can experiment to see how training the algorithm using different combinations of input variables affects the performance of the model when it is applied to the evaluation data.
- *Select the best machine learning algorithm for your needs.* Use experimentation to compare the performance of different machine learning algorithms to see the impact on the estimated skill of the model.
- *Adjust settings of the learning algorithm to optimize its performance.* You can experiment to see how you can adjust various settings (called *hyperparameters*) to tune the performance of the model, and choose settings that produce the best model for your needs.

Hypothesis

In scientific experiments, the hypothesis is a provisional idea or educated guess that requires additional investigation, experimentation, or evaluation to be proved true or false. In machine learning, the *hypothesis* is a candidate machine learning model that you create to test its performance, particularly whether it is able to produce the outcome that you require. You program the hypothesis to employ a particular function that you hope will produce the intended outcome. Like any hypothesis, you use it as the basis for additional investigation, experimentation, and evaluation.

Through experimentation, you improve upon the hypothesis. You provide the hypothesis with a *sample* of historical data to train it, which produces a model. You then test the model's performance—for example, trying it out on a new dataset and evaluating the effectiveness of its predictions based on that data.

When you have refined the model through experimentation and have a result that can serve as your finished solution, you have essentially produced a *target function*, which you can use to find output data (answers) for inputs to real problems.

Hypothesis Testing

An important part of your role as a machine learning practitioner is to select the best model for a given task. For example, when you apply two machine learning methods to predict outcomes on a dataset, you have to choose one model over the other, presumably the model whose effectiveness best meets your requirements. In the case of a prediction model, this probably means the model with the best *estimated* effectiveness when making predictions on new data. The problem is, the effectiveness estimate is just that—an estimate. The difference in estimated effectiveness could be real, or it might be due to statistical chance. Fortunately, statistical hypothesis testing can help you determine which one is actually better in practice.

Hypothesis testing focuses on the **null hypothesis**, which is the assumption that there is no statistically significant (i.e., real) difference between the two models under comparison. So there are two outcomes of such a test:

- **Insufficient evidence to reject the null hypothesis.** In other words, observed differences in model effectiveness are likely due to statistical chance.
- **Sufficient evidence to reject the null hypothesis.** In other words, observed differences in model effectiveness are likely due to a difference in the models.

Consider how you might train two models on the same data, where one model has undergone additional dimensionality reduction to remove a feature that appears to be noisy. The reduced model may end up giving you a better result (whatever specific result you may be looking for). Due to the probabilistic nature of machine learning algorithms, it's entirely possible that the "improvement" in the reduced model is due to random chance. You must therefore test the models to verify this. If it turns out that the change in results was due to *real* differences, you can reject the null hypothesis. Otherwise, you cannot—and if you can't reject the null hypothesis for the new model, you can't be confident that the model is truly more effective.



Note: It's important to understand that you're not *accepting* the null hypothesis; you are merely failing to reject it. You cannot say with absolute certainty that the null hypothesis is true, so it's not correct to say you accepted it.

Type I and Type II Errors

Because hypothesis testing is done on sample data and not an entire population, the tests are susceptible to errors. These errors are classified as follows:

- **Type I**—You rejected the null hypothesis, but the null hypothesis was actually true.
- **Type II**—You failed to reject the null hypothesis, but the null hypothesis was actually false.

Hypothesis Testing Methods

There are many methods that you can use to conduct a hypothesis test, some of which are more relevant than others when applied to certain problems. The following table briefly describes some of the most prominent methods.

Testing Method	Description
A/B test	An A/B test compares two different values of the same variable in order to determine which value is most effective. A common example is serving online customers two different versions of the same web page, and then choosing the page that leads to the best conversion. So, your hypothesis might be that adding a specific UI widget increases conversion rates, whereas the null hypothesis would state that this widget has no real effect on conversion rates.

Testing Method	Description
<i>z-test</i>	A <i>z</i> -test is used to compare the mean of two distributions when the standard deviation of a population is known. For example, among a population of students, the mean score on an exam is 76. If you selected a sample of 40 students from this population, the mean of this sample might be 78. You'd then compare this sample mean to the mean of some other random sampling of 40 students to see if the increase in score is significant. In other words, the null hypothesis is that the selected 40 students have comparable exam scores to a random sampling of students. A <i>z</i> -test is performed by calculating the standard deviation of the sample, then using this to calculate the aforementioned <i>z</i> -score. The <i>z</i> -test is most applicable to larger sample sizes, typically above 30.
<i>t-test</i>	A <i>t</i> -test is an alternative to the <i>z</i> -test in that it compares the mean of two distributions in which the population standard deviation is <i>not</i> known. A <i>t</i> -test estimates the population standard deviation by incorporating the standard deviation of the sample. The <i>t</i> -test is most applicable to smaller sample sizes, typically below 30.
<i>Analysis of variance (ANOVA)</i>	An ANOVA test compares the mean of multiple distributions. In the standard approach, an ANOVA test evaluates the effect that a single independent variable has on three or more sample groups. For example, among a population of medical patients, you can test the effect of a specific type of treatment. The null hypothesis would state that the specified treatment has the same effect on one random sampling as it does on multiple other random samplings. ANOVA as a single test is more useful than conducting multiple <i>t</i> -tests for each sample, as ANOVA considers the variation within and between all of the samples. This minimizes the chance of errors.
<i>Chi-squared test</i>	A chi-squared (χ^2) test compares the effect of categorical variables. For example, you might categorize berries as being the color white or the color red. You also categorize berries as being poisonous or not poisonous. A chi-squared test attempts to answer the question, "Does the color of a berry affect whether or not it's poisonous?" If yes, then the two categorical variables are said to be dependent. The null hypothesis of such a test is that the variables are independent—i.e., one variable has no significant effect on the other.

p-value

The results of a hypothesis test should help you reject or fail to reject the null hypothesis, all while avoiding type I and type II errors as much as possible. One common testing output is a ***p*-value**. The *p*-value is the probability of obtaining a result from the test given that the null hypothesis is true. In other words, the *p*-value enables you to determine whether or not to reject or fail to reject the null hypothesis if the value is lower or higher than a specified level of statistical significance. This level of significance, also referred to as the alpha value, is something you must determine beforehand. There is no "right" alpha, but it's common to set the alpha at 0.05 (5%). Ultimately, you compare the *p*-value returned from the test to the alpha value you decided on earlier. This will lead to two possible outcomes:

- If ***p*-value > alpha**, fail to reject the null hypothesis.
- If ***p*-value <= alpha**, reject the null hypothesis.

So, if your *p*-value is 0.03, and you decided on an alpha of 0.05, you'd reject the null hypothesis.

Note that *p*-value is *not* the probability that the null hypothesis is true or false. Nor does it definitively tell you whether or not the null hypothesis is true or false. It simply provides you with a level of confidence about either rejecting or failing to reject the null hypothesis. It is for this reason that some statisticians have suggested alternative ways to measure confidence.

Alpha and Beta Values

Another way to think about the alpha value is that it is the rate of type I errors. Likewise, there is also a beta value, which is the rate of type II errors. Beta is not as commonly used as alpha, because reducing type I errors is often more important to the practitioner than reducing type II errors. However, you may still want to consider the beta value during hypothesis testing.

Confidence Interval

A **confidence interval** returns a range of values for which there is some specified probability that the true value lies within that range. The specified probability is called the confidence level; a confidence level of 95% is commonly used. This means that there is a 95% probability that the range of values (the interval) will contain the true value. So, if the interval is (10, 30) for a range of mean values at a confidence level of 95%, there is a 95% chance that a true mean value is not going to be less than 10 or greater than 30. Unlike *p*-values, confidence intervals are able to show the likely effects in the population; if a value lies outside the interval, there is good evidence that it does not exist in the population. Confidence intervals and *p*-values are not mutually exclusive, however, and you can certainly choose to use both when deciding whether or not to reject the null hypothesis.

Machine Learning Algorithms

When you set up your initial machine learning model (your hypothesis), you'll have to select an algorithm or algorithms that you'll use to produce the outcome you require. Note that some outcomes may be supported by several different algorithms. For example, if you needed to perform a regression task such as predicting a person's life expectancy based on various inputs—lifestyle factors, date of birth, gender, and so forth—you could use linear regression, random forest, logistic regression, or one of several other algorithms.

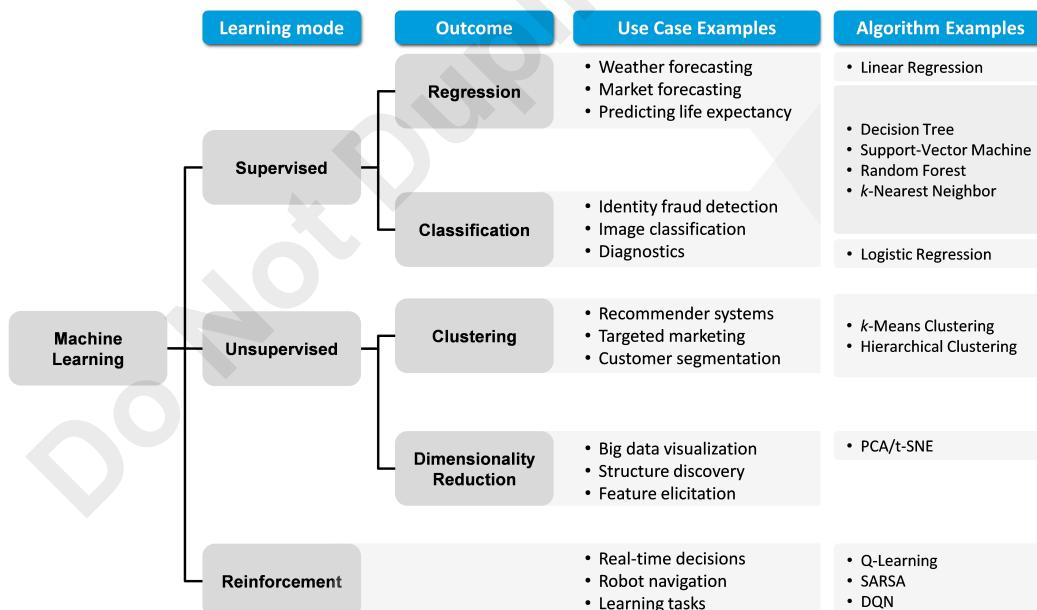


Figure 3-1: Example machine learning algorithms.

Some algorithms, depending on how they are used, can support various types of outcomes. For example, a random forest could be used for various types of classification *or* regression tasks.



Note: It helps to think about the algorithms you might use even before you are ready to apply them, since it may affect how you need to prepare data.

Algorithm Selection

Selecting the right algorithm for a machine learning model can seem overwhelming because there are numerous supervised and unsupervised machine learning algorithms, each one employing its own approach to the learning process. The "right" algorithm depends on the situation, and there may be more than one algorithm that solves a particular problem. Even highly experienced data scientists may not be able to immediately select the best algorithm for a particular situation without some experimentation.

Guidelines for Setting Up a Machine Learning Model

Machine learning models are commonly created and refined over multiple iterations. It may take you some time to achieve a model that meets your requirements. Consider these guidelines as you evaluate your machine learning models.



Note: All of the Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Factors that Guide Evaluation of a Machine Learning Model

When evaluating a machine learning model, consider factors such as:

- How well the model meets use cases and other requirements
- How much preprocessing is required
- How effective the model is
- How explainable the model is
- How long it takes to build a model
- How long it takes to make predictions using the model
- How well the model can scale to handle expected quantities of data

ACTIVITY 3-1

Setting Up a Machine Learning Model

Data File

/home/student/CAIP/Workflow/Workflow-Housing.ipynb

Before You Begin

If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the **CAIP/Workflow/Workflow-Housing.ipynb** notebook. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled "Build and test a linear regression model - Round 1". Select **Cell→Run All Above**.

Scenario

You have analyzed the dataset and are now ready to create your first pass at a machine learning model—your hypothesis, which you will use for experimentation and testing. You will start by setting up a linear regression model.

1. Why would you use a linear regression algorithm to produce a real estate price estimator?

2. Create a linear regression model and fit it using the training features and labels.

- a) Scroll down to view the cell titled **Build and test a linear regression model - Round 1**, and examine the code listing beneath it.

Build and test a linear regression model - Round 1

```
In [ ]: 1 from sklearn.linear_model import LinearRegression
          2 from time import time
          3
          4 # Create a linear regression model
          5 regressor = LinearRegression()
          6
          7 # Fit the model using training data and labels
          8 start = time()
          9 regressor.fit(data_train, data_train_labels);
         10 end=time()
         11 train_time = (end - start) * 1000
         12 print('Model took {:.2f} milliseconds to fit.'.format(train_time))
```

- Line 1 imports the scikit-learn `LinearRegression` library, which will be used to create a linear regression model.
 - Line 2 imports the `time` library. You will use this to profile the model's performance, timing how long it takes the model to train.
 - Line 5 creates an object from the `LinearRegression()` class, which provides the algorithm that will create the model.
 - Line 8 starts the timer.
 - Line 9 trains the regression model, providing the training data and labels.
 - Lines 10, 11, and 12 calculate and report the time it takes to fit the model.
- b) Select the cell that contains the code listing, and select **Run**.
- c) Observe the output.
- The model is quickly processed to fit the dataset you provided, with the time it took shown in the output.
 - The model now resides in memory (in the `regressor` variable). Of course, now you need to test that the model is actually able to make predictions. You can use the testing dataset for this purpose.



Note: As you can see, except in the case of very large and complicated datasets (which can take several days to process), it may take significantly less time to actually set up the model than it takes to prepare the dataset.

3. Use the holdout dataset to test the model.

- a) Scroll down to view the cell titled **Use the holdout dataset to test the model**, and examine the code listing beneath it.

Use the holdout dataset to test the model

```
In [ ]: 1 # Evaluate the model's performance using test data and labels
          2 score = regressor.score(data_test, data_test_labels)
          3 'Score: {:.%}'.format(int(round(score * 100)))
```

- Line 2 calls the `LinearRegression` object's `score()` function to run a set of predictions on each row of house data in the testing features dataset (`data_test`). It compares those predictions to the actual prices of those houses, which are in `data_test_labels`.
 - The `score()` function uses an appropriate algorithm to rate the performance of the regression model. The best possible score that can be returned is 100%. The score can be negative.
 - Line 3 outputs the score.
- b) Select the cell that contains the code listing, and select **Run**.

- c) Examine the score for your linear regression model.

```
'Score: 70%'
```

You may be able to make some adjustments to improve on this score.

4. Compare predicted prices to the actual prices.

- a) Scroll down to view the cell titled **Compare predicted values to actual values**, and examine the code listing beneath it.

Compare predicted values to actual values

```
In [ ]: 1 predicted_prices = regressor.predict(data_test)
          2 predictions = data_test_labels.copy()
          3 predictions['predicted'] = predicted_prices
          4
          5 # View examples comparing actual prices to predicted prices
          6 with pd.option_context('float_format', '${:,.2f}'.format): print( predictions.head(10) )
```

- Line 1 calls the `predict()` function to generate a new dataframe containing predicted prices for the houses in the test dataset.
 - Line 2 makes a copy of the dataframe that contains the labels (correct answers), and line 3 then adds the predicted prices to the dataframe as a new column. Having both columns side by side in the same dataframe will make it easy to compare them.
 - Line 6 will display the first 10 rows of the dataframe, so you can see a sample of the results comparing actual prices to the predicted prices. It applies formatting to show the numbers as U.S. dollars with commas and two decimal places.
- b) Select the cell that contains the code listing, and select **Run**.
 c) Examine the output.

	price	predicted
735	\$365,000.00	\$451,576.87
2830	\$865,000.00	\$745,528.73
4106	\$1,038,000.00	\$1,234,144.11
16218	\$1,490,000.00	\$1,659,505.67
19964	\$711,000.00	\$737,851.90
1227	\$211,000.00	\$284,352.79
18849	\$790,000.00	\$832,187.44
19369	\$680,000.00	\$490,462.65
20164	\$384,500.00	\$392,922.89
7139	\$605,000.00	\$471,310.48

- The first ten actual prices can be compared with the predicted prices.
- While the predicted prices and actual prices correlate roughly, the current level of accuracy may be inadequate. The model could use some improvement.

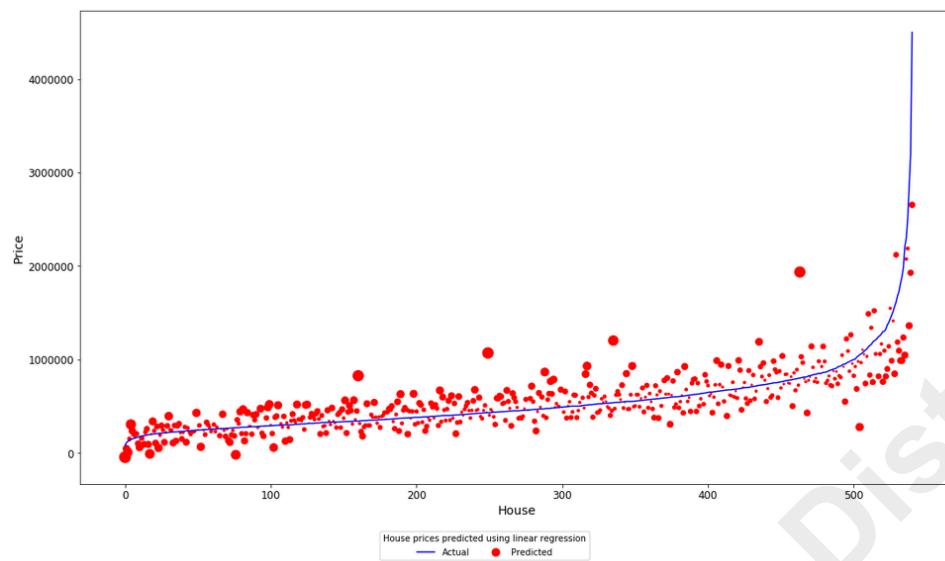
5. Create a chart to compare the predictions to the actual values.

- a) Select the next code cell, and examine the code.

```
In [ ]: 1 def compare_pred_to_actual(chart_description):
2
3     N = 10 # Plot every Nth value to save time and space
4     pred_df = predictions.sort_values('price')[::N]
5
6     pred_df['diff'] = pred_df['price'] - pred_df['predicted']
7     pred_df['recnum'] = np.arange(len(pred_df))
8     pred_df['error_pct'] = abs(pred_df['diff']/pred_df['price'])*100
9
10    ax = plt.figure(figsize=[18,10])
11    plt.ylabel('Price')
12    plt.xlabel('House')
13    plt.plot(pred_df['recnum'], pred_df['price'], color='blue');
14    plt.scatter(pred_df['recnum'],
15                pred_df['predicted'],
16                pred_df['error_pct'],
17                color='red');
18
19    ax.legend(['Actual','Predicted'],
20              loc='lower center',
21              ncol=2,
22              title=chart_description)
23
24    plt.show()
25
26 # Compare the predicted prices to actual prices
27 compare_pred_to_actual('House prices predicted using linear regression')
```

- The `compare_pred_to_actual()` function is defined in lines 1 through 24. This function generates a combination line/scatter chart that will compare predicted prices (shown as dot markers) to actual prices (plotted in a line).
- Lines 3 and 4 will create a subset of house records that samples every 10th record. This will remove some of the detail from the chart, but will save processing time and memory, and will still give a good impression of the model's effectiveness.
- Lines 6 through 8 add columns to the dataset, calculate the difference between the predicted and actual price, adding a record number which can be plotted along the x-axis, and calculating the percent of difference between the actual and predicted price.
- The statements in lines 10 through 17 plot the line chart and scatter chart.
- The statement in lines 19 through 22 generates a legend. The chart description passed into the function is shown as the legend's title.
- Line 24 displays the chart.
- Line 27 calls the function defined in lines 1 through 24, passing in a chart description that notes the chart shows prices predicted using a linear regression model.

b) Select Run.



- The chart shows a blue line representing actual prices and red dots showing predicted prices.
- The greater the percent of error, the larger the dot.
- You can use this chart as a baseline to visually compare the quality of predictions as you improve the model. As the model improves, the dots should get smaller and clustered more closely to the blue line.

TOPIC B

Train the Model

After the initial setup, it may take multiple tests and refinements to produce a model that meets your requirements.

Iterative Tuning

As you work with machine learning models, you must constantly attend to various challenges that will lead to ineffective and faulty results. Machine learning is probabilistic, based on variance in the data. So a typical approach to improving the effectiveness of a model is to use different samples for training and testing, to reflect the types of challenges the model will encounter when used with real data. As you test the model over multiple iterations, you'll make refinements, and test it again.

In statistics, it is often said that "all models are wrong—but some are *useful*." The point is that statistical models will always contain some degree of error, due to variations based on probability. In machine learning, the goal is to produce a model that is correct *enough* as to be useful, even though the predictions may not be exactly correct or always correct. A model that is useful for its intended task is commonly described as *skillful*. There are degrees of skill; some models are more useful than others. Improving a model's skill is the ultimate goal of the iterative tuning process.

Bias

As you progress through the machine learning workflow, you should be careful to avoid *bias*. Bias is a systematic error in data analysis that results in a faulty model.

For example, a *selection bias* occurs when the training dataset doesn't truly represent the population the model will ultimately be applied to. For example, suppose you are developing a model to predict heating costs for a building. Your selection of training data might bias the results if it covers only two months out of a year, or if it came from a single year that was unusually cold.

Reporting bias occurs when the training data is missing observations that were not reported. For example, some types of crimes may be more likely to be reported than others. Using data with a reporting bias to train a machine learning model may skew the outcome.

Attrition bias occurs when the training data excludes participants that dropped out over time. For example, a model might be developed to predict the success rate of a particular treatment over time. Some participants might drop out of treatment, and a well-intentioned data analyst might remove their data from the training dataset under the rationale that the outcomes for dropouts would skew results. However, the fact that some participants will not complete the treatment program is an important part of the predictions the model should make.

You must always be careful not to introduce bias. Using the testing dataset will help you prove the skill of your model. However, you should extract the testing dataset before you become too familiar with the data it contains to ensure your decisions during the modeling process are not influenced by data you examine within the testing set. This undesirable influence, called a *data snooping bias*, may (consciously or unconsciously) lead you to tailor the model to the testing dataset, thereby making it less skillful for new datasets you will process later on.

Compromises

Tuning a model typically involves making compromises, and finding the "sweet spot" that results in a model that meets business requirements. While you may strive for perfection in your job, as an AI practitioner, this may translate to meeting business requirements and not getting perfect scores for your machine learning models.

For example, you might compromise some of the model's performance in favor of:

- **Simplicity of the data pipeline**—With "just a little more data" you might be able to build an even better model. On the other hand, you might end up creating a complex data pipeline that is time consuming, hard to maintain, and way more complicated than you really need. In many cases it may be sufficient to have just enough skill to get actionable insights from the model.
- **Model interpretability**—When humans delegate decisions to machine learning processes, in many cases humans are still accountable to understand and explain the rationale for those decisions. Some models may produce slightly better performance at the expense of model interpretability. In some cases, the additional performance may not be worth the loss of being able to understand and explain decisions made by the model.
- **Resources needed to train and operate a model**—You may be able to gain some performance points by using additional resources, more time, more memory, more CPUs or GPUs, and so forth. But fewer resources may produce adequate performance while saving time and money.

Model Generalization

When you train a machine learning prediction model, you'll typically want it to make a reasonably good prediction on any new datasets that it might encounter—beyond the dataset that was originally used to train it. This characteristic is called **generalization**.

When you use a particular dataset to train a model, you run the risk of **overfitting** the model to work with the dataset you're using for training. In other words, you may tune the model to very closely match the unique data in the training dataset, but find that the model does not generalize well to new datasets.

When you have overfitted the data, there are a few approaches you might take to resolve the problem:

- Work with a larger sample of data.
- Use regularization to reduce some of the model's sensitivity to noise—for example, by using a simpler model with fewer parameters or using fewer attributes in the training data.
- Clean and prepare the data better to reduce noise in the data itself.

Cross-Validation

As you shift from training a model with the training dataset to testing the model on new data, you'll need a way to estimate how well the model performs—for example, to ensure that it hasn't been overfitted or underfitted to patterns in the training dataset, or influenced by noise in the data. After all, a model may perform well on training data but not do so well when it's faced with a new dataset it hasn't seen before.

Cross-validation (also called rotation estimation or out-of-sample testing) provides a way to measure how well the model is able to generalize to new test data. There are actually several cross-validation techniques:

- The holdout method
- k -fold cross-validation
- Stratified k -fold cross-validation
- Leave- p -out cross-validation

k -Fold Cross-Validation

In **k -fold cross-validation**, the data is split into k groups (folds). One group is the test set, and the remaining groups form the training set. The model trains and then evaluates its performance. Then, the groups rotate: a different group is designated as the test set, and the rest are used in the training set. Once again, the model trains and evaluates its performance. This process repeats k times. Then,

the average error across all of these trials is calculated. The advantage of this approach is that it minimizes variance, as every data point is used to both train and test at some point. However, because the training and testing must be done k times, this method adds overhead to both time and processing power. A practical rule of thumb is to set k between 5 and 10.

A related method is **stratified k -fold cross-validation**. This scheme helps to minimize variance and bias issues by ensuring that each train/test fold is a good representation of the data as a whole. In binary classification, if 30% of the data is in class 1 and 70% is in class 2, then 30% of the data in each fold will be in class 1, and 70% will be in class 2. The stratification method is therefore most suitable in cases of class imbalance.

Leave- p -Out Cross-Validation

Leave- p -out cross-validation (LPOCV) is essentially the k -fold method, but with k equal to all data points in the set (n). So, $n - p$ data points are used in training, and p data points are used to test. This is repeated for all possible combinations of data that fit this split. Like with k -fold, the average error across these trials is then calculated.

It is common to leave p at 1, because as p increases, the amount of trial combinations increases dramatically, leading to significant performance issues. This approach is also called **leave-one-out cross-validation (LOOCV)**. LOOCV is commonly used to minimize bias in smaller datasets, though it may not perform well with larger datasets due to increased variance and performance concerns.



Note: Performance issues may not only be due to the size of the dataset. The complexity of the calculations being performed also has some bearing on performance.

ACTIVITY 3-2

Dealing with Outliers

Before You Begin

If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the **CAIP/Datasets/Workflow-Housing.ipynb** notebook. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled "Identify outliers". Select **Cell→Run All Above**.

Scenario

One way to improve a model is to check your training data for noise such as outliers, which may mislead the algorithm and hinder the fit. When you find faulty data values, you can determine the best way to handle them—for example, by correcting them or by simply dropping them from the dataset.

When you observed the histograms for price and bedrooms, there was a lot of white space on the right side of the chart. You'll investigate these features to see if they have outliers in the high end of the range.

1. Identify outliers that you must deal with.

- Scroll down to view the cell titled **Identify outliers**, and examine the code listing beneath it.

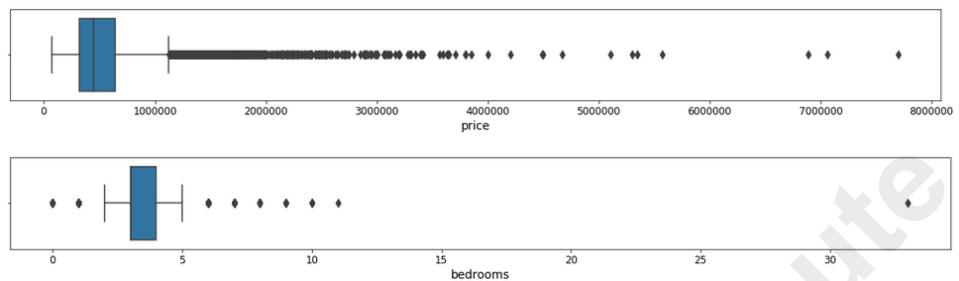
Identify outliers

```
In [ ]: 1 feature_list = ['price', 'bedrooms']
          2
          3 for feature in feature_list:
          4     plt.figure(figsize=(20,2))
          5     bplot = sb.boxplot(x=feature, data=data_raw, orient="h", fliersize=7)
```

- Line 1 identifies the features that you will include in a box plot to find outliers.
- Line 3 iterates through each feature in the list you defined in line 1, repeating the steps in lines 4 and 5 for each feature.
- Line 4 specifies the figure size available to the box plot, and line 5 actually generates it, specifying the name of the feature to be plotted, the dataframe that holds the data, and formatting options for the box plot.

- Select the cell that contains the code listing, and select **Run**.

- c) Examine the output.



- In both boxplots, the colored boxes show where the middle 50% of the values occur. The vertical lines on either side of the boxes (the "whiskers") show the boundaries of the minimum ($Q1 - 1.5 \times IQR$) and maximum ($Q3 + 1.5 \times IQR$) for the distributions.
- The upper boxplot shows the distribution of `price` values in the original dataset. There is a long tail on the right, with the distribution tapering off around \$5,500,000, followed by a gap, and three more outliers on the far right.
- The lower boxplot shows the distribution of `bedrooms` values in the original dataset. The distribution tapers off around 11 bedrooms, and then there is a single outlier with more than 30 bedrooms. That value is likely an error.

2. Examine the data values for the outliers.

- a) Scroll down to view the cell titled **Examine data values in the outliers**, and examine the two code cells beneath it.

Examine data values in the outliers

```
In [ ]: 1 # Houses with a value above $6,000,000
          2 data_train.loc[data_train_labels['price'] > 6000000]

In [ ]: 1 # Houses with more than 11 bedrooms
          2 data_train.loc[data_train['bedrooms'] > 11]
```

- In the first code cell, Line 2 contains a statement that will return a subset of records from `data_train`. For any row in the labels where the price is more than 6 million dollars, the corresponding row from the training dataset will be returned.
 - Line 2 in the second code cell does something similar, returning records from the training set where the number of bedrooms is more than 11.
 - When run, the statements in these two cells will display lists of the outlier values you saw in the boxplots.
- b) Select the first (upper) code cell, and select **Run**.
- c) Select **Run** again to run the second cell.

- d) Examine the data values for the outliers.

1 # Houses with a value above \$6,000,000
2 data_train.loc[data_train_labels['price'] > 6000000]
sqft_living grade bathrooms view sqft_basement bedrooms lat waterfront floors yr_renovated sqft_lot yr_built condition long zipcode
3914 10040 11 4.50 2 2360 5 47.6500 1 2.0 2001 37325 1940 3 -122.214 9800
9254 9890 13 7.75 4 1030 6 47.6305 0 2.0 0 31374 2001 3 -122.240 9803
7252 12050 13 8.00 3 3480 6 47.6298 0 2.5 1987 27600 1910 4 -122.323 9810
1 # Houses with more than 11 bedrooms
2 data_train.loc[data_train['bedrooms'] > 11]
sqft_living grade bathrooms view sqft_basement bedrooms lat waterfront floors yr_renovated sqft_lot yr_built condition long zipcode
15870 1620 7 1.75 0 580 33 47.6878 0 1.0 0 6000 1947 5 -122.331 9810



Note: You may have to scroll slightly to the right and left to see all of the columns.

- In the upper listing, the `price` values are not shown since the prices are stored in the separate labels dataset, but you can see that all three of these (most expensive) houses have a high grade, large number of bedrooms, bathrooms, and high square footage.
- The lower listing shows a house with 33 bedrooms. It only has 1,620 square feet of living space, so this value is almost certainly incorrect. You could either fix the value or just drop the record from the training set.



Note: Because the dataset is large, these outliers may not have much impact on the model, but it won't hurt to remove them from the training set either. Doing so in this activity will enable you to see how it's done in Python.

3. Drop outliers from the training dataset.

- a) Scroll down to view the cell titled **Drop outliers from the training dataset**, and examine the code cell beneath it.

Drop outliers from the training dataset

```
In [ ]: 1 print(f'{len(data_train):d} houses in the training dataset')
2
3 # Keep only the rows for houses priced $6M or less
4 data_train = data_train.loc[data_train_labels['price'] <= 6000000]
5 data_train_labels = data_train_labels.loc[data_train_labels['price'] <= 6000000]
6 print(f'{len(data_train):d} houses remain after dropping those priced over $6M')
7
8 # Keep only the rows for houses with 11 or fewer bedrooms
9 data_train_labels = data_train_labels.loc[data_train['bedrooms'] <= 11]
10 data_train = data_train.loc['bedrooms' <= 11]
11 print(f'{len(data_train):d} houses remain after dropping those with more than 11 bedrooms')
```

- Line 4 filters the training dataset to include only records whose price is less than six million dollars.
 - Line 5 performs the same filter operation on the training labels. It is important to keep these two sets parallel, dropping exactly the same rows from each.
 - Lines 9 and 10 are similar to lines 4 and 5, dropping records from the training dataset and labels to remove any records for houses with more than 11 bedrooms.
 - The print statements in lines 1, 6, and 11 provide a before-and-after summary so you can see the results of dropping the outliers.
- b) Select the cell that contains the code listing, and select **Run**.

- c) Examine the output.

```
16209 houses in the training dataset
16206 houses remain after dropping those priced over $6M
16205 houses remain after dropping those with more than 11 bedrooms
```

After dropping the four outliers, 16,205 records remain in the dataset.

Feature Transformation

Items represented using similar numbers are easier to compare. For example, `age` and `income` would have very different ranges of values. But if you scaled the values in both columns so the maximum `age` and `income` were both 1, they would be easier to analyze and compare from a statistical perspective. To find someone with an average `age` and `income`, you could look for values near .5 for both features. If you were looking for a young person with a high income, you could look for an `age` of less than .5 and an `income` near 1. Once you find the records you're looking for, you can refer back to their original data values to determine the actual `age` and `income`.

It helps to represent values in a similar scale when you are working directly with numbers, but it also helps when you are using visualization tools. When you create charts, it will be easier to plot similarly scaled features for direct comparison.

In addition to scale, in some cases it can help to adjust the way data is *distributed*. Even though skewed distributions are not considered normal according to statistical measures, they are in fact all too common in real world data, and they can make it cumbersome to perform statistical analysis.

Various types of mathematical transformations can be applied to data to remove skew and normalize the scale. **Feature transformations** often facilitate data analysis—not only by humans, but by machine learning algorithms as well. Applying transformations to machine learning data can make some patterns in data more evident, thereby improving the algorithm's performance.

Transformation Functions

Various functions can be used to transform features. Which function you should choose is based on the circumstances. A few examples of such functions are described here.

Function	Description
Normalization	<ul style="list-style-type: none"> Scales data items to fit within a range such as 0 to 1. Various methods may be used, but can be obtained through division. For example, if the minimum data value is 0, divide each item by the maximum. If the minimum is not 0, subtract the minimum from each item, and then divide by the difference between maximum and minimum. May leave the distribution skewed.
Log	<ul style="list-style-type: none"> Obtained by calculating the log base 10 of x, log base e of x ($\ln(x)$), or log base 2 of x. Reduces skewness. Can be used only on positive numbers.
Cube Root	<ul style="list-style-type: none"> Obtained by raising each data item to a power of $1 / 3$. Reduces skewness, but not as strongly as a log function.



Note: These are just a few common examples. Depending on the circumstances, various other functions may be appropriate, such as sigmoid, hyperbolic tangent, reciprocals, Box–Cox, Yeo–Johnson, O'Brien, z-score, and so forth. Many different transformation functions are implemented in Python through libraries such as numpy and scipy.stats.

Do Not Duplicate or Distribute

ACTIVITY 3–3

Scaling and Normalizing Features

Data File

/home/student/CAIP/Workflow/Workflow-Housing.ipynb

Before You Begin

If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the **CAIP/Workflow/Workflow-Housing.ipynb** notebook. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled "Show statistics for the training features". Select **Cell→Run All Above**.

Scenario

As you continue with your optimization of the model, you will look for additional problems with the training data. You will now look for data values that should be adjusted to produce a better linear regression model.

1. Compare the ranges of sqft_living and price.

- Scroll down to view the cell titled **Show statistics for the training features**, and examine the code listing beneath it.

Show statistics for the training features

```
In [ ]: 1 # Show statistics for the features we'll be using, to prepare for feature scaling.
2 with pd.option_context('float_format', '{:.2f}'.format):
3     print(data_train['sqft_living'].describe(), '\n')
4     print(data_train_labels['price'].describe())
```

- Line 2 sets the context for lines 3 and 4, so number values will be shown with two decimal places.
 - Line 3 outputs descriptive statistics for the values in the `sqft_living` column.
 - Line 4 outputs descriptive statistics for the values in the `price` column.
- Select the cell that contains the code listing, and select **Run**.

- c) Examine the output.

```

count    16205.00
mean     2071.71
std      899.46
min      290.00
25%     1427.00
50%     1910.00
75%     2544.00
max      9640.00
Name: sqft_living, dtype: float64

count    16205.00
mean     536227.60
std      348666.79
min      75000.00
25%    320000.00
50%    450000.00
75%    640000.00
max     5110800.00
Name: price, dtype: float64

```

- The minimum and maximum values for `sqft_living` are 290 and 9,640.
- The minimum and maximum values for `price` are 75,000 and 5,110,800.
- The scale of these values is quite different.

2. Examine the distribution of `sqft_living` and `price`.

- a) Scroll down to view the cell titled **Compare the scale and distribution of price and sqft_living**, and examine the code listing beneath it.

Compare the scale and distribution of price and sqft_living

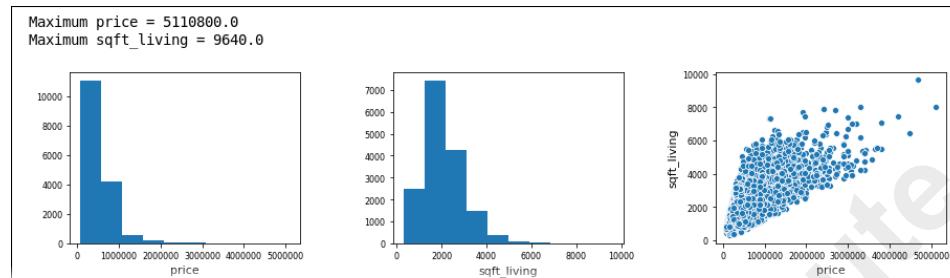
```

In [ ]: 1 # Compare scale and distribution of price and sqft_living
2 def compare_price_sqft():
3
4     print('Maximum price =', data_train_labels.loc[data_train_labels['price'].idxmax()]['price']);
5     print('Maximum sqft_living =', data_train.loc[data_train['sqft_living'].idxmax()]['sqft_living']);
6
7     fig = plt.figure(figsize=(15,3))
8     fig.subplots_adjust(wspace=.4)
9
10    plt.rc('axes', titlesize=9) # fontsize of the axes title
11    plt.rc('axes', labelsize=11) # fontsize of the x and y labels
12    plt.rc('xtick', labelsize=8) # fontsize of the tick labels
13    plt.rc('ytick', labelsize=8) # fontsize of the tick labels
14
15    ax1=fig.add_subplot(1, 3, 1)
16    plt.xlabel('price')
17    plt.hist(data_train_labels['price'], label='price');
18
19    ax2=fig.add_subplot(1, 3, 2)
20    plt.xlabel('sqft living')
21    plt.hist(data_train['sqft_living'], label='sqft_living');
22
23    # View relationship between price and sqft_living
24    ax2=fig.add_subplot(1, 3, 3)
25    sb.scatterplot(x=data_train_labels['price'], y=data_train['sqft_living']);
26
27 compare_price_sqft()

```

- Lines 4 and 5 show the maximum values for `price` and `sqft_living`.
 - Lines 7 through 13 prepare to create a visualization.
 - Lines 15 through 17 plot a histogram to show the distribution of `price` values.
 - Lines 19 through 21 plot a histogram to show the distribution of `sqft_living` values.
 - Lines 24 and 25 plot a scatterplot to show the relationship between `sqft_living` and `price`.
 - All of the lines described above are contained within the `compare_price_sqft()` function, which is called in line 27 to generate the charts.
- b) Select the cell that contains the code listing, and select **Run**.

- c) Examine the output.



- The maximum `price` is 5,110,800, while the maximum `sqft_living` is 9,640. These values are in very different scales.
- The left two charts show the distributions of `price` and `sqft_living` skewing to the right.
- The chart on the right, the scatterplot, shows a cone-shaped dispersion. This occurs because the values for `sqft_living` and `price` spread differently. Statisticians call this condition **heteroscedasticity**, and it limits the effectiveness of linear regression.
- To optimize the linear regression model, you should transform the values in these two columns to be more similar in scale and spread.

3. Transform `price` and `sqft_living` to be more similar in scale and spread.

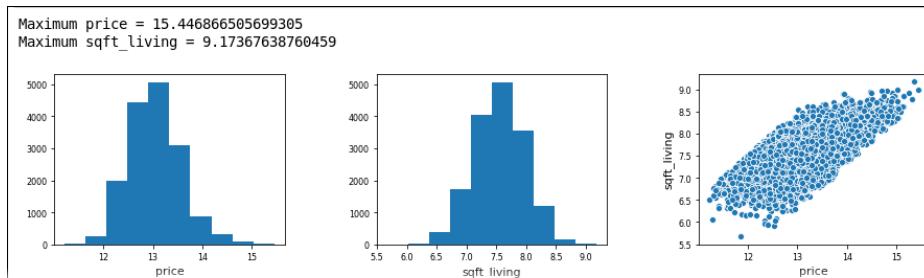
- a) Scroll down to view the cell titled **Transform price and sqft_living and compare results**, and examine the code listing beneath it.

Transform price and sqft_living and compare results

```
In [ ]: 1 # Apply a log transformation to scale price and sqft_living
2 data_train['sqft_living'] = np.log(data_train['sqft_living'])
3 data_train_labels['price'] = np.log(data_train_labels['price'])
4
5 # Log transformation must be applied to test dataset as well
6 data_test['sqft_living'] = np.log(data_test['sqft_living'])
7 data_test_labels['price'] = np.log(data_test_labels['price'])
8
9 # Compare scale and distribution of price and sqft_living
10 compare_price_sqft()
```

- Lines 2 and 3 apply a log transformation to the two columns in the training dataset.
 - Lines 6 and 7 apply the log transformation to the same columns in the testing dataset.
 - Line 10 calls the `compare_price_sqft()` function again, so you can see the results of the log transformation on the two columns.
- b) Select the cell that contains the code listing, and select **Run**.

c) Examine the output.



- After transformation, the maximum `price` is about 15.45, and the maximum `sqft_living` is 9.17. These values are now in a much more similar scale than they were before.
- Both `price` and `sqft_living` now show a normal distribution curve, with the skew eliminated.
- The rightmost chart, the scatterplot, is no longer cone-shaped. The highest and lowest edges are roughly parallel. This makes it more suitable for linear regression.

The Bias–Variance Tradeoff

When you develop a machine learning model, your goal will be to find the "sweet spot" that makes a good compromise between high bias and high variance. The model may err on the side of *high bias* or *high variance*, and these two types of errors are inversely related. In other words, as one increases, the other decreases. The sweet spot is the point where the smallest number of total errors are produced.

High bias refers to assumptions you make to simplify a model that make the target function less complex, easier to understand, and easier for a machine to learn. Some algorithms, such as linear regression, tend to err on the side of high bias. High bias models tend to have lower predictive performance on complex problems that fail to meet the simplifying assumptions they have made.

Variance refers to the extent to which the machine learning algorithm will adapt to a new set of data. A high variance model will be quite complex, perhaps changing its approach significantly from one dataset to another, based on subtle patterns and relationships among the inputs. This will make it very adaptable to different datasets, but it also adds complexity. Generally, algorithms that do not use linear models like linear regression are higher in variance than linear models.

Even when you find the sweet spot, a certain amount of error, called **irreducible error**, will always remain. This amount of error cannot be reduced further, due to the way the problem is framed and variables that you never identified or chose not to deal with.

The balance between bias and variance is typically found through experimentation. You rig a machine learning model with controls that you can adjust (parameters). By repeatedly running the model, and adjusting these external controls, you can find the best configuration that produces a good balance.

High Bias	The Sweet Spot	High Variance
May <i>underfit</i> the training set	<i>Good enough fit</i>	May <i>overfit</i> the training set
More simplistic	Just complex enough	More complex
Less likely to be influenced by true relationships between features and target outputs	Skillful in finding true relationships between features and target outputs while not overly influenced by noise	More likely to be influenced by false relationships between features and target outputs ("noise")

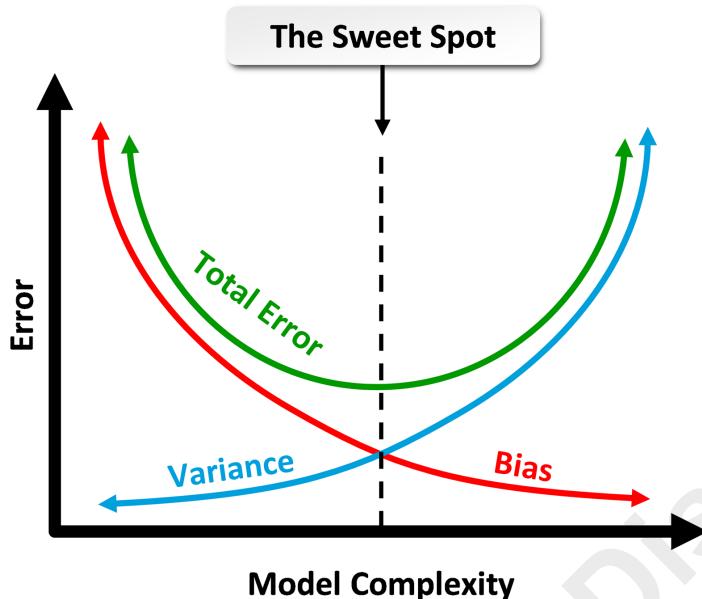


Figure 3–2: Finding a bias-variance balance that doesn't overfit or underfit the model.

Parameters

The mathematics of machine learning involves the use of parameters, or configurable values that are of importance to the machine learning process in multiple ways. In machine learning, parameters can be divided into two groups: model parameters and hyperparameters.

A **model parameter** is a parameter that is derived from the machine learning model as it undergoes the training process. In other words, model parameters are what is actually "learned" by the model while it performs calculations on the training data. These parameters therefore determine how well the model makes predictions and other intelligent decisions. As they are derived from the training process, model parameters are typically not configured by the machine learning practitioner, but by the algorithms and mathematical functions that comprise the machine learning process. One example of a model parameter is a coefficient of an independent variable for linear regression models; i.e., the model parameter is what the variable is multiplied by.

A **hyperparameter** is a parameter that is set on the algorithm itself and not the learning model. This means that a hyperparameter is provided *before* training, typically by the machine learning practitioner. The practitioner tunes hyperparameters so that the eventual model will be better at estimating the model parameters, thereby improving the model's learning performance. For example, in an artificial neural network, the number of hidden layers is a hyperparameter and must be configured before training.



Note: Model parameters are said to be *internal* to the model, whereas hyperparameters are *external* to the model.

Regularization

As mentioned, one common issue that machine learning models face is overfitting. Providing more training examples can help to address overfitting, but that approach is not always feasible.

Regularization is the technique of simplifying a machine learning model by constraining the model parameters, which helps the model avoid overfitting to the training data. This typically involves

forcing one or more model parameters to only include values within a small range, or forcing parameters to 0. This helps to minimize the effect of outliers on the model.

In a machine learning model, you can control the amount of regularization by setting the λ (lambda) hyperparameter. As you increase the value of λ , the model will become less likely to overfit to the training data. This is because you are decreasing variance in the model. But decreasing variance increases bias, so you must be careful not to make λ too large. This would underfit the model, preventing it from making useful predictions on the training data.

One common method for selecting a λ value is to use cross-validation to randomly sample the data several times for one λ value, then repeat the process for different λ values. You can then choose the λ value that best minimizes the total error.



Note: λ is the Greek letter lambda.

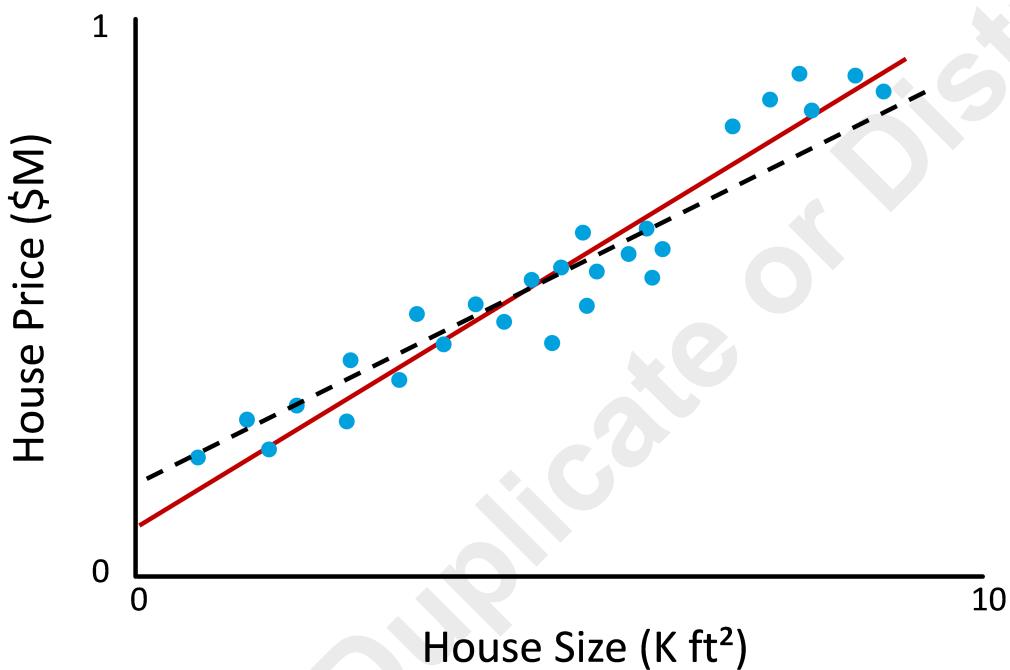


Figure 3-3: A linear model without regularization (solid red line) vs. with regularization (dashed black line). The former fits the training data better, but the latter generalizes to new data better.

Models in Combination

The process of preparing data often consumes a large portion of the time on a machine learning project. In contrast, setting up and testing a hypothesis model based on that data may take considerably less time. In light of this, you may find it beneficial to try out several different types of algorithms that are able to produce the type of outcome you require. Run different algorithms on your test data, tune them, and compare their performance, selecting one that performs the best or best meets your requirements.

In some cases you may achieve best results by using multiple algorithms in combination, processing the data in stages. For example, you might use one algorithm to perform dimensionality reduction on the dataset, and then use another algorithm to make predictions.



Note: Ensemble methods formally combine multiple machine learning algorithms to possibly produce better results than any one of the single algorithms they employ. Ensemble methods essentially use "the wisdom of the crowd," with each algorithm producing its best prediction, which it then casts as a "vote." The winning answer is the one that is returned by the ensemble method.

Processing Efficiency

Time management is a significant aspect of machine learning. In this context, it doesn't necessarily mean managing your *own* time, although that is certainly important on any type of project, including machine learning. In this context it refers to the time your processing hardware (CPUs and GPUs) require in order to train and run a machine learning model. This may not be much of a concern when you are working with relatively small datasets like those used in this course. But real world datasets can be huge and complex, possibly taking hours or even days to be processed by a machine learning model.

You might consider throwing money and hardware at the problem, buying or leasing access to fast hardware that will more than meet your requirements. While capable hardware may be at least part of the solution to some problems, it's easy to waste money on hardware that is really not necessary to meet your business requirements.

Guidelines for Training and Tuning the Model

Follow these guidelines when training and tuning a model.

Manage the Time Needed for Processing

Following are some of the general strategies you might employ to reduce or manage the time needed for processing.

- **Set realistic performance requirements.** You may not need the most skillful model that time and money can produce. Set your performance requirements for the model to align with business requirements. In some cases, you may not need a very high-performing model to get the actionable insights you require.
- **Avoid unnecessary complexity.** In general, you should use the simplest or most computationally efficient method that meets your requirements.
- **Employ appropriate math and logic techniques.** Some types of calculations can be accomplished slowly through brute force tactics, or much more quickly if you know the right "tricks." For example, for some matrix operations, you can perform calculations using programming loops, but from a processing perspective, it might be much more efficient to use matrix math.
- **Learn through experimentation.** Processing times are influenced by numerous factors. Knowing which factors to change comes through experimentation and experience.
- **Select the right hardware.** Make sure the systems you use are appropriate for the tasks you need to accomplish, and they are appropriately configured.
- **Improve the structure of your datasets.** Some algorithms take much longer to process datasets when data items have high standard deviations or columns are in vastly different ranges. Optimize datasets as needed for the types of tasks you need to perform on them.

ACTIVITY 3–4

Refitting and Testing the Model

Data File

/home/student/CAIP/Workflow/Workflow-Housing.ipynb

Before You Begin

If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the **CAIP/Workflow/Workflow-Housing.ipynb** notebook. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled "Build and test a linear regression model - Round 2". Select **Cell→Run All Above**.

Scenario

You will now test the model again, to see whether the adjustments you made to the datasets will improve the results.

1. Compare the scaling of sqft_living and price.

- Scroll down to view the cell titled **Build and test a linear regression model - Round 2**, and examine the code listing beneath it.

Build and test a linear regression model - Round 2

```
In [ ]: 1 from sklearn.linear_model import LinearRegression
2
3 # Create a linear regression model and fit it using the training data
4 regressor = LinearRegression()
5
6 start = time()
7 regressor.fit(data_train, data_train_labels);
8 end=time()
9 train_time = (end - start) * 1000
10 print('Model took {:.2f} milliseconds to fit.'.format(train_time))
11
12 # Evaluate the model's performance
13 score = regressor.score(data_test, data_test_labels)
14 'Score: {:.%}'.format(int(round(score * 100)))
```

- Lines 1 through 14 create and fit a linear regression model similar to the one you created in Round 1.
 - The model will be re-fit using the updated training data.
- Select the cell that contains the code listing, and select **Run**.

- c) Examine the output.

```
Model took 14.45 milliseconds to fit.
'Score: 77%'
```

- The score has increased from 70% to 77%.
- Improvements you made to the dataset have led to improved results.



Note: In addition to the size and complexity of the dataset and the complexity of the algorithm, other factors affect the amount of time needed to fit a model, including the processor speed, background processes currently running, and so forth. Your computer may have taken a different amount of time to fit the model than what is shown here.

2. Compare predicted prices to the actual prices.

- a) Scroll down to view the cell titled **Compare the first ten predictions to actual values**, and examine the code listing beneath it.

Compare the first ten predictions to actual values

```
In [ ]: 1 # y_pred is the predicted prices that will be produced by testing
2 predicted_prices = regressor.predict(data_test)
3 predictions = data_test_labels.copy()
4 predictions['predicted'] = predicted_prices
5
6 # View examples of the transformed prices
7 with pd.option_context('float_format', '{:.2f}'.format): print(predictions.head(10))
```

- Again, this code is similar to code you ran in Round 1.
 - This will show you the first ten rows of actual prices compared to predicted prices, so you can see how well the new model is making its predictions.
- b) Select the cell that contains the code listing, and select **Run**.
- c) Examine the output.

	price	predicted
735	\$12.81	\$12.95
2830	\$13.67	\$13.44
4106	\$13.85	\$14.01
16218	\$14.21	\$14.58
19964	\$13.47	\$13.45
1227	\$12.26	\$12.62
18849	\$13.58	\$13.59
19369	\$13.43	\$13.04
20164	\$12.86	\$12.92
7139	\$13.31	\$12.90

- Because the prices were scaled using a log transformation, the values don't reflect the actual prices.
- You will have to transform these prices back to actual values. Python's `math` library provides an `exp()` function, which you can use to reverse the `log()` function.

3. Convert prices back to actual values.

- a) Scroll down to view the cell titled **Convert the prices back to actual values**, and examine the code listing beneath it.

Convert the prices back to actual values

```
In [ ]: 1 # Need to call exp() function to convert back from log value to actual price.
2 import math
3 predictions = predictions.applymap(math.exp)
4
5 # View examples of the actual and predicted prices
6 with pd.option_context('float_format', '${:.2f}'.format): print(predictions.head(10))
```

- Line 2 imports the `math` library.
 - Line 3 calls the pandas library's `applymap()` function to apply the `exp()` function to every data item in the `predictions` dataframe.
 - Line 6 outputs the first ten rows of `predictions`, showing the values with formatting for U.S. dollars.
- b) Select the cell that contains the code listing, and select **Run**.
 c) Examine the output.

Round One Predictions

	price	predicted
735	\$365,000.00	\$451,576.87
2830	\$865,000.00	\$745,528.73
4106	\$1,038,000.00	\$1,234,144.11
16218	\$1,490,000.00	\$1,659,505.67
19964	\$711,000.00	\$737,851.90
1227	\$211,000.00	\$284,352.79
18849	\$790,000.00	\$832,187.44
19369	\$680,000.00	\$490,462.65
20164	\$384,500.00	\$392,922.89
7139	\$605,000.00	\$471,310.48

Round Two Predictions

	price	predicted
735	\$365,000.00	\$420,860.44
2830	\$865,000.00	\$689,455.76
4106	\$1,038,000.00	\$1,220,057.79
16218	\$1,490,000.00	\$2,139,532.72
19964	\$711,000.00	\$696,523.60
1227	\$211,000.00	\$301,303.20
18849	\$790,000.00	\$800,320.80
19369	\$680,000.00	\$462,471.84
20164	\$384,500.00	\$408,170.22
7139	\$605,000.00	\$398,681.64

- The prices have been transformed back to reflect real prices.
- Although the overall score has improved, by comparing individual predictions in the two rounds, it may not be easy to see that the predicted prices are now generally closer to the actual prices.

4. Create a visualization to show the accuracy of the predictions.

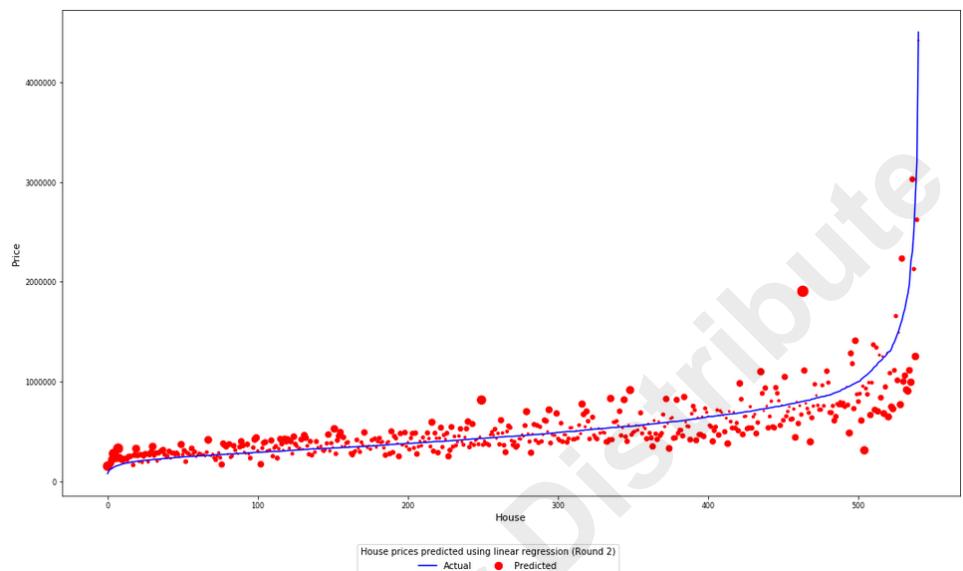
- a) Scroll down to view the cell titled **Compare predicted values to actual values (Round 2)**, and examine the code listing beneath it.

Compare predicted values to actual values (Round 2)

```
In [ ]: 1 # Compare the predicted prices to actual prices
2 compare_pred_to_actual('House prices predicted using linear regression (Round 2)')
```

Line 2 calls the `compare_pred_to_actual()` function that was previously defined, to generate a new chart.

- a) Select the cell that contains the code listing, and select **Run**.



- If you compare back to the previous chart, you can see that there has been some improvement.
- The dots are a bit more clustered toward the blue line, and generally smaller than before.

5. Try a different algorithm.

- a) Scroll down to view the cell titled **Try a different algorithm**, and examine the code listing beneath it.

Try a different algorithm

```
In [ ]: 1 # Create a model using the random forest algorithm.
2 from sklearn.ensemble import RandomForestRegressor
3
4 rnd_forest = RandomForestRegressor(n_estimators=100,random_state=0)
5
6 start = time()
7 rnd_forest.fit(data_train, data_train_labels)
8 end=time()
9 train_time = (end - start) * 1000
10 print('Model took {:.2f} milliseconds to fit.'.format(train_time))
11
12 score = rnd_forest.score(data_test, data_test_labels)
13 print('Score: {:.2f}'.format(score * 100)))
```

This code is similar to the code you used to create a linear regression model, only now you are creating a model using a random forest class that employs a combination of various regression algorithms to come up with the best fit.

- b) Select the cell that contains the code listing, and select **Run**.



Note: Be patient, as the random forest algorithm will take much longer to finish than the linear regression. This operation may take half a minute or more to complete.

- a) Examine the output.

```
Model took 33,572.89 milliseconds to fit.
Score: 89%
```

- Although this model took longer to fit, it produced a much better score than the linear regression model.
- Often, you'll have to make tradeoffs between time and performance, deciding whether measures like accuracy and recall outweigh the need for producing quick results.

6. Create a visualization to show the accuracy of the random forest predictions.

- a) Scroll down to view the cell titled **View examples of the actual and predicted prices**, and examine the code listing beneath it.

View examples of the actual and predicted prices

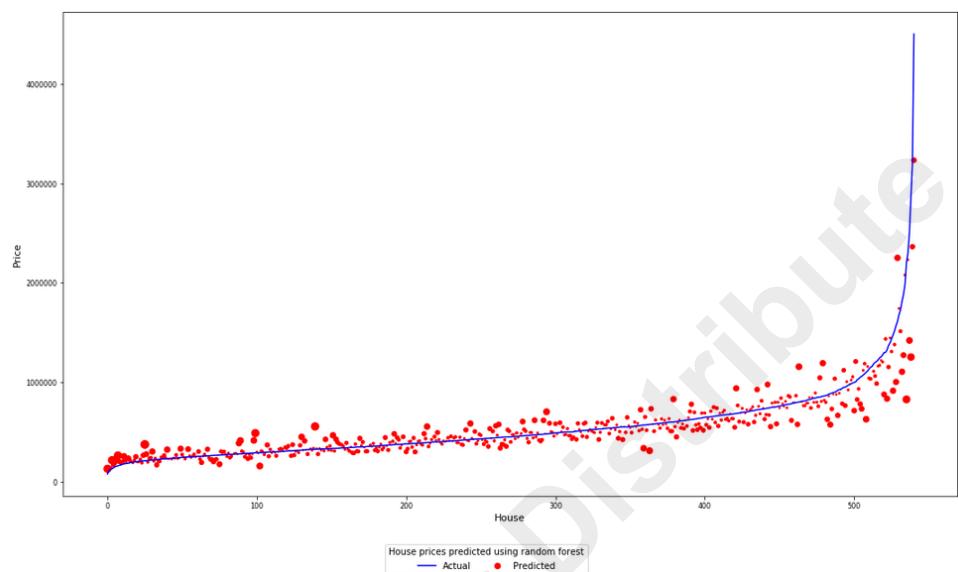
```
In [ ]: 1 predicted_prices = rnd_forest.predict(data_test)
2
3 predictions = data_test_labels.copy()
4 predictions['predicted'] = predicted_prices
5
6 # Scale the prices back to actual values.
7 predictions = predictions.applymap(math.exp)
8
9 # View examples of the actual and predicted prices
10 with pd.option_context('float_format', '${:.2f}'.format): print(predictions.head(10))
```

- b) Select the cell that contains the code listing, and select **Run**.
 c) Examine the output.

	price	predicted
735	\$365,000.00	\$367,229.33
2830	\$865,000.00	\$797,830.56
4106	\$1,038,000.00	\$1,112,227.86
16218	\$1,490,000.00	\$1,866,465.53
19964	\$711,000.00	\$709,144.08
1227	\$211,000.00	\$251,855.28
18849	\$790,000.00	\$870,082.54
19369	\$680,000.00	\$562,736.77
20164	\$384,500.00	\$413,123.06
7139	\$605,000.00	\$534,529.34

These predicted prices tend to be more accurate than those you produced with the linear regression model.

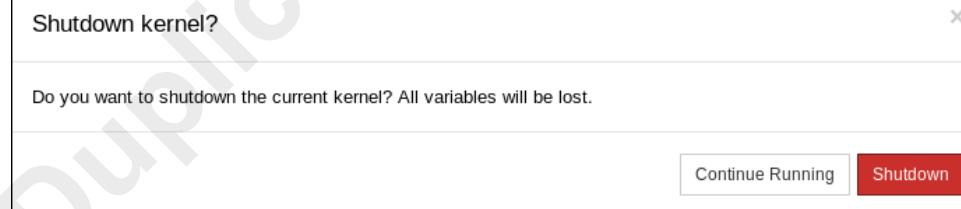
- d) Run the last code cell and examine the output.



The last code cell generates an updated bar chart comparing the actual prices with the prices predicted by the random forest model. The prices in this model tend to be closer than those shown in the chart for the linear regression model.

7. Shut down the notebook.

- a) Select **Kernel→Shutdown**.



- b) Select **Shutdown**.
c) Close the **Workflow-Housing** tab in Firefox, but keep a tab open to the file hierarchy in Jupyter Notebook.

Summary

In this lesson, you set up and trained a hypothesis model.

What strategies might you employ when determining which algorithm you'll use initially?

What concerns do you have about bias in your own datasets?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Do Not Duplicate Or Distribute

4

Finalizing a Model

Lesson Time: 50 minutes

Lesson Introduction

When you have finished training and tuning a machine learning model, you can turn your attention to deploying it. This may amount to producing a report based on your findings, or it may be much more involved, particularly if it will be incorporated into repeatable processes or become part of a software solution.

Lesson Objectives

In this lesson, you will:

- Communicate the findings of a machine learning project back to the organization.
- Incorporate a machine learning model into a long-term business solution.

TOPIC A

Translate Results into Business Actions

The final stage of the machine learning workflow is applying the model and presenting the results. In some cases you'll have to report your findings back to stakeholders. You'll want to communicate the results in a way that your audience can understand.

Know Your Audience

Unless you were creating a model just to answer your own questions, once you have trained and tuned a good model, you'll have an audience you need to report back to. Your audience may be a single person, a group of stakeholders, or perhaps an entire organization. They may work for the same company you work for, or they may be external clients.

In many cases, the audience for your report may consist of many different people, with different knowledge, needs, and expectations. To produce a good report, then, you'll need to know your audience so you can identify what information you must provide, and how you must provide it to communicate it effectively.

Visualization for Presentation

Early in the project, you may have used visualizations to explore the dataset and come up with a hypothesis for the model. You may also have used visualizations as you trained and tested the model, to evaluate its performance and verify that it is working as you intended. Visualizations provide a powerful means of conveying patterns in numbers. Just as they are helpful to you during the development of a model, they can be very helpful to assist your audience in understanding the results produced by your machine learning model.

Guidelines for Presenting Your Findings

Follow these guidelines when you report the results of a machine learning project back to project stakeholders.



Note: All of the Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Put Together a Machine Learning Presentation

A typical business audience will need to know the problem you set about to solve, and your proposed solution. At its core, a machine learning solution will involve data and a model that operates on that data, so you'll need to explain how you utilized those components, sticking to information that is relevant for the audience, and providing the kinds of details they need to know. Primarily, they will need to understand how the results of your project impact the business. These components, then, should make up the essential parts of your presentation, described here.

- **An executive summary or abstract of the project**—A report on a data analysis project may contain many details and complex findings. To help your audience prepare to understand what you will present in detail, start with a brief summary of the high points—what you set out to do, and what you found as the answer or solution.
- **The business problem you set about to solve**—Set the stage by describing the problem and identifying how a solution might be measured in terms of business metrics, such as a percentage of increased sales, reduced maintenance costs, increased production rates, and so forth.
- **The data**—Describe, in terms your audience can relate to, the data that was used, where it came from, and any relevant explanations of how you had to modify the data to make it useful. If

appropriate, identify additional data it would be helpful to collect in the future to support ongoing analysis.

- **The hypothesis**—What you initially surmised from analyzing the available data, and how your solution evolved, sticking with details that would be relevant to your audience. Avoid diving in to technical topics like the algorithms, programming tools, and statistical techniques you used unless they are relevant to your audience, and you can present them in a way your audience will understand.
- **The solution**—Identify what the final model revealed, clearly identifying its business implications. Use examples and visualizations to help your audience understand. Be transparent, describing any assumptions you made, identifying where you might make improvements, and any risks the solution might introduce, including risks and possible ethical challenges posed to the company, staff, users, and customers.

Communicate Your Findings Clearly

When communicating your finding in a report or presentation:

- **Don't drown your audience in a sea of numbers.** Adapt the presentation for different audiences you must address, including only information relevant to that audience, avoiding jargon, and explaining things in a way that audience will understand.
- **Use visuals strategically.** Use charts or diagrams when they more clearly illustrate patterns in the data than words or numbers, particularly where they support important points you have to make about your findings. Experiment with various chart types and formatting options as needed until you come up with a configuration that clearly communicates the point you're trying to make with little or no explanation.
- **Use formatting, labels, and legends.** When displaying numbers, use commas and decimal points to help viewers read the numbers. If values are really close in size, use more decimal places to show differences in values. Use currency symbols for money values. Include labels, legends, and other captions where they will help the reader understand what they're looking at. Align columns of numbers on the right, or at the decimal point so they are more easily compared.
- **Avoid suspense.** A good presentation flows well and tells a story, but not in the way a mystery novel tells a story. You will likely provide your audience with a lot of information to process, and they will have questions and concerns. Help them understand and buy in to your conclusions by anticipating the types of questions you think they will have, and sharing important points early on.
- **Provide context for details.** If you must tunnel down into details showing numbers, tables, and charts, be sure to connect them back to the main idea or point you're trying to communicate.
- **Be honest and transparent.** Be clear regarding how you obtained results. Don't hide significant data or results—even if they don't fully support your hypothesis, objectives, or proposed solution.
- **Identify key assumptions you have made.** This will provide context and a rationale for the results you've produced.
- **Check your work.** Although your machine learning models may be impeccable, the quality of your findings may be called into question if your presentations contain errors.
- **Invite feedback.** Make sure your audience has a chance to ask questions so you can provide clarification and promote buy-in.
- **Provide supplementary communication channels for those with different communication styles.** Some people may be hesitant to ask questions during a presentation, and may be more comfortable asking you follow-up questions one-on-one or through online messaging. Consider inviting them to share feedback through such side channels.
- **Provide supplementary documentation for those with different informational needs.** Some people may require more details or technical depth than others. While it may not be appropriate to bog down your presentation slides or report with details that are only of interest to a small portion of your audience, consider directing those who require more information to view supplemental documents that you've posted on the company intranet, a network directory, or some other secure location where they can view the additional details.

- **Test it before you deliver it.** Depending on the importance of the outcome of your presentation, consider testing it before you deliver it. Pass it by reviewers who can represent your target audience and will give you honest feedback. Ask them questions to assess how well the presentation gets important points across. Do a dry run of your delivery. Make revisions as needed.

Use Visualization Tools to Present Your Findings

When communicating your finding in a report or presentation:

- Label axes and chart titles with meaningful names.
- Use legends as needed to identify what values are shown in a chart.
- Don't rely solely on color coding, but when you must use it, use color palettes that can be detected by people with color blindness.
- Avoid adding unnecessary details or decoration that hinder readability, such as backgrounds or themes.

ACTIVITY 4-1

Translating Results into Business Actions

Scenario

The CapitalR Real Estate company has hired you to develop new pricing models that will enable real estate agents to more accurately predict the prices that houses will sell for. They want to avoid situations where houses stay on the market too long because they're overpriced, or where houses sell quickly but are underpriced, not providing as large a return for sellers or as large a commission for agents.

In your analysis, you were somewhat surprised when your findings seemed to debunk the old real estate adage—"It's all about location, location, location!" When it comes to the pricing of houses in your region, you determined that other factors play a stronger role than location. You found that the strongest predictors for price are the size of the home and its grade, a rating of the house's overall construction quality and features, evaluated by the tax assessor.

While there is some correlation between house prices and neighborhoods, general location features such as whether the house has a good view or is situated on waterfront property don't seem to matter as much as the size of the house or the quality of the building itself. The new machine learning pricing models you developed incorporate many of the subtleties influencing prices, and in your evaluations, you have demonstrated that the model has been able to predict houses that are overpriced, which end up languishing on the market and eventually selling for a price close to what the model predicted.

In some ways, the model runs contrary to old principles that agents have used for years to come up with prices. You want the real estate board to adopt your pricing model. As you head into a presentation to sell them on using it, you're concerned that there may be some resistance to adopting it.

1. How can you present your findings to the real estate board so they not only understand how your pricing model is different from what they've done in the past, but so they are also convinced they should use it?

2. The model is complex and subtle in its calculations. In the past, agents have used a simple paper worksheet to calculate a listing price, but the new model is too complex for that.

How might you make it easy for agents to get pricing for a house based on the new model?

Do Not Duplicate or Distribute

TOPIC B

Incorporate a Model into a Long-Term Business Solution

You've developed a machine learning model that you need to incorporate into a long-term business solution, such as an enterprise application. Now you need to put the model into production, possibly working with a team to develop, test, and deploy it.

Put a Model into Production

Some machine learning models you develop may not simply result in a one-time recommendation or answer. They may instead represent the prototype for a new capability that you must roll into existing products or processes, perhaps a new feature or function in software running on the organization's website, mobile app, or enterprise application.

When you put a machine learning model into production, it is not finished by simply writing up a report. You'll need to consider how you'll deploy the algorithm on a software platform, perhaps involving different code than you used to develop the initial model. You may hand your prototype off to software developers to be incorporated into a larger solution. A machine learning model put into production will require further development, testing, and maintenance over time.

Production solutions may involve parallelization, involving multiple CPUs on a single computer, multiple GPUs, or cluster computing across multiple nodes. You may need to utilize cloud services to provision systems that can meet your needs.

Production Algorithms

If you use a software library in Python or R to identify a model that performs well, the algorithm provided in that library may not be optimized for the environment or use cases you require for the production implementation. For example, you may need to deploy a software solution over cloud services that don't use the same environment you used to develop your proof of concept.

Furthermore, the libraries you used may be optimized for research and general use by a single user, rather than for production use on a server. The machine learning algorithm you prefer to use may not be available on the platform you must deploy it on, so it may be necessary to find an alternative algorithm when you take a solution into production. In situations like these, consider your initial work to be a prototype and be open to using more appropriate tools for production use.

Of course, the process of training and optimization will need to be repeated when using a different implementation of the algorithm, but your prototype will go a long way toward understanding the problem and the expected outcomes.

Pipeline Automation

Automated pipelines function by enabling a series of data wrangling tasks to be chained together, to help automate a machine learning workflow. Various tools support this capability. For example, scikit-learn provides a `Pipeline` utility that you can use for this purpose.

Automating the data pipeline instead of performing these tasks manually not only reduces the tedium involved in performing the steps and saves time, but it also helps to ensure the process is performed exactly the same way each time, avoiding errors like leaking data from the training set into the test set, forgetting to perform various data cleanup or transformation tasks, and so forth.

As you identify improvements in the workflow, you can implement them in the automation script to ensure they'll be incorporated every time the task is performed. When you incorporate a model into

ongoing, repeated business processes or applications, it is a good idea to automate the processes that provide these processes with data.

Testing and Maintenance

Machine learning is subject to the **CACE principle**—*changing anything changes everything*. In other words, changes to the feature set, transformations done on data, learning rate and other parameters, and many other factors can affect the model's performance.

You may implement a machine learning model that works with data that is updated and accumulated on an ongoing basis. In business, many things change over time. Often this happens subtly, and the changes are insignificant from one batch of data to the next. Or changes might be quite abrupt. But at some point, changes from the patterns that were present when the model was initially trained may be significant enough that they reduce the model's skill.

Also, changes in *software* may affect the performance of the model. Many developers use third-party software libraries to implement data pipelines and machine learning algorithms. When these libraries are updated, changes in how they operate may lead to changes in the machine learning model's performance.

When machine learning becomes part of a long-term business solution, it may be necessary to periodically retrain the model and test it to ensure the model still performs as required. Aspects of this can be automated. For example, you could use automated monitoring and testing tools to track the model's performance over time and raise alarms if its skill drops below a minimum level. Some models can be configured to perform online learning and update themselves periodically.



Note: Be careful when setting up self-updating models in a production environment. Typically it is more prudent to manage the model update process in a separate "staged" environment, pushing the model out to production only after you have thoroughly verified it.

Consumer-Oriented Applications

Machine learning is not only used for data analysis, but has also been implemented in applications used directly by consumers. Programmers can use machine learning APIs (application programming interfaces) when developing applications for Android, iOS, Windows, Linux, and numerous other operating systems. Machine learning APIs enable applications to perform a wide range of AI/ML tasks, such as text recognition, face detection, barcode scanning, image labeling, object detection and tracking, landmark recognition, language identification and translation, smart message replies and typing suggestions, and so forth. Some machine learning APIs use the device's network connection to offload heavy processing tasks to cloud services, but some APIs perform AI/ML tasks directly on the device, in real time.

Guidelines for Incorporating Machine Learning into a Long-Term Solution

Follow these guidelines when incorporating machine learning models into applications or other long-term solutions.

Maintain Models Incorporated into Applications

When you incorporate machine learning models into software applications, make sure the programming team follows these guidelines to ensure the continued integrity of the model.

- **Create software unit tests that validate the data model every time new code is pushed out to the application.** Changes in code or data may adversely affect the machine learning model. Modular software design with unit tests that address performance of the ML model can help to mitigate this problem.

- **Protect the model from upstream changes to the data pipeline.** If possible, work with outside providers of data to ensure that they will not make changes that will adversely affect your model's data pipeline. Avoid depending on inputs that are likely to change over time. Provide a configuration layer at the front end of the data pipeline so you can easily adapt to changes in the input data over time.
- **Protect downstream consumers of the model.** When you are unaware of other processes that consume outputs produced by the ML model, changes to your model, such as retraining or changes to inputs and outputs, may adversely affect them. By implementing access controls, you will be aware of all consumers and will be able to provide them with proper notifications when changes are anticipated. Require that outside components request access to use the model and/or its outputs so you can track consumer processes.
- **Design for periodic retraining of the model.** Model performance typically decays over time as inputs evolve. It is generally a good idea to periodically retrain a model with recent data.
- **Provide a versioning system for various machine learning components used in the application.** Consider providing version numbers for components that may change over time, including model configuration and parameters, data pipeline including specific features used, and training and validation datasets.
- **Put appropriate protections in place for the data pipeline and all outputs.** The organization must ensure that data is protected for security (the company's confidential data, for example) and privacy (user data protected by laws and regulations).

ACTIVITY 4–2

Incorporating a Model into a Long-Term Solution

Scenario

You are working with the CapitalR Real Estate company to incorporate a pricing calculator (based on your machine learning model) into a web-based application that agents will use to manage real estate listings. The web developers you are working with have not worked with machine learning before. Even though you won't be involved in the actual programming of the web application, you have been contracted to provide guidance to the development team. As part of those responsibilities, you will help to write software requirements and test cases for the application.

1. What market factors might affect the real estate pricing model, and how frequently should the model be updated to account for them?

 2. What software requirements might you include to ensure the pricing calculator remains accurate, considering the various market factors you identified in the previous question?
-

Summary

In this lesson, you learned how to translate the results of a machine learning project into business actions, including how to present your findings to stakeholders. You also examined various issues you'll need to manage when you incorporate a machine learning model into production.

In your machine learning projects, to whom will you typically have to present your findings, if anyone?

Where do you see your machine learning models being deployed, and how will this affect your management and maintenance of the model?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Do Not Duplicate Or Distribute

5

Building Linear Regression Models

Lesson Time: 3 hours

Lesson Introduction

Now that you've gone through the general machine learning workflow, you can start to apply that workflow to different machine learning algorithms. One of the most fundamental algorithms for predicting data is linear regression. You actually created a linear regression model as part of the earlier workflow discussion, but now you'll dive deeper into how this algorithm works and how it can solve your complex business problems.

Lesson Objectives

In this lesson, you will:

- Build regression models using linear algebra.
- Build regularized regression models using linear algebra.
- Build iterative linear regression models.

TOPIC A

Build Regression Models Using Linear Algebra

There are several ways you can apply a linear regression algorithm to a given problem, creating a machine learning model in the process. To start with, you'll use simple linear algebra to create a regression model that can predict numerical data.

Linear Regression

In the field of statistics, regression analysis is the technique of identifying the relationships between variables. These variables are categorized as either dependent or independent. A dependent variable is one whose variation you are interested in studying. An independent variable has a potential effect on the variation of dependent variables—in other words, it helps explain what is happening to the dependent variable.

Linear regression is a type of regression analysis in which there is a linear relationship between one independent variable and one dependent variable. If plotted on a graph, this relationship would form a straight line, and the slope of that line between any two points on the graph would be the same.

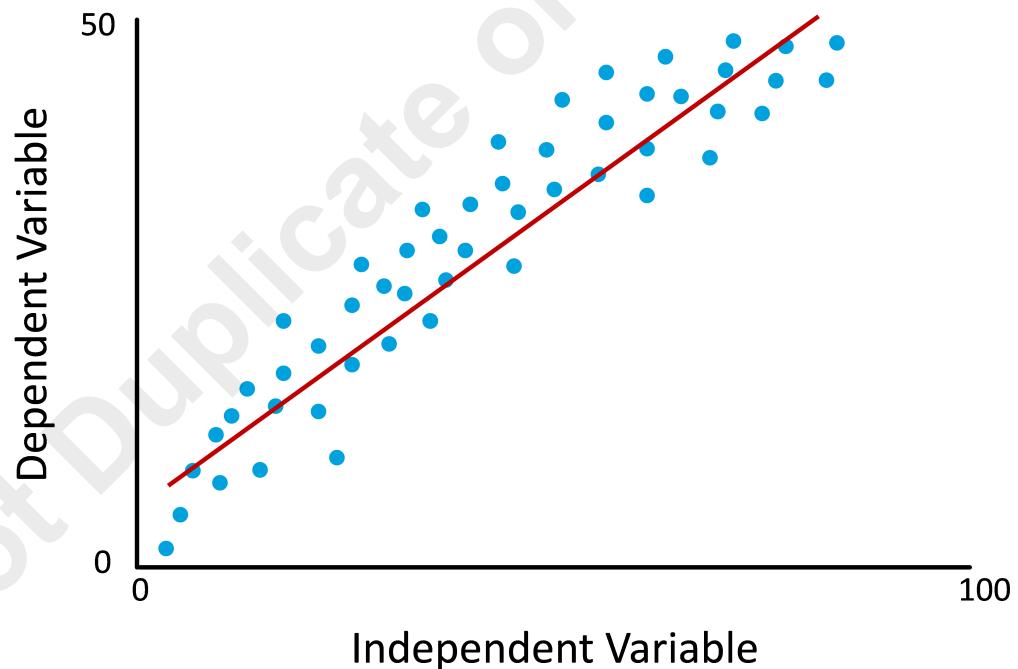


Figure 5–1: A linear regression model in which the data fits to a straight line.

Linear Equation

It helps to understand linear regression by first considering how data may be plugged into a simple linear equation. This equation generates a straight line fit to linear data. You may recognize the linear equation as it is a foundational concept in statistics courses taught in schools:

$$y = mx + b$$

Where:

- y is the y-value for a data example (the dependent variable).
- x is the x-value for a data example (the independent variable).
- m is the slope of the line, calculated by dividing the change in y by the change in x .
- b is the intercept—the value of y when x is 0.

Linear Equation Data Example

Let's say you're trying to map how many miles a vehicle has been driven (x) and the effect that has on the vehicle's monetary value (y). You have 15 historical data examples to draw from, as recorded in the following table.

Miles Driven in Thousands	Value in Thousands of Dollars
.10	48.70
1.09	47.30
8.71	43.90
28.54	44.25
41.92	39.50
57.13	45.60
89.54	27.20
97.12	34.00
103.00	39.20
125.43	27.40
139.10	31.85
145.02	20.65
163.21	24.40
194.98	43.00
224.63	13.25



Note: For example purposes, this data is just in raw form and hasn't undergone feature engineering.

When graphed, that data might look something like the following.

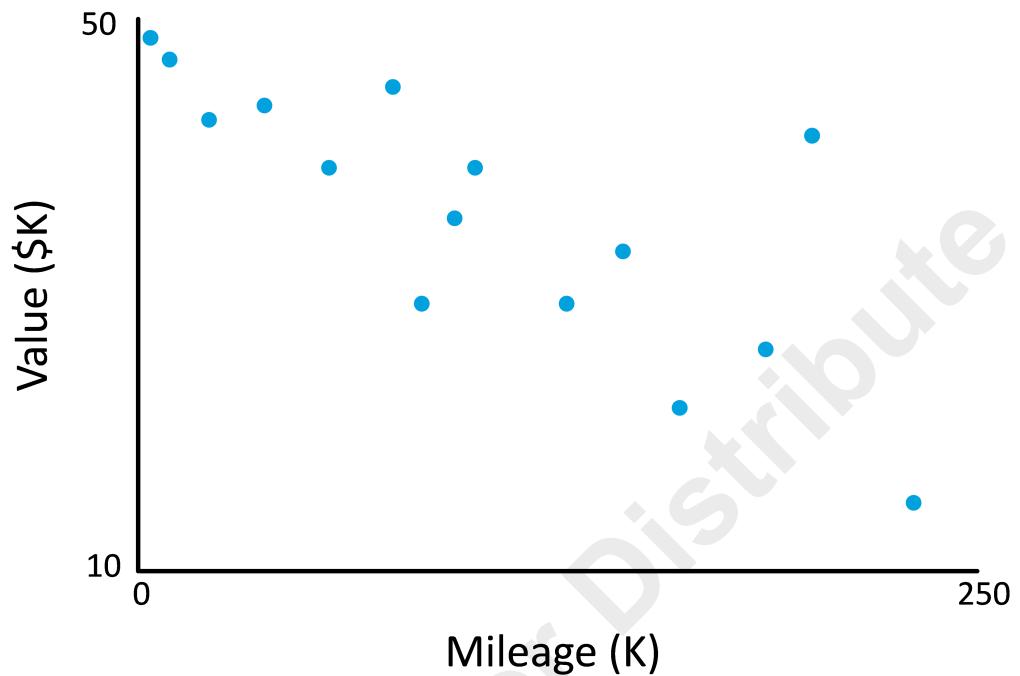


Figure 5–2: Mapping a vehicle's worth as it correlates to miles driven.

Straight Line Fit to Example Data

Using the previous data, the calculation of the slope (m) is -0.116 and the intercept is 46.342 . So, plugged into the linear equation, this is:

$$y = -0.116x + 46.342$$

This provides you with the line of best fit for the data which, when graphed, looks something like the following.

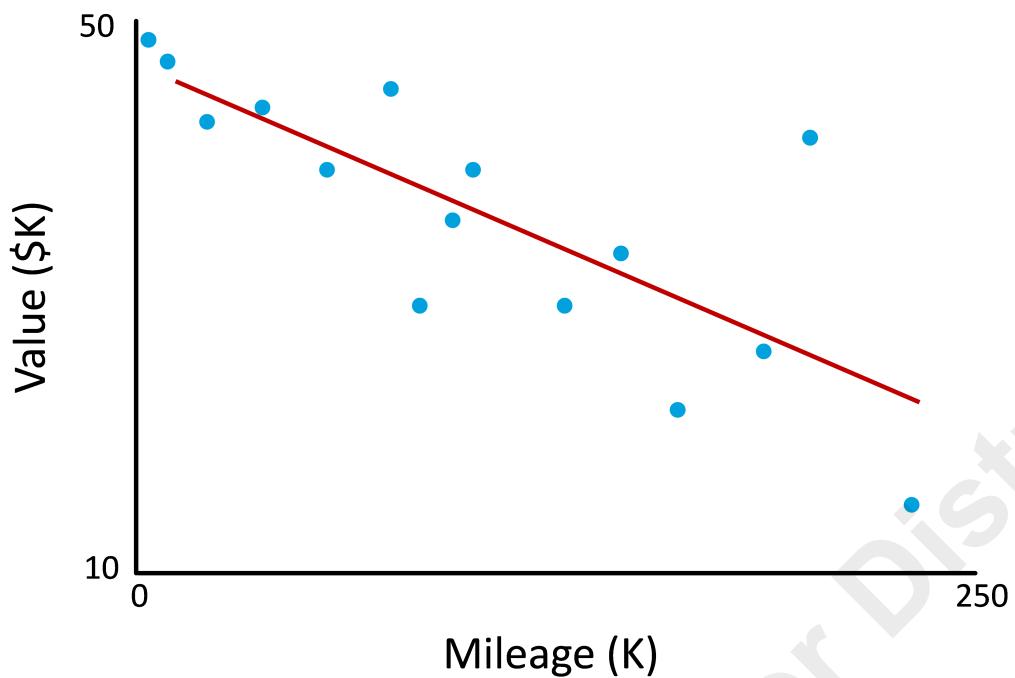


Figure 5–3: Fitting a straight line to the vehicle dataset.

Now, let's say you want to predict the value of your car after driving it exactly 100,000 miles. You'd simply plug that value in for x like so:

$$y = (-0.116)100 + 46.342 = 34.74$$

So, this very simple linear model has predicted that your car will be worth around \$34,742. When graphed, this might look like the following.

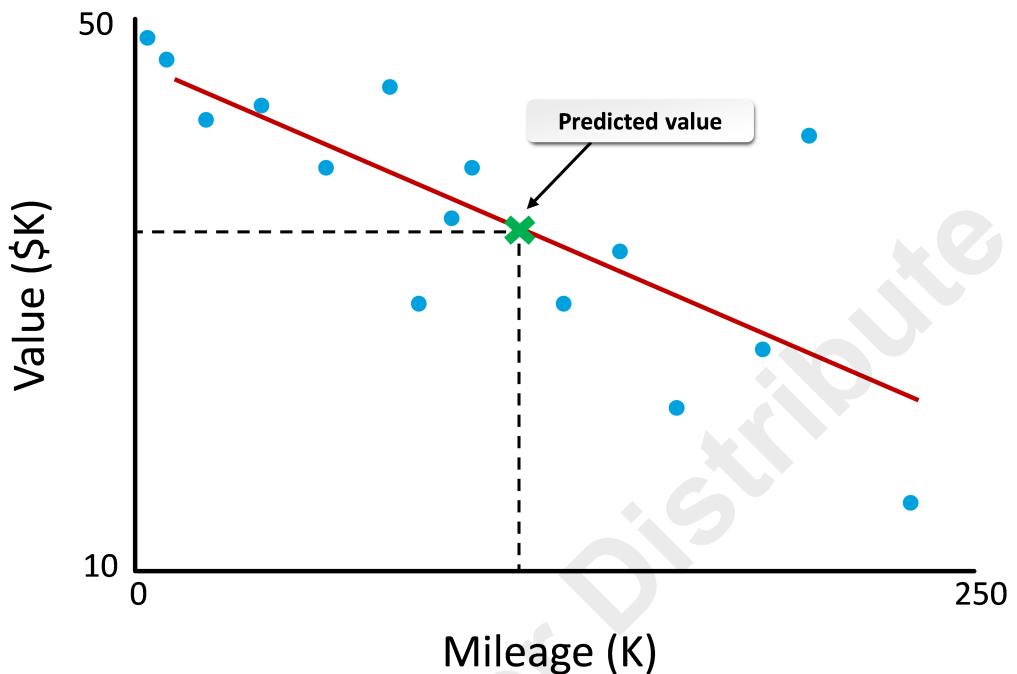


Figure 5–4: Making a prediction based on the line of best fit.

Linear Equation Shortcomings

The linear equation is a simple example of making predictions based on a dataset, but there are more powerful ways of applying linear regression to machine learning. The linear equation may not work well or as expected with data that cannot be fit linearly. It also fails to account for multiple predictors—after all, mileage is not the only factor that influences how much a car is worth. You have other factors like the class of vehicle (e.g., sedan vs. tractor trailer), the make, the model, the year of manufacture, the year of sale, the climate it's primarily driven in, who owned it, how well the previous owner treated it, and many more. All of these potentially contribute to the worth of the vehicle.

Machine learning tasks often necessitate somewhat more complex approaches to linear regression.

Linear Regression in Machine Learning

In a machine learning problem, the linear regression algorithm's objective is to find the difference between the input training data and the predictive line fit that the model generates. This difference is called the error or cost. Each feature of the dataset, as well as any permutations generated by a human practitioner during the feature engineering process, will have a corresponding parameter θ_i that the model must solve for.



Note: The θ symbol is the Greek letter theta.

A basic linear model in machine learning can be expressed as:

$$\hat{y} = \theta_0 + \theta_1 \cdot x$$

Where:

- \hat{y} is the variable you're trying to predict (the dependent variable).
- θ_0 is the intercept (equivalent to b in the linear equation).
- θ_1 is a model parameter (equivalent to m in the linear equation). A model parameter is an independent variable relevant to the problem, and is usually just a feature of the dataset.
- x is the independent variable of interest—the features you'd extract and pass into the model.



Note: Notice the use of \hat{y} (pronounced "y-hat") instead of y . \hat{y} is often used to mean an estimate or prediction from a model.

Linear Regression in Machine Learning Example

Using the vehicle value example from before, you train the model on historical data with multiple features. You can construct a linear model based on one or more of these features. For simplicity's sake, you want to start by mapping a single feature—a vehicle's mileage. This can be plugged into the formula as:

$$\text{vehicle_value} = \theta_0 + \theta_1 \cdot \text{mileage}$$

When you map this linear function to more features, you can compare how well those functions fit to a straight line. If the line fits relatively straight through the data, that particular feature has a strong correlation with the prediction variable.

Linear regression is commonly used in supervised learning to predict numerical values (the dependent variables) that increase or decrease based on multiple features (the independent variables).

Matrices in Linear Regression

Because a linear model represents n number of examples in a training set, there would be n number of linear equations for each relevant value of x and y . To calculate all of these instances, linear models represent the data in matrices. The single-parameter linear model equation can be restated as a vector of y values being equal to a matrix of x values multiplied by the model parameters. As an equation, this is:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$



Note: The column of all 1s in the x matrix is necessary because these 1s are being multiplied by the constant θ_0 intercept value, whereas the column of variable x values is being multiplied by θ_1 .

For simplicity's sake, consider how these matrices would be filled in using just the last two points of the vehicle value dataset: (194.98, 43.00) and (224.63, 13.25). This would give you:

$$\begin{bmatrix} 43.00 \\ 13.25 \end{bmatrix} = \begin{bmatrix} 1 & 194.98 \\ 1 & 224.63 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

You could use this to find the model parameters θ_0 and θ_1 , but it's better to pre-multiply each side of the equation by the inverse of the \mathbf{x} matrix. Any matrix multiplied by its inverse is an identity matrix. An **identity matrix** is a matrix of all zeros except for the main diagonal, which consists of all 1s. This matrix can be removed from the equation, which leaves us with:

$$\begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

When the values are plugged in:

$$\begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} 7.58 & -6.58 \\ -0.03 & 0.03 \end{bmatrix} \begin{bmatrix} 43.00 \\ 13.25 \end{bmatrix} = \begin{bmatrix} 238.76 \\ -0.89 \end{bmatrix}$$

Now that you have the model parameters, you can create a straight line fit of the data using a linear equation. For only using the last two data points, this would be $y = -0.89x + 238.76$. Of course, in a real scenario, you'd use the entire dataset as the matrix values rather than just two instances, so your model parameters would change to account for this.

Normal Equation

One issue with these matrices is that you cannot take the inverse of a non-square matrix. So, when you plug in all 15 data points for the vehicle value dataset, you end up with a 15×2 matrix for the x values. The solution for this is to take the pseudoinverse (specifically, the Moore–Penrose inverse) of the matrix. This involves *transposing* the matrix of x values. The equation that takes this pseudoinverse is called the **normal equation**. The normal equation is a closed-form solution, meaning that it will directly provide you with the model parameters that lead to the best possible fit to the training data.

To get to this normal equation, consider how the matrix math discussed previously can be rewritten:

$$\mathbf{y} = \mathbf{X} \cdot \boldsymbol{\theta}$$

Where:

- $\boldsymbol{\theta}$ is a matrix of the model parameters (e.g., m and b for slope and intercept).
- \mathbf{x} is a matrix of the x values.
- \mathbf{y} is the vector of y values.

Then, you can multiply each side of the equation by the transpose of \mathbf{x} (i.e., \mathbf{x}^T):

$$\mathbf{X}^T \mathbf{y} = \mathbf{X}^T \mathbf{X} \cdot \boldsymbol{\theta}$$

Note that $\mathbf{x}^T \mathbf{x}$ is now a square matrix. You can then apply the inverse of $\mathbf{x}^T \mathbf{x}$ to both sides of the equation:

$$(\mathbf{X}^T \mathbf{X})^{-1} \cdot \mathbf{X}^T \mathbf{y} = (\mathbf{X}^T \mathbf{X})^{-1} \cdot \mathbf{X}^T \mathbf{X} \cdot \boldsymbol{\theta}$$

Now, the right side of the equation (other than $\boldsymbol{\theta}$) can be converted into an identity matrix because a matrix is being multiplied by its inverse. So, this can simply be removed.

The normal equation can finally be expressed as:

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \cdot \mathbf{X}^T \mathbf{y}$$

By doing all of these calculations on the vehicle worth dataset, you'd come up with $\theta_0 = 46.342$ and $\theta_1 = -0.116$ —the same as the intercept and slope mentioned before. This is a more robust way of solving linear regression problems than just a simple linear equation.

Linear Model with Higher Order Fits

Some data cannot easily be fit with a straight line. For example, in the following graph, the temperature of a particular location during the month of June fluctuates depending on the hour of day.

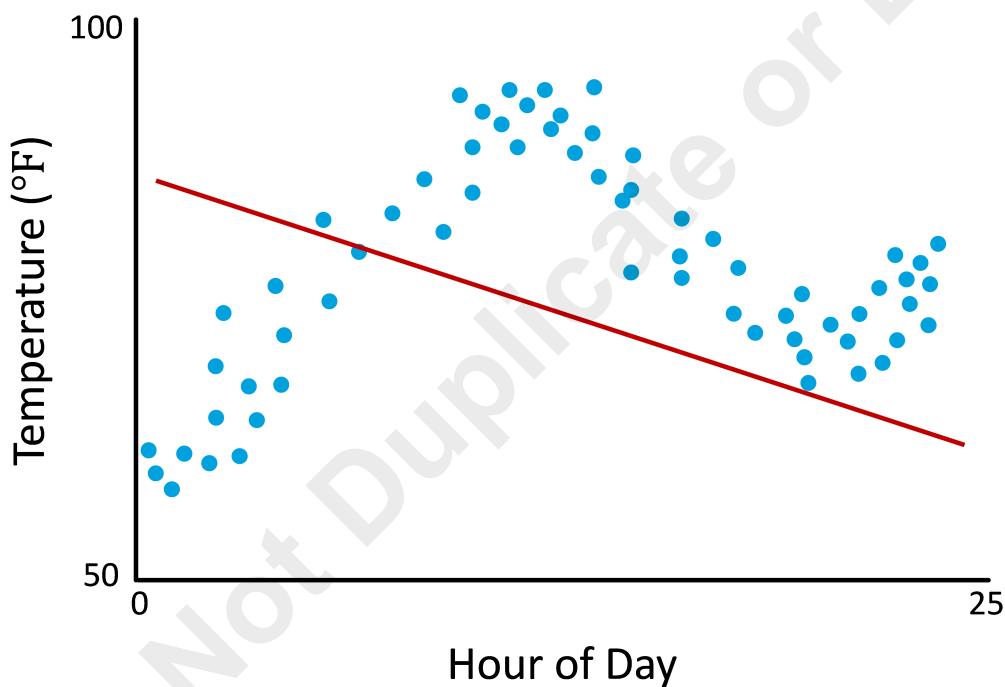


Figure 5-5: Attempting to fit a straight line to complex, non-linear data.

To obtain a higher order fit, you add more columns of data to the \mathbf{x} matrix. This can be represented as:

$$\begin{bmatrix} 1 & x_1^1 & x_1^2 & \cdots & x_1^k \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_n^1 & x_n^2 & \cdots & x_n^k \end{bmatrix}$$

In this case, k represents the total number of columns you're adding to the matrix. The new columns can be any types of transformations of the x values, like logarithm or exponent. For example:

$$\begin{bmatrix} 1 & x_1^1 & \exp(x_1) & \log(x_1) & \cdots & x_1^k \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_n^1 & \exp(x_n) & \log(x_n) & \cdots & x_n^k \end{bmatrix}$$

When applied to a linear equation, this would be:

$$\begin{bmatrix} \theta_0 \\ \vdots \\ \theta_k \end{bmatrix} = \begin{bmatrix} 1 & x_1^1 & \exp(x_1) & \log(x_1) & \cdots & x_1^k \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_n^1 & \exp(x_n) & \log(x_n) & \cdots & x_n^k \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Then, of course, it could be run through the normal equation. Ultimately, the new line's fit will change depending on what k is. Larger k values will tend to overfit, so keep that in mind.

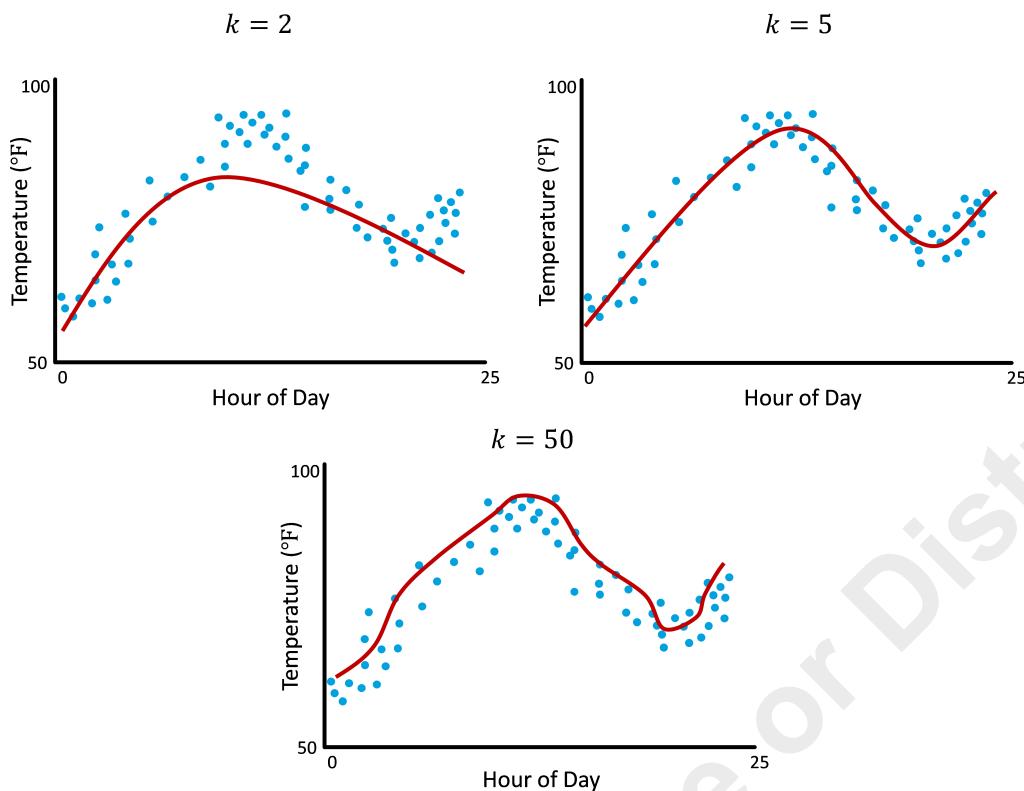


Figure 5–6: New line fits based on different k values.

Linear Model with Multiple Parameters

In the previous example, there was only one model parameter (besides the intercept): θ_1 . Depending on the data, there can be many more. So, with multiple parameters, linear regression can be expressed as a series of weighted features:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_d x_d$$

Where:

- d is the total number of model parameters (features).
- θ_i is the i^{th} model parameter (and the intercept).
- x_i is the i^{th} value of the feature.

This can also be represented in sigma notation:

$$\hat{y} = \theta_0 + \sum_{i=1}^d \theta_i x_i$$

In matrix form, inputting multiple predictive features is essentially the same as adding permutations to higher order models: you add more columns to the \mathbf{x} matrix. Each column represents an

additional feature that the model learns from. So, with several factors going into determining a vehicle's value, the matrix might end up looking something like this:

$$\begin{bmatrix} 1 & \text{mi}_1 & \text{make}_1 & \text{mod}_1 & \text{yr}_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \text{mi}_n & \text{make}_n & \text{mod}_n & \text{yr}_n \end{bmatrix}$$

As before, this can be plugged into the normal equation to calculate the model parameters θ_i . You can also combine this multi-parameter approach with a higher order approach by including permutations of the feature values. This helps to generate a non-linear fit to data that has multiple predictive features.

Cost Function

To find the best possible fit for a linear model, you need a way to determine the values for the model parameters (θ_i) that will lead to this best fit. Doing so involves evaluating the performance of the model on the training data. Rather than evaluating how *well* the model makes predictions, in linear regression, it's more common to evaluate how *poorly* it predicts—in other words, its cost.

A **cost function** attempts to quantify the error between the predicted values and the actual labeled training values. It does this by calculating the difference between the two. In other words, the machine learning model identifies how bad it is at estimating the relationship between the independent and dependent variables (x and y). A significant part of the learning process is the act of minimizing this cost function by determining the optimal model parameters. The normal equation, for example, is a method of minimizing the cost function.

Mean Squared Error (MSE)

There are several different methods for selecting a cost function. In linear regression, the simplest and often most effective cost function is the **mean squared error (MSE)**. Once the error (difference) between the predicted value and the actual value is calculated, that error is squared. Then, the average of these squared errors is taken. The reason for squaring the error is because errors can either be positive or negative; by simply taking the average of these values, the average will trend toward 0, which would not be very useful. Squaring the errors before taking the average makes them all positive.

This can be expressed as:

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Where:

- n is the size of the training set, i.e., the total number of examples.
- y_i is the actual value of a variable.
- \hat{y}_i is the predicted value of the variable.
- $J(\boldsymbol{\theta})$ is the cost function itself.

Mean Absolute Error (MAE)

An alternative for calculating the cost function is the **mean absolute error (MAE)**, which calculates the average difference in values (the absolute value between y and \hat{y}) without considering the sign (i.e., positive or negative) of those values. One advantage of MAE over MSE is that MSE tends to minimize errors that are much less than 1, while also exaggerating errors that are much greater than 1; MAE is not as susceptible to this. However, MSE is often preferred because it is differentiable (a derivative exists for all values in the function) and it matches what the linear equation is minimizing.

MAE can be expressed as:

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Coefficient of Determination

The **coefficient of determination**, also referred to as R^2 , is a statistical measure that indicates how much of a dependent variable's variance is explainable by a statistical model. In other words, it is the ratio of explained variation to total variation.

R^2 is commonly used as a scoring method for evaluating the performance of a linear regression model. R^2 is always between 0 and 1. If the regression line were able to perfectly pass through every data point, the R^2 would be 1; on the other hand, as the line misses more and more points, its R^2 decreases, as it becomes less and less able to explain the variance. For example, an R^2 of 0.76 indicates that 76% of the variance in the target variable (i.e., the label you're trying to predict) is explainable by the model. The other 24% cannot be explained by the model.

It is important to note that a high R^2 value does not always imply a more skillful model, and vice versa. A model with a low R^2 may have better predictive power than a model with a high R^2 value. Even if the R^2 changes dramatically from model to model, the predictive error may still be exactly the same. This is because a strong connection between variables does not always imply that one variable has strong causality with another. Therefore, it's usually preferable to minimize a cost function like MSE rather than attempt to optimize R^2 .



Note: The coefficient of determination is not to be confused with the correlation coefficient. The latter is R , while the former is R^2 . As you might expect, the former takes the latter and multiplies it by itself. Whereas R^2 measures explained variance, R measures linear relationships between variables.

Linear Model Code Example

The following is a snippet of Python code that builds a simple linear regression model. The `dataset` object is a hypothetical dataset of 5,000 vehicles, each with 10 possible features (mileage, make, model, etc.). The data has already been normalized and standardized. The label in this case is the value of the vehicle, which is the feature in column 10. In this example, the model is trained on just one feature (mileage).

```
# Import applicable modules.
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error as mse
import matplotlib.pyplot as plt

dataset = np.loadtxt('/path/vehicle_data.csv', delimiter = ',') # Load CSV file.
```

Licensed For Use Only By: Abdulwahab Alweban dev.abdulwahab@gmail.com Jan 8 202

```

X = dataset[:, 0] # Extract mileage feature.
X = X[:, np.newaxis] # Convert to NumPy matrix.
y = dataset[:, 9] # Extract label (vehicle value).
y = y[:, np.newaxis]

n_split = int(len(dataset) / 2) # Split dataset in two.
X_train = X[:n_split] # Extract first half of examples for train set.
y_train = y[:n_split]
X_test = X[n_split:] # Extract second half of examples for test set.
y_test = y[n_split:]

lin_reg = linear_model.LinearRegression() # Create linear regression object.
lin_reg.fit(X_train, y_train) # Train model on data.
predict = lin_reg.predict(X_test) # Use test data for prediction.

cost = mse(y_test, predict) # Calculate cost with MSE.
print('Cost (MSE): {:.2f}'.format(cost))

plt.plot(X_train, y_train, 'bo') # Plot training data.
plt.plot(X_test, predict, color = 'red') # Plot line of best fit.
plt.xlabel('Mileage (K)')
plt.ylabel('Value ($K)')
plt.show()

```

Note that the `LinearRegression()` class in scikit-learn automatically computes the normal equation under the hood. To prove that this is the case, you can also compute the model parameters manually, like so:

```

eq_part_1 = np.linalg.inv(X_train.transpose().dot(X_train))
eq_part_2 = X_train.transpose().dot(y_train)
model_params = eq_part_1.dot(eq_part_2)

```



Note: NumPy's `dot()` function returns the dot product of two arrays (in other words, it multiplies matrices). The `linalg.inv()` function computes the inverse of a matrix, and `transpose()` computes the transpose.

Normal Equation Shortcomings

While the normal equation tends to work very well for many machine learning regression tasks, it is not the perfect solution in every scenario. Besides the non-square matrix issue that is solved by taking the pseudoinverse, there are two general issues that arise when using this method of linear algebra:

- Memory issues with large datasets. If the training data includes a very large number of examples and/or features, computing the inverse of the data matrix will require an impractical amount of memory. This problem necessitates an iterative method for calculating the cost function.
- Overfitting issues. Datasets with large numbers of features may fail to generalize to new test data. One common method of reducing overfitting is to simplify the model through regularization.

Guidelines for Building a Regression Model Using Linear Algebra

Follow these guidelines when you are building simple linear regression models.



Note: All of the Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Build a Linear Regression Model

When building a simple linear regression model:

- Use linear regression in supervised learning to predict the change in value of a numeric dependent variable in relation to one or more independent variables.
- Consider that some data is not easily fit by a straight line—i.e., higher order fits.
- Use various transformations on the data, like exponent and logarithm, and add them as new features to account for higher order fits.
- Observe how adding more feature transformations changes a higher order fit, and consider that adding too many will lead to overfitting.
- Consider that linear regression models are usually not evaluated based on how well they fit, but on how poorly they fit (the cost).
- Prefer using a cost function like mean squared error (MSE) over R^2 when evaluating the skill of a linear regression model.
- Prefer using MSE over mean absolute error (MAE).
- Keep in mind the shortcomings of the closed-form normal equation: that large datasets will lead to memory issues due to the computational cost of inverse matrix calculations, and that it is prone to overfitting.

Use Python for Linear Regression

The scikit-learn `LinearRegression()` class enables you to construct a machine learning model using a basic linear regression algorithm. The following are some of the objects and functions you can use to build such a model.

- `model = sklearn.linear_model.LinearRegression()` —This constructs a model object that uses the linear regression algorithm.
- `model.fit(X_train, y_train)` —Fit a set of training data to the model. The first argument is a dataframe or array with the training set (not including labels), whereas the second argument is a dataframe or array of just the labels.
- `model.score(X_test, y_test)` —Return the coefficient of determination (R^2) score for the model given validation/test data. The first argument is a dataframe or array with the validation/test set (not including labels), whereas the second argument is a dataframe or array of just the labels.
- `model.predict(X_test)` —The model makes predictions on the validation/test set, without being given labels.
- `sklearn.metrics.mean_squared_error(y_test, prediction)` —Return the mean squared error (MSE) of the model given validation/test data. The second argument is an object created by `model.predict()`, as in the previous item.
- `model.coef_` —An attribute that lists the optimal model parameters generated during training.



Note: `X_train/val/test` and `y_train/val/test` are conventional variable names that are used throughout this course.

ACTIVITY 5-1

Building a Regression Model Using Linear Algebra

Data Files

/home/student/CAIP/Linear Regression/LinearRegression-PowerPlant.ipynb

/home/student/CAIP/Linear Regression/power_plant_data/cc_power_plant_data.csv

Before You Begin

Jupyter Notebook is open.

Scenario

IOT Company supplies network-connected sensors used to measure a variety of factors in industrial settings. One application of these sensors is in power plants, where they measure ambient conditions inside and outside the plant, like temperature and humidity. These conditions can have an impact on the efficiency of the power plant's energy output.

You've been given a dataset with hourly measurements over a six-year period, and are tasked with predicting the energy output of the plant given certain measurements. Because the energy output is measured in megawatts (MW), a numeric value, you'll use a simple linear model to make these predictions.



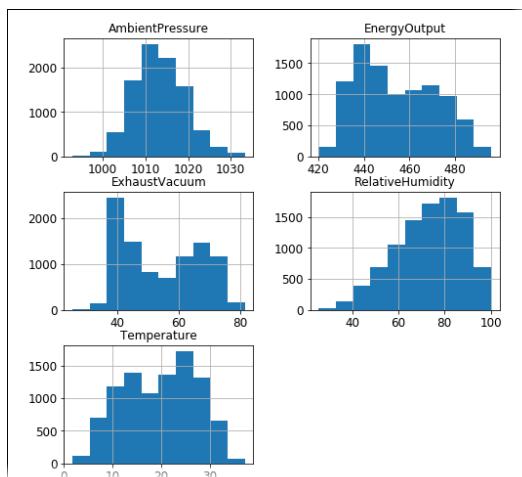
Note: This dataset was retrieved from the UCI Machine Learning Repository at: <https://archive.ics.uci.edu/ml/datasets/combined+cycle+power+plant>.

1. From Jupyter Notebook, select **CAIP/Linear Regression/LinearRegression-PowerPlant.ipynb** to open it.
2. Import the relevant libraries and load the dataset.
 - a) View the cell titled **Import software libraries and load the dataset**, and examine the code listing below it.
 - b) Select the cell that contains the code listing, then select **Run**.
 - c) Verify that **cc_power_plant_data.csv** was loaded with 9,568 records.
3. Get acquainted with the data.
 - a) Scroll down and view the cell titled **Get acquainted with the dataset**, and examine the code listing below it.
 - b) Select the cell that contains the code listing, then select **Run**.
 - c) Examine the output.
 - The training set includes 9,568 rows and 5 columns.
 - All of the columns contain float values.
 - There is no missing data; all rows have values for every column.
 - Each column refers to a particular measurement taken from sensors placed around the power plant:
 - **Temperature** is the temperature of the system in Celsius.
 - **ExhaustVacuum** measures the pressure of air as it is pushed out of the system.
 - **AmbientPressure** is the air pressure surrounding the system.

- `RelativeHumidity` measures humidity at a given temperature.
- `EnergyOutput` is the net electrical energy output by the system in megawatts.
- Each row represents an hourly average for each measurement over a six-year period.
- The `EnergyOutput` column will be treated as the label the model will try to predict.

4. Examine the distribution of the features.

- Scroll down and view the cell titled **Examine the distribution of the features**, and examine the code listing below it.
- Select the cell that contains the code listing, then select **Run**.
- Examine the output.



- The distribution for `AmbientPressure` is roughly symmetrical.
- The distribution for `RelativeHumidity` appears left-skewed.
- The other features' distributions are more varied.

5. Examine a general summary of statistics.

- Scroll down and view the cell titled **Examine a general summary of statistics**, and examine the code listing below it.
- Select the cell that contains the code listing, then select **Run**.

- c) Examine the output.

```

Temperature    ExhaustVacuum    AmbientPressure    RelativeHumidity    \
count          9568.00          9568.00          9568.00          9568.00
mean           19.65            54.31            1013.26           73.31
std            7.45             12.71             5.94             14.60
min            1.81              25.36            992.89            25.56
25%           13.51             41.74            1009.10           63.33
50%           20.34             52.08            1012.94           74.97
75%           25.72             66.54            1017.26           84.83
max            37.11             81.56            1033.30           100.16

EnergyOutput
count          9568.00
mean           454.37
std            17.07
min            420.26
25%           439.75
50%           451.55
75%           468.43
max            495.76

```

- Compared to the other features, `AmbientPressure` and `EnergyOutput` seem to exhibit a low amount of variance, as their minimum and maximum values are relatively close together.
- Likewise, the scale of these two features seems out of alignment with the other features. However, feature scaling does not improve a simple linear model's skill, so you'll leave the features as they are.

6. Look for columns that correlate with `EnergyOutput`.

- Scroll down and view the cell titled **Look for columns that correlate with `EnergyOutput`**, and examine the code listing below it.
- Select the cell that contains the code listing, then select **Run**.
- Examine the output.

```

Correlations with EnergyOutput
EnergyOutput      1.000000
AmbientPressure   0.518429
RelativeHumidity  0.389794
ExhaustVacuum     -0.869780
Temperature        -0.948128
Name: EnergyOutput, dtype: float64

```

All four of the features have a decent amount of correlation, with `AmbientPressure` having the highest positive correlation (i.e., as the pressure goes up, so does the energy output) and `ExhaustVacuum` and `Temperature` having the highest negative correlation (i.e., as they increase, the energy output decreases). You'll therefore use all of these features during training.

7. Split the datasets.

- Scroll down and view the cell titled **Split the datasets**, and examine the code listing below it. This code will split the dataset and training labels.
- Select the cell that contains the code listing, then select **Run**.
- Examine the output.

```

Original set:      (9568, 5)
-----
Training features: (7176, 4)
Test features:     (2392, 4)
Training labels:   (7176, 1)
Test labels:       (2392, 1)

```

The original training dataset has now been split into two: one set to continue using as training, the other to use for testing. Note that `EnergyOutput` label has been removed from the `x` matrices and placed into its own `y` vector.

8. Create a linear regression model.

- a) Scroll down and view the cell titled **Create a linear regression model**, and examine the code listing below it.

Create a linear regression model

```
In [ ]: 1 from sklearn.linear_model import LinearRegression
2
3 lin_reg = LinearRegression(fit_intercept = False)
4 start = time()
5 lin_reg.fit(X_train, np.ravel(y_train))
6 end = time()
7 train_time = (end - start) * 1000
8
9 # Score using the test data.
10 score = lin_reg.score(X_test, y_test)
11
12 print('Linear regression model took {:.2f} milliseconds to fit.'.format(train_time))
13 print('Variance score on test set: {:.0f}%'.format(score * 100))
```

The `LinearRegression()` class is being used to fit a simple linear model. The `fit_intercept` argument determines whether or not the intercept is calculated; in this case, it is being set to `False` for the sake of simplicity.

- b) Select the cell that contains the code listing, then select **Run**.
 c) Examine the output.

```
Linear regression model took 25.64 milliseconds to fit.
Variance score on test set: 91%
```

By default, the `score()` method for a `LinearRegression()` object returns the R^2 value of the prediction, also known as the coefficient of determination. It summarizes the amount of variance that the model explains; so, in this case, 91% of the variance in the dependent variable (`EnergyOutput`) is explainable.

Since there are better ways of measuring a model's skill—by using a cost function to measure prediction error, for example—you won't be using R^2 as the desired metric.

9. Compare the first ten predictions to actual values.

- a) Scroll down and view the cell titled **Compare the first ten predictions to actual values**, and examine the code listing below it.
 This code will generate new columns for the predicted and actual energy output, and show a sample of ten records for comparison.
- b) Select the cell that contains the code listing, then select **Run**.
 c) Examine the output.

	Temperature	ExhaustVacuum	AmbientPressure	RelativeHumidity	PredictedEnergyOutput	ActualEnergyOutput
7466	25.68	70.32	1009.08	90.48	436.20	431.32
8561	17.94	62.10	1019.81	82.65	457.51	453.55
7777	10.60	41.17	1018.38	87.92	474.29	478.47
176	12.28	40.55	1005.72	98.56	464.26	472.47
6342	9.75	52.72	1026.03	78.53	477.28	473.41
74	10.25	41.46	1018.67	84.41	475.27	479.28
2155	23.73	63.94	1010.70	87.10	442.35	441.78
9057	16.84	39.63	1004.67	82.98	457.90	466.06
1675	24.78	68.63	1013.96	43.70	445.15	448.43
771	25.72	59.21	1012.37	66.62	443.16	437.72

10. Compare the error between predicted and actual values.

- a) Scroll down and view the cell titled **Calculate the error between predicted and actual values**, and examine the code listing below it.

- b) Select the cell that contains the code listing, then select **Run**.
- c) Examine the output.

```
Cost (mean squared error): 25.87
```

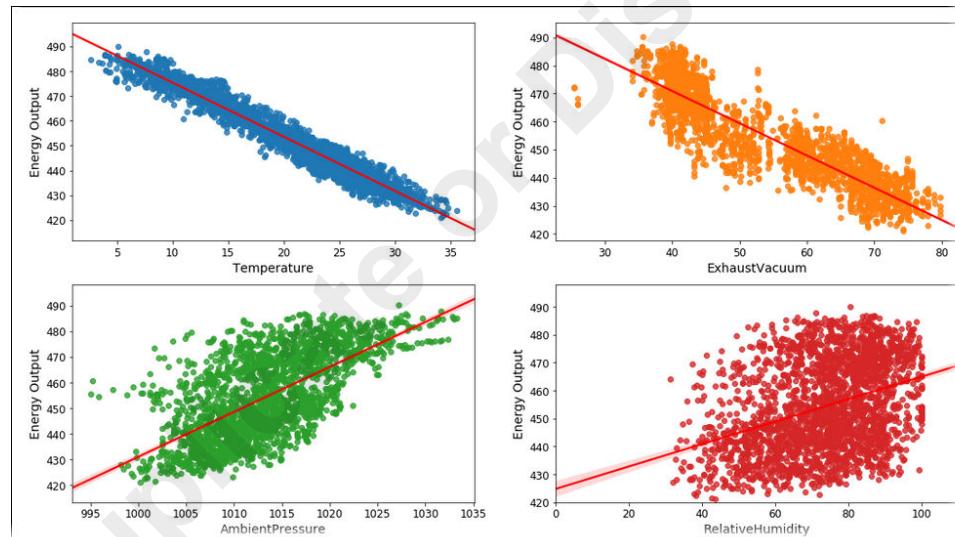
The result is the mean squared error (MSE) for the model. The lower the MSE, the higher the model's predictive skill. Different models based on different datasets will lead to different MSE values, so there is not one objectively "correct" value to shoot for.

11. Plot lines of best fit for all features.

- a) Scroll down and view the cell titled **Plot lines of best fit for all features**, and examine the code listing below it.

This code will produce four plots enabling you to view the lines of best fit for temperature, exhaust vacuum, ambient pressure, and relative humidity.

- b) Select the cell that contains the code listing, then select **Run**.
- c) Examine the output.



- Each feature is plotted against the `EnergyOutput` label, with a line of best fit generated from the linear regression.
- Note that the appearance of each scatter plot seems to align with the correlations identified earlier. For example, `Temperature` and `ExhaustVacuum` both exhibited high negative correlation with the `EnergyOutput`, so their data points support a straight line fit rather well. On the other hand, `RelativeHumidity` had a much lower correlation, and its data points are scattered in such a way that a straight line does not fit as tightly.

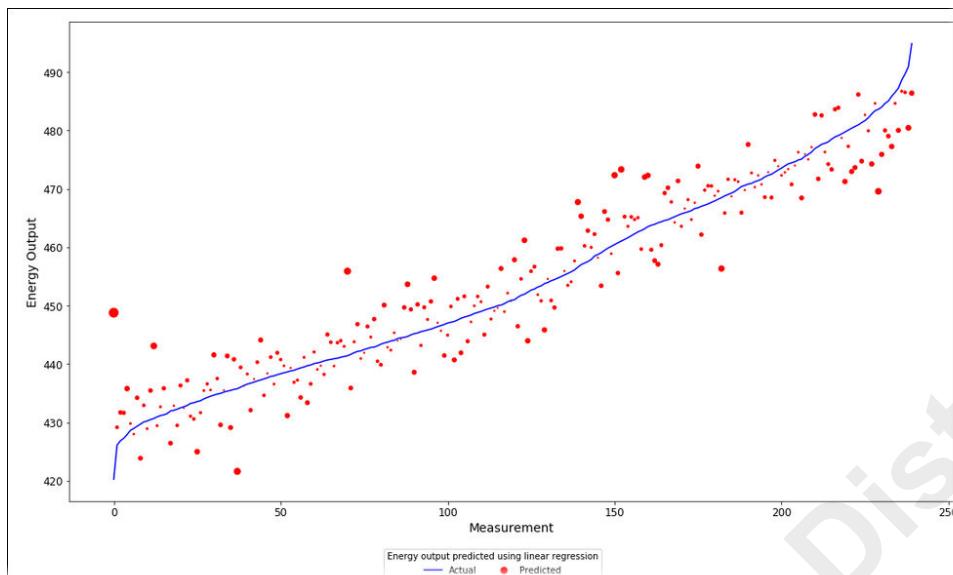
12. Compare predicted values to actual values.

- a) Scroll down and view the cell titled **Compare predicted values to actual values**, and examine the code listing below it.

This code will generate a combination line chart and scatter plot to visually compare the predicted values to actual values.

- b) Select the cell that contains the code listing, then select **Run**.

- c) Examine the output.



- The dataset has been sorted by the actual energy output per hourly measurement, which is shown as the blue line.
- The predicted energy output at each hourly measurement is shown as a red dot in a scatter plot. The larger the dot, the higher the amount of error between predicted and actual values.
- For the most part, the predicted output corresponds rather closely to the actual output.

13. Retrieve the model parameters.

- Scroll down and view the cell titled **Retrieve the model parameters**, and examine the code listing below it.
This code will print out the model parameters that the linear regression model uses to make its predictions.
- Select the cell that contains the code listing, then select **Run**.
- Examine the output.

```
Temperature coefficient: -1.6668474737282652
ExhaustVacuum coefficient: -0.27545747113877944
AmbientPressure coefficient: 0.5025758077163675
RelativeHumidity coefficient: -0.09689119750793702
```

These coefficients are the parameters that were derived from the linear regression model. They are used by the model to make predictions on input data.

14. Manually calculate the model parameters using the normal equation.

- Scroll down and view the cell titled **Manually calculate the model parameters using the normal equation**, and examine the code listing below it.
This code manually forms the normal equation that is used in simple linear regression.
- Select the cell that contains the code listing, then select **Run**.

- c) Examine the output.

```
[[ -1.66684747]
 [ -0.27545747]
 [ 0.50257581]
 [ -0.0968912 ]]
```

As you can see, plugging the training data into the normal equation gives you the exact same model parameters. This is because the `LinearRegression()` class uses the normal equation to generate a predictive model.

15. Shut down this Jupyter Notebook kernel.

- a) From the menu, select **Kernel**→**Shutdown**.
 - b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
 - c) Close the **LinearRegression-PowerPlant** tab in Firefox, but keep a tab open to **CAIP/Linear Regression/** in the file hierarchy.
-

ACTIVITY 5–2

(Optional) Building a Linear Regression Model to Predict Diabetes Progression

Data File

/home/student/CAIP/Optional/LinearRegression.ipynb

Before You Begin

Jupyter Notebook is open.

Scenario

You're doing consultant work for Greene City Physicians Group (GCPG), a medical practice that provides treatment in several different fields. One of those fields is endocrinology. The endocrinologists treat hundreds of different diabetic patients, helping them manage the disease. Preventing the disease from reaching the more severe stages is of primary importance, but knowing when a patient is at risk of progressing to the later stages is not always easy.

The endocrinologists have supplied you with an anonymized dataset of past diabetic patients, including various medical attributes that might be indicators of disease progression. You've been asked to help the doctors predict the progression of the disease in patients. Since this is an ordinal numeric value, you'll create a linear regression model.

The following are the attributes of the dataset:

Attribute	Description
age	The patient's age in years.
sex	The patient's sex.
bmi	The patient's body mass index (BMI).
bp	The patient's average blood pressure.
s1–s6	Six different blood serum measurements taken from the patient.
target	A measurement of the disease's progression one year after a baseline.



Note: Sometimes, when you run code containing logic errors or bugs, you may corrupt the data contained in variables you created in previous code cells. To clear out such problems, you can select **Kernel→Restart & Clear Output**, then run each code cell again.

1. Open the linear regression notebook.

- From Jupyter Notebook, select **CAIP/Optional/LinearRegression.ipynb** to open it.



Note: Be careful *not* to open the solution file.

- Observe the notebook.

Placeholder code cells have been provided in which you can add your own code. Comments provide hints on tasks you might perform in each code cell. The first code cell has already been completely programmed for you.

2. Import software libraries and load the dataset.

- Select the code listing under **Import software libraries and load the dataset**.
- Run the code and examine the results.

The dataset is loaded for you. There are 442 records.

3. Get acquainted with the dataset.

- In the code block under **Get acquainted with the dataset**, write code to convert the loaded dataset to a `DataFrame`, view the data types for each feature, and then view the first 10 records.



Note: Except for the label, the features have already been standardized.

- Run the code and verify that the shape of the training data is shown.

4. Examine the distribution of various features.

- In the code block under **Examine the distribution of various features**, write code to plot distribution histograms for all features.
- Run the code and verify that the distributions are shown.

5. Examine a general summary of statistics.

- In the code block under **Examine a general summary of statistics**, write code to view summary statistics (mean, standard deviation, min, max, etc.) for each feature.
- Run the code and verify that the summary statistics are printed.

6. Look for columns that correlate with target (disease progression).

- In the code block under **Look for columns that correlate with target (disease progression)**, write code to view the correlation values for each feature compared to the label.
- Run the code and examine the feature correlations.

7. Split the label from the dataset.

- In the code block under **Split the label from the dataset**, write code to split the data into train and test sets, as well as split the label from the features.
- Run the code and observe the shape of the split datasets.

8. Drop columns that won't be used for training.

- In the code block under **Drop columns that won't be used for training**, write code to drop the three features that have the least correlation with the label.
One of these features is being dropped because it is categorical. You could technically encode that feature instead, but for simplicity's sake, you'll drop it.
- Run the code and verify that the columns were dropped.

9. Create a linear regression model.

- In the code block under **Create a linear regression model**, write code to construct a basic linear regression class object, then fit the training data to that object.
- Run the code.

10. Compare the first ten predictions to actual values.

- In the code block under **Compare the first ten predictions to actual values**, write code to make predictions on the test set. Then, view the first 10 records with two new columns: the actual label value for that record (disease progression), and the value that your model predicted for that record.

- b) Run the code and observe the predictions.

11.Calculate the error between predicted and actual values.

- a) In the code block under **Calculate the error between predicted and actual values**, write code to print the mean squared error (MSE) for the model's predictions on the test set.
- b) Run the code and observe the error value.

12.Plot lines of best fit for four features.

- a) In the code block under **Plot lines of best fit for four features**, write code to generate scatter plots for the four features that exhibited the strongest correlation with the label. Also plot a line of best fit for each feature on top of its scatter plot.
- b) Run the code and examine the plots.

13.In Jupyter Notebook, open **CAIP/Optional/LinearRegression-Solution.ipynb and compare it to the code you wrote.**



Note: Since there are many ways to write code to do the same basic thing, don't expect your code to match exactly. The important thing is that your code accomplishes the same basic goals, and returns similar results.

14.Shut down the Jupyter Notebook kernels.

- a) From the menu, select **Kernel->Shutdown**.
- b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
- c) Close the **LinearRegression** tab in Firefox.
- d) Repeat this process to shut down and close the **LinearRegression-Solution** kernel.
- e) Return to **CAIP/Linear Regression/** in the file hierarchy.

TOPIC B

Build Regularized Regression Models Using Linear Algebra

The simple model you created earlier works well in many cases, but that doesn't mean it's the optimal approach. Linear regression can be enhanced by the process of regularization, which will often improve the skill of your machine learning model.

Regularization Techniques

Recall that regularization is the process of simplifying a model by applying constraints to the model parameters. It does this by applying the λ hyperparameter. More specifically, this hyperparameter is part of a term that is added to the cost function. This term penalizes the model if its parameter values are too high, and therefore keeps the parameter values small. There are actually several techniques for applying regularization to a model, each of which uses a different regularization term. The three regularization techniques that have found the most success are:

- Ridge regression
- Lasso regression
- Elastic net regression

In most cases, whichever specific technique you choose, it's a good idea to apply at least some form of regularization while training a linear regression model. However, you should only perform regularization during training; when you evaluate the model's performance after training, you must not use the regularization hyperparameter.

Ridge Regression

Ridge regression uses a mathematical function called an ℓ_2 norm to implement its regularization term. The ℓ_2 norm is the sum of square coefficients, and the objective is to minimize it. This helps to keep the model parameter weights small, reducing overfitting. In the following equation, the ridge regression term is applied to an MSE cost function:

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \lambda \sum_{i=1}^d \theta_i^2$$

Where:

- $J(\boldsymbol{\theta})$ is the cost function (now with a regularization term applied).
- $\text{MSE}(\boldsymbol{\theta})$ is the cost function before regularization. This is the same as the mean squared error (MSE) equation.
- λ is the regularization hyperparameter.
- d is the total length of $\boldsymbol{\theta}$.
- θ_i is a model parameter.



Note: Since the summation starts at 1, θ_0 is not regularized.

Ridge regression is suitable in datasets featuring a large number of features, each having at least some predictive power. This is because ridge regression helps to reduce overfitting without actually removing any of the features entirely.

Collinearity

Ridge regression addresses the issue of **collinearity**, in which multiple features exhibit a linear relationship—in other words, the features are either exactly or very closely related in terms of how they influence the dependent variable. Collinearity therefore makes it difficult to determine how each feature has an affect on the output independently.

Normal Equation with Ridge Regression

When using ridge regression, the normal equation becomes:

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \cdot \mathbf{X}^T \mathbf{y}$$

Where \mathbf{I} is an identity matrix. The identity matrix is d by d in size. In the case of regularization, the only difference is that the top-left value in the identity matrix is a 0 so that θ_0 isn't regularized.

Lasso Regression

Lasso regression uses the ℓ_1 norm to implement its regularization term. The ℓ_1 norm forces the coefficients of the least relevant features to 0—in other words, removing them from the model. Like with ridge regression, this helps the model avoid overfitting to the training data. When applied to MSE, the lasso regression term looks like the following:

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \lambda \sum_{i=1}^d |\theta_i|$$

As opposed to ridge regression, lasso regression is suitable in datasets that have only a small or moderate number of features that have moderate predictive power. Lasso regression is able to eliminate the rest of the features that have no significant effect on the data, which may lead to better model performance than keeping and reducing them, as in ridge regression. The elimination of unnecessary features essentially means that lasso regression performs a type of dimensionality reduction—specifically, feature selection.



Note: Lasso regression initially stood for "least absolute shrinkage and selection operator regression," but is primarily known as just its shortened version.

Elastic Net Regression

Elastic net regression uses a weighted average of both ridge and lasso regression as part of its regularization term. It is therefore an attempt to leverage the best of both ℓ_1 and ℓ_2 norms. Along with λ , elastic net regression also uses the α ratio hyperparameter. An α of 0 uses the ℓ_1 regularization term, and an α of 1 uses the ℓ_2 term. So, in elastic net, the objective is to find a value between 0 and 1 that optimizes the regularization of the model. The elastic net term can be defined as follows:

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \lambda \left(\frac{1-\alpha}{2} \sum_{i=1}^d \theta_i^2 + \alpha \sum_{i=1}^d |\theta_i| \right)$$



Note: α is the Greek letter alpha.

As you can see, the elastic net equation incorporates both the ℓ_1 and ℓ_2 norms, as well as the α hyperparameter that defines the ratio between the two.

Like lasso regression, elastic net regression tends to work well when there are only a small to moderate number of features that are actually relevant (though, depending on α , it can also equal ridge regression). Elastic net regression is typically preferable, as lasso regression may not perform optimally when the number of features far outnumber the training examples. Likewise, elastic net regression tends to perform better in situations where multiple features exhibit high correlation. Pure ridge regression may still be ideal if removing even a small number of features could impair the model's predictive skill.



Caution: In scikit-learn, the λ hyperparameter is actually called `alpha`, whereas α is called `l1_ratio`.

Guidelines for Building a Regularized Linear Regression Model

Follow these guidelines when you are building regularized linear regression models.

Build a Regularized Linear Regression Model

When building a regularized linear regression model:

- Perform regularization only during training.
- Consider using ridge regression with datasets that have a large number of features, each having at least some predictive power.
- Consider using lasso regression with datasets that have a small to moderate number of features that have moderate predictive power.
- Consider using elastic net regression with datasets that have a small to moderate number of features that are relevant.
- Prefer to use elastic net regression over lasso regression in most cases.
- Prefer to use ridge regression if removing even a small number of features can have a negative affect on the model's skill.
- Consider applying multiple types of regularization to a model in order to identify which performs better.
- Prefer to use any type of regularization over not using regularization at all.

Use Python for Regularized Linear Regression

Several scikit-learn classes can provide regularization techniques to linear regression. The three basic classes that provide this directly are `Ridge()`, `Lasso()`, and `ElasticNet()`. The following are some of the objects and functions you can use to build such models.

- `model = sklearn.linear_model.Ridge(alpha = 0.1)` —This constructs a model object that uses ridge regression (ℓ_2 norm). The `alpha` parameter defines the strength of regularization to apply, which, in this case, is 0.1.
- `model = sklearn.linear_model.Lasso(alpha = 0.1)` —This constructs a model object that uses lasso regression (ℓ_1 norm). This uses the same `alpha` parameter.

- `model = sklearn.linear_model.ElasticNet(alpha = 0.1, l1_ratio = 0.5)` —This constructs a model object that uses elastic net regression (a weighted average of both ℓ_1 and ℓ_2 norms). In addition to using `alpha`, the object uses the `l1_ratio` parameter to add more weight to either type of regularization. A value of 0 is the same as ridge regression, whereas a value of 1 is the same as lasso regression. In this case, a value of 0.5 specifies an equal weight of both.
- You can use these class objects to call the same `fit()`, `score()`, and `predict()` methods as with `LinearRegression()`. You can also generate the MSE using `mean_squared_error()` and return the model parameters using the `coef_` attribute.

Do Not Duplicate or Distribute

ACTIVITY 5–3

Building a Regularized Linear Regression Model

Data File

/home/student/CAIP/Linear Regression/LinearRegression-Boston.ipynb

Before You Begin

Jupyter Notebook is open.

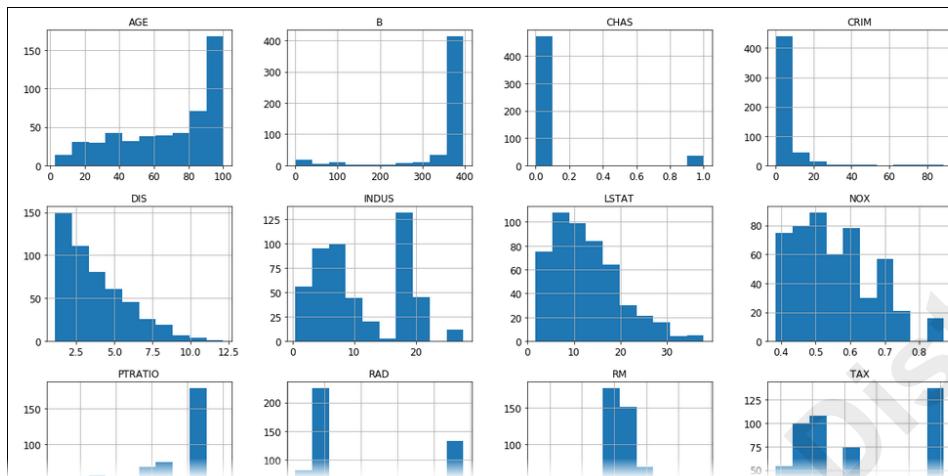
Scenario

You want to apply your house valuation models to more than just the examples provided in the King County dataset. You also have access to a dataset that includes data about houses in Boston, Massachusetts. Once again, you want to be able to train a model to predict the value of a home in this area given several factors. However, simple linear regression is not necessarily the best way to address this problem. You want to avoid running into issues where the model overfits to the training data, making it less useful in generalizing to new data. So, you'll apply the technique of regularization to your linear models to help simplify the models and avoid overfitting. Rather than just arbitrarily choosing one type of regularization, you'll evaluate all three (ridge, lasso, and elastic net), and then choose which one performs the best according to your requirements.

1. From Jupyter Notebook, select **CAIP/Linear Regression/LinearRegression-Boston.ipynb** to open it.
2. Import the relevant libraries and load the dataset.
 - a) Select the code cell beneath the **Import software libraries and load the dataset** title, then select **Run**.
 - b) Verify that 506 records were loaded.
This dataset actually comes pre-packaged with scikit-learn, and can be easily loaded into an array through the `load_boston()` function. There's no need to load a file stored on the local machine.
3. Get acquainted with the dataset.
 - a) Scroll down and select the code cell beneath the **Get acquainted with the dataset** title, then select **Run**.
 - b) Examine the output.
 - The training set includes 506 rows and 14 columns.
 - All of the columns contain float values.
 - There is no missing data; all rows have values for every column.
 - Each column measures some factor about a house in Boston. Some examples include:
 - **CRIM** is the per-capita crime rate of the area.
 - **CHAS** refers to the "Charles River dummy variable"—if 1, the land bounds the river; if 0, it does not.
 - **NOX** is the level of nitrogen oxides (NO_2) in the area. High levels of NO_2 can cause health issues.
 - **RM** is the average number of rooms per house.
 - **AGE** is the proportion of occupied units built before 1940.
 - **DIS** is the weighted mean distance to several employment centers around Boston.
 - **TAX** is the property tax rate per \$10,000.
 - **PTRATIO** is the ratio of pupils (students) to teachers in the school district.
 - **target** also refers to **MEDV**, the median value of the house in thousands of dollars.

4. Examine the distribution of various features.

- Select the code cell beneath the **Examine the distribution of various features** title, then select **Run**.
- Examine the output.



Some highlights include:

- The age of the houses seems to be left-skewed, indicating that most houses in the dataset are quite old.
- The mean distance to employment centers is right-skewed, indicating that most houses are close to such centers.
- The NO_2 levels seem to fluctuate, though most tend to be on the low end.
- The number of rooms seems to form a roughly symmetrical distribution.
- The median house value (i.e., target) also seems to have a reasonably symmetrical distribution, with perhaps a few high outliers.

5. Examine a general summary of statistics.

- Select the code cell beneath the **Examine a general summary of statistics** title, then select **Run**.
- Examine the output.

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	\
count	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	
mean	3.61	11.36	11.14	0.07	0.55	6.28	68.57	3.80	9.55	408.24	
std	8.60	23.32	6.86	0.25	0.12	0.70	28.15	2.11	8.71	168.54	
min	0.01	0.00	0.46	0.00	0.39	3.56	2.90	1.13	1.00	187.00	
25%	0.08	0.00	5.19	0.00	0.45	5.89	45.02	2.10	4.00	279.00	
50%	0.26	0.00	9.69	0.00	0.54	6.21	77.50	3.21	5.00	330.00	
75%	3.68	12.50	18.10	0.00	0.62	6.62	94.07	5.19	24.00	666.00	
max	88.98	100.00	27.74	1.00	0.87	8.78	100.00	12.13	24.00	711.00	
	PTRATIO	B	LSTAT	target							
count	506.00	506.00	506.00	506.00							
mean	18.46	356.67	12.65	22.53							
std	2.16	91.29	7.14	9.20							
min	12.60	0.32	1.73	5.00							
25%	17.40	375.38	6.95	17.02							
50%	19.05	391.44	11.36	21.20							
75%	20.20	396.23	16.96	25.00							
max	22.00	396.90	37.97	50.00							

The feature scales seem to be disproportionate, especially when comparing lower-scale features like **CRIM** (whose mean is 3.61) and higher-scale features like **TAX** (whose mean is 408.24). Unlike with simple linear regression, regularized linear regression will benefit from some feature scaling. When features are scaled, the regularization penalty can be applied equally to the data, rather than giving undue weight to certain features. Scaling can also help reduce the effect of outliers on regularized linear regression.

6. Look for columns that correlate with target (median house value).

- Select the code cell beneath the **Look for columns that correlate with target (median house value)** title, then select **Run**.
- Examine the output.

```
Correlations with median house value
target    1.000000
RM        0.695360
ZN        0.360445
B         0.333461
DIS       0.249929
CHAS      0.175260
AGE       -0.376955
RAD        -0.381626
CRIM      -0.388305
NOX       -0.427321
TAX        -0.468536
INDUS     -0.483725
PTRATIO   -0.507787
LSTAT     -0.737663
Name: target, dtype: float64
```

It looks like many of these features have at least some correlation with the median house value, whether positive or negative. The lowest correlation is **CHAS**, the categorical variable that indicates whether a house's property borders the Charles River. Because the correlation appears to be weak, and because the feature is categorical and not numeric (like the other features), you'll drop this feature from training.

7. Split the label from the dataset.

- Select the code cell beneath the **Split the label from the dataset** title, then select **Run**.
- Examine the output.

Rather than using the holdout method to split the datasets into training set and validation/test set, you'll be training the data using cross-validation. Cross-validation can help improve the model's ability to generalize to new data. For now, you're just extracting the label from the training set (x) and placing it in its own vector (y).

8. Drop columns that won't be used for training.

- Select the code cell beneath **Drop columns that won't be used for training**, then select **Run**.
- Examine the output and observe that the **CHAS** column was dropped.

```
Columns before drop:
['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
Columns after drop:
['CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
```

9. Standardize the features.

- a) Scroll down and view the cell titled **Standardize the features**, and examine the code listing below it.

Standardize the features

```
In [ ]: 1 def standardize(X):
2     result = X.copy()
3
4     for feature in X.columns:
5         result[feature] = (X[feature] - X[feature].mean()) / X[feature].std() # z-score formula.
6
7     return result
8
9 X = standardize(X)
10
11 print('The features have been standardized.')
```

This function includes the *z*-score formula, which is used to standardize the data.

- b) Select the cell that contains the code listing, then select **Run**.

The features have been standardized.

- c) Scroll down and select the next code listing cell.

```
1 with pd.option_context('float_format', '{:.2f}'.format):
2     print(X.describe())
```

- d) Select **Run**.
e) Examine the output.

	CRIM	ZN	INDUS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
count	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00
mean	0.00	0.00	-0.00	0.00	-0.00	-0.00	0.00	-0.00	0.00	0.00	-0.00	-0.00
std	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
min	-0.42	-0.49	-1.56	-1.46	-3.88	-2.33	-1.27	-0.98	-1.31	-2.70		
25%	-0.41	-0.49	-0.87	-0.91	-0.57	-0.84	-0.80	-0.64	-0.77	-0.49		
50%	-0.39	-0.49	-0.21	-0.14	-0.11	0.32	-0.28	-0.52	-0.46	0.27		
75%	0.01	0.05	1.01	0.60	0.48	0.91	0.66	1.66	1.53	0.81		
max	9.92	3.80	2.42	2.73	3.55	1.12	3.96	1.66	1.80	1.64		

	B	LSTAT
count	506.00	506.00
mean	0.00	-0.00
std	1.00	1.00
min	-3.90	-1.53
25%	0.20	-0.80
50%	0.38	-0.18
75%	0.43	0.60
max	0.44	3.55

Each feature has been transformed to have a mean value of 0 and a standard deviation of 1. As a result, the values have been scaled down.

10. Train the model and calculate its scores.

- a) Scroll down and view the cell titled **Train a model and calculate its scores**, and examine the code listing below it.

Train a model and calculate its scores

```
In [ ]: 1 from sklearn.model_selection import cross_val_score
2 from sklearn.model_selection import cross_val_predict
3 from sklearn.metrics import mean_squared_error as mse
4
5 # Use cross-validation to split/train datasets.
6 def model_train(model):
7     predict = cross_val_predict(model, X, np.ravel(y), cv = 5)
8     score = cross_val_score(model, X, np.ravel(y), cv = 5).mean()
9     cost = mse(y, predict)
10
11     print('Mean variance score on test set: {:.0f}%'.format(np.round(score * 100)))
12     print('Cost (mean squared error): {:.2f}'.format(cost))
13
14 print('The function to train the model has been defined.')
```

This function both performs cross-validation on the dataset and trains the model:

- On line 6, the function takes `model` as an argument, which is the class object of whatever machine learning algorithm you wish to use.
- On line 7, rather than using the standard `fit()` method to begin training, the `cross_val_predict()` method performs both cross-validation and the actual training on the data it splits. The first argument it takes is the model, and the next two arguments are the datasets to train the model on (`X` and `y`).
- The `cv` argument specifies the number of folds to use in cross-validation. In other words, by supplying an integer, k -fold cross-validation will automatically be performed. Specifically, stratified k -fold is used as the default. You can also specify a different kind of cross-validation (like LOOCV), but for this dataset, stratified k -fold appears to be adequate.
- On line 8, the `cross_val_score()` method returns the R^2 score for the training, much like the standard `score()` method. However, it computes a score for all of the folds, so those five scores are being averaged to produce the overall score.
- On line 9, the mean squared error (MSE) for the predictions is being calculated.
- Line 14 prints a message to indicate the function has been defined.

- b) Select the cell that contains the code listing, then select **Run**.

11. Evaluate several regularized linear regression models.

- a) Scroll down and view the cell titled **Evaluate several regularized linear regression models**, and examine the code listing below it.

Evaluate several regularized linear regression models

```
In [ ]: 1 from sklearn.linear_model import LinearRegression
2 from sklearn.linear_model import Ridge
3 from sklearn.linear_model import Lasso
4 from sklearn.linear_model import ElasticNet
5
6 # Create non-regularized and regularized linear regression models.
7 def model_eval(a, l1):
8     for name, model in [
9         ('None', LinearRegression()),
10        ('Ridge', Ridge(alpha = a, solver = 'cholesky')),
11        ('Lasso', Lasso(alpha = a)),
12        ('Elastic net', ElasticNet(alpha = a, l1_ratio = l1))]:
13
14     print('Regularization: {}'.format(name))
15     print('-----')
16     model_train(model)
17     print('\n')
18
19 print('The function to evaluate the linear regression models has been defined.')
```

This function calls `model_train()` using four different algorithms:

- Simple linear regression (no regularization).
- Ridge regression.
- Lasso regression.
- Elastic net regression.

In addition:

- On line 7, the function takes two arguments: `a` is the regularization hyperparameter (i.e., λ) that is applied to all regularization algorithms; and `l1` is the regularization penalty ratio to use for elastic net. The lower this value, the closer the penalty is to the ℓ_2 norm (ridge); the higher the value, the closer the penalty is to the ℓ_1 norm (lasso).
- The `solver` argument for ridge regression refers to how the algorithm will minimize the cost function. In this case, `cholesky` is just the standard closed-form solution (the normal equation).

- b) Select the cell that contains the code listing, then select **Run**.

```
The function to evaluate the linear regression models has been defined.
```

By placing this code in a function, it can easily be repeated multiple times for comparison, passing in different parameters.

- c) Scroll down and select the next code listing cell.

```
In [ ]: 1 model_eval(1, 0.5)
```

- d) Select **Run**.

- e) Examine the output.

```
1 model_eval(1, 0.5)

Regularization: None
-----
Mean variance score on test set: 34%
Cost (mean squared error): 36.63

Regularization: Ridge
-----
Mean variance score on test set: 35%
Cost (mean squared error): 36.37

Regularization: Lasso
-----
Mean variance score on test set: 32%
Cost (mean squared error): 40.50

Regularization: Elastic net
-----
Mean variance score on test set: 33%
Cost (mean squared error): 39.97
```

- In this function call, you used an arbitrary starting point of 1 as the regularization hyperparameter.
 - You used an elastic net penalty of 0.5, which favors neither the ℓ_1 nor ℓ_2 norm.
 - For these hyperparameters, ridge regression appears to have both the highest variance score and the lowest MSE. The latter is of particular interest, as this is what you're trying to minimize.
 - Lasso and elastic net regression seem to have performed worse than the simple, non-regularized linear model.
 - You'll reconfigure these hyperparameters to hopefully get better results.
- f) Scroll down and select the next code listing cell.

```
In [ ]: 1 model_eval(0.1, 0.3)
```

- g) Select Run.
h) Examine the output.

```
1 model_eval(0.1, 0.3)

Regularization: None
-----
Mean variance score on test set: 34%
Cost (mean squared error): 36.63

Regularization: Ridge
-----
Mean variance score on test set: 34%
Cost (mean squared error): 36.60

Regularization: Lasso
-----
Mean variance score on test set: 39%
Cost (mean squared error): 35.53

Regularization: Elastic net
-----
Mean variance score on test set: 42%
Cost (mean squared error): 33.68
```

- In this function call, you changed the regularization hyperparameter to 0.1.
- You changed the elastic net penalty to 0.3, which favors the ℓ_2 norm (ridge).
- For these hyperparameters, elastic net regression appears to have the lowest MSE—even lower than the ridge regression model from the previous training round. The improvement may be slight, but you won't always see massive gains through regularization.

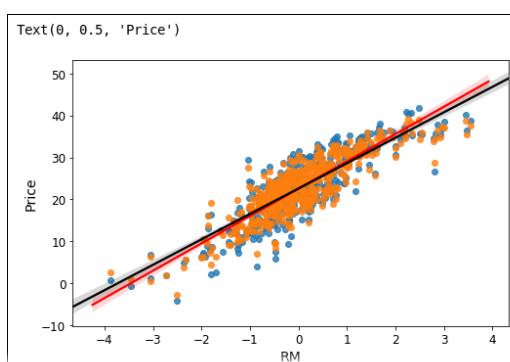
12. Are you satisfied with this MSE value? In other words, would you stop there and finalize the model? Why or why not?

13. Plot lines of best fit for the RM (average number of rooms) feature.

- Scroll down and view the cell titled **Plot lines of best fit for the RM (average number of rooms) feature**, and examine the code listing below it.

This code will generate a chart showing the line of best fit for the linear regression model.

- Select the cell that contains the code listing, then select **Run**.
- Examine the output.



- RM is being demonstrated here because it had one of the strongest correlations with the median house value.
- The red line is the unregularized line of best fit, whereas the black line is the line of best fit using elastic net with the hyperparameters that were deemed "good enough."
- The blue dots are the unregularized predictions, whereas the orange dots are the regularized predictions.
- As you can see, regularized regression produced slightly different predictions with a slightly different line of best fit, which reflects the evaluation you did earlier.

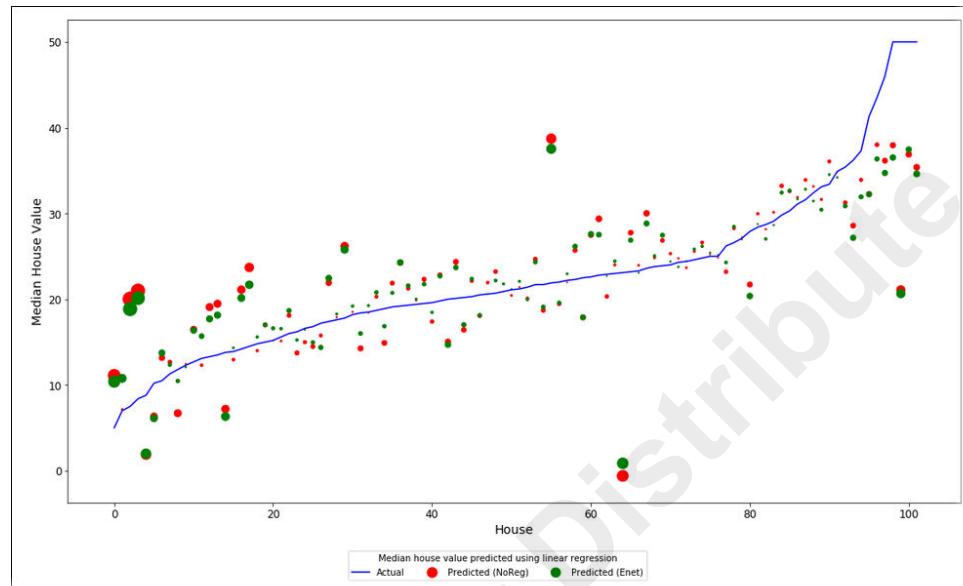
14. Compare predicted values to actual values.

- Scroll down and view the cell titled **Compare predicted values to actual values**, and examine the code listing below it.

This code will generate a combination line chart and scatter plot to visually compare the predicted values to actual values.

- Select the cell that contains the code listing, then select **Run**.

- c) Examine the output.



- The dataset has been sorted by the actual median house value, which is shown as the blue line.
- The predicted value of a house using unregularized linear regression is shown as a red dot in a scatter plot. The larger the dot, the higher the amount of error between predicted and actual values.
- The predicted value of a house using elastic net regression is shown as a green dot in a scatter plot.
- Each model gives slightly different predictions, and some predictions are closer to the target than others.

15. Shut down this Jupyter Notebook kernel.

- a) From the menu, select **Kernel**→**Shutdown**.
 - b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
 - c) Close the **LinearRegression-Boston** tab in Firefox, but keep a tab open to **CAIP/Linear Regression/** in the file hierarchy.
-

TOPIC C

Build Iterative Linear Regression Models

The closed-form approach to linear regression, whether regularized or not, is good at solving problems in certain types of datasets. However, you may encounter datasets on which the normal equation struggles. This is where iterative approaches come into play.

Iterative Models

The closed-form nature of the normal equation means it will directly provide you with the model parameters that best minimize the cost function. The expense of computing the matrix inverse as applied to large datasets necessitates a different approach. In an iterative approach, the model attempts to minimize the cost function by gradually approaching the minimum after repeated calculations. An iterative model is not always as effective as the normal equation at finding a "true" minimum, but it often provides a good enough result.

Feature scaling tends to be more important for iterative methods than the normal equation. This is because features of different scales can impair the model's ability to gradually approach the desired cost minimum. So, always consider scaling your features for training an iterative regression model.

Gradient Descent

In machine learning, the most common iterative method for minimizing the cost function in a linear model is the gradient descent algorithm. In *gradient descent*, the model parameters are tuned over several iterations by taking gradual "steps" down a slope, toward a minimum error value.

Think of a cost function (like MSE) as a valley. The top edges of the valley are the values of θ that do not minimize the cost function. The bottom of the valley is the value of θ that best minimizes the cost function—also called the minimum. Gradient descent selects a value for θ by starting at a random location. It then calculates the partial derivative of each model parameter with respect to the cost function, then updates each value of θ (i.e., it *steps* in a direction). This new θ value is one in which the model parameters give a cost that is lower than the previous trial. This process repeats, and as it does, it takes more and more steps down the valley until finally converging at the minimum (represented as $\hat{\theta}$).

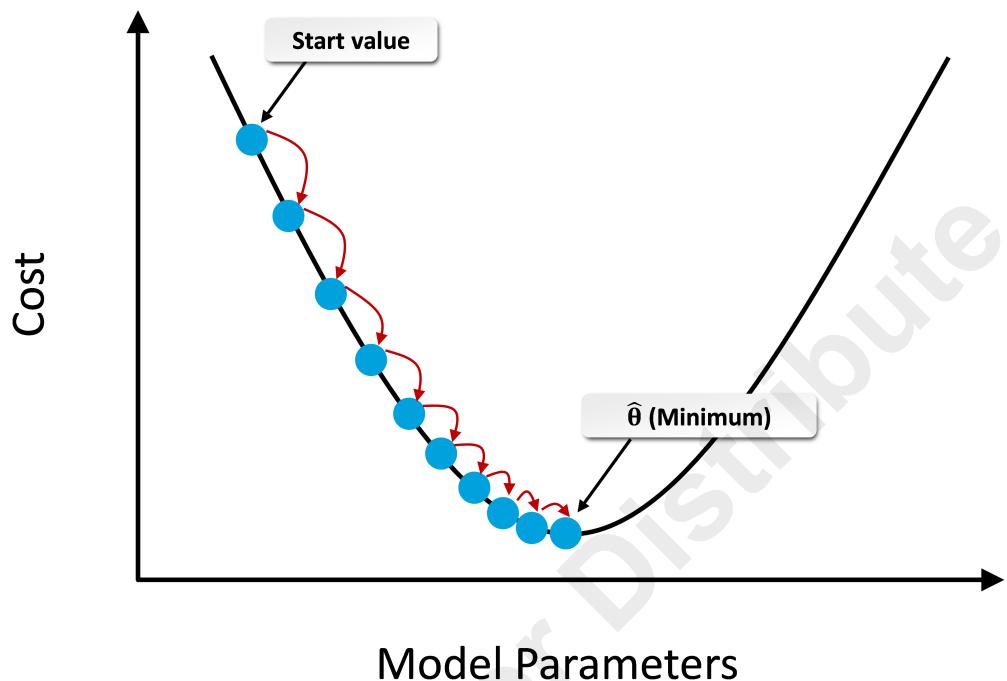


Figure 5–7: The gradient descent algorithm converges at the value that minimizes the cost function.

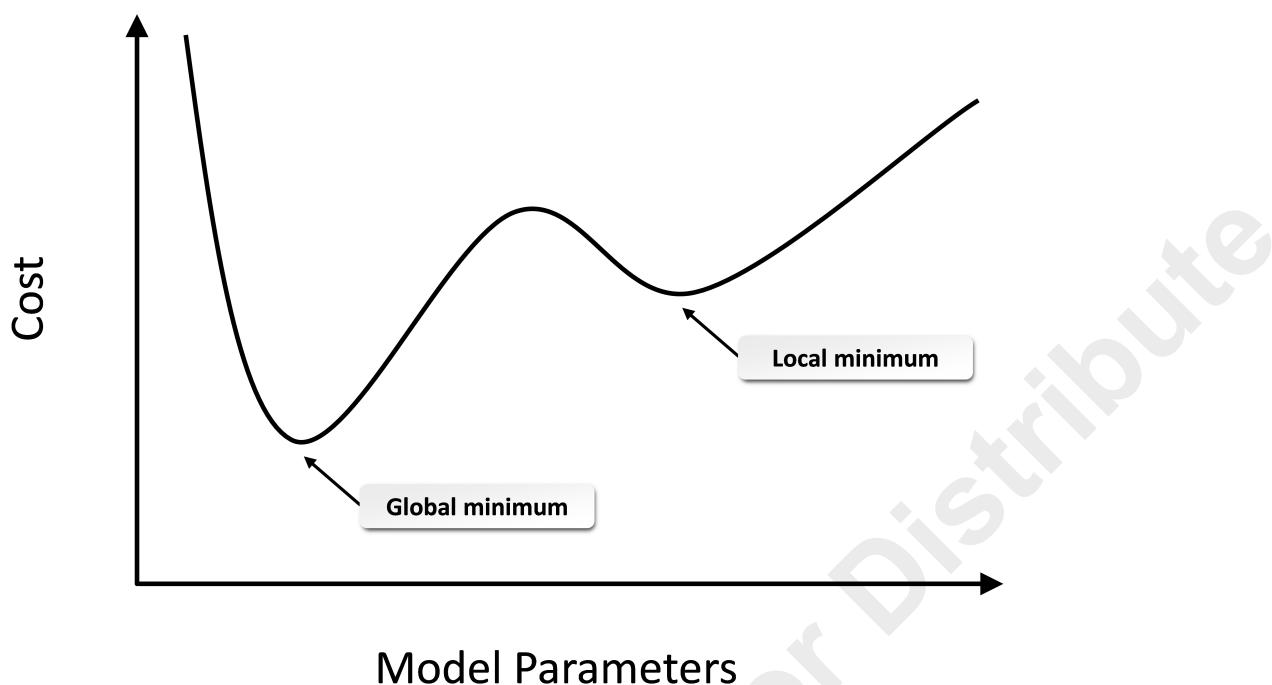
Gradient descent performs better than the normal equation in datasets that have a large number of features or examples, because calculating the inverse of a very large matrix of values takes a great deal of memory. Gradient descent avoids these memory issues and, depending on the type of gradient descent, may be able to work with virtually any size dataset. A good rule of thumb is that a dataset with 10,000 or more examples and/or features will usually benefit from gradient descent. In any given scenario, if it's feasible to train a model with the normal equation, then do so—otherwise, consider an iterative approach.



Note: Gradient descent can be applied to many types of problems, not just linear regression.

Global Minimum vs. Local Minima

MSE is a convex cost function—it looks like a simple valley, as shown in the previous figure. This means that it has only one minimum. However, some cost functions are non-convex and can actually alternate with peaks and valleys. In these cases (as in the following figure), there may be more than one minimum. A local minimum is the lowest point for any one valley (of which there may be several), whereas the global minimum is the lowest possible point overall. So, there may be several local minima, but only one global minimum. With these kind of cost functions, gradient descent is not guaranteed to converge at the global minimum. Whenever possible, use a convex cost function.



Learning Rate

In gradient descent, the size of each "step" down the valley is called the **learning rate**. The learning rate is a hyperparameter (α) of the model that you must tune to get the best results. If the learning rate is small, more "steps" will be required to converge at the minimum. Too small, and the descent could end up taking a very long time.

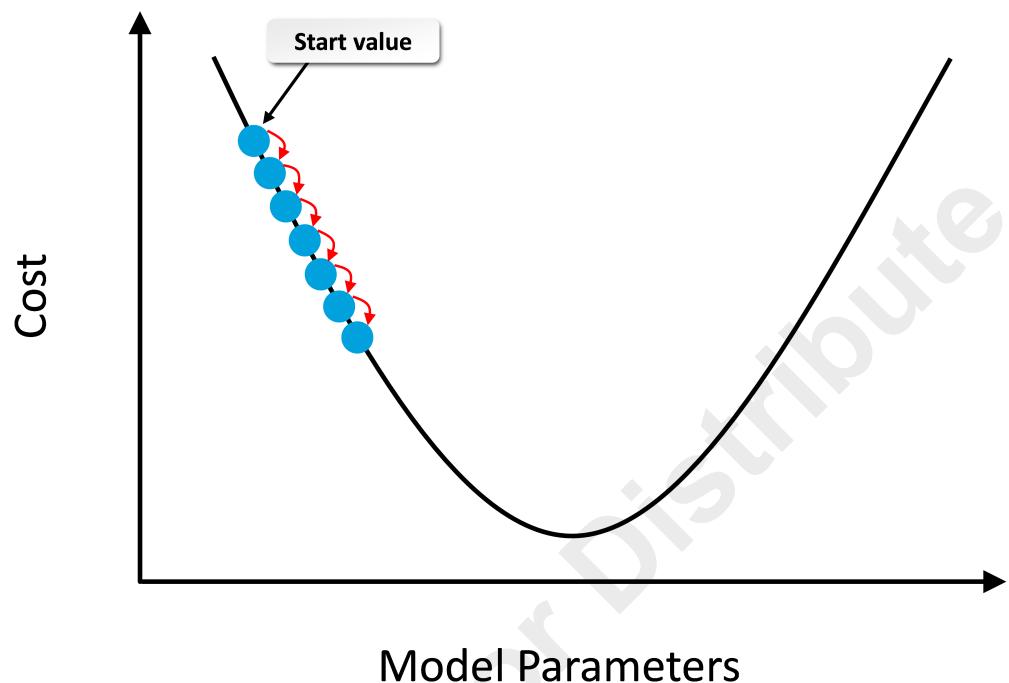


Figure 5–8: In this example, it will take a long time to reach the minimum because the learning rate is too small.

However, if the learning rate is too large, each "step" may jump to the opposite curve and actually end up higher than it was before. The model will therefore diverge instead of converging at a minimum.

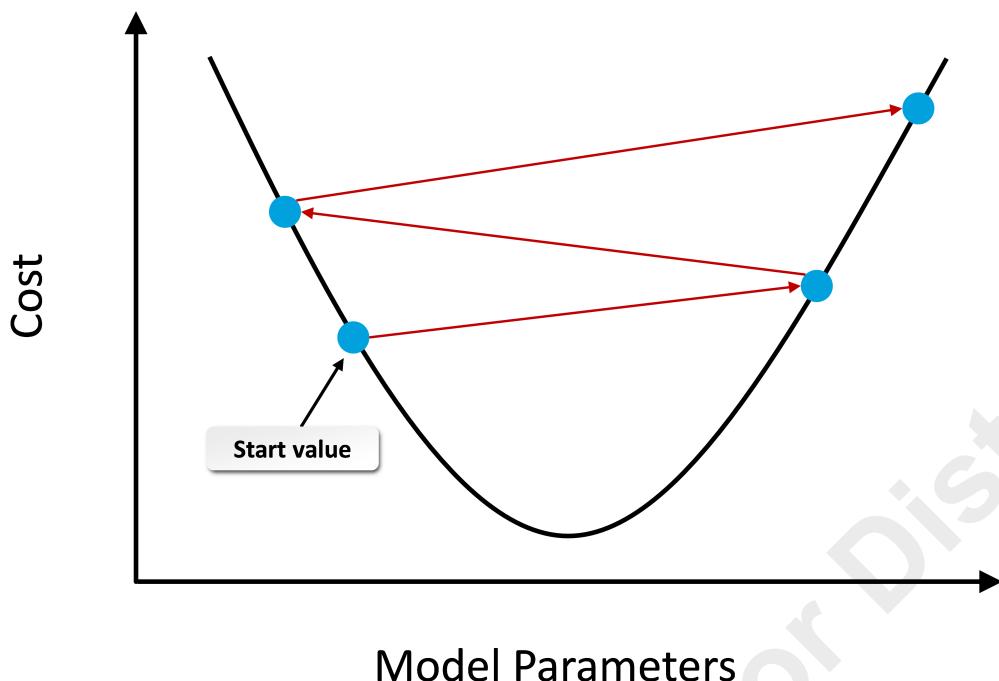


Figure 5–9: In this example, the model diverges because the learning rate is too large.

To avoid these issues, consider tuning the learning rate hyperparameter after several epochs. An epoch is one instance of the algorithm training over the entire dataset. If the error value decreases after an iteration, you can increase the learning rate slightly (e.g., around 5%). If the error value increases after an iteration, you can decrease the learning rate more significantly (e.g., around 50%). However, in most cases, you'll usually just start with a large learning rate and then gradually decrease it over time.



Note: This is one advantage the normal equation has over gradient descent—there's no need to tune a learning rate.

Gradient Descent Techniques

There are actually many different techniques for implementing gradient descent. The differences between these techniques are most relevant during the training process, as they tend to lead to models with similar levels of skill and predictive power after training. The following table gives a brief overview of these techniques.



Note: A deeper dive into these techniques is beyond the scope of this course.

Gradient Descent Technique	Description
Batch gradient descent (BGD)	This approach uses the entire training dataset to calculate the step-wise gradients. Each data example has its own unique gradient, and in BGD, the model parameters are updated based on the average gradient of all examples. BGD is not commonly used because it takes a very long time to compute the gradients for large datasets.

Gradient Descent Technique	Description
Stochastic gradient descent (SGD)	The stochastic approach selects a data example at random and computes its gradient at every step. This approach is therefore much quicker than the batch method. When one example updates the model parameters, it may "cancel out" the updates from a previous example. However, when this is done over a sufficient number of iterations, progress is slowly made in the right direction.
Stochastic average gradient (SAG)	This approach is very similar to SGD, except that SAG retains a "memory" of the previously computed gradients. This enables it to converge faster than SGD. However, it only supports ℓ_2 regularization and may present performance issues in datasets with a large number of examples.
Mini-batch gradient descent (MBGD)	This approach attempts to find a middle ground between batch and stochastic. Mini-batch gradient descent selects a group of examples at random and steps in the direction of the average gradient from all examples in the mini-batch. This can lead to better performance than SGD, especially when GPUs are involved in the computations.

Guidelines for Building an Iterative Linear Regression Model

Follow these guidelines when you are building linear regression models that iteratively minimize cost.

Build an Iterative Linear Regression Model

When building an iterative linear regression model:

- Consider using an iterative technique like gradient descent when your data set has 10,000 or more examples and/or features.
- Tune the learning rate hyperparameter after several epochs to avoid divergence or slow convergence issues.
- Consider starting with a high learning rate and gradually decrease it over time to minimize error.
- Prefer stochastic gradient descent (SGD) or stochastic average gradient (SAG) over batch gradient descent (BGD) in most cases to minimize the time it takes to minimize error.
- Consider using mini-batch gradient descent (MBGD) as a tradeoff between BGD and SGD.

Use Python for Iterative Linear Regression

Several scikit-learn classes that implement machine learning algorithms enable you to select how cost is minimized. This is usually specified through the class object's `solver` parameter. You can also perform iterative linear regression directly through the `SGDRegressor()` class, which gives you more options to customize the iterative approach (in this case, through stochastic gradient descent). The following are some of the objects and functions you can use to build such models.

- `model = sklearn.linear_model.Ridge(alpha = 0.1, solver = 'sag')` —This constructs a ridge regression object as before, but the `solver` parameter specifies that the algorithm should use SAG—an iterative approach—to minimize cost.
- `model = sklearn.linear_model.SGDRegressor(penalty = 'l2', alpha = 0.1, learning_rate = 'constant', eta0 = 0.05)` —This constructs a model object that uses SGD to iteratively minimize the cost function. The `penalty` and `alpha` parameters specify the type and strength of regularization to apply (if any), whereas `learning_rate` and `eta0` determine how to "step" through the descent and what the initial learning rate is, respectively.
- You can use these class objects to call the same methods and return the same attributes mentioned previously.

ACTIVITY 5–4

Building an Iterative Linear Regression Model

Data File

/home/student/CAIP/Linear Regression/LinearRegression-Boston-Iterative.ipynb

Before You Begin

Jupyter Notebook is open.

Scenario

As you continue expanding the scope of your real estate models, you realize that you'll quickly be training on very large datasets—datasets with hundreds of thousands of examples, and hundreds or perhaps thousands of features. The typical closed-form solution of the normal equation will start being a burden on training performance, slowing down the process to a potentially unacceptable degree. So, you'll try implementing an alternative cost minimization technique like gradient descent to hopefully cut down on training time. You'll begin by retraining your Boston housing model.

1. From Jupyter Notebook, select **CAIP/Linear Regression/LinearRegression-Boston-Iterative.ipynb** to open it.
2. Import the relevant libraries and load the dataset.
 - a) Select the code cell beneath the **Import software libraries and load the dataset** title, then select **Run**.
 - b) Verify that 506 records were loaded.
3. Split the datasets.
 - a) Scroll down and select the code cell beneath the **Split the datasets** title, and examine the code listing below it.
 - b) Select **Run**.
You'll be training the following models using a holdout set rather than cross-validation. This is for demonstration purposes, as the stochastic nature of cross-validation in scikit-learn makes it more difficult to compare the results of the two algorithms you'll use in this activity.
4. Drop columns that won't be used for training.
 - a) Scroll down and view the cell titled **Drop columns that won't be used for training**, and examine the code listing below it.
 - b) Select the cell that contains the code listing, then select **Run**.
5. Standardize the features.
 - a) Scroll down and select the code cell beneath the **Standardize the features** title, then select **Run**.
The data is being prepared in the same way as before. The *z*-score is applied to the features in order to scale their values.
6. **Why is it important to scale down features, such as through standardization, when using an iterative cost minimization technique like gradient descent?**

7. Train a model and calculate its scores.

- a) Scroll down and select the code cell beneath the **Train a model and calculate its cost** title, then select **Run**.

This is the same `model_train()` function as before; the only differences are that each model's training will be timed, and the variance score is not used.

8. Evaluate linear regression models using both closed-form and iterative solutions.

- a) Scroll down and view the code cell beneath the **Evaluate linear regression models using both closed-form and iterative solutions** title.

Evaluate linear regression models using both closed-form and iterative solutions

```
In [ ]: 1 from sklearn.linear_model import Ridge
2 from sklearn.linear_model import SGDRegressor
3
4 # Create closed-form and iterative ridge regression models.
5 def model_eval(eta):
6     for name, model in [
7         ('Ridge regression (closed form)', Ridge(alpha = 0.1, solver = 'cholesky')),
8         ('Ridge regression (gradient descent)', SGDRegressor(penalty = 'l2',
9             alpha = 0.1,
10            tol = 1e-3,
11            learning_rate = 'constant',
12            eta0 = eta,
13            random_state = 2))]:
14
15     print('Model: {}'.format(name))
16     print('-----')
17     model_train(model)
18     print('\n')
19
20 print('The function to evaluate the linear regression models has been defined.')
```

As before, this function calls `model_train()` using multiple algorithms:

- The ridge regression algorithm. This model uses the closed-form solution to minimize the cost function, with an arbitrary regularization strength (`alpha`) value.
- A similar linear algorithm, but one that uses stochastic gradient descent (SGD) to minimize the cost function.

For the `SGDRegressor()` algorithm, the following hyperparameters are configured:

- The `penalty` is the type of regularization to use. The objective is to try to compare the speed of each model using the same or very similar settings, so the ℓ_2 norm (ridge) is being used.
- The `alpha` value is the same so that the regularization strength of both models is comparable.
- The `tol` argument is the stopping criterion; when a value is supplied, the iterations will stop when the amount of loss is greater than the optimal loss minus the `tol` value. The default is $1e-3$, which is supplied here.
- The `learning_rate` is actually the scheduling technique used to configure the learning rate during descent. Setting this to `constant` means that the learning rate will stay at its initial value (`eta0`) and won't change. The learning rate determines the number of steps in gradient descent, and the more steps the descent takes, the more time it takes to train the model.
- The `eta0` argument is the initial learning rate. This is what you'll supply to the `model_eval()` function as a way to tune your model.

- b) Select the cell that contains the code listing, then select **Run**.

9. Which of the following describes stochastic gradient descent (SGD)?

- The model has a "memory" of previously computed gradients.
- A data example is selected at random and its gradient is computed for every step.
- A group of data examples is selected at random, and the model steps in the direction of the average gradient from all examples in the group.
- All data examples have their gradients computed for every step.

10. Continue evaluating the linear regression models using different initial learning rates.

- a) Scroll down and select the next code listing cell.

```
In [ ]: 1 model_eval(0.09)
```

- b) Select **Run**.
c) Examine the output.

```
Model: Ridge regression (closed form)
-----
Linear regression model took 5.35 milliseconds to fit.
Cost (mean squared error): 21.88

Model: Ridge regression (gradient descent)
-----
Linear regression model took 1.63 milliseconds to fit.
Cost (mean squared error): 65.36
```

- In this function call, you used an arbitrary starting point of 0.09 as the initial learning rate ($\eta_{t=0}$).
 - The mean squared error (MSE) for the SGD model appears significantly worse.
 - Because the dataset is so small, the time to fit will vary. On larger datasets, where calculating the normal equation consumes too much memory, the SGD model should take less time to fit.
- d) Scroll down and select the next code listing cell.

```
In [ ]: 1 model_eval(0.08)
```

- e) Select **Run**.
f) Examine the output.

```
Model: Ridge regression (closed form)
-----
Linear regression model took 2.56 milliseconds to fit.
Cost (mean squared error): 21.88

Model: Ridge regression (gradient descent)
-----
Linear regression model took 1.53 milliseconds to fit.
Cost (mean squared error): 47.05
```

- In this function call, you decreased the learning rate slightly.
- The MSE for the SGD model seems to have improved, but is still worse than the closed-form model.

- g) Scroll down and select the next code listing cell.

```
In [ ]: 1 model_eval(0.05)
```

- h) Select **Run**.

- i) Examine the output.

```
Model: Ridge regression (closed form)
-----
Linear regression model took 2.12 milliseconds to fit.
Cost (mean squared error): 21.88

Model: Ridge regression (gradient descent)
-----
Linear regression model took 1.09 milliseconds to fit.
Cost (mean squared error): 31.89
```

- The SGD model seems to be gradually improving as you progressively decrease the learning rate.

- j) Scroll down and select the next code listing cell.

```
In [ ]: 1 model_eval(0.01)
```

- k) Select **Run**.

- l) Examine the output.

```
Model: Ridge regression (closed form)
-----
Linear regression model took 2.21 milliseconds to fit.
Cost (mean squared error): 21.88

Model: Ridge regression (gradient descent)
-----
Linear regression model took 1.09 milliseconds to fit.
Cost (mean squared error): 25.25
```

- By lowering the learning rate significantly, the SGD model is getting close to the closed-form solution in terms of its ability to minimize error.

11. You could continue to tune the learning rate, but the SGD model will not lead to a lower error than the closed-form solution.

Why is this?

12. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.
 - In the **Shutdown kernel?** dialog box, select **Shutdown**.
 - Close the **LinearRegression-Boston-Iterative** tab in Firefox, but keep a tab open to the file hierarchy in Jupyter Notebook.
-

Summary

In this lesson, you built several linear regression models, starting with the closed-form normal equation approach. To minimize overfitting and improve the skill of your regression model, you applied regularization techniques. Lastly, you alleviated memory consumption concerns on large datasets by using an iterative approach to minimizing cost in a linear regression model. Ultimately, any of these linear regression approaches will be applicable to tasks in which you must predict numerical data from a given set of features.

What type of data in your organization do you think might be conducive to a linear regression analysis?

Do you believe your data is best approached by a closed-form solution or an iterative one? Why?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Do Not Duplicate Or Distribute

6

Building Classification Models

Lesson Time: 4 hours

Lesson Introduction

You've built regression models that can tackle linear data, but that is just one common application of machine learning. Another major application is the act of classifying data. A data example *is* something, and simultaneously *is not* something else. In this lesson, you'll use various techniques for building classification machine learning models.

Lesson Objectives

In this lesson, you will:

- Train binary classification models.
- Train multi-class classification models.
- Evaluate classification models.
- Tune classification models.

TOPIC A

Train Binary Classification Models

To begin with, you'll train some binary classification models using a few different algorithms. Each algorithm may be ideal for solving a certain type of classification problem, so you need to be aware of how they differ.

Linear Regression Shortcomings

Consider a machine learning model in which you're trying to predict heart disease in patients (1 for yes, 0 for no). One of the features in this dataset is a patient's cholesterol. If you graphed this function using standard linear regression, it might look something like this:

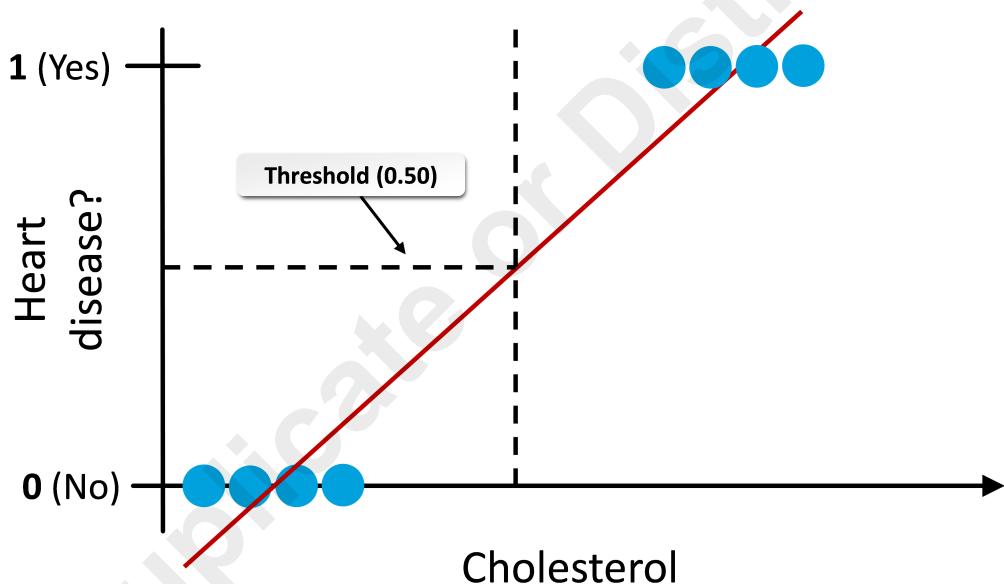


Figure 6-1: Fitting a straight line to a classification problem.

Based on the straight line fit, you can determine a reasonable threshold. Anything above this threshold is put in the positive class; anything below is put in the negative class. In this case, the threshold is around 0.50. So, any predictions above 0.50 (to the right of the threshold) are classed as 1 (has heart disease), and any predictions below 0.50 (to the left of the threshold) are classed as 0 (no heart disease). So, standard linear regression seems like it might be useful for this task. But, what if there's an outlier?

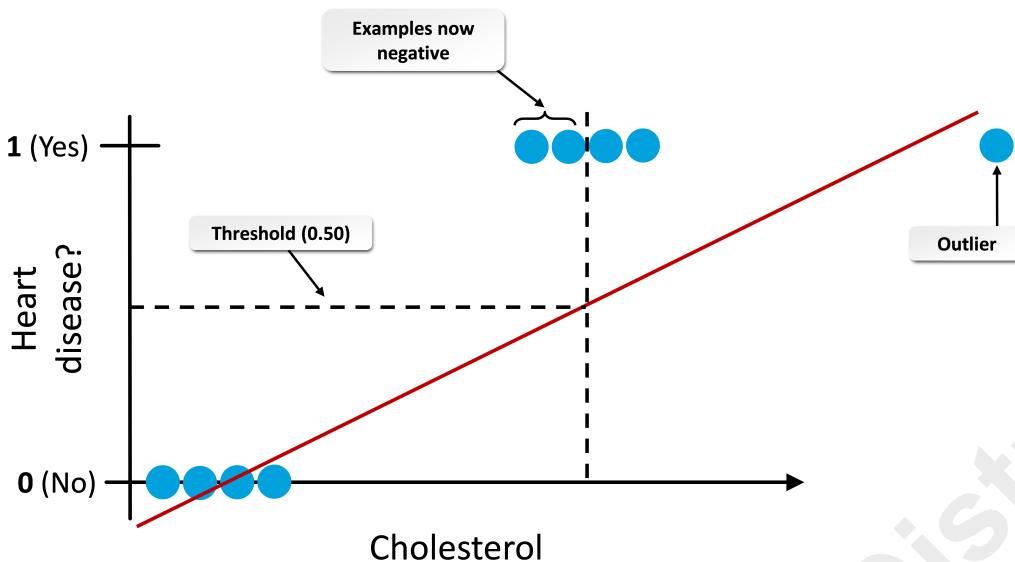


Figure 6-2: An outlier skews the line and changes the classification results.

As you can see, this skews the line of best fit. In order to maintain a 0.50 threshold, the threshold must move to a new position. Now, some examples that were classified as positive (has heart disease) are now being classified as negative (no heart disease).

Logistic Regression

Logistic regression is a type of linear regression in which the output is a prediction between 0 and 1. In the field of machine learning, this means that logistic regression is suitable for solving classification problems where standard linear regression is not. The value that a logistic regression algorithm outputs is called a **logistic function**. This is a type of *sigmoid function*. Instead of a straight line, the logistic function takes an *S* shape, as shown here:

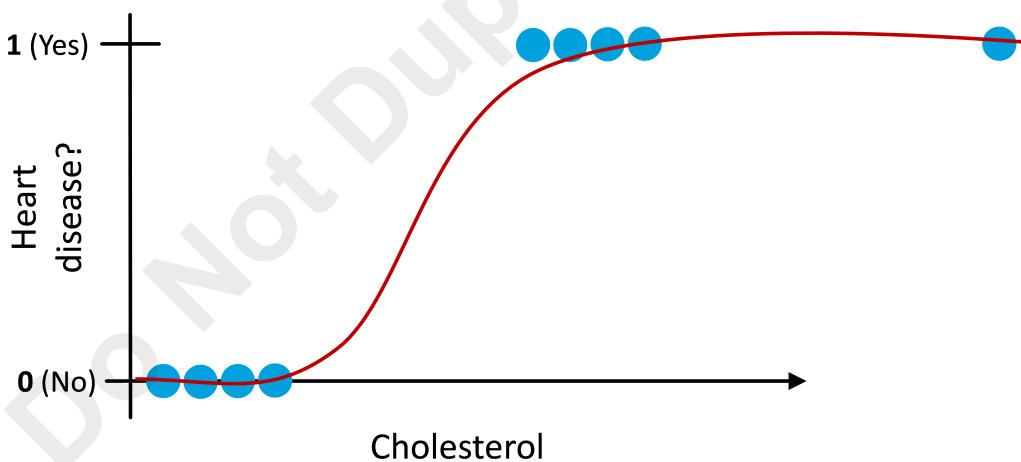


Figure 6-3: A sigmoid function used in logistic regression.

Notice that function is better able to fit the outlier, and that in doing so, it takes an *S* shape.

The logistic function can be expressed as the following equation:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Where:

- t is the prediction between negative infinity and positive infinity.
- e is the natural logarithm base.

The prediction value that is output by the logistic function is a value between 0 and 1. The t in this formula is essentially the linear model formula with multiple parameters that was described earlier. The logistic function's goal is to estimate the θ model parameters, much like in linear regression.

Decision Boundary

The division line that separates negative classes and positive classes is the **decision boundary**. This determines what class an example belongs to based on its prediction value between 0 and 1. A decision boundary can be expressed as follows, where \hat{p} is the predicted value and \hat{y} is the classification:

$$\begin{aligned} \text{if } \hat{p} \geq 0.50, \hat{y} &= 1 \\ \text{if } \hat{p} < 0.50, \hat{y} &= 0 \end{aligned}$$

So, if the model calculates a prediction value greater than or equal to 0.50, then it classifies that instance as a 1 (has heart disease). If the prediction value is less than 0.50, then the model classifies the instance as a 0 (no heart disease). When graphed on a sigmoid function, this would look something like the following:

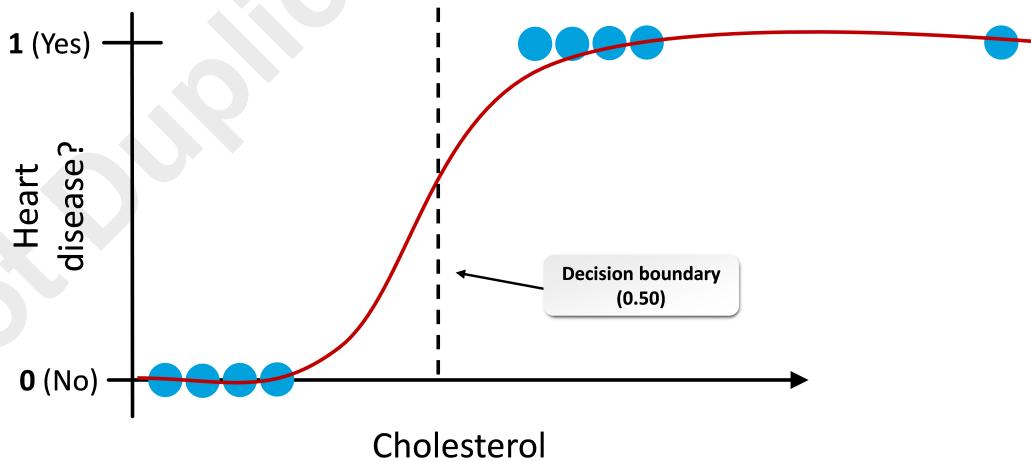


Figure 6–4: A sigmoid function applied to a classification problem. Note that the decision boundary in this case was selected arbitrarily.

As you can see, the decision boundary on a sigmoid function is able to account for the outlier, improving the algorithm's ability to classify the data examples.

You can configure a different boundary depending on the data and the context of the problem you're trying to solve. You can also analyze and evaluate logistic regression model performance using a technique like ROC curves.

Cost Function for Logistic Regression

Unlike with linear regression, you cannot effectively use the mean squared error (MSE) cost function with logistic regression. This would result in a non-convex function with multiple local minima, making it difficult to minimize the cost function. Instead, you can use the following cost function for logistic regression:

$$\begin{aligned} \text{if } y = 1, c(\theta) &= -\log(\hat{p}) \\ \text{if } y = 0, c(\theta) &= -\log(1 - \hat{p}) \end{aligned}$$

Where:

- y is the observed classification.
- $c(\theta)$ is the cost function.
- \hat{p} is the predicted probability.

As the algorithm's prediction (t) approaches 0, $-\log(t)$ increases significantly if the observed classification (y) is in the positive class (1). If y is in the negative class (0), $-\log(t)$ will increase significantly as t approaches 1. This is as expected, because the farther a prediction is away from the actual observed value, the worse it is at minimizing cost, so the cost function will be very high.

When the predicted value starts approaching the actual observation, that cost value shrinks. This cost function is applied to the whole training set as an average of the cost of all training examples, known as the *log loss*.

The normal equation cannot minimize the logistic regression cost function, but thankfully, gradient descent can. As long as your learning rate is properly tuned, gradient descent will be able to converge on the global minimum for the cost function.

A Simpler Alternative for Classification

Consider the following graph, in which the features of vehicle weight and vehicle size are plotted against each other. The purpose is to classify the vehicle as either a car or plane (red circles vs. blue triangles).

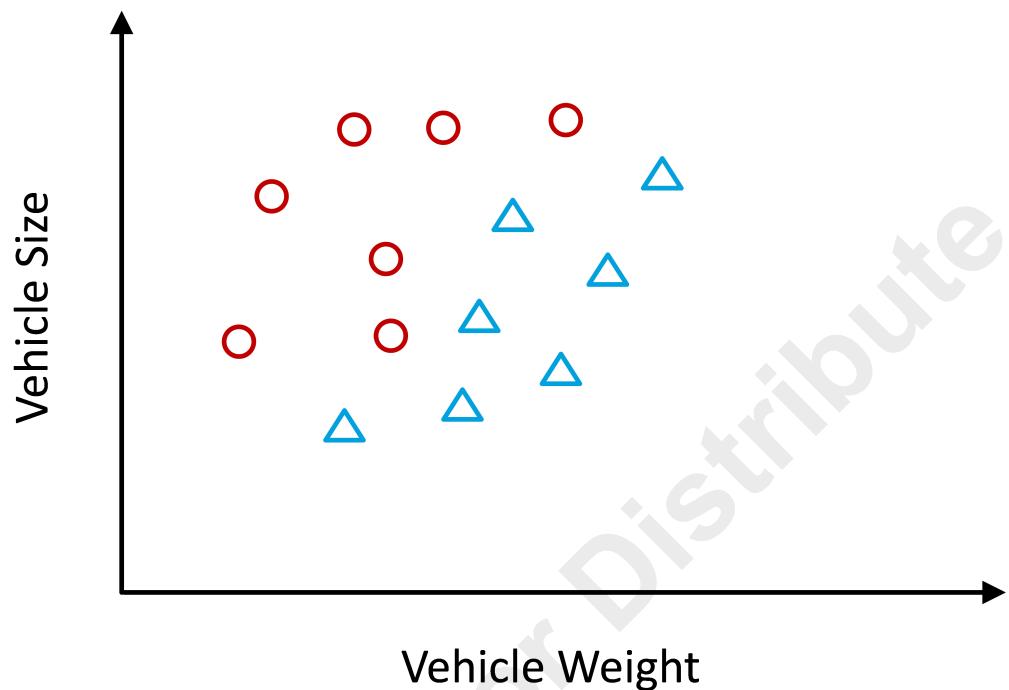


Figure 6–5: Data examples plotted for vehicle weight and vehicle size.

You could use a logistic regression model to predict these classifications. However, consider how you might determine the class for a new test example, depicted in the following figure as a green X.

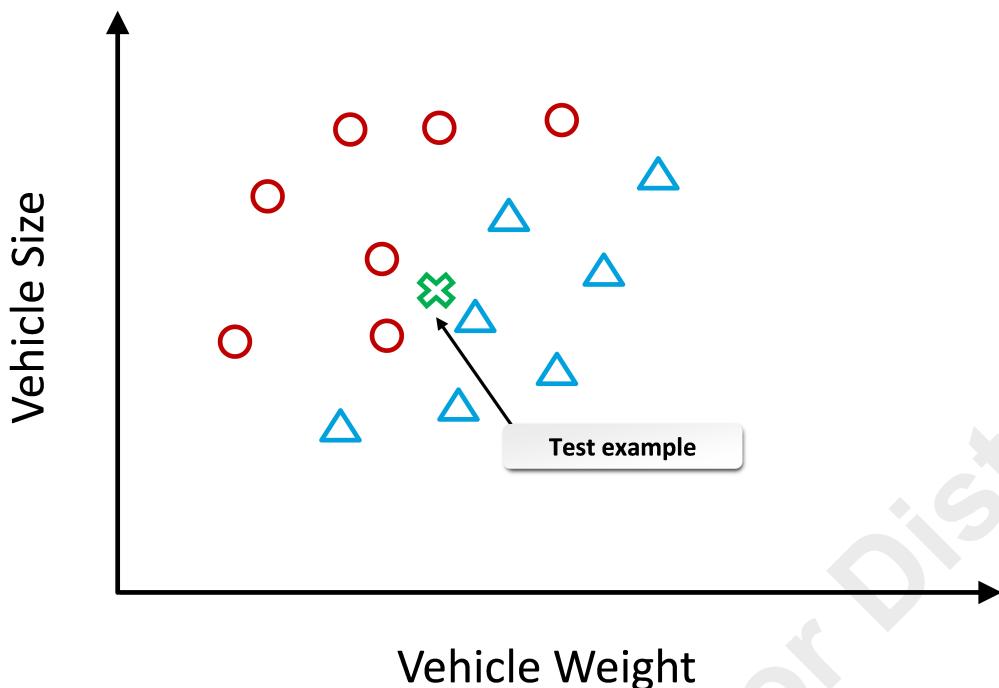


Figure 6-6: A new data example needs a classification.

Rather than use logistic regression, you can use an alternative classification algorithm called k -nearest neighbor to determine which class this new test point belongs to.

k -Nearest Neighbor (k -NN)

With the **k -nearest neighbor (k -NN)**, a data example is placed in a class based on its similarities to other data examples. These similarities are derived from the feature space (i.e., the combined vectors of training features). For example, you still want to classify a vehicle as either a car or a plane. If a vehicle has four wheels, is used on land, lacks wings, is smaller in size, and weighs less, it is more similar to vehicles labeled as cars than those labeled as planes. Therefore, k -NN will classify the example as a car.

The k in k -NN defines the number of the data examples that are the closest neighbors of the example in question. "Closest" in this case refers to the distance between data points when they are mapped to the feature space. Using k , k -NN takes a plurality vote of these neighboring points to determine how to classify the example in question. In the following figure, $k = 3$, so the algorithm takes a vote of the three nearest neighbors. Since two of the three neighbors are in class 0 (red circles), the data example is classified as a 0.

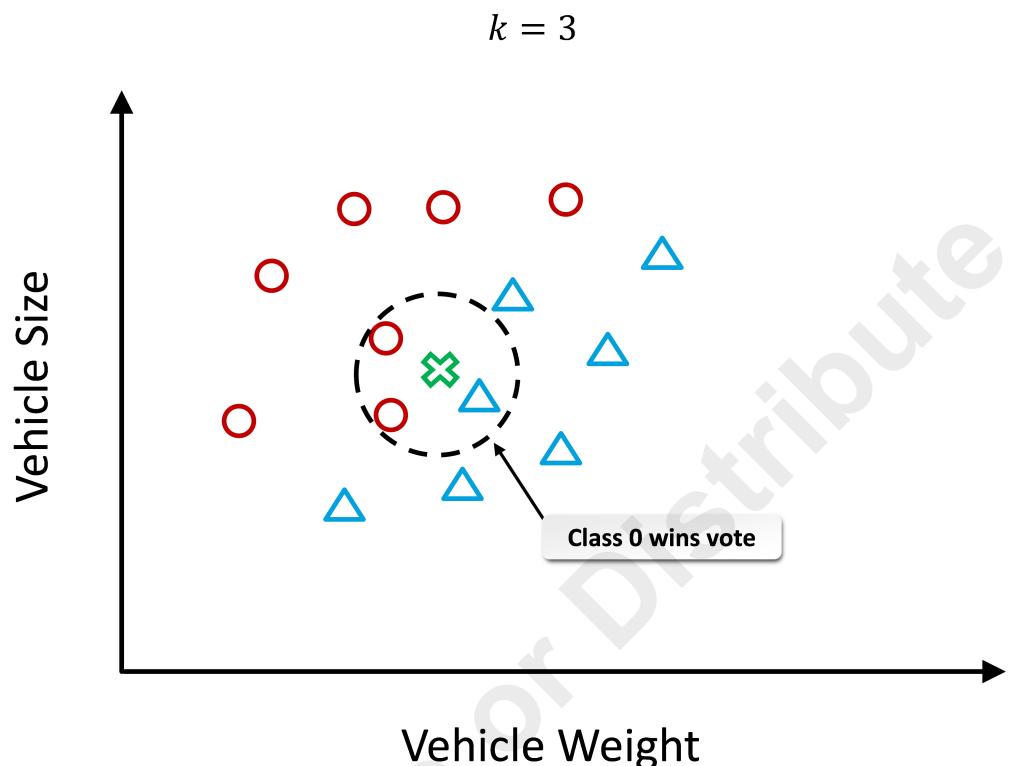


Figure 6–7: *k*-NN classification where $k = 3$.



Note: *k*-NN can also solve regression problems, but it is most often used in classification.

k Determination

Like many hyperparameters in machine learning, the choice of k in k -NN is ultimately dependent on the nature of the training data and the output you're looking for. There is not necessarily one perfect value for k for all known circumstances. The larger you make k , the less "noisy" the dataset becomes with respect to classification; in other words, the effect of anomalies or mislabeled examples is reduced. However, this also leads to less distinct boundaries between classes, and can increase computational overhead. Lower k values are better at keeping these boundaries distinct, but are less effective at minimizing the effect of noise on the data.

Consider the following example. In the figure, the k value from before (3) has been tuned to a new value (5) on the same training set. The model now classifies the example as a 1 instead of a 0 because there are more 1 votes (blue triangles) than 0 votes between the five nearest neighbors. A change of k like this can completely change the classification a data example receives.

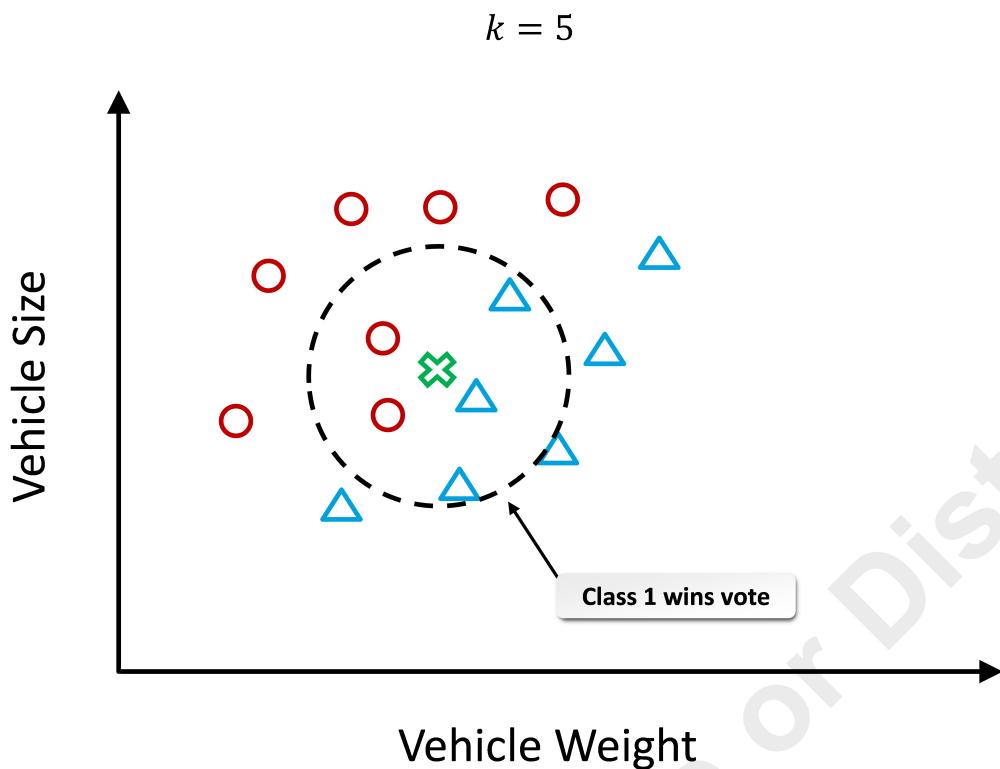


Figure 6-8: Changing k in a k -NN model can lead to different results.

One rule-of-thumb when selecting k is to always make k odd when you have only two classifiers; this eliminates the chance of there being a tie in the vote. Another approach to selecting k is called bootstrapping, and it involves taking the square root of the total number of data examples in the training set. The resulting value is approximately the k you should consider using. You can also use a performance testing technique like cross-validation to help you determine an acceptable value for k .

Logistic Regression vs. k -NN

As you've seen, both logistic regression and k -NN can be used to solve classification problems. You should weigh their general advantages and disadvantages when deciding which is most appropriate for your needs. Also, if time permits and you have enough computational power, you can run a dataset through both algorithms to evaluate their comparative performance.

One key difference is that k -NN does not really train a model in the same way as a typical machine learning algorithm like logistic regression. The model doesn't really improve its classification abilities through learning; it merely calculates the distance between data examples mapped to a feature space and determines the class. Likewise, the output from k -NN is the classification itself and not a probability. In one sense, this makes k -NN simpler to implement, as no time needs to be spent on actually training the model.

However, k -NN can take a very long time to actually make a prediction when there are many data examples and features. Think of a dataset with tens of thousands of examples and hundreds of features. Every single example will need to be mapped to a large feature space, and then, the k -NN algorithm must compute the distance between the example you're trying to predict and *every* data example in the set in order to determine the distance values for k . Then, it needs to sort all of these distances. Logistic regression, because it has been trained to determine optimal model parameters, is much faster in making predictions on new data.

So, for simplicity of implementation and when working with smaller datasets, k -NN may be the ideal approach. Logistic regression is ideal when datasets are large and predictions must be generated relatively quickly.

Guidelines for Training Binary Classification Models

Follow these guidelines when you are training binary classification models using logistic regression and k -NN.



Note: All of the Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Train a Binary Classification Model

When training a binary classification model:

- Consider using logistic regression rather than linear regression for classification tasks.
- In k -NN, consider that lower k values make class boundaries more distinct while being less effective at minimizing noise, and vice versa.
- Consider making k odd for binary classification to avoid a tie vote.
- Consider using a bootstrapping method of selecting k by taking the square root of the number of examples in the dataset.
- Consider using cross-validation to help determine a good k value.
- Consider using k -NN over logistic regression when simplicity of implementation is key, and when working with smaller datasets.
- Consider using logistic regression over k -NN when predictions must be quick, and when working with larger datasets.
- Time permitting, run a dataset through both algorithms to determine which performs the best.

Use Python for Binary Classification

The scikit-learn `LogisticRegression()` class enables you to construct a machine learning model using a logistic regression algorithm. The following are some of the objects and functions you can use to build such a model.

- `model = sklearn.linear_model.LogisticRegression(penalty = 'l2', C = 0.05, solver = 'sag')` —This constructs a model object that uses the logistic regression algorithm. In this case, the model is using regularization and an iterative approach to minimizing cost.
- `model.fit(X_train, y_train)` —Fit a set of training data to the model.
- `model.score(X_test, y_test)` —Return the accuracy score for the model given validation/test data.
- `model.predict(X_test)` —The model makes classification predictions on the validation/test set.
- `model.predict_proba(X_test)` —The model provides the raw probability estimates for each classification decision.
- `model.coef_` —An attribute that lists the optimal model parameters generated during training.

The scikit-learn `KNeighborsClassifier()` class enables you to construct a machine learning model using a k -NN algorithm. The following are some of the objects and functions you can use to build such a model.

- `model = sklearn.neighbors.KNeighborsClassifier(n_neighbors = 3)` —This constructs a model object that uses the k -NN algorithm. In this case, k is 3.
- You can use this class object to call the same `fit()`, `score()`, `predict()`, and `predict_proba()` methods as with `LogisticRegression()`. However, `predict_proba()` will return the fraction of neighbors that voted for each label, rather than a true prediction probability.

ACTIVITY 6–1

Training Binary Classification Models

Data Files

/home/student/CAIP/Classification/Classification-Titanic.ipynb
 /home/student/CAIP/Classification/titanic_data/test.csv
 /home/student/CAIP/Classification/titanic_data/train.csv

Before You Begin

Jupyter Notebook is open.

Scenario

You work for the History department at a state college. An upcoming lecture focuses on one of the most well-known disasters in history—the sinking of the RMS *Titanic*. The department wants to teach students about the multiple factors that may have influenced who survived and who didn't. Instead of just lecturing students on the subject matter and having them accept it passively, you want them to be able to reach their own conclusions and verify those conclusions through hands-on experience. So, you decide to build a machine learning model that will help students do just that.

You'll use a real-world dataset that includes various statistics about the *Titanic* passengers. The label in this case is whether or not the passenger survived. So, you'll train a couple classification models that will try to predict who survived and who didn't. Ultimately, you want students to be able to feed the model their own data—for example, everyone in the class could be a "passenger" with their own characteristics (age, sex, number of siblings, etc.)—so that you can show them whether or not *they* would survive the disaster. This will hopefully make the lecture a little more "real" to the students, while also being a fun academic exercise.

While you plan to make interacting with the machine learning model more user friendly at some point, you first need to build the code base.

1. From Jupyter Notebook, select CAIP/Classification/Classification-Titanic.ipynb to open it.

2. Import the relevant libraries and load the dataset.

- Select the code cell beneath the **Import software libraries and load the dataset** title, then select **Run**.
- Verify that **test.csv** and **train.csv** are in the project folder, and that the latter was loaded with 891 records.

There are separate testing and training datasets provided. You'll split the training dataset to create a validation set as well. But first you'll get familiar with the data.

3. Get acquainted with the data.

- Select the code cell beneath the **Get acquainted with the dataset** title, then select **Run**.
- Examine the output.
 - The training set includes 891 rows and 12 columns.
 - 5 columns contain integer values, 5 contain strings (objects), and 2 contain floats.
 - Most columns have a value in every row, except for `Age` (714 records, or 177 missing), `Cabin` (204 records, or 687 missing), and `Embarked` (889 records, or 2 missing).
 - Several columns are self-explanatory, but for those that aren't:

- Survived with a value of 0 means the passenger did not survive; 1 means they did. This is the label of interest.
- Pclass is the class of ticket. In other words, a value of 1 means the passenger was traveling first class, the most expensive and highest quality service. This also acts as a proxy for the passenger's socioeconomic status.
- SibSp refers to the number of siblings plus a spouse that the passenger had onboard.
- Parch refers to the number of parents and children that the passenger had onboard.
- Ticket is the ticket's identification number.
- Fare is the cost of the passenger's ticket, which likely correlates with Pclass.
- Cabin refers to where on the ship the passenger lodged.
- Embarked is a letter that refers to the port where the passenger embarked from: "C" for Cherbourg, "Q" for Queenstown, and "S" for Southampton.
- Several columns (Sex, Ticket, Cabin, Embarked) contain non-numeric values. These will have to be converted to numeric values somehow if they are to be used by the learning algorithm.
- Some data values shown as NaN (not a number) are missing, and will need to be handled somehow.
- The features have different ranges, so they will need to be scaled so that they exert the same influence over the model.
- It might be helpful to investigate the meaning (if any) of the letter/number combinations in Ticket and Cabin. However, since so many values are missing for Cabin, it might be necessary to just drop this feature.

4. Examine a general summary of statistics.

- Select the code cell beneath the **Examine a general summary of statistics** title, and then select **Run**.
- Examine the output.

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.00	891.00	891.00	714.00	891.00	891.00	891.00
mean	446.00	0.38	2.31	29.70	0.52	0.38	32.20
std	257.35	0.49	0.84	14.53	1.10	0.81	49.69
min	1.00	0.00	1.00	0.42	0.00	0.00	0.00
25%	223.50	0.00	2.00	20.12	0.00	0.00	7.91
50%	446.00	0.00	3.00	28.00	0.00	0.00	14.45
75%	668.50	1.00	3.00	38.00	1.00	0.00	31.00
max	891.00	1.00	3.00	80.00	8.00	6.00	512.33

- The mean for Survived is 0.38. Since features in this column contain a 1 (survived) or 0 (did not survive), this means that 38% of passengers in the training set survived.
- The minimum Age shows that the youngest passenger in the set was less than a year old (0.42). The maximum Age was 80 years old.
- The mean age was 29.7 years old.

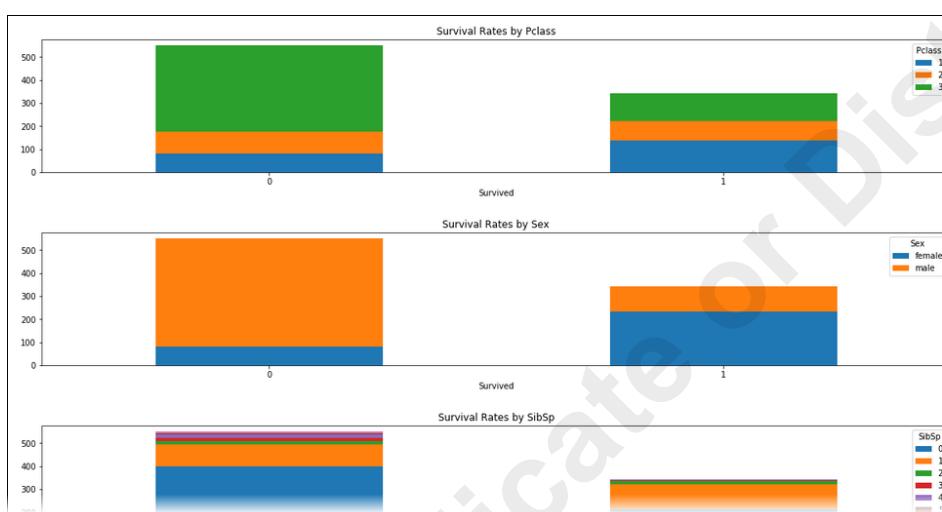
5. Given what you know about the dataset thus far, what features do you think might influence the survival rates?

6. Unlike with linear regression, attempting to find a correlation between a feature and a classification label is not useful. In other words, there's no need to run code to compare the *Titanic* features to the *Survived* label.

Why is such a correlation not relevant in classification problems like this one?

7. Use stacked bar visualization to show survival numbers.

- Scroll down and select the code cell beneath the **Use stacked bar visualization to show survival numbers** title, then select **Run**.
- Examine the output.

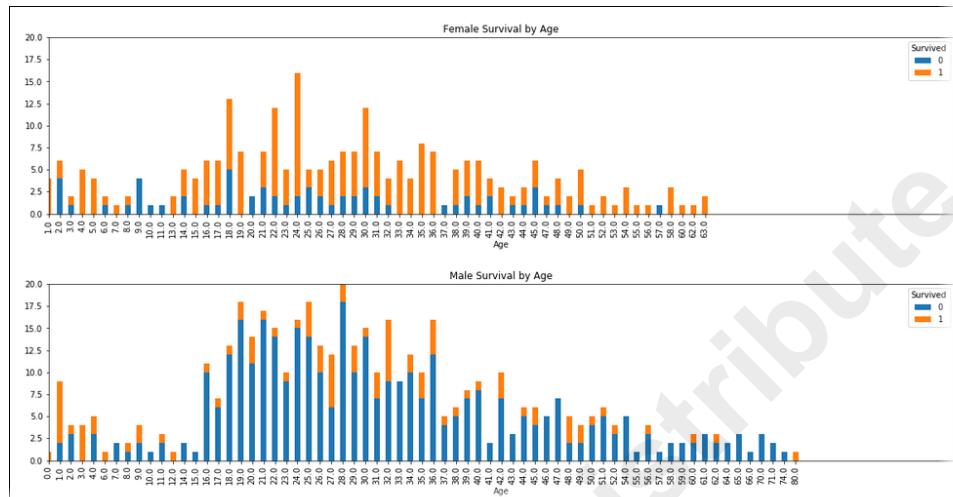


- In general, passengers were more likely to perish than survive.
- The majority of people who perished were in third class. However, keep in mind that these graphs are measuring absolute values and not proportions—there may have been more people in third class than in any other class.
- The majority of people who perished were male.
- The majority of people who perished had no family with them.
- People who embarked from Cherbourg were more likely to survive than perish.
- People who embarked from Southampton were more likely to perish than survive.

8. Look for relationships between survival, age, and sex.

- Scroll down and select the code cell beneath the **Look for relationships between survival, age, and sex** title, then select **Run**.

- b) Examine the output.



As expected, female passengers had higher survival rates than male passengers. Likewise, fewer of the elderly survived as compared to younger passengers, though this is more obvious for male than female passengers.

9. Identify columns with missing values.

- a) Scroll down and select the code cell beneath the **Identify columns with missing values** title, then select **Run**.
 b) Examine the output.

```
Number of missing values:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin          687
Embarked        2
dtype: int64
```

`Age` and `Cabin` have many missing values that will need to be resolved somehow.

10. Split the datasets.

- a) Scroll down and select the code cell beneath the **Split the datasets** title, then select **Run**.
 b) Examine the output.

```
Original set:      (891, 12)
-----
Training features: (668, 11)
Validation features: (223, 11)
Training labels:   (668, 1)
Validation labels: (223, 1)
```

The original training dataset has now been split into two: one set to continue using as training, the other to use as validation. Note that the `Survived` label has been removed from the `X` matrices and placed into its own `y` vector.

11. Identify columns that should be modified or deleted from the training set.

- Scroll down and select the code cell beneath the **Identify columns that should be modified or deleted from the training set** title, then select **Run**.
- Examine the output.

PassengerId	Pclass		Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
439	440	2	Kvillner, Mr. Johan Henrik Johannesson	male	31.0	0	0	C.A.18723	10.5000	NaN	S
617	618	3	Lobb, Mrs. William Arthur (Cordelia K Stanlick)	female	26.0	1	0	A/5.3336	16.1000	NaN	S
242	243	2	Coleridge, Mr. Reginald Charles	male	29.0	0	0	W/C.14263	10.5000	NaN	S
82	83	3	McDermott, Miss. Bridget Della	female	NaN	0	0	330932	7.7875	NaN	Q
398	399	2	Pain, Dr. Alfred	male	23.0	0	0	244278	10.5000	NaN	S

- PassengerId and Name are not relevant for the model and should be dropped from the training set. However, they may be needed for reporting results, so they should stay in the validation and test sets.
- You'll convert the Sex and Embarked columns to numeric codes.
- Many of the Name values include an honorific (personal title) such as "Mr.", "Mrs.", "Miss.", etc. You'll extract these and convert them to numeric codes.

12. Determine how to handle ticket values.

- Scroll down and select the code cell beneath the **Determine how to handle ticket values** title, then select **Run**.
 - Examine the output.
- It seems as though it would be difficult to convert ticket codes into something useful, so they will be dropped from the training set.

13. Identify all personal titles and embarked port codes.

- Scroll down and select the code cell beneath the **Identify all personal titles and embarked port codes** title, then select **Run**.
- Examine the output.

```
Titles: ['Mr' 'Mrs' 'Miss' 'Master' 'Don' 'Rev' 'Dr' 'Mme' 'Ms' 'Major' 'Lady'
'Sir' 'Mlle' 'Col' 'Capt' 'the Countess' 'Jonkheer']
Embarked locations: ['S' 'C' 'Q' nan]
```

You'll convert these to numeric values.

14. Perform common preparation on the training and validation sets.

- Scroll down and view the cell titled **Perform common preparation on the training and validation sets**, and examine the code listing below it.
Common cleaning and feature engineering tasks will be performed on the data.
- Select the cell that contains the code listing, then select **Run**.
- Examine the output.
 - Missing values for Age and Fare are being filled in with the median of those values. For Embarked, you're using the mode to fill in missing values.
 - Since SibSp and Parch seem to both be related, you're combining them into a single column, SizeOfFamily. Anyone traveling alone has a family size of 1.
 - If a passenger has a personal title, it is being removed from Name and placed into its own Title column.
 - Each unique Title, Sex, and Embarked value is being encoded with a number value, starting at 1. This will make them easier to work with.
 - Missing TitleEncoding values are being filled in with the mode.

15. Preview current training data.

- Scroll down and select the code cell beneath the **Preview current training data** title, and examine the code listing below it.
- Select the cell that contains the code listing, then select **Run**.
- Examine the output.
 - As mentioned previously, `PassengerId` and `Name` don't contain information that will be useful for training the model.
 - `Cabin` contains coding that might be useful, but numerous records are missing this data, so you'll drop it.
 - `Ticket` likewise might contain useful encoding, but you don't have enough information to convert it to a number.
 - Some columns containing text values (`Title`, `Sex`, `Embarked`) have been replaced with a numeric code, and can be dropped.

16. Drop columns that won't be used for training.

- Scroll down and view the cell titled **Drop columns that won't be used for training**, and examine the code listing below it.
Unused columns will be dropped from the dataset.
- Select the cell that contains the code listing, then select **Run**.
- Examine the output and observe the new set of columns in the `X` matrix (for both training and validation).

```
---- TRAINING ----
Columns before drop:
['PassengerId', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'SizeOfFamily',
'Title', 'SexEncoding', 'EmbarkedEncoding', 'TitleEncoding']

Columns after drop:
['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'SizeOfFamily', 'SexEncoding', 'EmbarkedEncoding', 'TitleEncoding']

--- VALIDATION ---
Columns before drop:
['PassengerId', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'SizeOfFamily',
'Title', 'SexEncoding', 'EmbarkedEncoding', 'TitleEncoding']

Columns after drop:
['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'SizeOfFamily', 'SexEncoding', 'EmbarkedEncoding', 'TitleEncoding']
```

- Scroll down and select the next code listing cell.

In []: 1 X_train.head()

- Select **Run**.
- Examine the output and observe the new columns.

	Pclass	Age	SibSp	Parch	Fare	SizeOfFamily	SexEncoding	EmbarkedEncoding	TitleEncoding
439	2	31.0	0	0	10.5000	1	2	1	1
617	3	26.0	1	0	16.1000	2	1	1	2
242	2	29.0	0	0	10.5000	1	2	1	1
82	3	27.5	0	0	7.7875	1	1	3	3
398	2	23.0	0	0	10.5000	1	2	1	4

17. Create a logistic regression model.

- Scroll down and select the code cell beneath the **Create a logistic regression model** title, then select **Run**.
- Examine the output.

```
Logistic regression model took 573.28 milliseconds to fit.
Score on validation set: 78%
```

- `LogisticRegression()` is a scikit-learn class that, as the name implies, generates a logistic regression model object. One of the important arguments that it takes is `solver`, which determines the method used to minimize the cost function. In this case, you're using `sag`, which refers to stochastic average gradient (SAG). Recall that this is similar to stochastic gradient descent (SGD), but has a "memory" of previous gradients for faster convergence. By using this `solver`, the algorithm will automatically perform ℓ_2 regularization.
 - The `C` argument is the hyperparameter that controls the strength of regularization applied to the model. For now, this is set to a relatively low value, which implements strong regularization.
 - The `max_iter` argument specifies the maximum number of iterations that the `solver` will perform in an effort to converge at a minimum. In this case, the default number of iterations (100) is not enough to converge, so the maximum is set higher.
 - The score for this model is around 78%.
- Scroll down and select the next code listing cell.

```
In [ ]:
1 # Use validation set to evaluate.
2 results_comparison = X_val.copy()
3 results_comparison['PredictedSurvival'] = log_reg.predict(X_val)
4 results_comparison['ActualSurvival'] = y_val.copy()
5 results_comparison['ProbPerished'] = np.round(log_reg.predict_proba(X_val)[:, 0] * 100, 2)
6 results_comparison['ProbSurvived'] = np.round(log_reg.predict_proba(X_val)[:, 1] * 100, 2)
7
8 # View examples of the predictions compared to actual survival.
9 results_comparison.head(20)
```

- Select **Run**.
- Examine the output.

Pclass	Age	SibSp	Parch	Fare	SizeOfFamily	SexEncoding	EmbarkedEncoding	TitleEncoding	PredictedSurvival	ActualSurvival	ProbPerished
400	3 39.00	0 0	7.9250	1	2			1 1	0 1	83.00	
450	2 36.00	1 2	27.7500	4	2			1 1	0 0	82.71	
846	3 30.25	8 2	69.5500	11	2			1 1	0 0	97.52	
516	2 34.00	0 0	10.5000	1	1			1 2	0 1	51.99	
42	3 30.25	0 0	7.8958	1	2			2 1	0 0	78.61	
247	2 24.00	0 2	14.5000	3	1			1 2	0 1	53.42	
22	3 15.00	0 0	8.0292	1	1			3 3	1 1	35.46	
867	1 31.00	0 0	50.4958	1	2			1 1	0 0	63.11	
410	3 30.25	0 0	7.8958	1	2			1 1	0 0	82.60	
252	1 62.00	0 0	26.5500	1	2			1 1	0 0	71.82	

In addition to the existing columns, the classification that was predicted by the model on the validation set is compared to the actual label value. The probabilities for each classification are also displayed in the right-most columns. These probabilities are the results that the logistic regression function generates and uses to determine what class to predict. For example, if `ProbPerished` is higher than `ProbSurvived`, then the model will have predicted a 0 for that particular passenger (i.e., did not survive).

18. Create a *k*-nearest neighbor model.

- Scroll down and select the code cell beneath the **Create a *k*-nearest neighbor model** title, then select **Run**.

- b) Examine the output.

```
Value of k: 27
KNN model took 9.14 milliseconds to fit.
Score on validation set: 76%
```

- This is an alternative to the logistic regression model built in the previous step. Both of these models perform binary classification, but it's useful to build multiple models to see if one performs better than another.
- The score for the k -nearest neighbors model is around 76%. This appears to perform worse than the logistic regression model, so you'll stick with the logistic regression model for the actual test set.
- In this case, k is generated by a bootstrapping method—taking the square root of the total number of data examples in the training set. To help avoid tie votes in a binary classifier, k is incremented by 1 if it is even.

- c) Scroll down and select the next code listing cell.

```
In [ ]: 1 # Use validation set to evaluate.
2 results_comparison = X_val.copy()
3 results_comparison['PredictedSurvival'] = knn.predict(X_val)
4 results_comparison['ActualSurvival'] = y_val.copy()
5 results_comparison['ProbPerished'] = np.round(knn.predict_proba(X_val)[:, 0] * 100, 2)
6 results_comparison['ProbSurvived'] = np.round(knn.predict_proba(X_val)[:, 1] * 100, 2)
7
8 # View examples of the predictions compared to actual survival.
9 results_comparison.head(20)
```

- d) Select Run.

- e) Examine the output.

The information here is in the same format as the logistic regression output from before, but the predictions are different because of the different model used.



Note: Because k -NN does not generate prediction probabilities in the same way that logistic regression does, the `predict_proba()` method is actually returning the fraction of neighbors that voted for each label. So, in the first row of results, ~93% of the neighbors voted for perished (25 of the total 27).

19. Use the logistic regression model to make predictions on the test data.

- a) Scroll down and select the code cell beneath the **Use the logistic regression model to make predictions on the test data** title, then select Run.

- b) Examine the output.

418 records are loaded from `test.csv`. Recall that this is the test set, separate from the training and validation sets. This test set does not include label values. It's this kind of data that the machine learning model must make predictions for in a production environment. In the context of your History class, this is the set that would include students' information as if they were passengers on the *Titanic*.

- c) Scroll down and select the next code cell.

```
In [ ]: 1 # Prepare the dataset and drop unneeded columns.
2 print('Preparing test data for prediction\n')
3 X_test = prep_dataset(X_test_raw.copy())
4 X_test = drop_unused(X_test.copy())
```

- d) Select Run.

- e) Examine the output.

As with the training and validation sets, you're performing the same data preparation steps to get the data in an optimal state.

- f) Scroll down and select the next code listing cell.

```
In [ ]: 1 # Show example predictions with the original test data.
2 results_log_reg = X_test_raw.copy()
3 results_log_reg['PredictedSurvival'] = log_reg.predict(X_test)
4 results_log_reg['ProbPerished'] = np.round(log_reg.predict_proba(X_test)[:, 0] * 100, 2)
5 results_log_reg['ProbSurvived'] = np.round(log_reg.predict_proba(X_test)[:, 1] * 100, 2)
6 results_log_reg.head(20)
```

- g) Select Run.
h) Examine the output.

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	PredictedSurvival	ProbPerished	ProbSurvived
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q	0	74.29	25.71
1	893	3	Wilkes, Mrs. (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	0	68.02	31.98
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	0	70.70	29.30
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	0	82.30	17.70
4	896	3	Hirvonen, Mrs. Alexander E (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	0	66.31	33.69
5	897	3	Svensson, Mr. Johan Cervin	male	14.0	0	0	7538	9.2250	NaN	S	0	81.58	18.42
6	898	3	Connolly, Miss. Kate	female	30.0	0	0	330972	7.6292	NaN	Q	1	36.70	63.30
7	899	2	Caldwell, Mr. Albert Francis	male	26.0	1	1	248738	29.0000	NaN	S	0	80.96	19.04
			Abrahim, Mrs. Joseph											

The logistic regression model has predicted values for this test set, fulfilling its ultimate purpose.

20. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **Classification-Titanic** tab in Firefox, but keep a tab open to **CAIP/Classification/** in the file hierarchy.

ACTIVITY 6–2

(Optional) Creating a Logistic Regression Model to Predict Breast Cancer Recurrence

Data Files

/home/student/CAIP/Optional/LogisticRegression.ipynb

/home/student/CAIP/Optional/breast_cancer_data/breast-cancer.csv

Before You Begin

Jupyter Notebook is open.

Scenario

Breast cancer is a leading cause of death, affecting millions of women across the globe each year. After a first incidence of breast cancer and subsequent remission of a year or more, cancer may reappear in the same location or elsewhere in the body, a situation known as recurrence. Early detection of a recurrence improves a woman's chance of survival, but follow-up care can be time-consuming and expensive. Based on the circumstances, some women are at greater risk for recurrence than others, so it would be very helpful to know which women are at greater risk, and should therefore have much more frequent follow-up care.

Given the extensive data available regarding breast cancer patients, machine learning can provide a valuable tool for helping to classify which patients are at greatest risk for recurrence. As you continue your work with the Greene City Physicians Group (GCPG), you will create a simple logistic regression model to demonstrate how this might be done.

A dataset has been provided to you. The dataset has already been cleaned and encoded, so it shouldn't require any additional preparation to use it for a simple logistic regression model. The dataset has the following attributes:

Attribute	Description
recurrence	Whether the patient had a recurrence event. <ul style="list-style-type: none"> 0 - Patient had no recurrence. 1 - Patient had a recurrence.
age_decade	Age of the patient at the time of diagnosis. Ages have been divided into bins for each decade, so a 33-year-old patient would have a value of 30.
meno_pre meno_lt_40 meno_ge_40	These three attributes identify the patient's age at menopause. <ul style="list-style-type: none"> meno_pre = 1 - The patient has not yet reached menopause. meno_lt_40 = 1 - The patient was less than 40 when reaching menopause. meno_ge40 = 1 - The patient was at least 40 when reaching menopause.
tumor_size	The largest diameter (in millimeters) of the excised tumor.
inv_nodes	The number (ranging from 0 to 39) of axillary lymph nodes that contain metastatic breast cancer in a histological examination.

Attribute	Description
node_caps	The outermost layer of a lymph node is a thin capsule made of connective tissue. Cancer may spread outside the lymph node but remain inside of the capsule. Or it may have penetrated the capsule and further spread into the surrounding tissue. <ul style="list-style-type: none"> • node_caps = 0 - The cancer remains contained within the capsule. • node_caps = 1 - The cancer has spread outside the capsule.
deg_malig	The histological grade of the tumor. Tumors that are grade 1 primarily contain cells that retain many of their normal characteristics. Grade 3 tumors contain cells that are largely abnormal, in which disease has spread considerably. <ul style="list-style-type: none"> • 1 - Still largely normal. • 2 - Somewhat abnormal. • 3 - Largely abnormal.
breast_left	These two attributes identify the presence of cancer in each breast.
breast_right	<ul style="list-style-type: none"> • breast_left = 1 - The patient had cancer in the left breast. • breast_right = 1 - The patient had cancer in the right breast.
irradiat	Whether the cancer has been irradiated. Cancer cells may be exposed to high-energy x-rays to destroy them, in the hope that the normal cells will regenerate, but cancerous cells will not. <ul style="list-style-type: none"> • 0 - The cancer has not been irradiated. • 1 - The cancer has been irradiated.



Note: Sometimes, when you run code containing logic errors or bugs, you may corrupt the data contained in variables you created in previous code cells. To clear out such problems, you can select **Kernel→Restart & Clear Output**, then run each code cell again.

1. Examine data in the file.

- Navigate to the **/home/student/CAIP/Optional/breast_cancer_data/breast-cancer.csv** file directly in Jupyter Notebook. Examine the column headings and refer to the various attributes described in the table above.
- Close the **breast-cancer.csv** tab to return to the list of files in Jupyter Notebook.

2. Open the logistic regression notebook.

- From Jupyter Notebook, select **CAIP/Optional/LogisticRegression.ipynb** to open it.



Note: Be careful *not* to open the solution file.

- Observe what has been provided for you in the notebook.

Placeholder code cells have been provided in which you can add your own code. Comments provide hints on tasks you might perform in each code cell. The first code cell has already been completely programmed for you.

3. Import software libraries and load the dataset.

- Select the code listing under **Import software libraries and load the dataset**.
- Run the code and examine the results.

The dataset is loaded for you. There are 286 records.

4. Get acquainted with the data structure and preview the records.
 - a) In the code block under **Get acquainted with the data structure and preview the records**, write statements to show the various features and their data types and to view the first five records.
 - b) Run the code cell and verify that a summary of the various features and their data types is shown, along with a sample of data in the first five records.
 5. Examine descriptive statistics.
 - a) In the code block under **Examine descriptive statistics**, write code to show descriptive statistics (count, mean, std, etc.) for the dataset.
 - b) Run the code cell and verify that descriptive statistics are shown for features in the dataset.
 6. Use histograms to visualize the distribution of various features.
 - a) In the code block under **Use histograms to visualize the distribution of various features**, write code to show a histogram for each attribute in the dataset.
 - b) Run the code cell and verify that the histograms are shown for each feature.
 7. Split the data into training and validation sets and labels
 - a) In the code block under **Split the data into training and validation sets and labels**, write code to perform the following tasks:
 - Import a function to split the dataset
 - Specify the column to be included in the label set (`recurrence`)
 - Specify columns to be included in the training and validation sets (all other columns)
 - Split the training set, validation set, and labels for both
 - Compare the number of rows and columns in the original data to the training and validation sets
 - b) Run the code cell and verify that the data has been split into training, validation, and label sets.
 - c) In the following two code cells, write and run code to preview a sample of the training data and labels.
 8. Build the model
 - a) In the code block under **Build the model**, write code to create a logistic regression model, and use the validation data and labels to score it.
 - b) Run the code cell and observe the score.
 9. Test the model
 - a) In the code block under **Test the model**, write code to perform the following tasks:
 - Add columns to a copy of the test data to compare predictions against actual values.
 - View examples of the predictions compared to actual recurrence.
 - b) Run the code cell.
 - c) Scroll if necessary, and compare the predicted values against the actual values.
 10. In Jupyter Notebook, open **CAIP/Optional/LogisticRegression-Solution.ipynb** and compare it to the code you wrote.
- 

Note: Since there are many ways to write code to do the same basic thing, don't expect your code to match exactly. The important thing is that your code accomplishes the same basic goals, and returns similar results.
11. Shut down the Jupyter Notebook kernels.
 - a) From the menu, select **Kernel→Shutdown**.
 - b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
 - c) Close the **LogisticRegression** tab in Firefox.
 - d) Repeat this process to shut down and close the **LogisticRegression-Solution** kernel.

-
- e) Return to **CAIP/Classification/** in the file hierarchy.
-

Do Not Duplicate or Distribute

TOPIC B

Train Multi-Class Classification Models

Not all classification problems are binary in nature—some require more than just a 0 or a 1. In this topic, you'll train a model to handle cases in which there are multiple ways to classify a data example.

Multi-Label Classification

The logistic regression and k -NN examples given previously work with a simple binary choice—something either *is* x , or is *not* x . Either a patient has heart disease, or does not. But, you may be faced with a classification problem that involves multiple labels, also called ***multi-label classification***. For example, say you need to analyze written text. Specifically, you need to classify nouns in a sentence. The machine learning model must classify the following:

- Category 1: Whether the noun is acting as a **subject** or **object**.
- Category 2: Whether the noun is **proper** or **common**.

All nouns in a clause are either subjects or objects, and one cannot be both. Likewise, all nouns are either proper (capitalized) or common (not capitalized). While the classes in each category are mutually exclusive, the same is not true *between* categories. You can have a proper subject, a common subject, a proper object, or a common object. That's four possible classification labels.

Multi-label problems can be solved by training multiple binary classification models, unless the labels themselves are correlated. You could train two models (one for category 1, the other for category 2) and then combine the results.

Multi-Class Classification

Multi-class classification is the process of placing a data example within a single class among three or more choices. This differs from multi-label classification in that the classes are mutually exclusive. An example cannot belong to class A and B and C; it must belong to A *or* B *or* C.

Consider the following example: You're back to analyzing written text, but this time, you're focusing on individual words. You're interested in classifying each word based on the way it is used in a sentence. It can be used as a:

- Noun
- Verb
- Adjective
- And so on...

When used in a sentence, a word either acts as a noun or some other non-noun type; it cannot be both. Therefore, this is a multi-class problem. Unlike with multi-label problems, multi-class problems require a different approach than just combining binary classifiers.

Multinomial Logistic Regression

Multinomial logistic regression, sometimes called softmax regression, is a method for solving multi-class problems. Multinomial logistic regression starts by computing a score represented by $(\mathbf{x})_k$:

$$(\mathbf{x})_k = (\boldsymbol{\theta}^{(k)})^T \cdot \mathbf{x}$$

Where:

- k is the class.
- θ is the vector of the model parameters.
- x is the vector of the feature values.



Note: This is very similar to the equation used to predict values in linear regression.

The scores are computed for every class k for vector x . Then, the scores are plugged into the softmax function. The softmax function calculates the exponent of each score divided by the sum of all exponents:

$$\sigma(x)_k = \frac{\exp(x)_k}{\sum_{j=1}^K \exp(x)_j}$$

Where:

- $\sigma(x)_k$ is the probability that example x belongs to class k given the score computed earlier.
- K is the total number of classes.

The end result is that, for each data example, the softmax function determines the class with the highest probability, where all probabilities add up to 1.

So, in the sentence, "I saw that movie yesterday," the softmax function generated the following probabilities for the word type of the word "saw":

- 84% verb.
- 13% noun.
- 3% adjective.

Thus, the algorithm predicts that the word "saw" in this sentence is a verb.

Cross-Entropy

Cross-entropy is a cost function used to evaluate the performance of a softmax function used in training. It is $-\log(\text{softmax})$, where the softmax function is evaluated at the ground truth (the actual label) of a data example. This result is then calculated over all examples, and the average of these results is the loss.

Cross-entropy essentially penalizes low probability scores for a particular class. The lower the probability, the higher the cost. When there are only two classes (binary classification), the cross-entropy function is essentially the same as the log loss function mentioned earlier. As with linear and binary logistic regression, you can minimize this cost function using a technique like gradient descent. The calculation for gradient descent using cross-entropy is more complex, but it fulfills the same purpose: to train the algorithm to minimize errors in prediction.

Guidelines for Training Multi-Class Classification Models

Follow these guidelines when you are training multi-class classification models using logistic regression.

Train a Multi-Class Classification Model

When training a multi-class classification model:

- Consider how a classification problem may require a data example being placed into two or more classes (multi-label).
- For multi-label problems, consider training multiple binary classification models, unless the labels are correlated.

- Consider how a classification problem may require a data example being placed into one of three or more possible classes (multi-class).
- For multi-class problems, consider training a multinomial logistic regression model.

Use Python for Multi-Class Classification

The scikit-learn `LogisticRegression()` class can also be used to construct a machine learning model using a multinomial logistic regression algorithm. The following are some of the objects and functions you can use to build such a model.

- `model = sklearn.linear_model.LogisticRegression(multi_class = 'multinomial')`
—This constructs a model object that uses the logistic regression algorithm. In this case, the model is enabling multi-class classification through the use of a multinomial logistic algorithm.
- You can use this class object to call the same methods and return the same attributes mentioned previously.

ACTIVITY 6–3

Training a Multi-Class Classification Model

Data File

/home/student/CAIP/Classification/Classification-Wine.ipynb

Before You Begin

Jupyter Notebook is open.

Scenario

Your hands-on *Titanic* lab was a success, and now other professors at the college are looking to apply machine learning to their chosen fields. The Chemistry department wants to help its students learn about the interactions between certain chemical compounds. Namely, the students should observe how slight variations in a substance's chemical makeup can change its overall form.

A professor from the Chemistry department has provided you with a dataset that lists various wines, each one with several characteristics. Each wine example is classified as a wine from a specific cultivar. However, unlike with the *Titanic* dataset, the wines can be grouped into one of *three* classes, not just two. So, a binary classification model isn't going to be enough. You'll need to build a multi-class classification model in order to predict the type of wine a particular sample is.

1. From Jupyter Notebook, select **CAIP/Classification/Classification-Wine.ipynb** to open it.
2. Import the relevant libraries and load the dataset.
 - a) Scroll down and select the code cell beneath the **Import software libraries and load the dataset** title, then select **Run**.
 - b) Verify that 178 records were loaded.
This dataset actually comes pre-packaged with scikit-learn, and can be easily loaded into an array through the `load_wine()` function. There's no need to load a file stored on the local machine.
3. Get acquainted with the dataset.
 - a) Scroll down and select the code cell beneath the **Get acquainted with the dataset** title, then select **Run**.
 - b) Examine the output.
 - The training set includes 178 rows and 14 columns.
 - All of the columns contain float values, except for the `target` column, which contains integer values. The `target` column is the label of wine the model must predict, classified as either 0, 1, or 2. With three possible classifications, this dataset presents a multi-class problem.
 - There is no missing data; all rows have values for every column.
 - Most of the columns describe a particular wine's chemical composition, like its alcohol level, magnesium level, number of phenols, etc. Some columns describe visual aspects of the wine, like its color intensity and hue.
 - Ultimately, this is a relatively clean and uniform dataset. However, it may still benefit from a bit of preparation.
4. Examine a general summary of statistics.
 - a) Scroll down and select the code cell beneath the **Examine a general summary of statistics** title, then select **Run**.

- b) Examine the output.

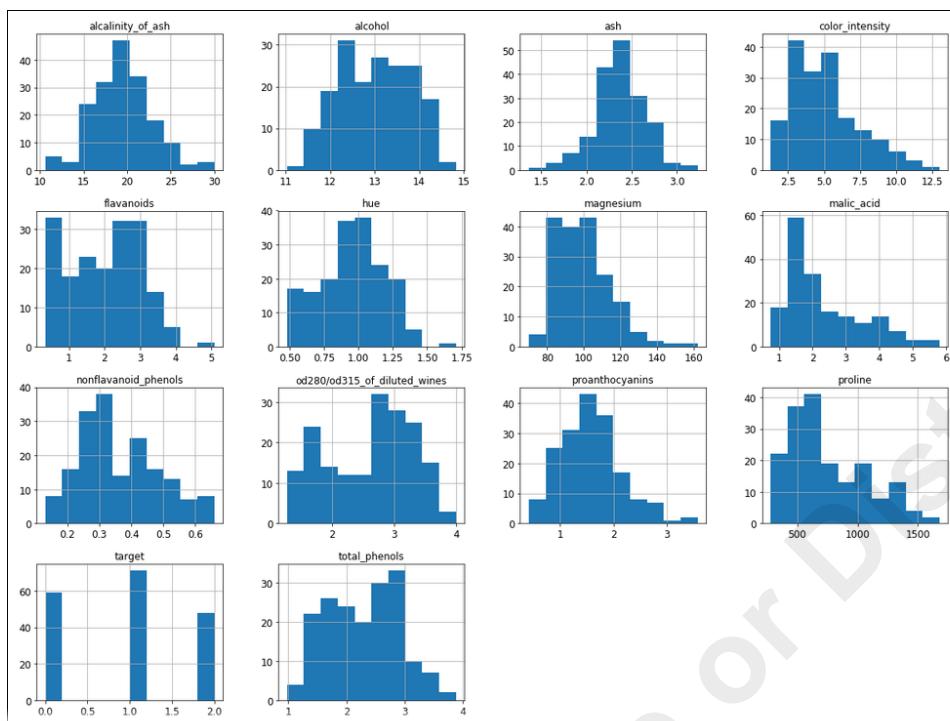
	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	\
count	178.00	178.00	178.00	178.00	178.00	
mean	13.00	2.34	2.37	19.49	99.74	
std	0.81	1.12	0.27	3.34	14.28	
min	11.03	0.74	1.36	10.60	70.00	
25%	12.36	1.60	2.21	17.20	88.00	
50%	13.05	1.87	2.36	19.50	98.00	
75%	13.68	3.08	2.56	21.50	107.00	
max	14.83	5.80	3.23	30.00	162.00	
	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins		\
count	178.00	178.00	178.00	178.00		
mean	2.30	2.03	0.36	1.59		
std	0.63	1.00	0.12	0.57		
min	0.98	0.34	0.13	0.41		
25%	1.74	1.20	0.27	1.25		
50%	2.35	2.13	0.34	1.56		
75%	2.80	2.88	0.44	1.95		
max	3.88	5.08	0.66	3.58		
	color_intensity	hue	od280/od315_of_diluted_wines	proline	target	
count	178.00	178.00	178.00	178.00	178.00	
mean	5.06	0.96	2.61	746.89	0.94	
std	2.32	0.23	0.71	314.91	0.78	
min	1.28	0.48	1.27	278.00	0.00	
25%	3.22	0.78	1.94	500.50	0.00	
50%	4.69	0.96	2.78	673.50	1.00	
75%	6.20	1.12	3.17	985.00	2.00	
max	13.00	1.71	4.00	1680.00	2.00	

- Unless you're a wine expert or knowledgeable in the chemistry of alcohol, you may not have much intuition as to which features are the most important in determining the class of wine.
- There don't appear to be many extreme outliers for any of the features, but you'll soon look at a visual distribution to be sure.
- One thing you should be able to identify is that the values in magnesium and especially proline are much higher than the others. Some scaling might be in order.

5. Examine the distribution of various features.

- a) Scroll down and select the code cell beneath the **Examine the distribution of various features** title, then select **Run**.

- b) Examine the output.



For the most part, these histograms seem to confirm the idea that there are not many extreme outliers in the dataset. You can also see that the class labels (`target`) are distributed pretty evenly.

6. Split the label from the dataset.

- a) Scroll down and select the code cell beneath the **Split the label from the dataset** title, then select **Run**.
 b) Examine the output.

Rather than using the holdout method to split the datasets into a training set and a validation/test set, you'll be training the data using cross-validation. For now, you're just extracting the label from the training set (`X`) and placing it in its own vector (`y`).

7. Transform magnesium and proline.

- a) Scroll down and select the code cell beneath the **Transform magnesium and proline** title, then select **Run**.
 You're transforming the two features you identified earlier as being out of scale with the rest of the data.

- b) Examine the output.

Transform magnesium and proline

```

1 # Apply a log transformation to scale 'magnesium' and 'proline'.
2 X = X.copy()
3 X['proline'] = np.log(X['proline'])
4 X['magnesium'] = np.log(X['magnesium'])
5
6 # Examine results of the transformation
7 with pd.option_context('float_format', '{:.2f}'.format):
8     print(X['magnesium'].describe())
9     print('-----')
10    print(X['proline'].describe())
11
12 X.head()

count    178.00
mean      4.59
std       0.14
min       4.25
25%      4.48
50%      4.58
75%      4.67
max      5.09
Name: magnesium, dtype: float64

-----
count    178.00
mean      6.53
std       0.42
min       5.63
25%      6.22
50%      6.51
75%      6.89
max      7.43
Name: proline, dtype: float64

   alcohol  malic_acid  ash  alcalinity_of_ash  magnesium  total_phenols  flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity  hue  od280/od
0    13.69        3.26  2.54             20.0    4.672829      1.83      0.56          0.50          0.80      5.88      0.96
1    12.67        0.98  2.24             18.0    4.595120      2.20      1.94          0.30          1.46      2.62      1.23
2    13.86        1.35  2.27             16.0    4.584967      2.98      3.15          0.22          1.85      7.22      1.01
3    13.73        1.50  2.70             22.5    4.615121      3.00      3.25          0.29          2.38      5.70      1.19
4    13.41        3.84  2.12             18.8    4.499810      2.45      2.68          0.27          1.48      4.28      0.91

```

- A simple logarithm has helped reduced the numeric range of these values. The maximum value for `magnesium` was 162.00; now it's 5.09. The maximum value for `proline` was 1680.00; now it's 7.43.
- This feature scaling is not entirely useful for standard regression models, but it *is* useful in speeding up gradient descent and for improving regularized regression models, both of which you'll implement.

8. Create a multinomial logistic regression model.

- a) Scroll down and select the code cell beneath the **Create a multinomial logistic regression model** title, then select **Run**.

Create a multinomial logistic regression model

```

1 from sklearn.linear_model import LogisticRegression
2
3 log_reg = LogisticRegression(solver='sag', multi_class='multinomial', max_iter=10000)
4
5 print('Multinomial logistic regression model created.')

```

- In scikit-learn, the same `LogisticRegression()` class is used for both binary classification and multi-class classification. The `multi_class` argument activates the latter. There are actually two different approaches; `multinomial` ensures that the softmax function is used to predict probabilities.
- The `solver` is using stochastic average gradient (SAG).

9. Train the model using stratified k -fold cross-validation to split the dataset.

- Scroll down and select the code cell beneath the **Train the model using stratified k -fold cross-validation to split the dataset** title, then select **Run**.
- Examine the output.

Train the model using stratified k -fold cross-validation to split the dataset

```

1 from sklearn.model_selection import cross_val_score
2 from sklearn.model_selection import cross_val_predict
3
4 # Train model and make predictions using test data.
5 start = time()
6 predict = cross_val_predict(log_reg, X, np.ravel(y), cv = 5)
7 end = time()
8 train_time = (end - start) * 1000
9
10 # Retrieve mean score of test folds.
11 score = cross_val_score(log_reg, X, np.ravel(y), cv = 5).mean()
12
13 print('Multinomial logistic regression model took {:.2f} milliseconds to fit.'.format(train_time))
14 print('Mean score on test sets: {:.0f}%'.format(np.round(score * 100)))

```

Multinomial logistic regression model took 349.64 milliseconds to fit.
Mean score on test sets: 95%

If you hadn't transformed those two out-of-scale features, the training process would have taken around 5 times longer.

- Scroll down and select the next code listing cell.

```

In [ ]: 1 # Retrieve prediction probabilities.
2 proba = cross_val_predict(log_reg, X, np.ravel(y), cv = 5, method = 'predict_proba')
3
4 # Use test set to evaluate.
5 results_comparison = X.copy()
6 results_comparison['magnesium'] = np.exp(results_comparison['magnesium'])
7 results_comparison['proline'] = np.exp(results_comparison['proline'])
8 results_comparison['PredictedWine'] = predict
9 results_comparison['ActualWine'] = y.copy()
10 results_comparison['ProbWine0'] = np.round(proba[:, 0] * 100, 2)
11 results_comparison['ProbWine1'] = np.round(proba[:, 1] * 100, 2)
12 results_comparison['ProbWine2'] = np.round(proba[:, 2] * 100, 2)
13
14 # View examples of the predictions compared to actual wine.
15 results_comparison.head(20)

```

- Select **Run**.
- Examine the output.

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od
0	13.69	3.26	2.54	20.0	107.0	1.83	0.56	0.50	0.80	5.88	0.96	
1	12.67	0.98	2.24	18.0	99.0	2.20	1.94	0.30	1.46	2.62	1.23	
2	13.86	1.35	2.27	16.0	98.0	2.98	3.15	0.22	1.85	7.22	1.01	
3	13.73	1.50	2.70	22.5	101.0	3.00	3.25	0.29	2.38	5.70	1.19	
4	13.41	3.84	2.12	18.8	90.0	2.45	2.68	0.27	1.48	4.28	0.91	
5	12.20	3.03	2.32	19.0	96.0	1.25	0.49	0.40	0.73	5.50	0.66	
6	13.83	1.65	2.60	17.2	94.0	2.45	2.99	0.22	2.29	5.60	1.24	
7	12.69	1.53	2.26	20.7	80.0	1.38	1.46	0.58	1.62	3.05	0.96	
8	14.75	1.73	2.39	11.4	91.0	3.10	3.69	0.43	2.81	5.40	1.25	
9	12.43	1.53	2.29	21.5	86.0	2.74	3.15	0.39	1.77	3.94	0.69	
10	13.05	5.80	2.13	21.5	86.0	2.62	2.65	0.30	2.01	2.60	0.73	
11	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	7.70	0.64	

- The magnesium and proline features you transformed earlier have now been scaled back to their original values.
- The predictions and actual values for each wine are displayed as their own columns.
- Each of the three classes has its own ProbWine# column, indicating what proportion of those three classes the model felt was most likely for a given example. The classification prediction is derived from the class with the highest probability.

10. Shut down this Jupyter Notebook kernel.

- a) From the menu, select **Kernel**→**Shutdown**.
 - b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
 - c) Close the **Classification-Wine** tab in Firefox, but keep a tab open to **CAIP/Classification/** in the file hierarchy.
-

Do Not Duplicate or Distribute

TOPIC C

Evaluate Classification Models

It's not enough to just train a model you think is best, and then call it a day. Unless you're using a very simple dataset or you get lucky, the default parameters aren't going to give you the best possible model for solving the problem. So, in this topic, you'll evaluate the performance of your classification models to see how they're performing.

Model Performance

A classifier algorithm examines a given input and identifies what class it belongs to. For example, a classifier might examine data describing a living organism and identify whether it is an animal or not. The classifier would label animals as "Yes" (positive), and other organisms (plants, fungi, bacteria, and so forth) as "No" (negative). As you train a classifier, there are various ways the model might succeed or fail, as depicted in this table.

True (Correct) Label	Predicted Label	Result	Assessment
Yes	Yes	Success	True positive (TP)
No	No	Success	True negative (TN)
No	Yes	Failure	False positive (FP)
Yes	No	Failure	False negative (FN)

Confusion Matrix

A **confusion matrix** is a method of visualizing the truth results of a classification problem. It helps you more easily examine the dimensions of a classification problem, as well as enumerate true positives/negatives and false positives/negatives within the predicted data. In a confusion matrix, the number of predictions is compared to the number of actual values in a table format. There are several ways you can structure this table; the following is one example, using a model with two classes (a binary classifier).

		Actual	
		Yes	No
Prediction	Yes	True positives	False positives
	No	False negatives	True negatives

Figure 6–9: A confusion matrix template.

In a real-world example, consider that you're training a supervised machine learning model to predict heart disease in patients. Whether or not the patient has heart disease is therefore the label. After training the model on the data, it'll make predictions that end up either being true or false and negative or positive. Plugging those values into a confusion matrix, you might get something like the following.

		Actual	
		Heart disease	No heart disease
Prediction	Heart disease	43	6
	No heart disease	3	212

Figure 6–10: A confusion matrix of heart disease predictions.

Using a table like this, it's much easier to identify the prevalence of errors in the model's prediction. To summarize the example:

- 43 patients were correctly predicted to have heart disease (true positives).
- 6 patients were incorrectly predicted to have heart disease when they actually do not (false positives).
- 3 patients were incorrectly predicted to not have heart disease when they actually do (false negatives).
- 212 patients were correctly predicted to not have heart disease (true negatives).

Classifier Performance Measurement

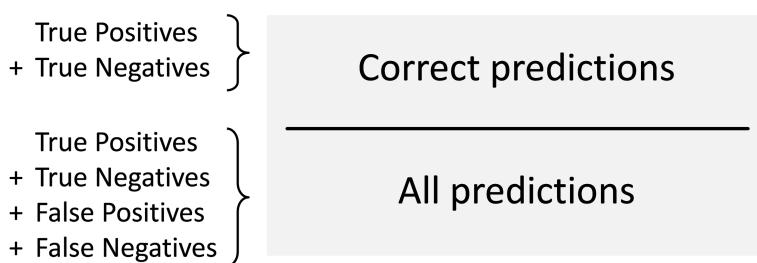
When training a classifier, you must evaluate how well it performs—not only when it succeeds in identifying positives or negatives, but also when it fails to correctly identify positives and negatives. Ideally, every prediction made by the model would be correct. But in the real world that is seldom possible, so you must tune the model to make compromises that will meet your needs. You tune the model to ensure correct identifications where they are essential, at the expense of allowing failures where they can be tolerated.

For example, in some situations, it might be better to have a model that ensures there will be no false negatives at the expense of allowing perhaps 30% of positive predictions to be false. For example, you'd probably want your machine learning model to identify every bridge that has a structural problem before you have a failure that harms someone. Saving lives would outweigh the cost of having someone investigate bridges that don't actually require any repairs. In other situations, it might be better to ensure there are no false negatives while allowing a small number of false positives.

Having various ways to measure how the model performs will help you optimize it for a particular situation.

Accuracy

Accuracy is a measure of how frequently each prediction is correctly deemed positive or negative.

**Figure 6-11: Accuracy.**

As shown here, accuracy is the number of correct predictions divided by the number of all predictions made. In other words, accuracy can be expressed as:

$$\frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

If accuracy is perfect (all predictions are correct), it will be measured as 1.0. If half the predictions are correct, the accuracy will be 0.5. These can be converted into percentages as needed.

Accuracy is intuitive, and the word itself has the connotation of being highly desirable. However, accuracy often proves to be an unreliable measure of model performance. Consider the example of identifying bridges with structural problems to predict those that might collapse. You create a model in which the label is "bridge has collapsed." That model would probably end up with a great many true negatives and almost no true positives, false positives, or false negatives, because bridge collapses are exceedingly rare. Here's what this might look like in a confusion matrix.

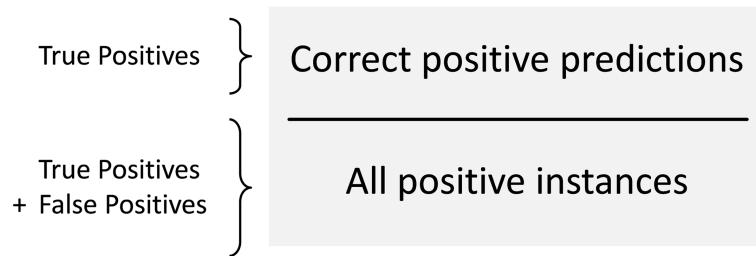
		Actual	
		Collapse	No collapse
Prediction	Collapse	1	1
	No collapse	2	512

Figure 6-12: The bridge collapse example in a confusion matrix.

As a result, the accuracy is extremely high—close to 99%. This makes accuracy nearly useless, as the machine learning model will become no better at predicting the rare cases in which a bridge shows signs of imminent collapse. Accuracy is therefore only useful in datasets where the label data is balanced.

Precision

Precision is a measure of how often the positives identified by the learning model are true positives.

**Figure 6-13: Precision.**

As shown here, precision is the number of correct positive predictions divided by the number of all positive predictions made. In other words, precision can be expressed as:

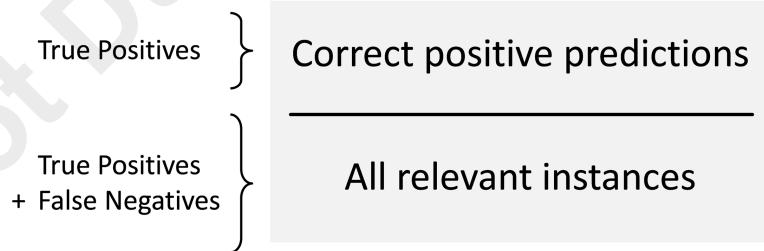
$$\frac{\text{TP}}{\text{TP} + \text{FP}}$$

Like accuracy, precision can be expressed as a number from 0 to 1.0 or as a percentage.

Precision is typically more useful than accuracy, especially in asymmetric datasets, but you still need to consider the context of the data and what outcomes you're looking for. Using the dataset, the model correctly predicted that 5 bridges were about to collapse. It also predicted that 2 bridges were about to collapse when they actually weren't. So, the precision would be $5 / 5 + 2$, or roughly 71%. This is more useful than accuracy, because it accounts for the unbalanced nature of the dataset with regard to the label. However, the problem with precision in this case is that it doesn't address cases in which a bridge will collapse but the model doesn't think it will (i.e., false negatives). Even just a single bridge collapsing is intolerable because of the damage it causes—if the model doesn't predict that instance, it may be considered a failure. Even if you set your tolerance higher than just one, precision will still come up short in assessing this model's performance.

Recall

Recall is the percentage of positive instances that are found by a model as compared to all relevant instances. A "relevant" instance is any instance that is actually true, even if the prediction is wrong.

**Figure 6-14: Recall.**

As shown here, recall is the number of correct positive predictions divided by the number of correct positive predictions plus the number of incorrect negative predictions. In other words, recall can be expressed as:

$$\frac{TP}{TP + FN}$$

Like accuracy and precision, recall can be expressed as a number from 0 to 1.0 or as a percentage.

As you've seen, precision might not be the most useful way to measure how good a machine learning model is at predicting bridge collapse. Let's try recall. In the bridge example, assume the model was able to correctly predict the collapse of 5 bridges. The model failed to predict the collapse of a single bridge. The recall would be $5 / 5 + 1$, or around 83%. Now, you have a better idea of how well your model performs with respect to its ultimate purpose—minimizing bridge collapse and the damage it can cause. Technically, the model could improve its recall by predicting that all bridges are about to collapse, leading to a recall of 100%, but this would make the model useless for prioritizing repair efforts.

Precision-Recall Tradeoff

In general, there is tradeoff between precision and recall. As you tune a machine learning model to emphasize *one* of these factors, you do so at the expense of de-emphasizing the *other*, as depicted through this table and chart.

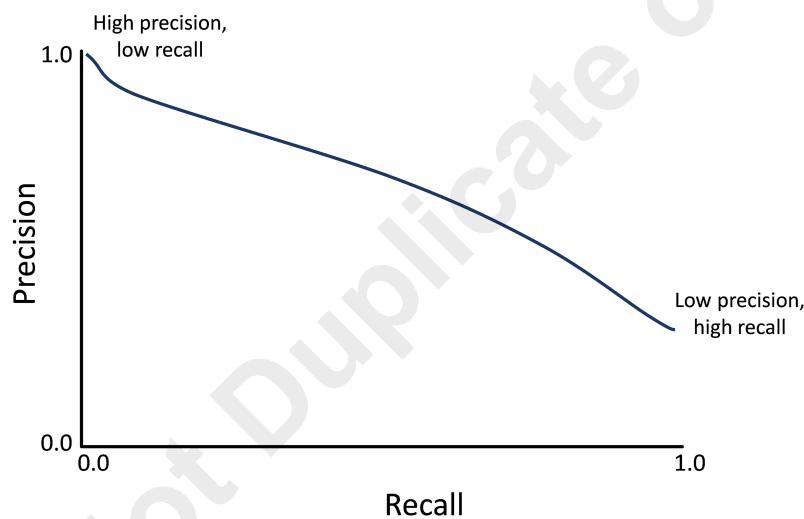


Figure 6–15: The precision and recall tradeoff.

Action	False Positives	False Negatives
Increase precision	Tend to decrease	Tend to increase
Increase recall	Tend to increase	Tend to decrease

Which factor you choose to emphasize depends on your business requirements.

F₁ Score

As you've seen, precision and recall are more useful in unbalanced datasets, but they come with a tradeoff. Sometimes, like in the bridge example, it's relatively clear which one is more useful. However, this is not always the case. Consider a machine learning task in which you want to determine whether or not a song is in the rock genre, given a short audio sample. A false positive (e.g., classifying a song as rock when it's not) is just as undesirable as a false negative (e.g., not classifying a song as rock even though it is); neither is particularly worse. So, what's the best way to measure performance in this case?

The **F₁ score** helps you find the optimal combination of both precision and recall. The F₁ score essentially just takes a weighted average (more precisely, a harmonic mean) of both precision and recall. The weighted average reduces the effect of extreme values. The F₁ score can be expressed as:

$$F_1 = 2 \left(\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \right)$$

So, assume that 100 songs were correctly classified as rock, and 10 songs were incorrectly classified as rock. The precision is ~91%. If 23 songs were not classified as rock even though they are, then the recall would be ~81%. Plug these values into the formula like so:

$$F_1 = 2 \left(\frac{.91 \cdot .81}{.91 + .81} \right)$$

The resulting F₁ score is ~.857 or around 86%.

To reiterate, the F₁ score is preferred when label values in the dataset are not distributed evenly, *and* neither precision nor recall are more useful than the other.

Receiver Operating Characteristic (ROC) Curve

A **receiver operating characteristic (ROC) curve** is a method of plotting the relationship between predicted "hits" versus false alarms. On the y-axis is the **true positive rate (TPR)** (the "hits"), which is essentially the same as the recall. On the x-axis is the **false positive rate (FPR)** (the false alarms), which is expressed as:

$$\frac{\text{FP}}{\text{FP} + \text{TN}}$$

Once these two values for a model are plotted on the graph, a line can be fit to the data. This line starts from the bottom left of the graph (0 FPR, 0 TPR) and continues to the top right of the graph (1 FPR, 1 TPR). Once the line is fit, it becomes the ROC curve.

ROC curves are useful because they help you tune a machine learning model to achieve the desired balance between the TPR and the FPR. In particular, they can help you evaluate two things. First, they can help you compare the performance of two or more models. Consider the following graph:

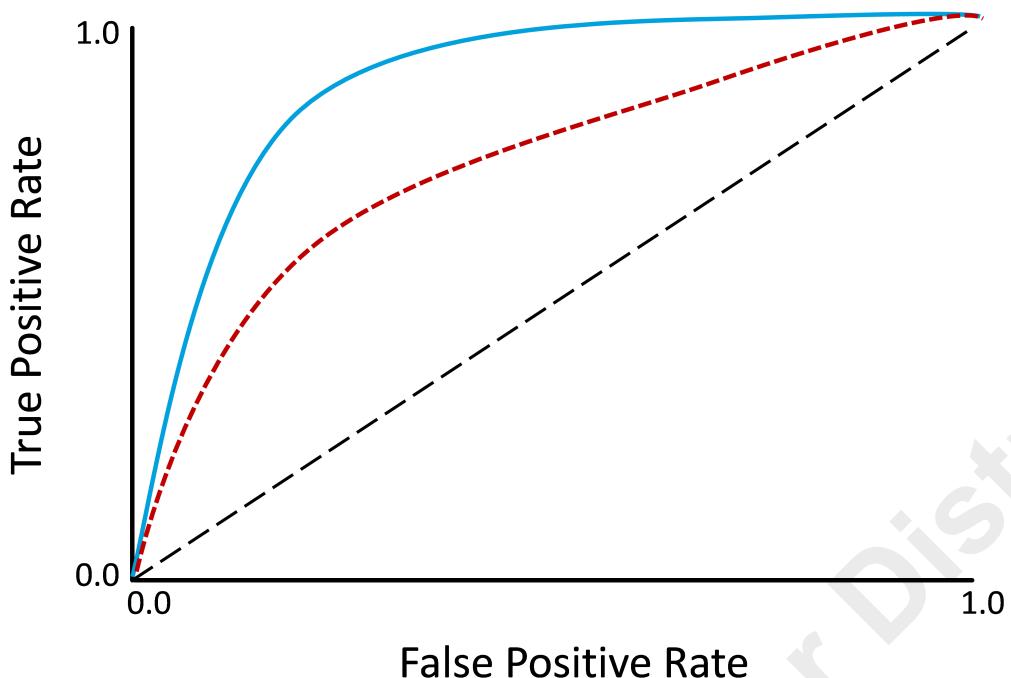


Figure 6-16: ROC curves plotted on a graph.

The diagonal dashed line (0.5, 0.5) represents a random guess. Therefore:

- Any data above and to the left of the dotted line is *better* than a random guess. This is what you want for your model.
- Any data near or at the dotted line is *no better* than a random guess. This is not what you want for your model.
- Any data below and to the right of the dotted line is *worse* than a random guess. This typically means there's a problem with the model.

A perfectly predictive model would be fit from the top-left corner of the graph (0, 1). This is unrealistic, but the more a model's curve approaches this top-left corner, the better it performs. So, in the figure, the model represented by the unbroken blue curve performs better than the model represented by the dotted red curve.



Note: You can use a confusion matrix to easily calculate the TPR and FPR.

Thresholds

The second way to use a ROC curve is to evaluate a model by itself, not in comparison to any other model. After all, one model may perform better than another, but still perform poorly overall. You could do an independent evaluation by configuring a **threshold**. Any data above this threshold is considered positive; anything below, negative. In other words, the threshold creates a decision boundary. In the case of predicting heart disease, you could set a threshold of 0.70. If the model makes a prediction of heart disease in a patient with a value of 0.75, that will be considered a 1 (has heart disease). If the prediction is 0.65, then that will be considered a 0 (no heart disease).

The problem with this method is that it can be very difficult to actually decide what a good threshold is for your data. As explained before, you'll want to treat data differently depending on what it represents. In the case of heart disease, you would want to put much more weight on

minimizing false negatives rather than false positives. So, what threshold would you set? Set it too low and the model will predict heart disease where there is none. This is due to a high true positive rate (TPR)/recall, also called *sensitivity*, which leads to a low true negative rate, also called *specificity*. Set the threshold too high and the model will miss people who do have heart disease. This is due to a high specificity, which leads to low sensitivity. The problem lies in judging the tradeoff between the two.

In this first example, the threshold is 0.50—right in the middle. The red Xs represent the actual negative values in the dataset, and the green check marks represent the actual positive values in the dataset. So, with this threshold, one false positive and two false negatives are generated by the predictive model.

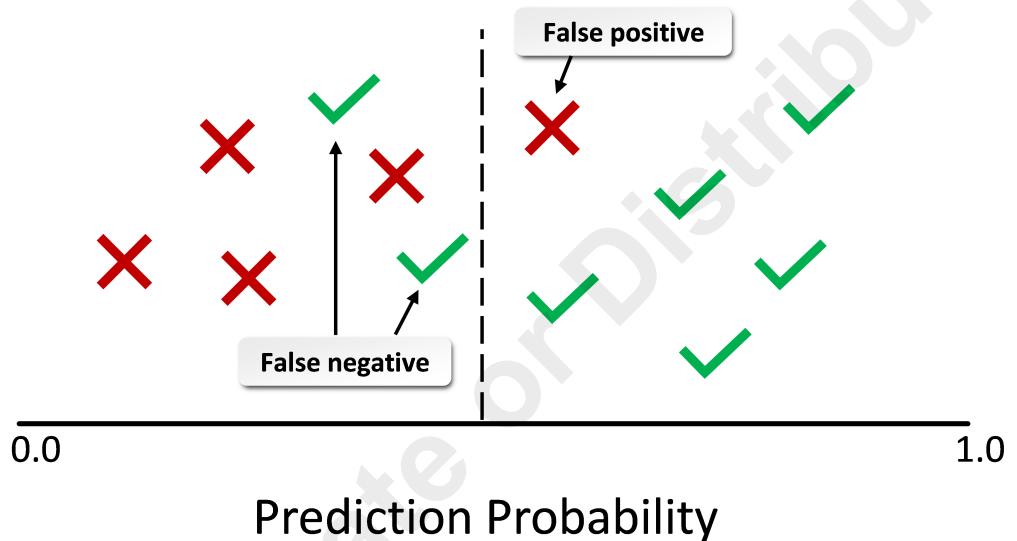


Figure 6–17: Plotting classification values with a threshold of 0.50.

Now, in this second example, the threshold is increased to around 0.70. Because of this, the false positive from before has been eliminated, and its data instance is now a true negative. However, this also added one more false negative.

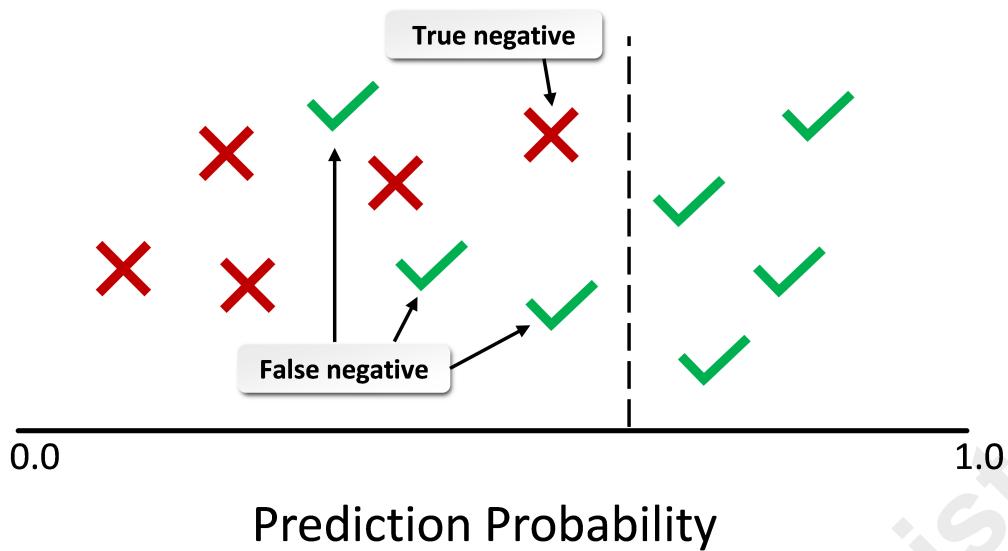


Figure 6–18: Moving the threshold to ~0.70.

Thankfully, there is another approach.

Area Under Curve (AUC)

The **area under curve (AUC)** is, as its name suggests, the total space that is under a model's ROC curve. The AUC can also be defined as the probability that the model will classify any positive instance higher than any negative instance, from 0 to 1.0. This minimizes the need to decide on a threshold, because the AUC considers *all* possible thresholds. So, AUC is a useful way to evaluate the overall performance of a classification model.

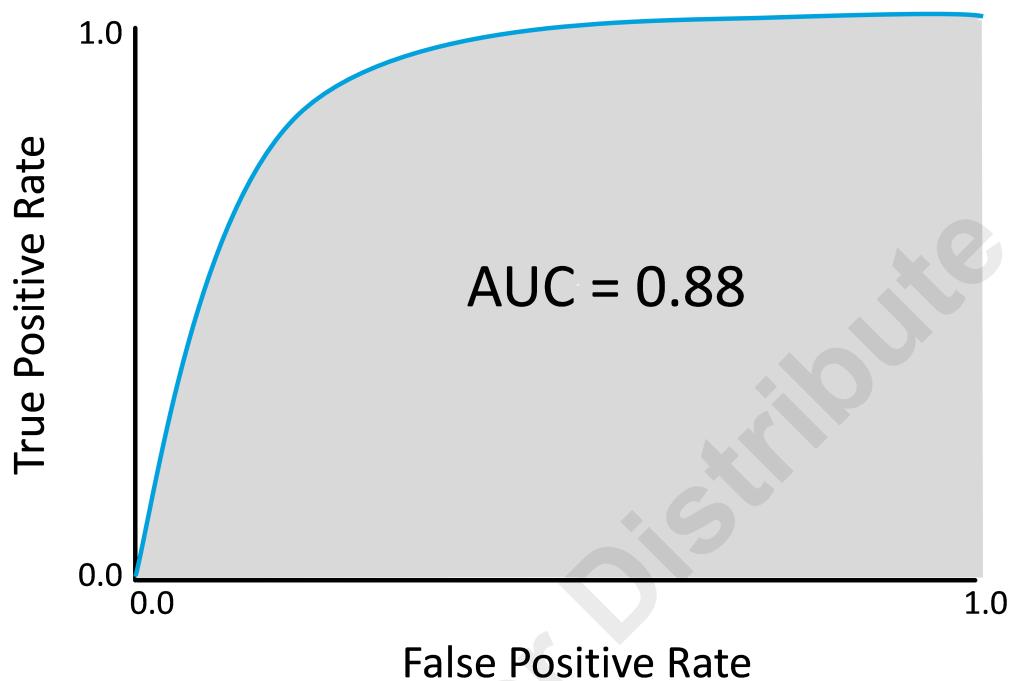


Figure 6-19: The AUC in this example is 0.88.

The AUC must at least be above 0.5 (random guessing), but there is no gold standard that applies to all data. You need to, once again, consider the context of your data and what it would mean to raise or lower the expected AUC. For diagnosing heart disease, you would only accept a very high AUC (>0.9), whereas in less critical fields, you'd be able to settle for a lower AUC. Consider researching AUC values for your particular industry or for the particular problem you are trying to solve.

However, the AUC of a ROC curve is not useful in datasets where there is a class imbalance, as it may be better to focus on minimizing one classification error rather than optimizing for multiple errors.



Note: Several math programming libraries can calculate the AUC for you.

Precision–Recall Curve (PRC)

If your data *does* have a class imbalance, it is preferable to plot data on a precision–recall curve. A [precision–recall curve \(PRC\)](#) plots precision values on the y-axis and recall values on the x-axis for different thresholds. This provides you with a way to visualize the tradeoff between these two values and focus on minimizing a particular type of error.

For example, the bridge collapse model marks a bridge as in danger of collapse (1) or not in danger of collapse (0). The classification in this problem is imbalanced; very few bridges will qualify as being in danger of collapse. You're therefore not all that interested in seeing how well the model predicts true negatives, because negatives make up the majority of the data. PRCs do not involve true negatives. Instead, they help you evaluate how well the model predicts the class in the minority (class 1) which (in this case) is the whole point of the model.

Consider the following PRC:

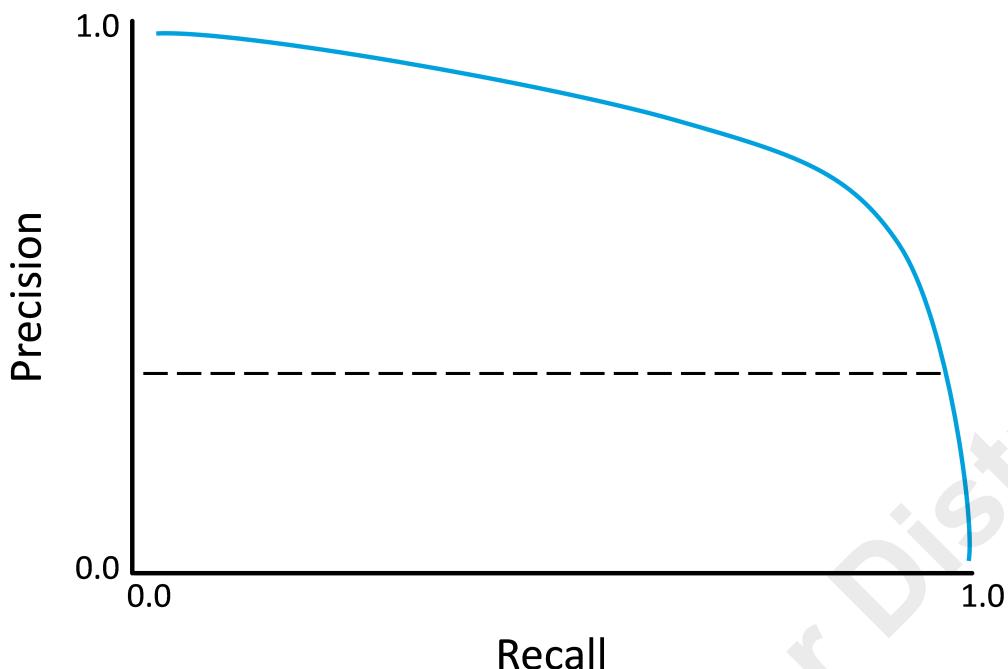


Figure 6–20: A skillful model, with the curve above the line.

The horizontal dashed line represents the "no-skill" line, which is the total positive instances divided by the total positive and negative instances. If there are equal numbers of positive and negative cases, the no-skill line will be 0.5. In any case, you want your curve to be above this line, otherwise the model is not useful. A model with perfect skill would be drawn from the top right of the graph (1.0, 1.0). In the example figure, the curve is above the no-skill line, so the model is considered skillful.

For a specific probability threshold, you can use the F_1 score mentioned earlier to summarize the precision–recall tradeoff. You can also calculate the AUC of a PRC, much like you can with a ROC curve, in order to summarize the tradeoff for all thresholds. Another common way to summarize a precision–recall curve is to calculate the average precision from all predicted probabilities for each threshold.

Guidelines for Evaluating Classification Models

Follow these guidelines when you are evaluating classification models.

Evaluate a Classification Model

When evaluating a classification model:

- Consider the significance of the four truth values for a classification problem: true positive, true negative, false positive, false negative.
- Use a confusion matrix to visualize the truth results of a classification problem.
- Consider that accuracy may not be useful in datasets with a class imbalance.
- Prefer precision and recall when the dataset has a class imbalance.
- Consider that recall may be better than precision when it's crucial that the model avoid false negatives (e.g., in predicting the presence of disease).
- Consider that there is a tradeoff between precision and recall—as one increases, the other tends to decrease.

- Use F_1 score when neither precision nor recall are more important than the other.
- Generate a ROC curve and its AUC to compare the true positive rate (TPR) with the false positive rate (FPR) for all thresholds.
- Prefer a precision–recall curve (PRC) to a ROC curve when there is a class imbalance in the dataset.
- Use average precision to summarize a PRC.
- Consider that there is no objectively "correct" evaluation metric for all problems, or even for a given type of problem.
- Consider applying multiple metrics to the same problem.

Use Python to Evaluate a Classifier

The scikit-learn `metrics` module has multiple functions you can use to evaluate a classification model. Note that the `prediction` argument is an object created by calling `predict()` on the model, whereas `prediction_proba` is the same, but for calling `predict_proba()`.

- `sklearn.metrics.confusion_matrix(y_test, prediction)` —Return the confusion matrix for the model given validation/test data.
- `sklearn.metrics.accuracy_score(y_test, prediction)` —Return the accuracy score for the model given validation/test data.
- `sklearn.metrics.precision_score(y_test, prediction)` —Return the precision score for the model given validation/test data.
- `sklearn.metrics.recall_score(y_test, prediction)` —Return the recall score for the model given validation/test data.
- `sklearn.metrics.f1_score(y_test, prediction)` —Return the F_1 score for the model given validation/test data.
- `sklearn.metrics.roc_curve(y_test, prediction_proba)` —Return the ROC curve for the model given validation/test data.
- `sklearn.metrics.roc_auc_score(y_test, prediction_proba)` —Return the ROC AUC score for the model given validation/test data.
- `sklearn.metrics.precision_recall_curve(y_test, prediction_proba)` —Return the PRC for the model given validation/test data.
- `sklearn.metrics.average_precision_score(y_test, prediction_proba)` —Return the average precision score for the model given validation/test data.

ACTIVITY 6–4

Evaluating a Classification Model

Data Files

/home/student/CAIP/Classification/Classification-Titanic-Tuned.ipynb
/home/student/CAIP/Classification/titanic_data/test.csv
/home/student/CAIP/Classification/titanic_data/train.csv

Before You Begin

Jupyter Notebook is open.

Scenario

Your *Titanic* classifier was a hit with students, but it still has room for improvement. You don't just want the model to make predictions, you want it to make *good* predictions. So, to improve the model's skill, you'll evaluate its performance in a variety of ways. Later, you'll tune the model based on your findings and expectations. This will hopefully lead to an optimized classifier that you and your students can be more confident in using.

1. From Jupyter Notebook, select **CAIP/Classification/Classification-Titanic-Tuned.ipynb** to open it.
2. Import the relevant libraries and load the dataset.
 - a) Select the code cell beneath the **Import software libraries and load the dataset** title, then select **Run**.
 - b) Verify that **test.csv** and **train.csv** are in the project folder, and that the latter was loaded with 891 records.
3. Split the datasets.
 - a) Scroll down and select the code cell beneath the **Split the datasets** title, then select **Run**.
This splits the dataset the same way as before.
4. Perform common preparation on the training and validation sets.
 - a) Scroll down and select the code cell beneath the **Perform common preparation on the training and validation sets** title, then select **Run**.
The data is being prepared in the same way as before.
5. Drop columns that won't be used for training.
 - a) Scroll down and select the code cell beneath the **Drop columns that won't be used for training** title, then select **Run**.
Irrelevant data is being dropped in the same way as before.
6. Create a logistic regression model.
 - a) Scroll down and select the code cell beneath the **Create a logistic regression model** title, then select **Run**.
 - b) Verify that the score on this model is 78%.
This is the same model that you created in the earlier activity. A score of 78% is not necessarily the best you can get out of the data, and the "score" itself is not the only way to assess the model, so you'll do some additional analysis.

7. Identify the class distribution in the dataset.

- Scroll down and select the code cell beneath the **Identify the class distribution in the dataset** title, then select **Run**.
- Examine the output.

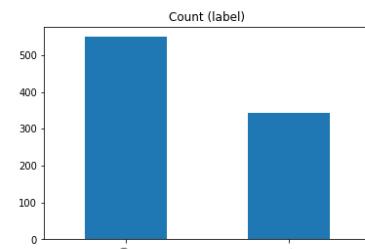
Identify the class distribution in the dataset

```

1 label_count = data_raw['Survived'].value_counts()
2 print('Perished (Class 0):', label_count[0])
3 print('Survived (Class 1):', label_count[1])
4
5 label_count.plot(kind = 'bar', title = 'Count (label)');

```

Perished (Class 0): 549
Survived (Class 1): 342



Since there were more passengers who perished than survived, the dataset appears to have a slight class imbalance. This can influence how useful certain metrics are in assessing the skill of the classification model. However, the imbalance is not necessarily significant.

8. Generate a confusion matrix.

- Scroll down and select the code cell beneath the **Generate a confusion matrix** title, then select **Run**.
This function is defined, but hasn't been called yet. When called, the function will use Seaborn to plot a heatmap of the predicted label values and actual label values. The heatmap also doubles as a confusion matrix.

9. Compute accuracy, precision, recall, and F_1 score.

- Scroll down and select the code cell beneath the **Compute accuracy, precision, recall, and F_1 score** title, then select **Run**.
This function, when called, will compute four statistical measures for the model: accuracy, precision, recall, and F_1 score. Note that the "score" you've calculated thus far for logistic regression (by calling `score()` on a model) is the same as the accuracy.

10. Generate a ROC curve and compute the AUC.

- Scroll down and select the code cell beneath the **Generate a ROC curve and compute the AUC** title, then select **Run**.
This function, when called, will plot a ROC curve. It will also compute the area under that curve, which is a common method for summarizing a ROC curve.

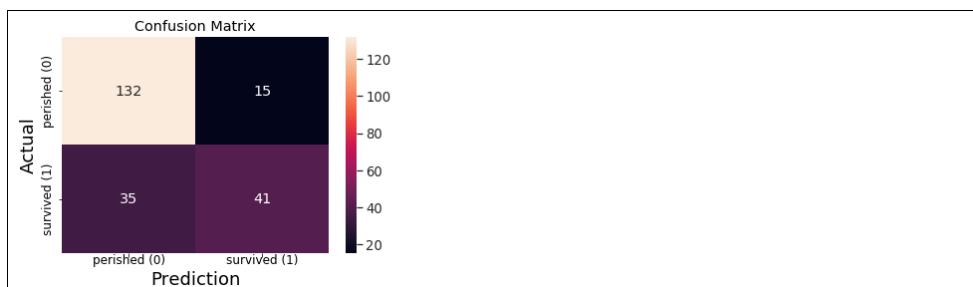
11. Generate a precision–recall curve and compute the average precision.

- Scroll down and select the code cell beneath the **Generate a precision–recall curve and compute the average precision** title, then select **Run**.
This function, when called, will plot a precision–recall curve. It will also compute the average precision score, which is a common method for summarizing a precision–recall curve.

12. Evaluate the initial logistic regression model.

- Scroll down and select the code cell beneath the **Evaluate the initial logistic regression model** title, then select **Run**.

- b) Examine the output.



To help orient you, each quadrant is plotted as follows:

- Top-left (132): *True negatives*
- Top-right (15): *False positives*
- Bottom-left (35): *False negatives*
- Bottom-right (41): *True positives*

13. While addressing the unique problem your classification model is trying to solve, it helps to contextualize confusion matrix results in real-world terms. When you have a fundamental understanding of the data and what you're expected to get out of it, you will make better decisions about what metrics to focus on improving.

What does each quadrant indicate in terms of predicting survivors of the *Titanic*?

14. Continue evaluating the initial logistic regression model.

- a) Scroll down and select the next code listing cell.

```
In [ ]: 1 model_scores(y_val, initial_predict)
```

- b) Select **Run**.
c) Examine the output.

```
Accuracy: 78%
Precision: 73%
Recall: 54%
F1: 62%
```

The familiar accuracy score is printed, as are precision, recall, and F_1 scores for this model.

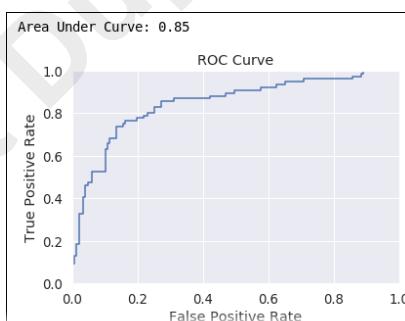
15. In what situation are precision and recall a better measure of a model's skill than accuracy?
16. Think about the problem unique to this *Titanic* dataset. In particular, consider the problem as it pertains to the model's accuracy, precision, recall, and F_1 score.
- Is there any one of these measures you'd be more interested in optimizing than the others? Why or why not?

17. Continue evaluating the initial logistic regression model.

a) Scroll down and select the next code listing cell.

```
In [ ]: 1 initial_predict_proba = log_reg.predict_proba(X_val)
          2
          3 roc(y_val, initial_predict_proba[:, 1])
```

b) Select Run.
c) Examine the output.

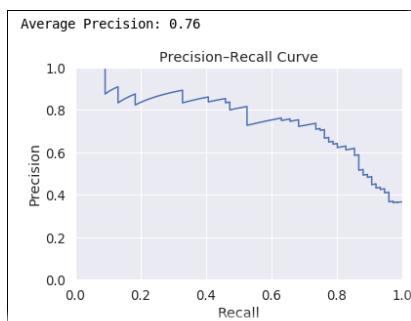


The ROC curve is above and to the left of graph, indicating that the model is performing better than a random guess. The AUC also confirms this, as it is well above 0.5.

d) Scroll down and select the next code listing cell.

```
In [ ]: 1 prc(y_val, initial_predict_proba[:, 1])
```

- e) Select Run.
- f) Examine the output.



The precision–recall curve is above and to the right of the no-skill line, indicating that the model is performing well with regard to the precision–recall tradeoff. The average precision confirms this.

- 18.What are the advantages of a ROC curve over a precision–recall curve, and vice versa? Given your domain knowledge, are you more interested in improving one of these curves over the other? Why or why not?

TOPIC D

Tune Classification Models

Now that you've evaluated your classification models, you want to put that evaluation to good use. In this topic, you'll attempt to improve the skill of your models, and you'll use your evaluation methods to confirm these improvements.

Hyperparameter Optimization

The previous evaluation metrics help you get a good idea of how your models are performing. The next step is to actually begin reconfiguring your model to improve its performance on those metrics, and ultimately, improve its overall predictive skill. While there are several data processing methods for improving a model, like re-engineering the training data through cleaning, standardization, normalization, etc., one of the most significant improvement techniques during training itself is hyperparameter optimization. **Hyperparameter optimization** is the process of repeatedly altering the hyperparameters that an algorithm uses to train a model in order to determine the set of hyperparameters that lead to the best or the desired level of model performance.

You could try to optimize the hyperparameters manually, evaluating performance after each iteration, but this can get tedious and is prone to error. There are several automated methods for hyperparameter optimization, including:

- Grid search
- Randomized search
- Bayesian optimization
- Genetic algorithms



Note: Hyperparameter optimization methods work for any algorithm that has hyperparameters, not just classification algorithms.

Grid Search

Grid search is a hyperparameter optimization method that simply takes a set (or grid) of parameter combinations, trains a model using each of those combinations, and then returns the combination that best optimizes an evaluation metric that you specify. Grid search also uses cross-validation to derive the optimal parameters by training over several folds. For example, you might construct a basic parameter grid for logistic regression like the following:

```
grid = [{}{'solver': ['liblinear'],
           'penalty': ['l1', 'l2'],
           'C': [0.01, 0.1, 1, 5, 10, 100]}]
```

In this case, `solver`, `penalty`, and `C` are all separate hyperparameters. Each hyperparameter is used in all possible combinations. So, grid search will first train on the dataset $1 \times 2 \times 6 = 12$ times. The actual search object would look something like this:

```
search = GridSearchCV(model, param_grid = grid, scoring = 'f1', cv = 5)
```

Because `cv` is 5, grid search will do 5-fold cross-validation for each combination—in other words, it will train $12 \times 5 = 60$ times. The `scoring` argument is the evaluation metric you want to optimize the model on. So, in this case, you're looking for the combination of hyperparameters that will return the best F_1 score. That optimal combination is returned and can then be used to construct a more ideal model.



Note: If you're not sure what values to add to the grid when it comes to numerical hyperparameters (like C), consider starting at a low value and then exponentially increase that value.

Randomized Search

While grid search will ultimately derive the optimal hyperparameters for a given metric, if your dataset has a very large feature space, grid search can take a very long time to exhaustively evaluate every possible combination of hyperparameters. **Randomized search** is an alternative approach in which random combinations of hyperparameters are used to train and evaluate the model. Rather than a grid of discrete values, hyperparameters in randomized search are typically defined in distributions; the algorithm will select a combination of values from these distributions for a certain number of specified iterations. Consider the following parameter distribution and accompanying randomized search object:

```
dist = {'solver': ['liblinear'],
        'penalty': ['l1', 'l2'],
        'C': sp_randint(1, 100)}
```

```
search = RandomizedSearchCV(model, param_distributions = dist, n_iter = 50,
                             scoring = 'f1', cv = 5)
```

The `solver` and `penalty` hyperparameters are the same discrete values as they were before, but the values for `C` are now a distribution of random integers that range from 1 to 100 (this what SciPy's `randint()` method does). For the `search` object, the `n_iter` argument specifies how many times a random combination of the hyperparameters is sampled—in this case, 50. As with grid search, this randomized search method will also perform cross-validation given a number of folds.

The fact that you can control the number of iterations of random hyperparameter sampling gives you more granular control over the training process, and subsequently, enables you to minimize training time in large feature spaces. While randomized search is not as thorough as grid search, and may not find the truly optimal hyperparameters in some cases, it will usually lead to adequate results.

Bayesian Optimization

Even randomized search is susceptible to training time issues, especially if you need to conduct many random sampling iterations over huge datasets. The algorithm needs to find a combination of random hyperparameters for each iteration, all while looking for a combination that optimizes the specified evaluation metric. **Bayesian optimization** is a hyperparameter optimization method that uses past samples to influence where sampling is conducted in subsequent iterations, enabling it to determine the next optimal space to sample from. This makes Bayesian optimization "smarter" than randomized search, because it is able to get to the optimal hyperparameters much faster than truly randomized sampling.

A simplified explanation of the Bayesian optimization process is as follows:

1. Start with an initial random sampling of the hyperparameter distribution space.
2. Evaluate the loss function (i.e., the chosen evaluation metric) for this space.
3. Use this evaluation to compute a posterior distribution of the loss function—in other words, capture the "beliefs" of the prior evaluation(s), then update those beliefs to account for everything that is currently known about the loss function.
4. Sample a new hyperparameter space that optimizes an acquisition function—in other words, given the posterior distribution, find the next space with the best tradeoff between loss optimum prediction and high prediction uncertainty.
5. Evaluate the loss from this new space.
6. Repeat steps 3 through 5 until some stopping criterion is met, such as a specified number of iterations or until the loss no longer improves.

Bayesian optimization is more complicated and harder to implement than grid search or randomized search, as it requires more careful tuning of the search configurations. However, it has been shown to be faster than grid search and randomized search in some scenarios.



Note: The default implementation of scikit-learn does not come with a Bayesian optimization library, but there are several third-party APIs that provide this functionality.

Genetic Algorithms

A **genetic algorithm** is an approach to optimization that is inspired by the theory of natural selection formulated by Charles Darwin. While genetic algorithms can optimize many types of problems, in the context of hyperparameter optimization, the process is generally as follows:

1. Create a "population" of randomly selected hyperparameter combinations. This population will have several members, each member a different hyperparameter combination.
2. Evaluate each member of the population by the chosen metric. (In genetic algorithms, this is commonly called the fitness function.)
3. "Breed" the highest-performing members of the population to create new members. There are two approaches to this, both of which may be used:
 - Crossover/recombination—Members (parents) combine to generate offspring that inherit traits from both parents. So, a child might have some hyperparameters from Parent A, and other hyperparameters from Parent B.
 - Mutation—Members produce offspring with slight random alterations to promote genetic diversity. So, a child might have a few hyperparameters that weren't present in its parents.
4. Kill off the rest of the low-scoring members.
5. Repeat steps 2 through 4 until some stopping criterion is reached, such as a specified number of iterations or until the fitness no longer improves.

Each iteration of the process is called a generation. The second generation is produced from the first; the third generation is produced from the second; and so on. The surviving parents are also kept for each generation, not just their offspring.

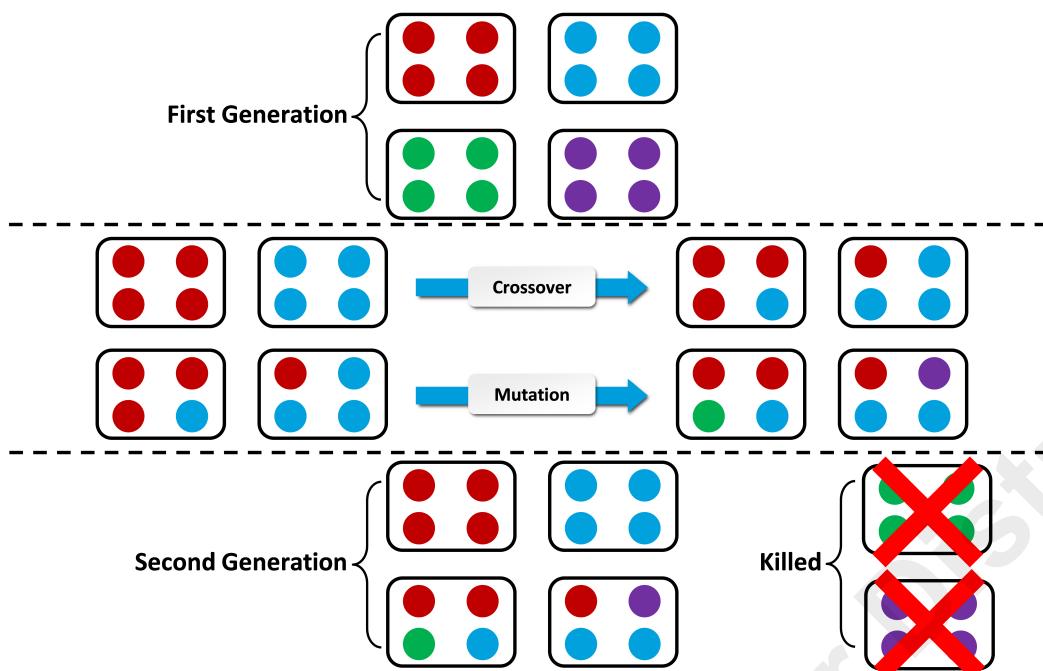


Figure 6–21: The hyperparameter optimization process applied by genetic algorithms.

One advantage to genetic algorithms is that they can avoid being stuck at local optima. This is because searches are executed in parallel from the population, rather than from any single point. Genetic algorithms are most effective when the number of hyperparameter combinations is relatively low. With a very large hyperparameter space, it can take a long time for genetic algorithms to derive the optimal hyperparameters. Because of these efficiency issues, they are typically not as common as other optimization techniques.



Note: The process of a genetic algorithm is also referred to as evolutionary optimization.

Guidelines for Tuning Classification Models

Follow these guidelines when you are tuning classification models.

Tune a Classification Model

When tuning a classification model:

- Use a search technique like grid search to find the optimal hyperparameters for a given model.
- Perform the search based on the metric(s) you're trying to optimize.
- Define a grid that includes the hyperparameters you want to try in combination.
- Prefer randomized search to cut down on search time, especially when there is a large field of values that you'd like to include in the search.
- Adjust the number of iterations in a randomized search, considering the tradeoff between search time and the quality of the results.
- Consider that Bayesian optimization can return results even faster than randomized search, but is more difficult to implement.
- Consider that genetic algorithms are not usually as useful as other optimization methods, especially with a large number of hyperparameter combinations.

- Use hyperparameter optimization for other types of machine learning problems—not just classification.

Use Python to Tune a Classifier

The scikit-learn `GridSearchCV()` and `RandomizedSearchCV()` classes can be used to perform hyperparameter optimization searches, while also performing cross-validation. The following are some of the objects and functions you can use to build such a model.

- `search = sklearn.model_selection.GridSearchCV(model, param_grid = grid, scoring = 'recall', cv = 5)` —This constructs a grid search object for the given machine learning model, using the provided parameter grid. In this case, the search will optimize based on recall, and will perform five-fold cross-validation.
- `search = sklearn.model_selection.RandomizedSearchCV(model, param_distributions = dist, n_iter = 100, scoring = 'recall', cv = 5)` —This constructs a grid search object for the given machine learning model. It takes a parameter distribution and the number of iterations with which to try the hyperparameter combinations.
- You can use these class objects to call the same `fit()`, `score()`, `predict()`, and `predict_proba()` methods as before, as well as any of the applicable `metrics` methods. For methods that return a value, the value that gets returned is from a model that uses the "optimal" hyperparameters that the search found. So, for a classifier, `score()` would return the accuracy of the model with the best hyperparameters, according to the search.



Note: The availability and functionality of these methods depends on the underlying model you've passed into the search object.

ACTIVITY 6–5

Tuning a Classification Model

Before You Begin

The CAIP/Classification/Classification-Titanic-Tuned.ipynb file is open in Jupyter Notebook, and you've executed the code through the point where you perform all of the initial evaluation.

Scenario

Now that you've assessed your *Titanic* model's skill, you'll attempt to improve that skill through hyperparameter optimization. Then, you'll evaluate the tuned model to verify whether or not it has actually improved, and how.

1. Fit a logistic regression model using grid search with cross-validation.

- Scroll down and view the cell titled **Fit a logistic regression model using grid search with cross-validation**, and examine the code listing below it.

Fit a logistic regression model using grid search with cross-validation

```
In [ ]: 1 from sklearn.model_selection import GridSearchCV
2
3 grid = [{solver: ['liblinear'],
4          'penalty': ['l1', 'l2'],
5          'C': [0.001, 0.01, 0.1, 1, 5, 10, 25, 50, 100]},
6          {'solver': 'sag',
7           'penalty': ['l2'],
8           'C': [0.001, 0.01, 0.1, 1, 5, 10, 25, 50, 100],
9           'max_iter': [10000]}}
10
11 search = GridSearchCV(log_reg, param_grid = grid, scoring = 'f1', cv = 5, iid = False)
12 search.fit(X_train, np.ravel(y_train));
13
14 print(search.best_params_)
```

The `grid` array holds the hyperparameters that grid search will use to train and evaluate the model. The array is actually divided into two dictionaries in order to keep valid arguments together, and to keep the grid search from trying arguments that are incompatible with one another. Each dictionary is as follows:

- Dictionary 1:
 - The `solver` is `liblinear`, which uses an iterative method called coordinate descent to solve classification problems. Compared to gradient descent, coordinate descent only updates one parameter at a time.
 - Both ℓ_1 and ℓ_2 regularization will be tried (`penalty`).
 - Various values for `C` (regularization strength) will be tried.
- Dictionary 2:
 - The `solver` uses stochastic average gradient (SAG).
 - Only ℓ_2 regularization will be tried, as SAG does not support ℓ_1 regularization.
 - The same values for `C` will be tried.
 - A maximum of 10,000 iterations will be tried in every case.

When `GridSearchCV()` is called:

- The hyperparameter array is passed in with the `param_grid` argument.
- The `scoring` argument is what the grid search is optimizing for. In this case, you're optimizing for F_1 score. You could optimize for whichever metric you'd prefer.
- By providing a `cv` value of 5, the dataset will be split using stratified k -fold cross-validation, where k is 5.

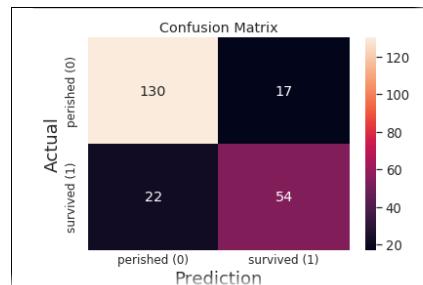
- Select the cell that contains the code listing, then select **Run**.
This will take a minute or two.
- Examine the output.

```
{'C': 10, 'penalty': 'l1', 'solver': 'liblinear'}
```

Grid search identified these hyperparameters as being the best combination for optimizing the F_1 score on the model.

2. Use a confusion matrix to evaluate the optimized model.

- Scroll down and select the code cell beneath the **Evaluate the optimized model** title, then select **Run**.
- Examine the output.



3. Compared to the initial model, how has the confusion matrix changed for the optimized model?

4. Continue evaluating the optimized model.

- Scroll down and select the next code listing cell.

```
In [ ]: 1 model_scores(y_val, search_predict)
```

- Select **Run**.
- Examine the output.

```
Accuracy: 83%
Precision: 76%
Recall: 71%
F1: 73%
```

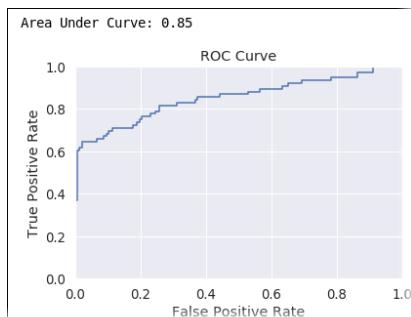
As you can see, all of the scores for this optimized model have improved. In particular, the recall and F_1 have improved the most, with the former increasing by 17%, and the latter increasing by 11%.

- Scroll down and select the next code listing cell.

```
In [ ]: 1 search_predict_proba = search.predict_proba(X_val)
          2
          3 roc(y_val, search_predict_proba[:, 1])
```

- Select **Run**.

- f) Examine the output.

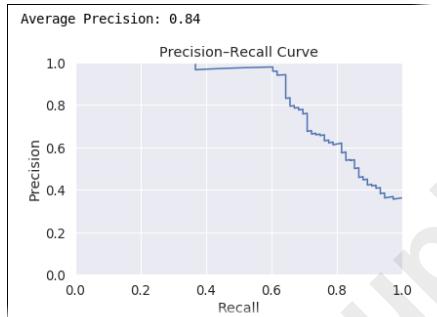


The ROC curve itself has changed, but the AUC is essentially the same as it was in the unoptimized model. You typically won't be able to improve every metric, which is why it's important to focus on the metrics you think are most relevant.

- g) Scroll down and select the next code listing cell.

```
In [ ]: 1 prc(y_val, search_predict_proba[:, 1])
```

- h) Select Run.
i) Examine the output.



Once again, the curve has changed. However, the average precision statistic appears to have improved by about 8%.

5. Most of the metrics have improved in the model that was optimized with grid search. However, this doesn't mean that you've automatically built the best possible classifier for this particular dataset.

What else might you do to continue improving your classification performance for this dataset?

6. Compare the logistic regression models' predictions on the test data.

Licensed For Use Only By: Abdulwahab Alweban dev.abdulwahab@gmail.com Jan 8 202

- Scroll down and select the code cell beneath the **Compare the logistic regression models' predictions on the test data** title, then select **Run**.
- Verify that the **test.csv** file was loaded.
- Scroll down and select the next code listing cell.

```
In [ ]: 1 # Show example predictions with the test data using the initial (unoptimized) model.
2 results_log_reg = X_test_raw.copy()
3 results_log_reg['PredictedSurvival'] = log_reg.predict(X_test)
4 results_log_reg['ProbPerished'] = np.round(log_reg.predict_proba(X_test)[:, 0] * 100, 2)
5 results_log_reg['ProbSurvived'] = np.round(log_reg.predict_proba(X_test)[:, 1] * 100, 2)
6 results_log_reg.head(10)
```

- Select **Run**, then verify that the initial model made predictions for the first ten test examples.
- Scroll down and select the next code listing cell.

```
In [ ]: 1 # Show example predictions with the test data using the model optimized with grid search.
2 results_log_reg = X_test_raw.copy()
3 results_log_reg['PredictedSurvival'] = search.predict(X_test)
4 results_log_reg['ProbPerished'] = np.round(search.predict_proba(X_test)[:, 0] * 100, 2)
5 results_log_reg['ProbSurvived'] = np.round(search.predict_proba(X_test)[:, 1] * 100, 2)
6 results_log_reg.head(10)
```

- Select **Run**, then verify that the optimized model made predictions for the first ten test examples.
- Compare both results and verify that the optimized model made different predictions on the same test set.

7. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **Classification-Titanic-Tuned** tab in Firefox, but keep a tab open to the file hierarchy in Jupyter Notebook.

Summary

In this lesson, you trained several different models to classify new data examples. Rather than stopping there, you improved your models by first evaluating their performance, then using various tuning techniques to make that performance better. By taking this approach, you can feel much more confident about your machine learning models—classification or otherwise. And, more importantly, you'll get better results for the problem at hand.

What type of data in your organization do you think might be conducive to classification?

Given the datasets you're likely to use and classification problems you're trying to solve, what evaluation metrics do you think you'll find most useful when tuning a classification model?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Do Not Duplicate Or Distribute

7

Building Clustering Models

Lesson Time: 2 hours, 30 minutes

Lesson Introduction

You've built models to tackle linear regression problems and classification problems. One of the other major machine learning tasks that you might want to engage in is clustering, a form of unsupervised learning. In this lesson, you'll see how a machine learning model can help you identify useful patterns even when the data you have to work with isn't labeled.

Lesson Objectives

In this lesson, you will:

- Build k -means clustering models.
- Build hierarchical clustering models.

TOPIC A

Build k-Means Clustering Models

The first clustering algorithm you'll work with is k -means clustering, one of the most popular. This algorithm is applicable to many common types of clustering problems and datasets.

k -Means Clustering

k -means clustering is an algorithm for unsupervised machine learning that groups like data examples together for the purpose of revealing patterns in the data. It does this by defining a set of k groups (clusters). Each data example is placed within the cluster whose center (called a centroid) is the closest to that data example. Closeness can be defined by a distance metric that is chosen during training. So, you end up with clusters of data that exhibit statistical similarity, as visualized in the following figure.

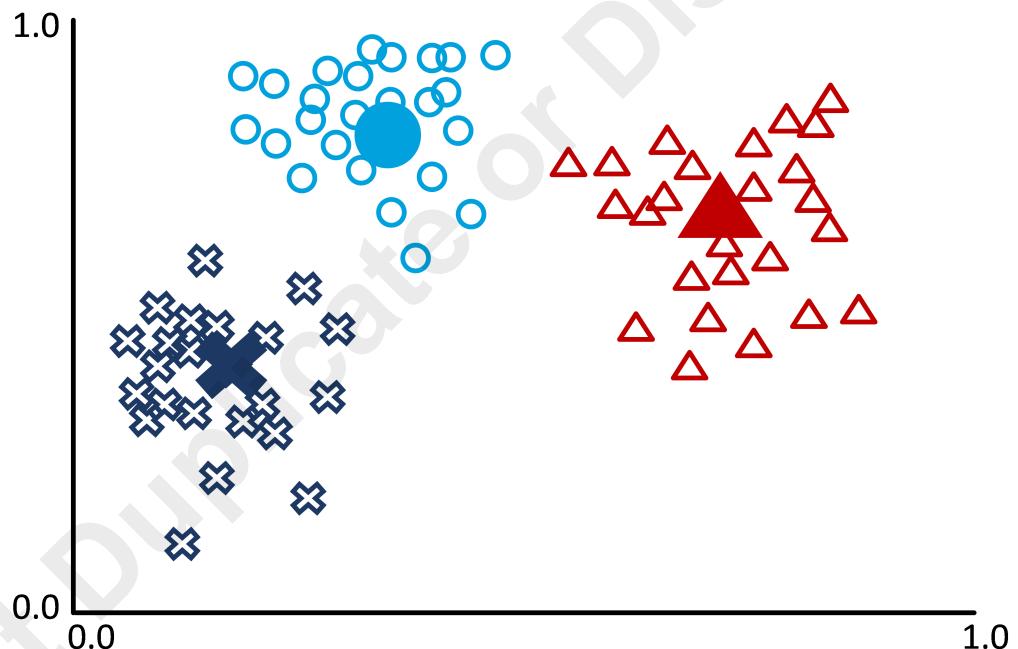


Figure 7-1: Three clusters of data, in which the large, filled-in shapes represent the centroids.

Once the algorithm assigns data examples to clusters, it recomputes each centroid by calculating the mean of all data examples in the centroid's cluster. This continues until the data examples no longer change clusters (i.e., the k -means converge), or until a specified number of iterations is met.

k -means clustering is often used in the field of customer segmentation to group customers based on similar characteristics. It has also found use in categorizing sensor data from IoT devices, like images and video.

k -Means Clustering vs. k -Nearest Neighbor

The k -means clustering algorithm described here is not to be confused with the k -nearest neighbor (k -NN) described earlier. The former is used to solve unsupervised problems by clustering data into groups, and an analyst can infer some meaning from those groups. The latter is used to solve

supervised problems, typically by classifying data examples based on their feature similarities with other examples.

Global vs. Local Optimization

The ultimate goal of k -means clustering is the minimization of cost, much like other machine learning algorithms. The global cost function $J(\theta)$ can be written as follows:

$$J(\theta) = \sum_{i=1}^n \min_j (\text{dist}(x_i, c_j))$$

Where:

- n is the total number of examples.
- x_i is the i^{th} data example.
- c_j is the j^{th} centroid.

So, going from right to left, the distance between a data example and a centroid is taken (**dist**). Then, the centroid with the minimum (nearest) distance to the example is taken (**min**). Lastly, the sum of all the nearest distances is calculated. Ultimately, this helps to find the centroids that minimize this total distance. However, it is usually not feasible to apply this optimization globally. If you tried every possible assignment of n data examples to every cluster, you'd end up with many, many possible combinations—even if your sample size and number of clusters is low. For example, with an n of 25 and 4 as the number of clusters, there are roughly 47 *trillion* possible assignments.

This is why k -means clustering is an iterative algorithm that requires local optimization. The process is as follows:

1. It begins by taking the number of desired clusters, then it randomly assigns centroids for each cluster.
2. Next, it assigns each data example to whatever centroid is currently closest. This is effectively the same as minimizing the cost of these assignments.
3. Then, the algorithm moves each centroid so that it is in the center of the data examples that were assigned to it. This is effectively the same as minimizing the cost of the centroids.
4. The process repeats until convergence or until an iteration maximum is met.

So, this iterative cost minimizing scheme is a more efficient approach to optimization. However, it doesn't always achieve the perfect global optimization, especially if the initial randomly selected centroids were placed in sub-optimal locations. You can re-initialize the k -means algorithm with different randomly chosen centroids to overcome this.

k Determination

In k -means clustering, the primary challenge is determining k (number of clusters). There are several evaluation metrics you can use to help you determine the optimal k . Before you use these metrics, however, you should start by assessing the problem you're trying to solve. Based on your knowledge of the domain, you may be able to place useful constraints on the data. For example, assume you have an unlabeled dataset of different fruits. In this case, you know that each example is one of three possible fruits, and that each feature describes properties of a fruit like its shape, size, color, etc. So, because of your domain knowledge, you know that you will only accept three clusters—no more, no less. This will prevent you from having to choose the number of clusters yourself.

Elbow Point

One method of determining k is to calculate the mean distance between each data example and its associated centroid. As k increases, the mean distance necessarily decreases. However, at some point, increasing k becomes pointless and doesn't reduce the mean distance in any significant way. The point at which the mean distance no longer decreases in a significant way is called the **elbow point**, and it's usually a good indicator of what k should be. Consider the following figure, in which k is plotted against the mean distance.

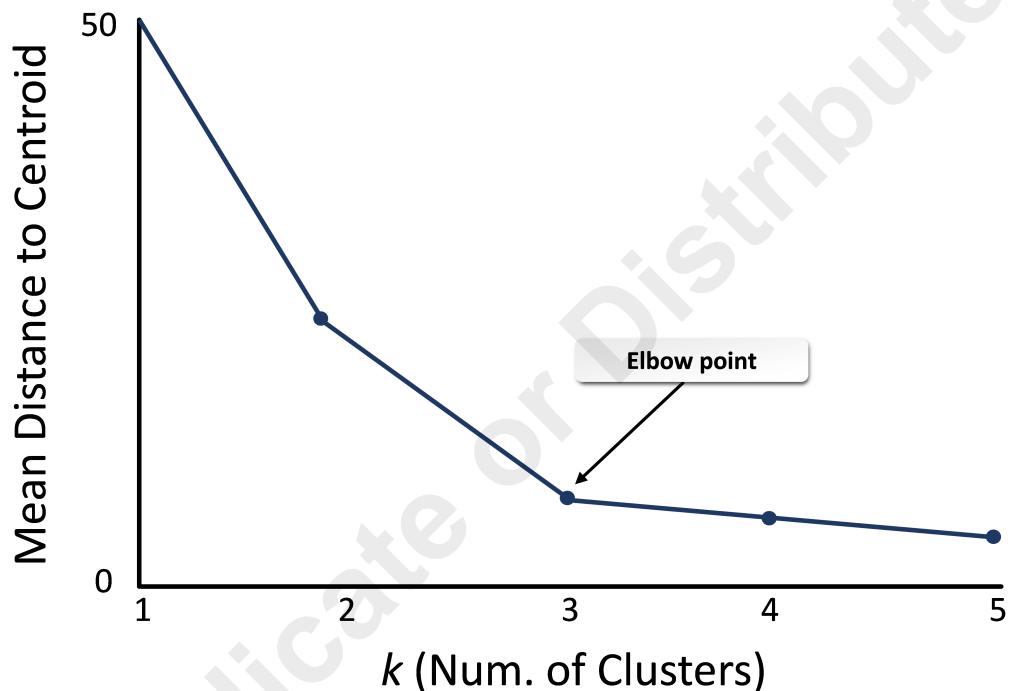


Figure 7-2: The elbow point indicates that k should be 3.

Cluster Sum of Squares

There are two ideal properties of a clustering model: the compactness of the cluster, and how well separated it is from other clusters. The better these properties are for a given problem, the better the outcome. To evaluate these properties, you can plug your data values into two separate equations.

The **within-cluster sum of squares (WCSS)** measures the compactness of clusters. It does this by measuring the distance between a given data point and its cluster centroid, then repeating this process for all other points in the cluster and calculating the mean distance. Smaller values of WCSS indicate that data points within a cluster are closer together, meaning that smaller values are ideal.

The **between-cluster sum of squares (BCSS)** measures the separation between clusters. It does this by measuring the distance between a given centroid and all other centroids, then repeating this process for all other centroids and calculating the sum. Larger BCSS values indicate clusters that are farther apart, meaning that larger values are ideal.

Silhouette Analysis

A **silhouette analysis** helps you evaluate the ideals of compactness and good separation in one mathematical equation. This equation calculates how well a particular data example fits within a

cluster as compared to its neighboring clusters. Each example is assigned a value, called a silhouette coefficient or silhouette score, between -1 and 1 . This value indicates the following:

- A high coefficient means that the example is far away from neighboring clusters, and therefore fits well within its own cluster.
- A coefficient close to 0 means the example is near the decision boundary between clusters.
- A coefficient in the negative means the example is closer to a neighboring cluster than its own, and is therefore likely to have been placed in the wrong cluster.

The ideal is to have your data examples as close to 1 as possible. Low coefficients indicate that you may need to adjust your k value. Typically, you'd calculate the coefficient for each data example and then group each into its respective cluster. You'd also calculate the average coefficient of each cluster, as well as the average of the entire model given k . Then, you'd do the same calculations for different k values and compare the average coefficients of the different k values. Ultimately, you'd select the k value that leads to the highest silhouette coefficient.

The values derived from a silhouette analysis are typically plotted on a graph. For each cluster, the data example with the highest coefficient is on top, with the lowest coefficient at the bottom. This forms a silhouette-like shape for each cluster. The cluster with the highest average coefficient is often placed on top, and the rest of clusters are placed in descending order.

In the following figure, the silhouette analyses of three different k values are plotted. For $k = 2$, the average silhouette coefficient is around $.578$; for $k = 3$, the average coefficient is around $.732$. For $k = 4$, the average silhouette coefficient is around $.492$. Since the average coefficient of $k = 3$ is highest, this suggests a better fit. Also note that the thickness of each plotted group in $k = 3$ is more evenly distributed than $k = 2$, as the model is no longer fitting many examples into a large group. This even distribution does not always lead to a better coefficient, however.

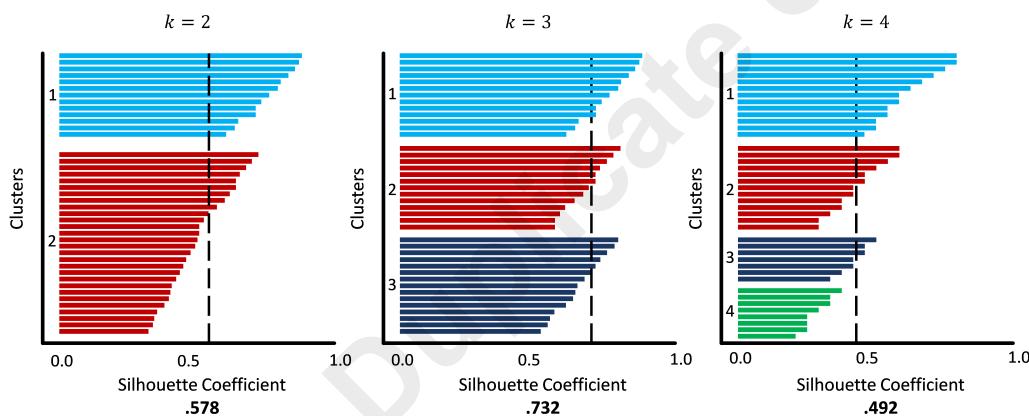


Figure 7-3: A silhouette analysis of three different k values. The dashed line indicates the average coefficient for that model.

Additional Cluster Analysis Methods

Another way to evaluate the performance of a clustering algorithm is the [Dunn index](#), which is calculated as follows:

1. The smallest distance between two data points *not* in the same cluster is calculated.
2. The largest distance between two data points *within* the same cluster is calculated.
3. The ratio of these values is calculated.

Unlike with the silhouette coefficient, the Dunn index will always be above 0 and has no maximum value. However, just like the silhouette coefficient, the higher the Dunn index, the better. A high Dunn index indicates good cluster compactness and separation.

The **Davies–Bouldin index** calculates the average ratio of the within-cluster distance and the between-cluster distance for each cluster as compared to its most similar cluster. Clusters that are both well-separated and compact will receive a better score. Like the Dunn index, the Davies–Bouldin index score starts at 0 and has no maximum value. However, for Davies–Bouldin, lower values are better.

Guidelines for Building a *k*-Means Clustering Model

Follow these guidelines when you are building *k*-means clustering models.



Note: All of the Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Build a *k*-Means Clustering Model

When building a *k*-means clustering model:

- Use *k*-means clustering for unsupervised learning to find groups of data points that share similarities.
- Apply *k*-means clustering to data that is spherical or overlapping in nature.
- Consider applying domain knowledge in determining *k*—you may know exactly or approximately how many clusters you need or is best for the problem.
- When domain knowledge isn't enough, use statistical analysis methods to help determine *k*.
- Use an elbow point graph to identify the value of *k* when the mean distance between a point and its centroid is no longer decreasing significantly.
- Use within- and between-cluster sum of squares (WCSS and BCSS) to evaluate the compactness of a cluster and how well separated it is for other clusters, respectively.
- Use a silhouette analysis to evaluate both compactness and separation at once.
- In silhouette analysis, consider choosing the *k* with the highest silhouette coefficient.
- Consider using additional cluster analysis techniques like the Dunn index and the Davies–Bouldin index.
- Consider that there is no objectively "correct" number of clusters for many unsupervised problems.
- Consider that different analysis methods may give you different results for an optimal *k* value, and that one is not necessarily more "correct" than another.

Use Python for *k*-Means Clustering

The scikit-learn `KMeans()` class enables you to construct a machine learning model using a *k*-means clustering algorithm. The following are some of the objects and functions you can use to build such a model.

- `model = sklearn.cluster.KMeans(n_clusters = 3)` —This constructs a model object that uses the *k*-means clustering algorithm. In this instance, the number of clusters specified is 3.
- `model.fit(X_train)` —Fit a set of training data to the model.
- `model.fit_predict(X_train)` —Fit a set of training data to the model, and return what cluster each example belongs to.
- `model.predict(X_test)` —Predict the cluster that each data example in a test set belongs to.
- `sklearn.metrics.silhouette_score(X_train, clusters)` —Return the mean silhouette score for all examples. Note that `clusters` is an object created by calling `fit_predict()` on the model.
- `sklearn.metrics.silhouette_samples(X_train, clusters)` —Return the silhouette score for all individual examples.

The scikit-learn library doesn't have a built-in method for generating an elbow analysis, but the third-party Yellowbrick library does:

- `visualizer = yellowbrick.cluster.KElbowVisualizer(model, k = (1, 10))` —This generates a visualization object by passing in a scikit-learn model and a range of k values to show.
- `visualizer.fit(X_train)` —Fit a set of training data to the visualizer.

Do Not Duplicate or Distribute

ACTIVITY 7-1

Building a k -Means Clustering Model

Data Files

/home/student/CAIP/Clustering/Clustering-Housing.ipynb

/home/student/CAIP/Clustering/housing_data/kc_house_data.csv

Before You Begin

Jupyter Notebook is open.

Scenario

A real estate company that's been working with the King County dataset has prospective buyers. Each buyer identifies which houses they are most interested in. However, due to certain circumstances—like a house being pulled from the market—they may not be able to get their top choices. So, the real estate agent needs to be able to recommend to them one or more houses that are similar to their top choices. Since there are so many factors that go into a buyer's choice, what is defined as "similar" is not easy to determine.

Earlier, you were able to train a machine learning model to predict the prices of houses. This was a supervised problem, because there was a target variable/label involved (price). However, datasets like these can be applicable to many problems, not just one. So, in this case, you want to be able to group, or cluster, houses together based on the similarity of their features. Because there is no label to predict, this presents an unsupervised problem—one you'll use k -means clustering to address.

1. Import the relevant libraries and load the dataset.

- From Jupyter Notebook, select **CAIP/Clustering/Clustering-Housing.ipynb** to open it.
- Select the code cell beneath the **Import software libraries** title, and then select **Run**.
- Select the code cell beneath the **Load the dataset** title, then select **Run**.
- Verify that **kc_house_data.csv** was loaded with 21,613 records and 21 columns.

Data files in this project: ['kc_house_data.csv'] Loaded 21613 records from ./housing_data/kc_house_data.csv.																
Dataset Rows and Columns: (21613, 21)																
id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	y		
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0		
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400		
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0		
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910		
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0		

5 rows × 21 columns

2. Engineer features as needed.

- Select the code cell beneath the **Engineer features as needed** title, then select **Run**.

- b) Scroll the table to the right as needed to verify that the `price_per_sqft` feature was created from `price` and `sqft_living`.

Engineer features as needed

# Create a price per square foot feature based on 'price' and 'sqft_living'.														
sqft_lot	floors	waterfront	view	... sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15	price_per_sqft	
5650	1.0	0	0	...	1180	0	1955	0	98178	47.5112	-122.257	1340	5650	188.050847
7242	2.0	0	0	...	2170	400	1951	1991	98125	47.7210	-122.319	1690	7639	209.338521
10000	1.0	0	0	...	770	0	1933	0	98028	47.7379	-122.233	2720	8062	233.766234
5000	1.0	0	0	...	1050	910	1965	0	98136	47.5208	-122.393	1360	5000	308.163265
8080	1.0	0	0	...	1680	0	1987	0	98074	47.6168	-122.045	1800	7503	303.571426

3. Use a k -means model to label every row in the dataset.

- a) Scroll down and view the cell titled **Use a k -means model to label every row in the dataset**, and examine the code listing below it.

Use a k -means model to label every row in the dataset

```
In [ ]: 1 from sklearn.cluster import KMeans
2
3 # Produce cluster labels.
4 def get_cluster_labels(cluster_count, X):
5
6     kmeans = KMeans(n_clusters = cluster_count,
7                      init = 'k-means++',
8                      random_state = 42)
9
10    kmeans.fit(X)
11    cluster_labels = kmeans.predict(X)
12
13    # Return the original DataFrame with the labels appended as a new column.
14    return cluster_labels
15
16 print('The function to produce cluster labels using a k-means model has been defined.')
```

- This function, when called, will create a k -means clustering model object using the number of clusters specified in the call.
- The `init` parameter of the `KMeans()` class determines how the model initializes centroids. The `k-means++` method is an improvement over the traditional k -means method, as it initializes centroids that are far away from each other, leading to better clustering.
- The function will return a `DataFrame` of the training dataset with a new column of cluster labels added.
- The `print` statement in line 16 shows that the function has been defined. (The function will be called later.)

- b) Select the cell that contains the code listing, then select **Run**.

The function to produce cluster labels using a k-means model has been defined.

4. Generate the cluster labels and attach them to the original dataset.

- a) Select the code cell beneath the **Generate the cluster labels and attach them to the original dataset** title, then select **Run**.

- b) Scroll the table to the right and examine the output.

Generate the cluster labels and attach them to the original dataset

```

1 # Initially cluster by latitude and longitude, just to verify we're using k-means model correctly.
2 feature_set_1 = ['lat', 'long']
3 X = housing_data[feature_set_1]
4
5 # Generate cluster labels for the housing data, assuming 4 clusters.
6 cluster_labels = get_cluster_labels(4, X)
7
8 # Append the cluster labels to a new column in the original dataset.
9 labeled_houses = housing_data.assign(c_label = cluster_labels)
10
11 # Show a preview of rows in the dataset with cluster labels added.
12 labeled_houses.head()

```

vin	sqft_lot	floors	waterfront	view	...	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15	price_per_sqft	c_label
.180	5650	1.0	0	0	...	0	1955	0	98178	47.5112	-122.257	1340	5650	188.050847	1
.570	7242	2.0	0	0	...	400	1951	1991	98125	47.7210	-122.319	1690	7639	209.338521	3
.770	10000	1.0	0	0	...	0	1933	0	98028	47.7379	-122.233	2720	8062	233.766234	3
.960	5000	1.0	0	0	...	910	1965	0	98136	47.5208	-122.393	1360	5000	308.163265	1
.680	8080	1.0	0	0	...	0	1987	0	98074	47.6168	-122.045	1800	7503	303.571429	2

- This code just uses two features—latitude and longitude of the houses—to perform the clustering on. This is just a preliminary step to see how the clustering works.
- On line 6, the code uses an arbitrary number of clusters (4) to call the clustering function with.
- The output shows which cluster each data example is placed in: 0, 1, 2, or 3.

5. Observe how many houses were distributed to each cluster.

- a) Select the code cell beneath the **Observe how many houses were distributed to each cluster** title, then select **Run**.
 b) Examine the output.

Observe how many houses were distributed to each cluster

```

1 for i in range(4):
2     num_in_clust = len(labeled_houses[labeled_houses['c_label'] == i])
3     print("Number of houses in cluster {} = {}".format(i, num_in_clust))

```

Number of houses in cluster 0 = 4273
 Number of houses in cluster 1 = 5104
 Number of houses in cluster 2 = 4810
 Number of houses in cluster 3 = 7426

Each cluster contains several thousand houses, with cluster 3 having the most.

6. Show clusters of homes on the map by location.

- a) Scroll down and view the cell titled **Show clusters of homes on the map by location**. Examine lines 1 through 22 in the code listing below it.

```

1 from folium.plugins import HeatMap
2
3 def show_on_map(labeled_house_dataset, map_title):
4
5     # To avoid overwhelming the visualization tool, we'll only plot every nth house.
6     n_homes = 20
7     mapping_set = labeled_house_dataset.sort_values(by = ['price'], ascending = False)[::n_homes]
8
9     # Descriptions of the building grades used in King County.
10    bldg_grades = ['Unknown', 'Cabin', 'Substandard', 'Poor', 'Low', 'Fair',
11                   'Low Average', 'Average', 'Good', 'Better',
12                   'Very Good', 'Excellent', 'Luxury', 'Mansion', 'Exceptional Properties']
13
14    # Generate the base map, centering on King County.
15    base_map = folium.Map(location = [47.5300, -122.2000],
16                          control_scale = True,
17                          max_zoom = 20,
18                          zoom_start = 10,
19                          zoom_control = True)
20
21    # Get price of most expensive house.
22    max_price = labeled_house_dataset.loc[labeled_house_dataset['price'].idxmax()]['price']
23

```

- The `show_on_map` function plots every 20th data point on a geographic map of the area, along with the clusters of that data.
- Lines 15 through 19 use the Folium library to display the geographic map.
- Line 22 defines the highest price of the houses in the dataset to use in scaling the rest of the points.

- b) Examine lines 24 through 58.

```

4.2
24
25     # Plot homes by price.
26     for index, row in mapping_set.iterrows():
27
28         # Add popup text. Click each point to show details.
29         popup_text = '<br>'.join(['KingCountyHousingSalesData',
30                               'Price: {:.0f}',
31                               'Cluster: {:.0f}',
32                               'Bedrooms: {:.0f}',
33                               'Bathrooms: {:.0f}',
34                               'Sqft Living: {:.0f}',
35                               'Location: [{:.3f}, {:.3f}]'])
36
37         popup_text = popup_text.format(row['price'],
38                                       row['c_label'],
39                                       row['bedrooms'],
40                                       row['bathrooms'],
41                                       row['sqft living'],
42                                       row['lat'],
43                                       row['long'])
44
45         cluster_value = int(row['c_label'])
46         scaling_value = (row['price'] / max_price)      # 1.0 for highest price.
47
48         folium.CircleMarker([row['lat'], row['long']],
49                             radius = 25 * scaling_value,
50                             weight = 3,
51                             fill = True,
52                             fill_color = '#000000',
53                             color = '#0000FF',
54                             fill_opacity = 0.8,
55                             opacity = 0.8,
56                             popup = popup_text).add_to(base_map)
57
58     # Heat map around each cluster.
59     cluster_max = labeled_house_dataset.loc[labeled_house_dataset['c_label'].idxmax()]['c_label'] + 1

```

- Line 25 begins a `for` loop that will iterate through every 20th row of the dataset, and will be used to plot the data for the houses.
- Lines 28 through 55 create the markers and popup text for the data points.

- c) Examine lines 60 through 80.

```

60   for cluster_num in range(0, cluster_max):
61
62     houses_in_same_cluster = labeled_house_dataset.loc[labeled_house_dataset['c_label'] == cluster_num]
63     house_locations = houses_in_same_cluster[['lat', 'long']].copy()
64
65     mean_price = houses_in_same_cluster['price_per_sqft'].mean()
66
67     cluster_name = f'Cluster {cluster_num} (mean ${mean_price:.0f} per sqft)'
68     feature_group = folium.FeatureGroup(name = cluster_name)
69     feature_group.add_child(HeatMap(house_locations, radius = 10))
70     base_map.add_child(feature_group)
71
72     folium.map.LayerControl('bottomright', collapsed = False).add_to(base_map)
73
74     # Add title to map.
75     map_html = f'<div style="position:fixed; top:10px; left:60px; z-index:9999"><b>{map_title}</b></div>'
76     base_map.get_root().html.add_child(folium.Element(map_html))
77
78   return base_map
79
80 print('The function to show the map has been defined.')

```

- The `for` loop that begins on line 60 iterates through each cluster.
 - Lines 62 through 70 create the heat map and display it along the dimensions of latitude and longitude, using the mean price per square feet of each cluster as a summary statistic.
 - The entire function returns the map when it is called.
 - The print statement in line 80 shows that the function has been defined.
- d) Select the cell that contains the code listing, then select **Run**.

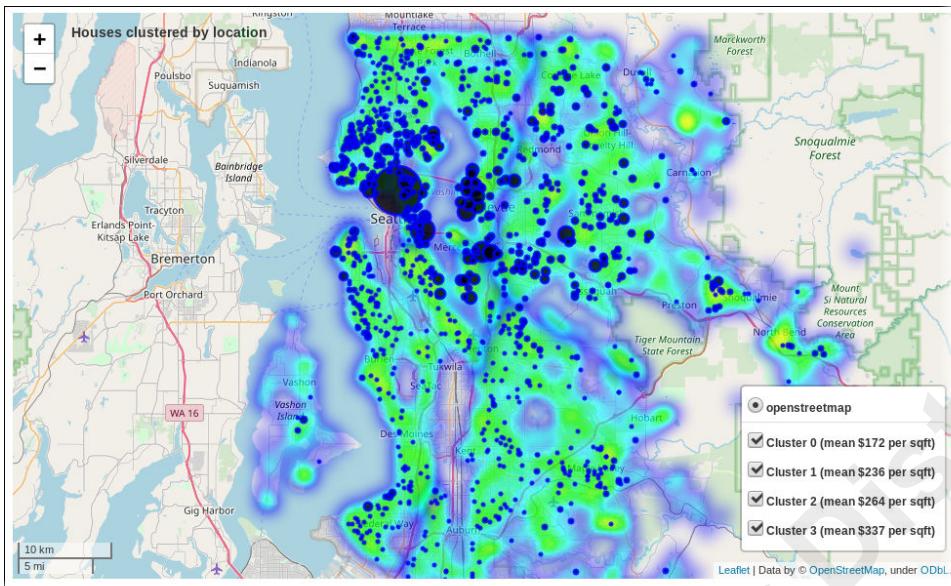
The function to show the map has been defined.

- e) Scroll down and select the next code listing cell.

```
In [ ]: 1 # View the results on the map.
          2 show_on_map(labeled_houses, 'Houses clustered by location')
```

- f) Select **Run**.

- g) Examine the output.



There are four clusters, each grouped by house location (latitude and longitude). You can check and uncheck the different clusters to display and hide those cluster heat maps, respectively.

7. How did the clusters form with regard to location?

8. Cluster by price per square foot.

- a) Scroll down and view the cell titled **Cluster by price per square foot**, and examine the code listing below it.

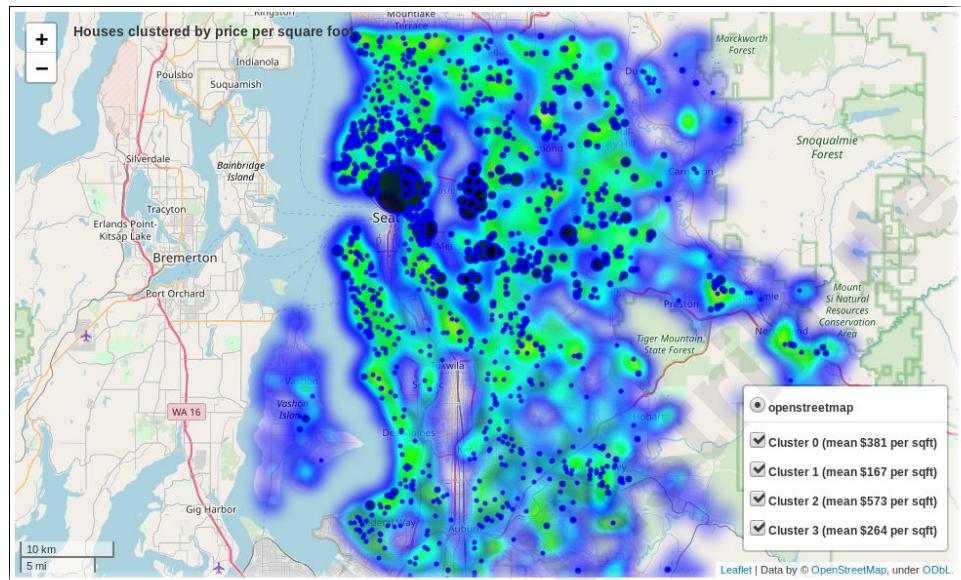
Cluster by price per square foot

```
In [ ]: 1 # Cluster homes by the price per square foot.
2 feature_set_2 = ['price_per_sqft']
3 X = housing_data[feature_set_2]
4
5 # Generate cluster labels for the housing data, assuming 4 clusters.
6 cluster_labels = get_cluster_labels(4, X)
7
8 # Append the cluster labels to a new column in the original dataset.
9 labeled_houses = housing_data.assign(c_label = cluster_labels)
10
11 # Show on the map.
12 show_on_map(labeled_houses, 'Houses clustered by price per square foot')
```

This is essentially the same code as before, except it's using `price_per_sqft` rather than `lat` and `long` (latitude and longitude).

- b) Select the cell that contains the code listing, then select **Run**.

- c) Examine the output.



9. How did the clusters form with regard to price per square foot?

10. Prepare to cluster by multiple features of interest to customers.

- a) Scroll down and view the cell titled **Prepare to cluster by multiple features of interest to customers**, and examine the code listing below it.

Prepare to cluster by multiple features of interest to customers

```
In [ ]: 1 # Specify the new dataset.
2 final_feature_set = ['sqft_living', 'bathrooms', 'bedrooms', 'grade', 'view', 'waterfront']
3 X = housing_data[final_feature_set]
4
5 print('A dataset containing features of interest to customers has been defined.)
```

Now that you're confident the clustering model is working, you'll generate the model using all of the features that are relevant to the prospective buyers' interests. These features are:

- The square footage of the house.
- The number of bathrooms.
- The number of bedrooms.
- The "grade" of the house.
- The quality of the house's view.
- Whether the house has a view to a waterfront.

- b) Select the cell that contains the code listing, then select **Run**.

A dataset containing features of interest to customers has been defined.

11. Use the elbow method to determine the optimal number of clusters.

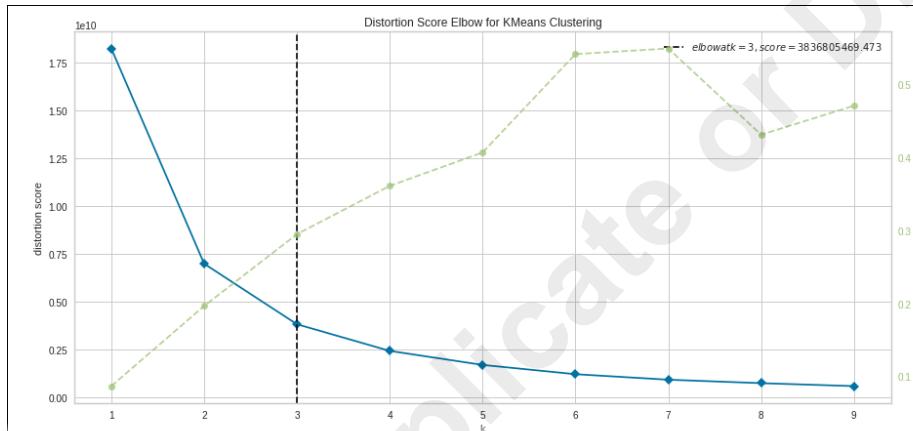
- Scroll down and view the cell titled **Use the elbow method to determine the optimal number of clusters**, and examine the code listing below it.

Use the elbow method to determine the optimal number of clusters

```
In [ ]: 1 from yellowbrick.cluster import KElbowVisualizer
2
3 # Use the elbow method to find the optimal number of clusters.
4 plt.rcParams["figure.figsize"] = (15, 7)
5
6 visualizer = KElbowVisualizer(KMeans(init = 'k-means++', random_state = 42), k = (1, 10))
7 visualizer.fit(X)
8 visualizer.poof();
```

Rather than supply an arbitrary value for the number of clusters, you should do some analysis to determine what the optimal number of clusters is. The elbow method, generated here by a visualization library called Yellowbrick, is one method of doing so.

- Select the cell that contains the code listing, then select **Run**.
- Examine the output.



This plot automatically locates the elbow for you: 3. As you can see, this is the point where adding more clusters gives diminishing performance returns. The dashed green line indicates the time taken to train the model at each cluster value.

12. Use silhouette analysis to determine the optimal number of clusters.

- a) Scroll down and view the cell titled **Use silhouette analysis to determine the optimal number of clusters**, and examine lines 1 through 33 in the code listing below it.

```

1 from sklearn.metrics import silhouette_score
2 from sklearn.metrics import silhouette_samples
3
4 # The number of clusters to try out.
5 range_n_clusters = [2, 3, 4, 5]
6
7 high_score = 0
8 optimum_n_clusters = 0
9
10 for n in range_n_clusters:
11
12     # Create k-means model and generate labels from the dataset.
13     Kmeans = KMeans(n_clusters = n, random_state = 10)
14     cluster_labels = Kmeans.fit_predict(X)
15     silhouette_avg = silhouette_score(X, cluster_labels)
16
17     print('\nWith {} clusters.'.format(n))
18     print(' - Average silhouette score:', silhouette_avg)
19
20     # Note the high score.
21     if silhouette_avg > high_score:
22         high_score = silhouette_avg
23         optimum_n_clusters = n
24
25     # Compute the silhouette scores for each sample.
26     sample_silhouette_values = silhouette_samples(X, cluster_labels)
27
28     # Prepare to plot charts side by side.
29     fig, (ax1, ax2) = plt.subplots(1, 2)
30     fig.set_size_inches(17, 5)
31     ax1.set_xlim([-0.1, 1])
32     ax1.set_ylim([0, len(X) + (n + 1) * 10])
33     y_lower = 10

```

- Line 5 passes in the arbitrary cluster values to try.
- Line 10 begins a `for` loop that iterates through each cluster value.
- Lines 13 through 15 create the k -means clustering model and calculate the highest mean silhouette coefficient of all samples.
- Lines 21 through 23 keep track of the highest mean coefficient of all samples.
- Line 26 computes the silhouette coefficient for each sample.
- Lines 29 through 33 begin plotting two charts side by side: the silhouette plot on the left, and a scatter plot of the clusters on the right.

- b) Examine lines 35 through 62.

```

35     # LEFT SIDE: SILHOUETTE PLOTS
36     for i in range(n):
37
38         # Plot silhouette for one cluster at a time.
39         ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]
40         ith_cluster_silhouette_values.sort()
41         size_cluster_i = ith_cluster_silhouette_values.shape[0]
42         y_upper = y_lower + size_cluster_i
43         color = cm.nipy_spectral(float(i) / n)
44         ax1.fill_betweenx(np.arange(y_lower, y_upper),
45                           0,
46                           ith_cluster_silhouette_values,
47                           facecolor = color,
48                           edgecolor = color,
49                           alpha = 1)
50
51         ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
52         y_lower = y_upper + 10 # Use 10 for the 0 samples.
53
54         # Log how many values were in this cluster.
55         print(' - Cluster {} includes {} values.'.format(i, size_cluster_i))
56
57         ax1.set_title('Silhouette Plots')
58         ax1.set_xlabel('Silhouette Coefficient Values')
59         ax1.set_ylabel('Cluster Label')
60         ax1.axvline(x = silhouette_avg, color = 'red', linestyle = '--')
61         ax1.set_yticks([])
62         ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

```

- Line 36 begins a `for` loop that iterates through each cluster value to plot the silhouette graphs on the left side of the screen.
- Lines 39 through 62 plot the silhouette for each cluster.

- c) Examine lines 64 through 92.

```

63      # RIGHT SIDE: SCATTER PLOTS
64
65      # Caption the various clusters.
66      colors = cm.nipy_spectral(cluster_labels.astype(float) / n)
67      centers = kmeans.cluster_centers_
68
69      # Plot the first two features.
70      ax2.scatter(X['sqft_living'],
71                  X['bathrooms'],
72                  marker = 'o',
73                  alpha = 0.7,
74                  s = 50,
75                  color = colors,
76                  edgecolor = 'black');
77
78      # Show a box at the center of each cluster, with the cluster number inside it.
79      ax2.scatter(centers[:, 0], centers[:, 1], marker = 's', c = 'white', alpha = 1.0, s = 200, edgecolor = 'black')
80      for i, c in enumerate(centers):
81          ax2.scatter(c[0], c[1], marker = '$%d$' % i, alpha = 1.0, s = 50, edgecolor = 'black')
82
83      # Axis labels.
84      ax2.set_title('Clustered Data')
85      ax2.set_xlabel(X.columns[0])
86      ax2.set_ylabel(X.columns[1])
87      plt.suptitle((f'Number of clusters = {n}'), fontsize = 16, fontweight = 'bold')
88
89      plt.show()
90
91      print(f'The highest score ({high_score}) was obtained using {optimum_n_clusters} clusters.')
92

```

- Lines 67 through 88 continue the `for` loop, this time plotting the cluster graph that will appear on the right side of the screen for each corresponding silhouette graph.
 - Lines 71 and 72 will use the square foot living space and number of bathrooms as the axes for the scatter plots.
 - Line 92 prints the highest silhouette coefficient and the number of clusters that obtained it. This can be used to determine the optimal number of clusters.
- d) Select the cell that contains the code listing, then select **Run**.
- e) Examine the output.

```

With 2 clusters:
- Average silhouette score: 0.5956028486082886
- Cluster 0 includes 15029 values.
- Cluster 1 includes 6584 values.

With 3 clusters:
- Average silhouette score: 0.55921974259051
- Cluster 0 includes 10918 values.
- Cluster 1 includes 8374 values.
- Cluster 2 includes 2321 values.

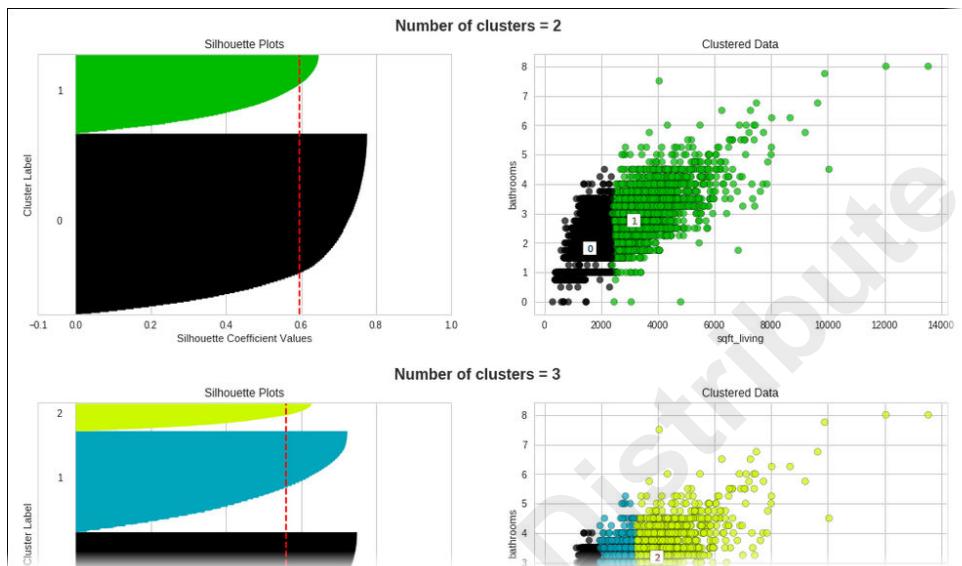
With 4 clusters:
- Average silhouette score: 0.5414443565247314
- Cluster 0 includes 7882 values.
- Cluster 1 includes 4552 values.
- Cluster 2 includes 1067 values.
- Cluster 3 includes 8112 values.

With 5 clusters:
- Average silhouette score: 0.5321936319368658
- Cluster 0 includes 7281 values.
- Cluster 1 includes 2447 values.
- Cluster 2 includes 388 values.
- Cluster 3 includes 6331 values.
- Cluster 4 includes 5166 values.

```

The highest score appears to be for the model with 2 clusters.

f) Examine the silhouette plots.

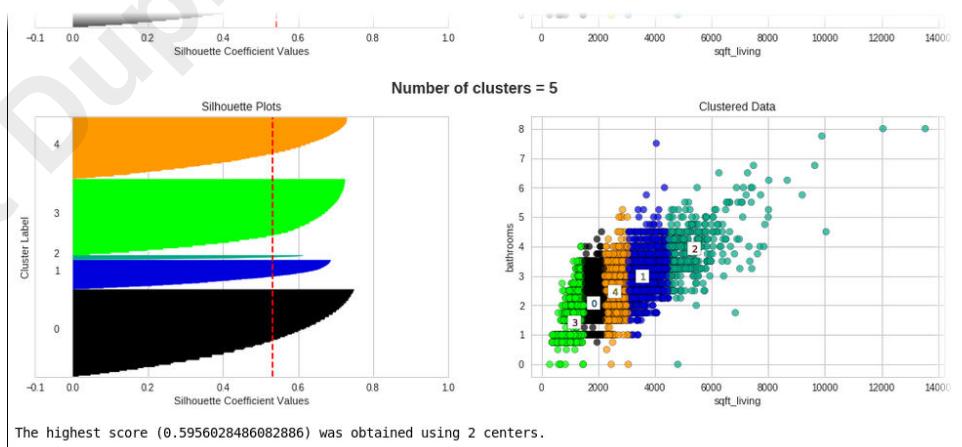


- The silhouette coefficient for each cluster value appears as a dashed red line on the silhouette plot.
- The scatter plot represents the corresponding clusters as they appear for square foot living space compared to number of bathrooms.



Note: Although it might look like cluster 1 in the first scatter plot has more examples than cluster 0, cluster 0 actually includes more, but they're more densely packed. You can scroll up to confirm the number of examples in each cluster.

g) Scroll as needed to see the remaining silhouette plots.



The silhouette method indicates that 2 clusters are optimal, because that amount of clusters returned the highest silhouette coefficient. This doesn't quite align with the elbow method's conclusions, but neither method is necessarily more valid than the other for all problems. For this particular scenario, you'll be using the conclusion of the silhouette analysis as your optimal number of clusters.

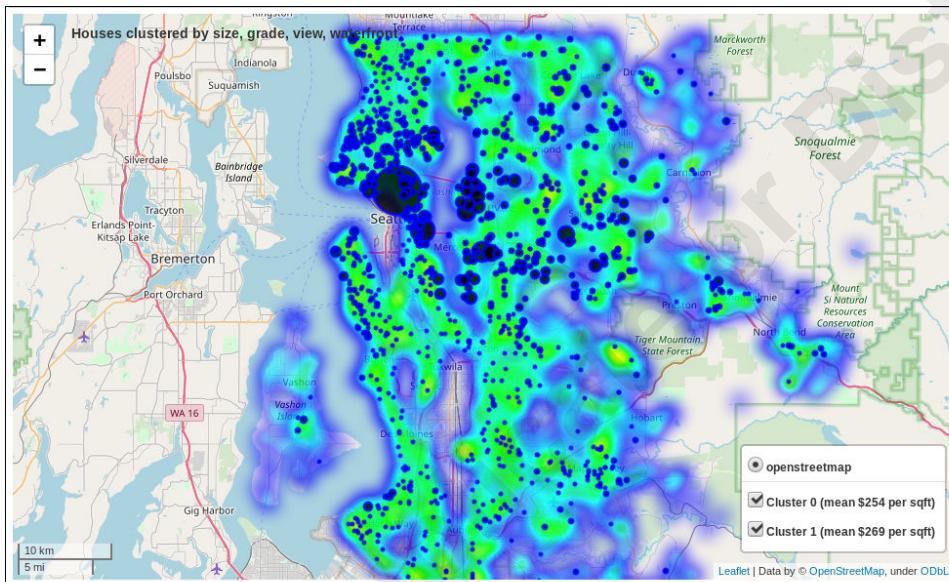
13. Use the optimal number of clusters to show the house groups.

- a) Scroll down and select the code cell beneath the **Use the optimal number of clusters to show the house groups** title, then select **Run**.

Use the optimal number of clusters to show the house groups

```
In [ ]: 1 # Generate cluster labels for the housing data based on the optimal number of clusters.
2 cluster_labels = get_cluster_labels(optimum_n_clusters, X)
3
4 # Append the cluster labels to a new column in the original dataset.
5 labeled_houses = housing_data.assign(c_label = cluster_labels)
6
7 # Show on the map
8 show_on_map(labeled_houses, 'Houses clustered by size, grade, view, waterfront')
```

- b) Examine the output.



14. Do you think this model is adequate in solving the problem of recommending similar houses to buyers who have expressed interest in a specific house? Why or why not?

15.What are some reasons why this model may need to be retrained over time?

16.Shut down this Jupyter Notebook kernel.

- a) From the menu, select **Kernel**→**Shutdown**.
 - b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
 - c) Close the **Clustering-Housing** tab in Firefox, but keep a tab open to **CAIP/Clustering/** in the file hierarchy.
-

ACTIVITY 7-2

(Optional) Building a Clustering Model for Customer Segmentation

Data Files

/home/student/CAIP/Optional/Clustering.ipynb

/home/student/CAIP/Optional/wholesale_customers_data/wholesale_customer_data.csv

Before You Begin

Jupyter Notebook is open.

Scenario

You work for Mixed Messages Media, a marketing firm. One of the firm's clients is a large wholesale distributor. The distributor sells many different kinds of products to various retail stores, but specializes in selling food products. As part of a marketing push, you've been hired to help the distributor with its customer segmentation approach. The distributor wants to be able to target their advertisements to specific retailers in order to maximize sales. You've been given historical data that includes annual spending figures for each of the distributor's retail clients for several product categories (paper products, frozen products, milk products, etc.). There is no label associated with this data, so your mission will be to try to assign the retailers to meaningful groups based on how much they spend for each type of product. So, you'll use a clustering approach to this data.

The following are the attributes of the dataset:

Attribute	Annual Spending On
Fresh	Fresh products.
Milk	Milk products.
Grocery	Grocery products.
Frozen	Frozen products.
Detergents_Paper	Detergent products and paper products.
Deli	Deli products.



Note: Sometimes, when you run code containing logic errors or bugs, you may corrupt the data contained in variables you created in previous code cells. To clear out such problems, you can select **Kernel→Restart & Clear Output**, then run each code cell again.

1. Open the clustering notebook.

- From Jupyter Notebook, select **CAIP/Optional/Clustering.ipynb** to open it.



Note: Be careful *not* to open the solution file.

- Observe the notebook.

Placeholder code cells have been provided in which you can add your own code. Comments provide hints on tasks you might perform in each code cell. The first code cell has already been completely programmed for you.

2. Import software libraries and load the dataset.

- Select the code listing under **Import software libraries and load the dataset**.
- Run the code and examine the results.

The dataset is loaded for you. There are 440 records.

3. Get acquainted with the dataset.

- In the code block under **Get acquainted with the dataset**, write code to view the data types for each feature, and then view the first 10 records.
- Run the code and verify that the shape of the training data is shown.

4. Examine the distribution of various features.

- In the code block under **Examine the distribution of various features**, write code to plot distribution histograms for all features.
- Run the code and verify that the distributions are shown.

5. Examine a general summary of statistics.

- In the code block under **Examine a general summary of statistics**, write code to view summary statistics (mean, standard deviation, min, max, etc.) for each feature.
- Run the code and verify that the summary statistics are printed.

6. Use a k -means model to label every row in the dataset.

- In the code block under **Use a k -means model to label every row in the dataset**, write code to create a k -means clustering object with 3 as the initial number of clusters.
- Write code to fit the training data to the clustering object, using only the fresh products and milk products features. Then, predict the cluster labels using this truncated dataset.
- Run the code.

7. Attach cluster labels to the original dataset.

- In the code block under **Attach cluster labels to the original dataset**, write code to append the cluster labels to the original dataset, then preview the first five rows.
- Run the code and observe the label assignments.

8. Show clusters of customers based on fresh products and milk products sale.

- In the code block under **Show clusters of customers based on fresh products and milk products sale**, write code to show a scatter plot of customer data, where fresh products is the x-axis and milk products is the y-axis. Ensure each cluster is distinguished by color.
- Run the code and observe the plotted clusters.

9. Use the elbow method to determine the optimal number of clusters.

- In the code block under **Use the elbow method to determine the optimal number of clusters**, write code to generate an elbow plot for 1 to 10 clusters. Fit the full training data to the model this time.
- Run the code and examine the suggested number of clusters based on the elbow point analysis.

10. Use silhouette analysis to determine the optimal number of clusters.

- In the code block under **Use silhouette analysis to determine the optimal number of clusters**, write code to print the silhouette scores of several k -means clustering models. The number of clusters for each model should range from 2 to 5.



Note: You can write code to visually plot the silhouettes if you want to, but just returning the scores is sufficient.

- b) Write code to print the number of clusters that received the highest silhouette score.
- c) Run the code and examine the suggested number of clusters based on the silhouette analysis.

11. Generate and preview cluster labels using the full dataset.

- a) In the code block under **Generate and preview cluster labels using the full dataset**, write code to generate a k -means clustering model and then fit the full training dataset to it. Use your own judgment to determine the desired number of clusters.
- b) Write code to predict the cluster labels, then append the labels to the dataset.
- c) Write code to show the first 20 rows in the dataset.
- d) Run the code and examine the cluster assignments.

12. In Jupyter Notebook, open **CAIP/Optional/Clustering-Solution.ipynb and compare it to the code you wrote.**



Note: Since there are many ways to write code to do the same basic thing, don't expect your code to match exactly. The important thing is that your code accomplishes the same basic goals, and returns similar results.

13. Shut down the Jupyter Notebook kernels.

- a) From the menu, select **Kernel->Shutdown**.
- b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
- c) Close the **Clustering** tab in Firefox.
- d) Repeat this process to shut down and close the **Clustering-Solution** kernel.
- e) Return to **CAIP/Clustering/** in the file hierarchy.

TOPIC B

Build Hierarchical Clustering Models

While k -means clustering is useful in many scenarios, it is not ideal in all. Certain datasets and problems may be more conducive to analysis using hierarchical clustering, which you'll use in the following topic.

k -Means Clustering Shortcomings

The k -means clustering algorithm is insensitive to data that is overlapping and works well with data that appears spherical in nature. However, consider the following graph:

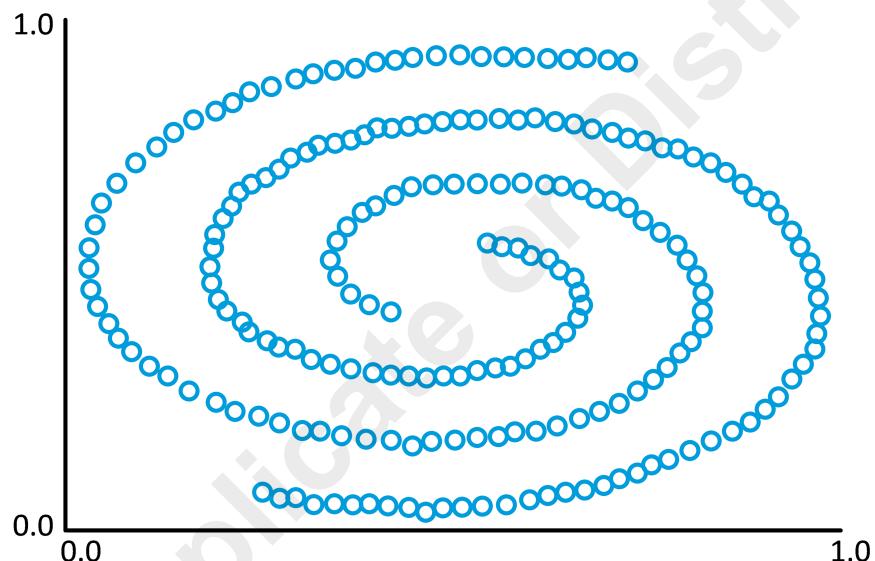


Figure 7–4: A spiral-shaped dataset.

This type of data distribution will present problems when trying to compute centroids until the means converge. For example, in the following figure, the graph on the left shows the initial centroid values, and the graph on the right shows the centroids after several iterations.

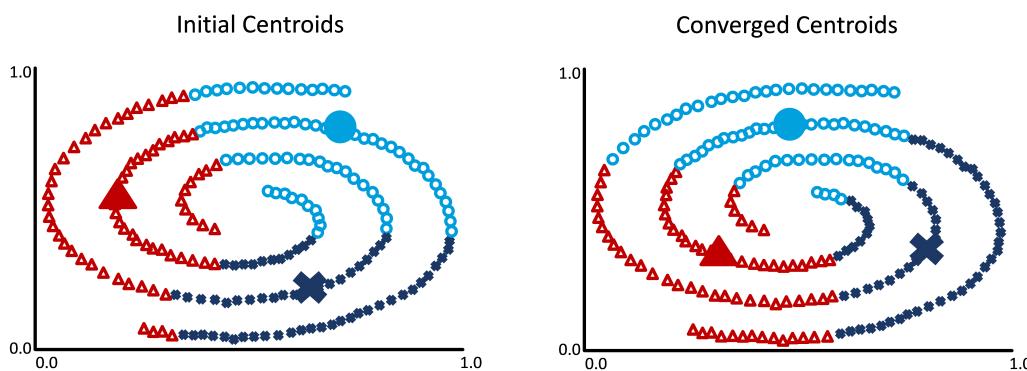


Figure 7-5: The clusters in this spiral-shaped dataset aren't particularly useful.

As you can see, the clusters don't do a good job of logically grouping similar data examples, even when the means start to converge. So, to summarize, k -means clustering is not very useful at clustering circular or spiral data—in other words, data that is well separated. This is where hierarchical clustering comes into play.

Hierarchical Clustering

Hierarchical clustering is a clustering method that, as the name implies, builds a hierarchy of groups in which similar data is placed. There are actually two approaches to hierarchical clustering: hierarchical agglomerative clustering (HAC) and hierarchical divisive clustering (HDC).

In **hierarchical agglomerative clustering (HAC)**, each data example starts out as its own cluster, then the two closest points are clustered together, then the next closest two points are clustered, and so on, until some stopping criterion is reached. HAC is referred to as a "bottom-up" approach to constructing a hierarchy of clusters. The following figure demonstrates the first four iterations of this algorithm on a small sample, using two clusters.

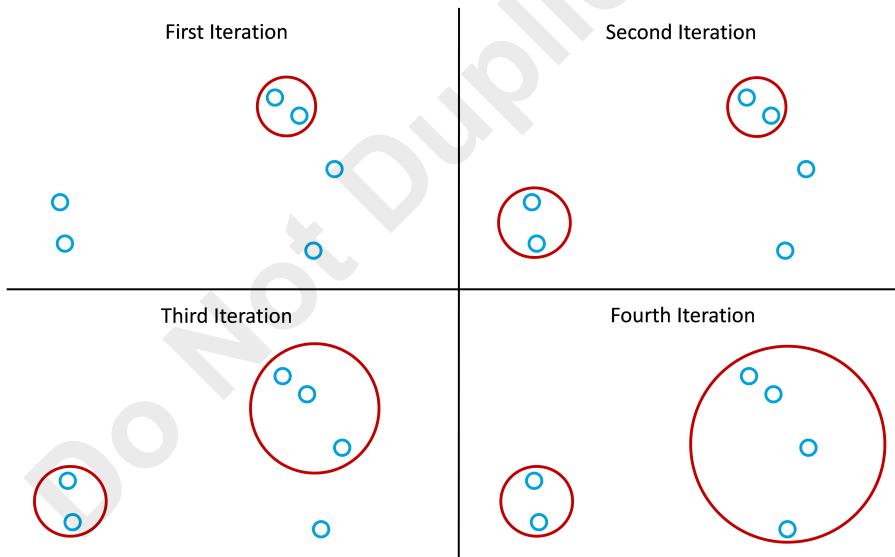


Figure 7-6: HAC starts merging the closest data points into clusters. A fifth iteration (not shown) would merge all of the points into one cluster.

Likewise, the following figure demonstrates HAC with three clusters.

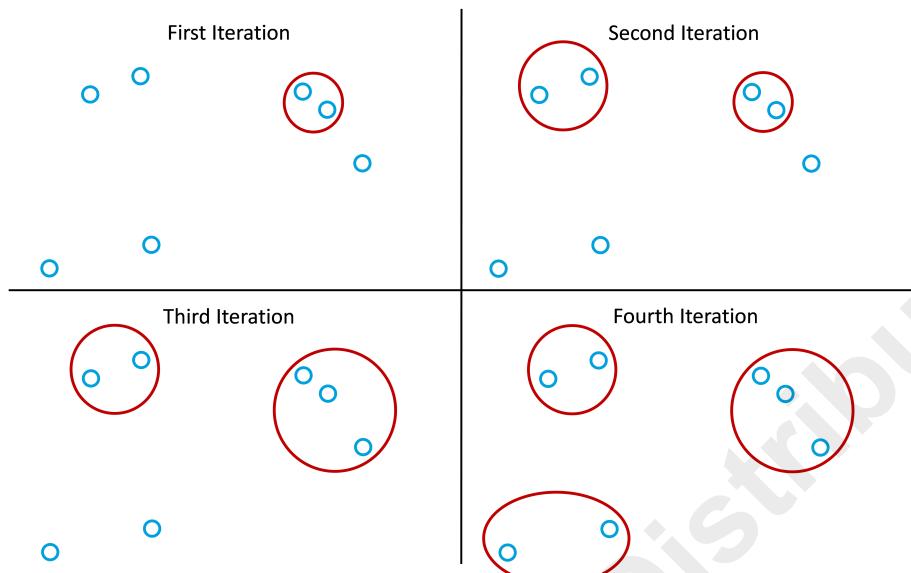


Figure 7-7: HAC applied to a three-cluster problem.

Hierarchical divisive cluster (HDC) is the opposite of HAC—all examples start out in the same single cluster, then this cluster is split using a particular method. The process continues until some stopping criterion is reached. The splitting method used in HDC can vary, and is more complex than simply taking the distance between two points like in HAC. This is because there are many different ways to split a cluster. HDC is referred to as a "top-down" approach to constructing a hierarchy of clusters.

While HAC has the advantage of its simplicity, HDC tends to be more efficient if the complete hierarchy is not created all the way to the bottom level. Also, HDC can be more skillful in its segmentation than HAC due to the higher level of complexity involved in its splitting decisions.

Hierarchical Clustering Applied to a Spiral Dataset

When applied to the spiral dataset shown previously, a hierarchical clustering algorithm (whether agglomerative or divisive) will likely produce the graph in the following figure.

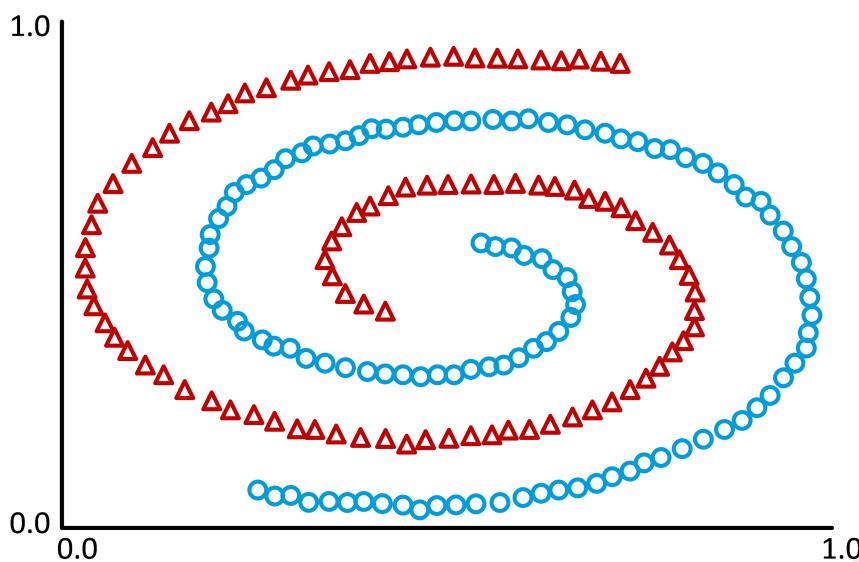


Figure 7-8: Hierarchical clustering applied to spiral data.

As you can see, the two clusters this hierarchical algorithm produced are segmented more logically than the ones produced by the k -means clustering algorithm.



Note: Hierarchical clustering can be applied to more than just datasets that make this exact shape. Any data that is well separated may be applicable to hierarchical clustering. One real-world example of data that might make this shape is topographical data placed on a map.

Hierarchical Clustering Applied to Spherical Data

Just like k -means clustering isn't all that useful on well-separated/spiral data, hierarchical clustering doesn't perform all that well on overlapping/spherical data. It will often lead to the majority of examples being placed into a single cluster or a few overly large clusters, with the outliers being placed into their own individual clusters.

When to Stop Hierarchical Clustering

Without any stopping criterion, HAC will eventually merge all data examples into a single cluster. Likewise, HDC will keep splitting clusters until each data point is by itself. Neither are ideal. So, you need to choose a point at which to stop the hierarchy from continuing in either case. As with k -means clustering, it helps to have domain knowledge of the problem you're trying to solve, as this might dictate how many clusters you need.

You won't always be so lucky, however. In cases where you don't have this level of domain knowledge, you can leverage much of the same techniques you used with k -means clustering to determine when to stop. For example, you can use silhouette analysis to try and push the silhouette coefficient as close to 1 as possible while trying different numbers of clusters. This can help ensure that data examples fit well within their clusters and are ideally far away from neighboring clusters. You can also use metrics like the Dunn index or the Davies–Bouldin index.

Dendrogram

One type of analysis unique to hierarchical clustering is the use of a dendrogram. A **dendrogram** is a diagram that represents a tree-like hierarchy. In the context of hierarchical clustering, a

dendrogram visually demonstrates which particular data points and clusters are either being merged (HAC) or split (HDC). The following figure shows an HAC dendrogram.

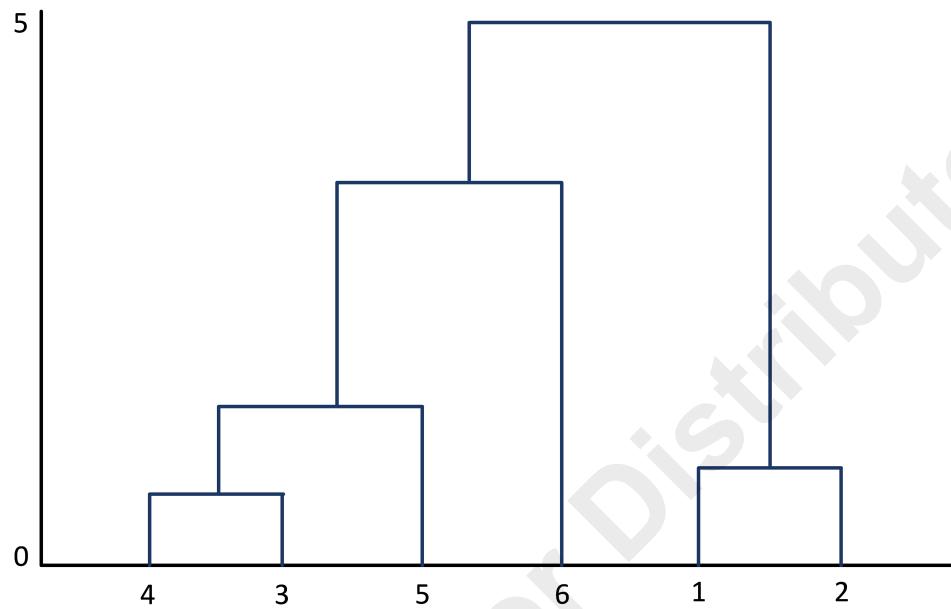


Figure 7-9: A dendrogram of hierarchical clustering data.

You start on the bottom (the x-axis) where the data examples are. As you go up the y-axis, each data example is placed into a cluster, and those clusters begin merging with other clusters. The y-axis is a measurement of how distant the data examples and clusters are from one another.

After plotting a dendrogram, you can identify a cutoff line that will indicate a stopping point. The cutoff line intersects the graph horizontally, and the number of vertical lines that it intersects is equal to the number of clusters.

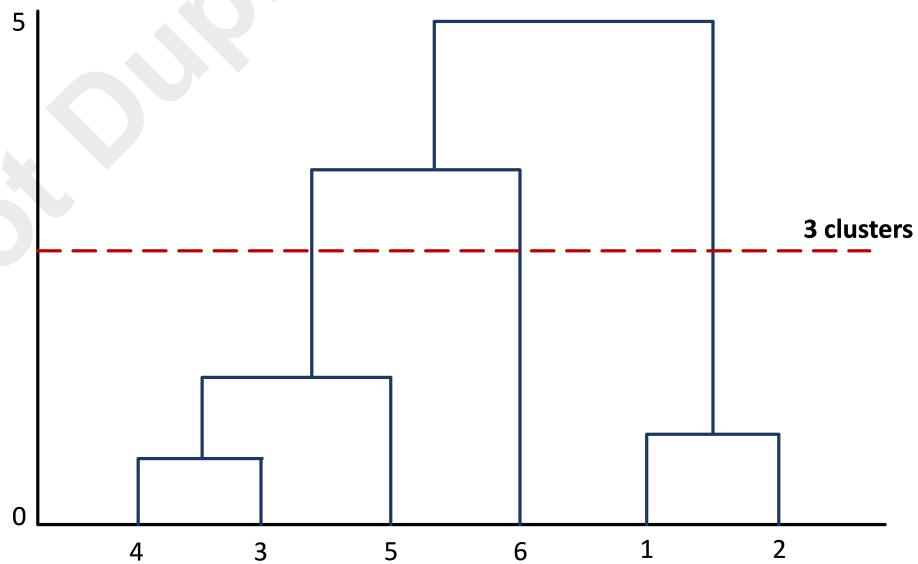


Figure 7-10: A cutoff point indicating that the number of clusters to stop on is 3.

There are several methods for determining where to place the cutoff line. As before, domain knowledge will be extremely helpful in your analysis. A statistical method for determining the cutoff line involves calculating the distance of the longest "branch" before it begins to merge. This can become more and more difficult as the size and complexity of the dendrogram increases, however. There are also more dynamic dendrogram cutting methods available in some programming libraries. For example, the Dynamic Tree Cut library for the R programming language can automate the cutoff determination process, and offers several additional advantages over a static analysis.



Note: One advantage that dendograms have over other analysis methods is that you don't need to re-run the training each time you want to experiment with a different number of clusters.

Guidelines for Building a Hierarchical Clustering Model

Follow these guidelines when you are building hierarchical clustering models.

Build a Hierarchical Clustering Model

When building a hierarchical clustering model:

- Use hierarchical clustering on data that is well separated in shape, and does not overlap.
- Consider using hierarchical agglomerative clustering (HAC) over hierarchical divisive clustering (HDC) if simplicity is desired; otherwise, consider using HDC to maximize clustering skill.
- Consider applying domain knowledge in determining the number of clusters to stop at.
- Use some of the same techniques as in k -means clustering to analyze clusters and determine the optimal number.
- Create a dendrogram to help you determine where to stop clustering using a cutoff line.
- To determine a cutoff point on a dendrogram, calculate the distance of the longest branch before it begins to merge.
- For complex dendograms, consider using dynamic cutting methods provided by some programming libraries.

Use Python for Hierarchical Agglomerative Clustering

The scikit-learn `AgglomerativeClustering()` class enables you to construct a machine learning model using an HAC algorithm. The following are some of the objects and functions you can use to build such a model.

- `model = sklearn.cluster.AgglomerativeClustering(n_clusters = 3, linkage = 'ward')` —This constructs a model object that uses the HAC algorithm. In this instance, the number of clusters specified is 3. The `linkage` argument specifies the distance metric to use between sets of examples.
- You can use this class object to call the same `fit()` and `fit_predict()` methods as before, as well as any of the applicable `metrics` methods. There is no `predict()` method, however.
- You can use the same analysis libraries like Yellowbrick to generate elbow point graphs.

To plot a dendrogram for additional cluster analysis, use the `hierarchy` module provided by SciPy:

- `linkage = scipy.cluster.hierarchy.linkage(X_train, method = 'ward')` —This constructs a `linkage` object to use with the dendrogram.
- `scipy.cluster.hierarchy.dendrogram(Z = linkage)` —This constructs a dendrogram based on the provided `linkage` object.

ACTIVITY 7–3

Building a Hierarchical Clustering Model

Data File

/home/student/CAIP/Clustering/Clustering-Moons.ipynb

Before You Begin

Jupyter Notebook is open.

Scenario

As part of your efforts to reveal logical groups of data through clustering, you've been presented with another unsupervised dataset. This one, however, includes data that is well separated. The typical k -means clustering algorithm may not be the ideal choice in this situation. So, you'll train a k -means model on the data and then compare it to the results of a hierarchical agglomerative clustering (HAC) model. The HAC model's different approach to clustering may be more applicable to this dataset. In addition, you'll evaluate and then try to select the optimal number of clusters for that model.

1. Open the notebook and load the dataset.

- From Jupyter Notebook, select **CAIP/Clustering/Clustering-Moons.ipynb** to open it.
- Select the code cell beneath the **Import software libraries** title, then select **Run**.
- Select the code cell beneath the **Load the dataset** title, then select **Run**.
- Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
Col1    1000 non-null float64
Col2    1000 non-null float64
dtypes: float64(2)
memory usage: 15.7 KB
None

   Col1      Col2
0  0.201460  0.958164
1  0.050975  0.305029
2  1.573935 -0.366780
3  -0.991440  0.094687
4  0.465245 -0.326029
```

This dataset is being artificially generated on the fly in order to produce a shape that is conducive to hierarchical clustering. Based on the arguments provided:

- There are 1,000 total samples.
- There are two feature columns, both having float values.
- Some random noise is added to the sample spread for a bit more realism.

2. Plot the data to identify its shape.

- Scroll down and select the code cell beneath the **Plot the data to identify its shape** title, then select **Run**.

- b) Examine the output.

Plot the data to identify its shape

```

1 fig, ax = plt.subplots()
2
3 # Plot the features
4 ax.scatter(X['Col1'],
5            X['Col2'],
6            marker = 'o',
7            alpha = 0.7,
8            s = 50,
9            color = 'black',
10           edgecolor = 'black');
11
12 # Axis labels.
13 ax.set_title('Raw Data')
14 ax.set_xlabel(X.columns[0])
15 ax.set_ylabel(X.columns[1])

```

The data takes the shape of two symmetrical crescent moons.

3. Use a clustering model to label every row in the dataset.

- a) Scroll down and view the cell titled **Use a clustering model to label every row in the dataset**, and examine the code listing below it.

Use a clustering model to label every row in the dataset

```

In [ ]: 1 from sklearn.cluster import AgglomerativeClustering
2 from sklearn.cluster import KMeans
3
4 # Produce cluster labels.
5 def get_cluster_labels(algorithm, cluster_count, X):
6
7     # Determine which model to generate.
8     if algorithm == 'hac':
9         model = AgglomerativeClustering(n_clusters = cluster_count, linkage = 'single')
10    if algorithm == 'kmeans':
11        model = KMeans(n_clusters = cluster_count, init = 'k-means++', random_state = 42)
12
13    model.fit(X)
14    cluster_labels = model.fit_predict(X)
15
16    # Return the original DataFrame with the labels appended as a new column.
17    return cluster_labels
18
19 print('Function to produce cluster labels has been defined.')

```

This function is similar to the function used to generate the cluster labels for the King County dataset. The only difference is that you can now select which type of clustering model to generate— k -means or agglomerative. For the agglomerative model:

- `n_clusters` is the same as it is in `KMeans()`—it defines the number of clusters.
- `linkage` refers to the linkage criterion, which specifies the distance metric to use when comparing two sets of data examples. In this case, the model is using the `single` method, which uses the minimum of the distances between all data examples in the two sets. This enables the model to determine which clusters to merge.

- b) Select the cell that contains the code listing, then select **Run**.

```
Function to produce cluster labels has been defined.
```

4. Plot the clusters.

- a) Scroll down and view the cell titled **Plot the clusters**, and examine the code listing below it.

Plot the clusters

```
In [ ]: 1 def plot_clusters():
2     fig, ax = plt.subplots()
3
4     # Caption the various clusters.
5     colors = cm.nipy_spectral(cluster_labels.astype(float) / 2)
6
7     # Plot the features.
8     ax.scatter(X['Col1'],
9                X['Col2'],
10               marker = 'o',
11               alpha = 0.7,
12               s = 50,
13               color = colors,
14               edgecolor = 'black');
15
16     # Axis labels.
17     ax.set_title('Clustered Data')
18     ax.set_xlabel(X.columns[0])
19     ax.set_ylabel(X.columns[1])
20
21
22 print('Function to plot the clusters has been defined.')
```

This function, when called, plots the data as before. The difference is on line 6, in which each of the model's cluster labels is being assigned to a different color. Each data point will be plotted in the color of whatever its cluster label is.

- b) Select the cell that contains the code listing, then select **Run**.

```
Function to plot the clusters has been defined.
```

5. Generate a k -means clustering model and plot the results.

- a) Scroll down and select the code cell beneath **Generate a k -means clustering model and plot the results** title, then select **Run**.

- b) Examine the output.

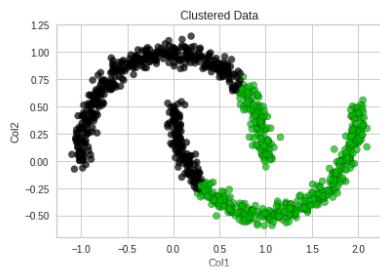
Generate a k-means clustering model and plot the results

```

1 # Generate cluster labels for the moons data, assuming 2 clusters.
2 cluster_labels = get_cluster_labels('kmeans', 2, X)
3
4 # Append the cluster labels to a new column in the original dataset.
5 labeled_data = X.assign(c_label = cluster_labels)
6
7 # Show a preview of rows in the dataset with cluster labels added.
8 display(labeled_data.head())
9
10 plot_clusters()

```

	Col1	Col2	c_label
0	0.201460	0.958164	0
1	0.050975	0.305029	0
2	1.573935	-0.366780	1
3	-0.991440	0.094687	0
4	0.465245	-0.326029	1



- Two clusters were generated using a *k*-means clustering model.
- The clusters don't seem to do a good job of supporting the separation between the crescents. Instead, the clusters seem to be defined by dividing the entire feature space in half.

6. Generate a hierarchical agglomerative clustering model and plot the results.

- a) Scroll down and select the code cell beneath the **Generate a hierarchical agglomerative clustering model and plot the results** title, then select **Run**.

- b) Examine the output.

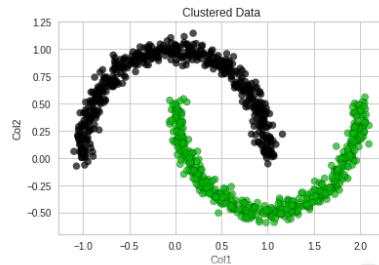
Generate a hierarchical agglomerative clustering model and plot the results

```

1 # Generate cluster labels for the moons data, assuming 2 clusters.
2 cluster_labels = get_cluster_labels('hac', 2, X)
3
4 # Append the cluster labels to a new column in the original dataset.
5 labeled_data = X.assign(c_label = cluster_labels)
6
7 # Show a preview of rows in the dataset with cluster labels added.
8 display(labeled_data.head())
9
10 plot_clusters()

```

	Col1	Col2	c_label
0	0.201460	0.958164	0
1	0.050975	0.305029	1
2	1.573935	-0.366780	1
3	-0.991440	0.094687	0
4	0.465245	-0.326029	1



- Two clusters were generated using a hierarchical agglomerative clustering (HAC) model.
- The clusters seem to do a better job of supporting the separation between the crescents.

7. Use the elbow method to determine the optimal number of clusters.

- a) Scroll down and view the code cell beneath the **Use the elbow method to determine the optimal number of clusters** title.

Use the elbow method to determine the optimal number of clusters

```

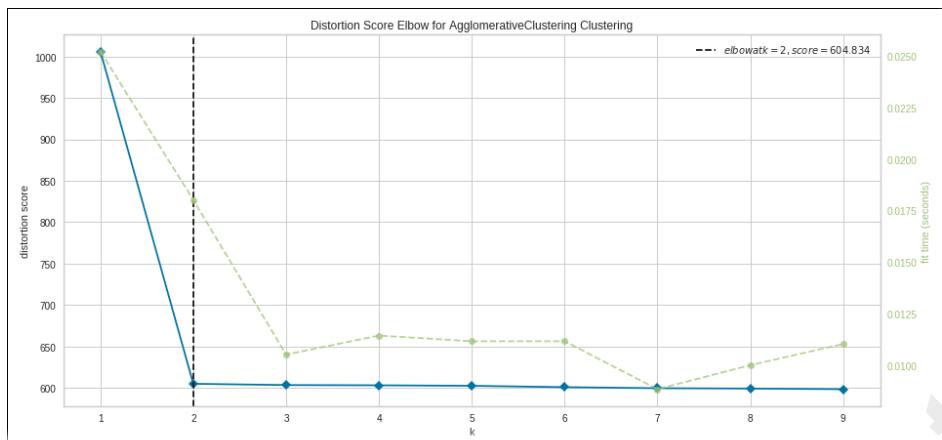
In [ ]: 1 from yellowbrick.cluster import KElbowVisualizer
2
3 # Use the elbow method to find the optimal number of clusters.
4 plt.rcParams["figure.figsize"] = (15, 7)
5
6 visualizer = KElbowVisualizer(AgglomerativeClustering(linkage = 'single'), k = (1, 10))
7 visualizer.fit(X)
8 visualizer.poof();

```

When using `single` as the linkage mode, silhouette analysis is not particularly useful. The silhouette for each cluster will likely show many negative values. This is because there are many instances where a point in cluster A is actually closer to centroid B, and vice versa. You can see this in the previous cluster plot—the green points at the inner tip of the green crescent are actually closer to the center of the black cluster than they are to their own (green) center.

- b) Select the cell that contains the code listing, then select **Run**.

- c) Examine the output.



The elbow method clearly indicates that the optimal number of clusters is 2. However, as with k -means clustering, there are other methods to determine k that are just as valid.

8. Plot a dendrogram for additional cluster analysis.

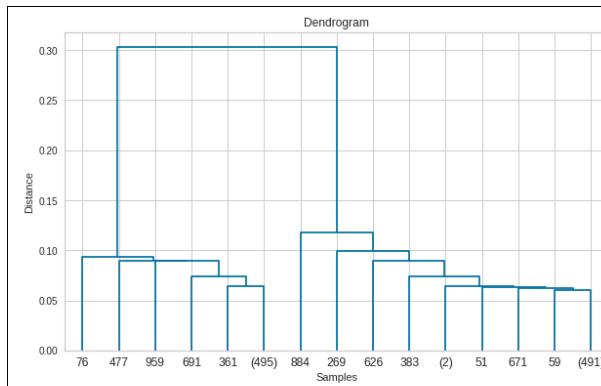
- a) Scroll down and view the cell titled **Plot a dendrogram for additional cluster analysis**, and examine the code listing below it.

Plot a dendrogram for additional cluster analysis

```
In [ ]: 1 import scipy.cluster.hierarchy as sch
2
3 # Specify same linkage used in training model.
4 linkage = sch.linkage(X, method = 'single')
5
6 def plot_dendrogram(cutoff = 0):
7     plt.figure(figsize = (10, 6))
8     plt.title("Dendrogram")
9     plt.xlabel("Samples")
10    plt.ylabel("Distance")
11
12    # Truncate dendrogram x-axis so it's readable.
13    dendrogram = sch.dendrogram(Z = linkage, truncate_mode = 'lastp', p = 15, color_threshold = cutoff)
14
15 plot_dendrogram()
```

- Line 4 defines a `linkage` object that will be used in the plot. This is the same `single` `linkage` criterion used before.
 - Line 13 creates the dendrogram with the defined linkage as the `Z` argument.
 - The `truncate_mode` will truncate the dendrogram based on the value provided by `p` (15), which determines how many examples are plotted.
- b) Select the cell that contains the code listing, then select **Run**.

- c) Examine the output.

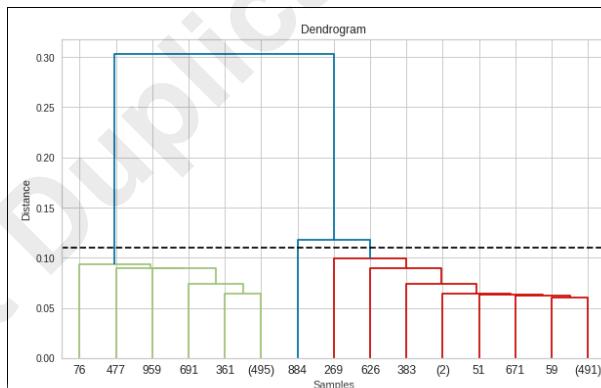


The x-axis plots the 15 examples, whereas the y-axis plots the distance values between examples and clusters.

- d) Scroll down and select the next code listing cell.

```
In [ ]: 1 cutoff = .11
          2
          3 plot_dendro(cutoff)
          4
          5 # Plot cutoff line based on specified distance.
          6 plt.axhline(y = cutoff, color= 'black', linestyle = '--')
```

- e) Select Run.
f) Examine the output.



A cutoff line is plotted as a way of determining an optimal number of clusters. In this case, the distance of the longest branch before it begins to merge is around a distance of 0.11, so that is being used as the cutoff's y-axis value. Since the cutoff passes through 3 branches, the number of clusters is 3. This aligns with the elbow analysis, but, like with k-means clustering, there is not necessarily one "best" cluster number for a given agglomerative clustering model.

9. If this were a real-world problem rather than an artificial dataset, how might domain knowledge of that dataset help you determine the optimal number of clusters?

10. Shut down this Jupyter Notebook kernel.

- a) From the menu, select **Kernel->Shutdown**.
- b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
- c) Close the **Clustering-Moons** tab in Firefox, but keep a tab open to the file hierarchy in Jupyter Notebook.

Summary

In this lesson, you used two major unsupervised learning techniques to cluster data. Although you're somewhat limited by the fact that the data isn't labeled, you can still use these techniques to group similar data examples together. This can help you identify patterns in data that otherwise may seem too disparate to analyze.

What type of data in your organization do you think might be conducive to clustering analysis?

What types of clustering evaluation metrics do you think you'll rely on most to determine the optimal number of clusters? Why?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

8

Building Decision Trees and Random Forests

Lesson Time: 3 hours

Lesson Introduction

You've built machine learning models from fundamental linear regression and classification algorithms. These algorithms can get you pretty far in many scenarios, but they are not the only algorithms that can meet your needs. In this lesson, you'll build machine learning models from decision trees and random forests, two alternative approaches to solving regression and classification problems.

Lesson Objectives

In this lesson, you will:

- Build decision tree models.
- Build random forest models.

TOPIC A

Build Decision Tree Models

To begin with, you'll build individual decision trees that can be applied to both regression and classification tasks.

Decision Tree

You've probably encountered the basic concept of a decision tree at some point. A **decision tree** is an arrangement of conditional statements and their conclusions in a branch–leaf structure. Like many concepts in this field, it's easier to grasp decision trees through visuals. Consider the following figure in which a bank determines whether or not a customer qualifies for "preferred" status.

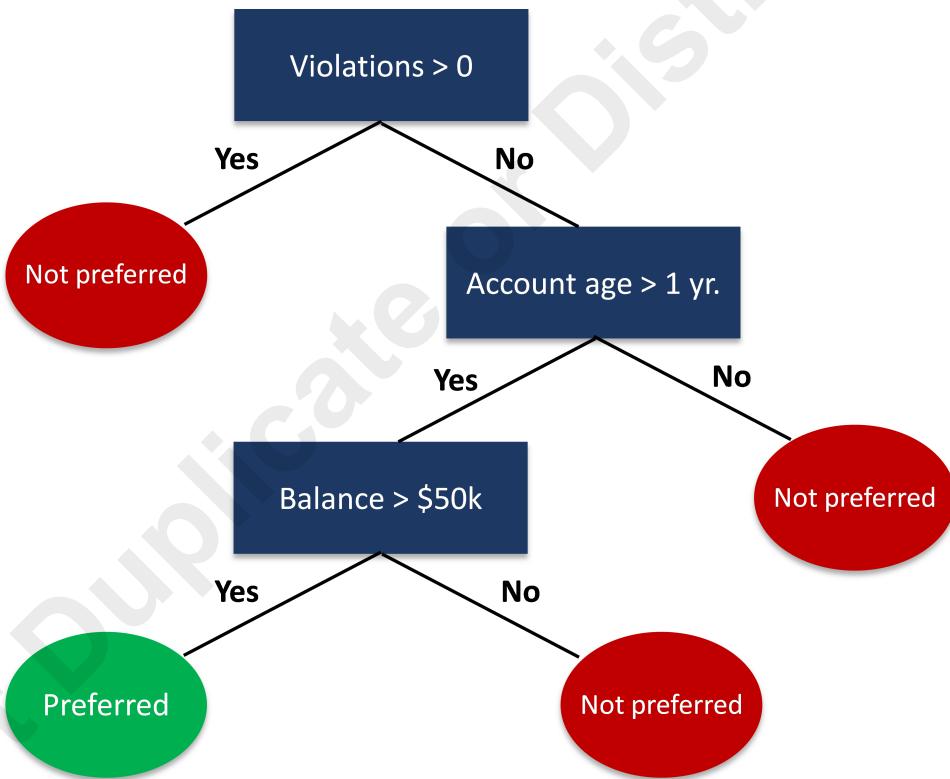


Figure 8-1: A decision tree determining the preferred status of customers.

So, a decision tree looks like a flowchart, where each "branch" represents a decision based on some condition and each "leaf" represents an output. The decision nodes receive input based on prior decisions until that particular branch terminates at a leaf.

In this example, the rectangles are the decision nodes, and the ovals are the output values. The output values in this case are classification labels. The decisions are either "Yes" or "No." Any data example (customer account) must start at the root decision node (number of violations). If the customer has had any violations, they are automatically disqualified. If not, they move on to the next node (account age). If their account is less than one year old, they do not qualify. If the account is over a year old, they continue to the final decision node (account balance). If their account balance

is less than \$50,000, they do not qualify. If their balance exceeds \$50,000, they qualify. Ultimately, any data example put through this decision tree will be given a classification based on its features.

Decision trees are used extensively in machine learning because they are simple to understand and do not require as much data preparation as some other algorithms. This is because the structure of the decision tree will be the same on a dataset regardless of procedures like scaling or feature engineering that you apply. They are not only capable of classification, but also regression tasks. Regression trees are very similar to classification trees, the main difference being that, instead of the output leaf being a class, it will be a numeric value.

Classification and Regression Tree (CART)

The **classification and regression tree (CART)** algorithm is one of the most popular decision tree algorithms in machine learning. CART uses the **Gini index** as a metric for constructing the decision tree based on the training dataset. The algorithm splits the training set into two based on a single feature with a threshold value. For example, it will first split the dataset based on the "Violations" feature, and it will choose a decision value—greater than 0, in this case. It makes this choice by using the Gini index cost function to determine the "purity" of the decision node. The purest decision node is one in which all data examples at the decision node belong to either class. The most impure decision node is one in which the data examples are split 50–50 between each class.

The following is the formula for Gini index:

$$G = 1 - \sum_{i=1}^c (p_i)^2$$

Where:

- p_i is the probability of a data example being placed in a class i , using a single feature and the label.
- c is the total number of classes.

The Gini index is calculated for each value of a feature, then a weighted sum is taken for those values to produce the ultimate Gini index for a feature. This process is repeated for the rest of the features in the dataset. The feature with the highest level of purity ($G = 0$), and therefore lowest Gini index, is chosen as the root decision node.



Note: In the context of decision tree splitting, the name "Gini index" is a reference to a measurement of income inequality, also called the Gini index or Gini coefficient. It is named after Corrado Gini, the statistician that developed the inequality measurement.

Gini Index Example

Consider the following dataset based on the banking example provided earlier.

Account age > 1	Violations > 0	Balance > \$50,000	Preferred
Yes	No	Yes	No
Yes	Yes	Yes	No
No	No	Yes	No
Yes	No	Yes	Yes
Yes	No	No	No

Account age > 1	Violations > 0	Balance > \$50,000	Preferred
Yes	No	Yes	Yes
No	No	Yes	Yes
No	Yes	Yes	No

First, we'll calculate the Gini index for "Account age." There are 5 "Yes" values for "Account age." There are 2 cases where "Account age" and "Preferred" are both "Yes." There are 3 cases where "Account age" is "Yes" and "Preferred" is "No." Plugged into the Gini index formula, this is:

$$G = 1 - ((2/5)^2 + (3/5)^2) = 0.48$$

Then, you apply the same formula to where "Account age" is "No." There are 3 "No" values for "Account age." There is 1 case where "Account age" is "No" and "Preferred" is "Yes." There are 2 cases where both "Account age" and "Preferred" are "No." Plugged into the Gini index formula, this is:

$$G = 1 - ((1/3)^2 + (2/3)^2) = 0.44$$

The weighted sum of both indices is equal to:

$$G = (5/10) \cdot 0.48 + (5/10) \cdot 0.44 = 0.46$$

CART then repeats this process for the other two features ("Violations" and "Balance"). Fastforwarding through the math, the resulting weighted Gini indices are:

Feature	Gini Index
Account age	0.46
Violations	0.38
Balance	0.43

Because "Violations" has the lowest Gini index, it is made the root decision node.

CART then uses the same Gini index cost function to split the dataset into another subset, evaluating the purity of data as before. It then splits this into a sub-subset, and so on. The splitting stops when there is no more impurity to reduce, or the tree reaches a depth specified in a hyperparameter.



Note: CART only works with binary values. It cannot construct decision trees in which a single node has three or more child nodes.

CART Hyperparameters

The following are some of the major hyperparameters associated with CART.

Hyperparameter	Description
max_depth	<p>Use this to specify how "deep" the tree should go, i.e., the number of decision nodes from root to farthest leaf, which relates to the number of times the data is split. If you don't specify a max depth, CART will keep splitting the dataset until G (purity) for all leaves is 0, or until all leaves are less than the value of <code>min_samples_split</code>.</p> <p>In their quest for achieving high levels of decision purity, decision trees tend to fall prey to overfitting, especially when the dataset has many features. The <code>max_depth</code> parameter is one way to regularize the training in order to avoid overfitting. There is not necessarily one ideal depth value for all scenarios; you must evaluate the model's performance to determine the ideal value for your specific task.</p>
min_samples_split	<p>Use this to specify how many samples (i.e., data examples) are required in order to split a decision node. By increasing this value, you constrain the model because each node must contain many examples before it can split. This can help minimize overfitting, but take care not to constrain the model too much, and thus underfitting it.</p>
min_samples_leaf	<p>Use this to specify how many samples are required to be at a leaf node. Again, increasing this value can reduce overfitting but also has the chance to lead to underfitting.</p>
splitter	<p>The default value for this parameter is "best", which calculates the Gini index as described previously. However, if you specify <code>splitter="random"</code>, the CART algorithm splits a feature at random and then calculates its Gini index. It repeats this process and identifies the split with the best Gini index. The advantages of this approach are that it requires less computational overhead and may be better at avoiding overfitting in certain cases. However, random splitting will not always be as good at making optimal decisions as the default setting.</p>



Note: The exact names of the parameters may differ based on the library you're using; the names in the previous table are based on scikit-learn.

Pruning

In keeping with the tree metaphor, **pruning** is the process of reducing the overall size of a decision tree by eliminating nodes, branches, and leaves that provide little value for the classification or regression problem at hand. This helps to mitigate the problem of overfitting, which often plagues larger trees. Even if you specify hyperparameters like max depth (often called "pre-pruning"), you may still wind up with an overly complex tree with many sections. So, pruning is another method for simplifying the model and improving its overall performance and skill.

The following table describes three major pruning methods that are applied after the tree is first generated, also known as "post-pruning."

Pruning Method	Description
Reduced-error pruning	<p>This form of pruning involves converting a subtree (a decision node, its immediate branches, and its immediate leaves) into a single leaf. The leaf in this case is the most common classification for that subtree. The decision tree is then evaluated on the test dataset, using a technique like cross-validation to split the training data. The error rate between the original decision tree and the tree with the conversion is then calculated. This process is applied to the other subtrees. The subtree with the highest reduction in error is pruned. This continues until the error rate can no longer be effectively reduced.</p> <p>Because it is relatively simple, reduced error pruning tends to be quicker than the other methods and leads to smaller trees. It has also been shown to exhibit similar levels of classification and predictive skill as other pruning methods. It is therefore a good first option, especially when combined with pre-pruning techniques.</p>
Minimum error pruning	<p>This form of pruning considers the expected error rate at each subtree given the most common classification for that subtree. This error rate is weighted according to the number of data examples that belong to each branch. Then, the expected error rate is calculated if that subtree were to be pruned (i.e., converted to a leaf). If the expected error rate of pruning the subtree is larger than not pruning it, then the subtree is kept as-is. If the expected error rate of pruning the subtree is smaller, then the subtree is pruned.</p> <p>Unlike with reduced error pruning, minimum error pruning does not involve a test dataset. However, it assumes that each class is equally likely, and it tends to perform poorly as the number of classes increases.</p>
Cost complexity pruning	<p>This form of pruning calculates the error cost of a subtree if it were converted to a leaf representing the most common classification. So, the error rate of a decision node after this conversion is multiplied by the node's proportion in the entire tree. This ends up being the error cost of the subtree if pruned. The error cost is also calculated if it were not pruned. The increase in error cost of pruning as compared to not pruning is divided by the total number of leaves in the subtree. The result is the "worth" of the subtree. The subtree with the lowest worth is pruned, creating a new decision tree. The process repeats, and a new tree is generated for each time a subtree is pruned. The decision tree with the highest classification performance on the test set is chosen.</p> <p>Like reduced-error pruning, cost complexity pruning also tends to perform well, though it is usually not as quick. Likewise, the training dataset must be split using cross-validation.</p>

C4.5

C4.5 is a decision tree algorithm that is an alternative to CART. Rather than the Gini index used by CART, C4.5 constructs trees with respect to the concepts of **entropy** and **information gain**.

Entropy refers to how "ordered" a dataset is. It is very similar to the concept of purity, in that a decision node with perfect entropy is one in which all data examples belong to a single class. However, instead of squaring each feature's class probability, entropy weights that probability by

multiplying it by a binary logarithm and multiplying that product by -1 . In addition, C4.5 can calculate entropy on not just a binary class, but any number of classes as well. Entropy is therefore calculated as follows:

$$E = -(p_1 \cdot \log_2 p_1 + p_2 \cdot \log_2 p_2 + \dots + p_n \cdot \log_2 p_n)$$

Where p_1 is the first class, p_2 is the second class, and p_n is the total number of classes.

The actual process of splitting decision nodes is determined by information gain. Information gain is the entropy of a parent decision node minus the entropy of its child decision nodes. In other words, information is *gained* as entropy is *reduced*. The feature with the highest information gain is prioritized as the root decision node. Much like with CART, the information gain process is repeated as the algorithm splits subsets and sub-subsets until some stopping criterion is reached.

In C4.5, it is not information gain alone, but the gain ratio that it used to determine the tree splitting. If two decision nodes have the same information gain, the decision node with the fewest number of distinct values is prioritized. This helps to reduce overfitting, as a feature with many distinct values is less likely to be useful in classifying a data example. A customer ID feature will have many distinct values because each customer has their own unique ID, but this feature will not be as important as, say, the customer's location, in determining what class the customer belongs to.

Related Algorithms

C4.5 is the successor to an older algorithm called **ID3**. In ID3, information gain is used without respect to gain ratio. If two decision nodes in ID3 have the same information gain, one is chosen at random. C4.5 is preferred, as, in addition to using gain ratio, it can handle missing values and enables pruning after the tree is created. It is also better than ID3 at handling continuous variables. ID3 is a legacy algorithm and should therefore not be used.

There is also a successor to C4.5 created by the same author, called C5.0. C5.0 has shown to be much faster and much more memory efficient than C4.5, while producing similar results. The full release of C5.0 is proprietary, though there is a single-threaded Linux version that is open source.

Continuous Variable Discretization

With continuous variables, because there are a significant number of possible values for a feature like age, a decision tree will start splitting off into decision nodes for each of those possible values. This can quickly lead to very large, very inefficient trees. With discrete variables, decision trees do not need to be split nearly as much.

The process of converting a continuous variable into a discrete variable is called **discretization**. This can greatly simplify and improve the performance of a decision tree model. The primary method of discretization is to take a continuous variable and place its values within specific, discrete intervals—a process also called **data binning**. An "age in years" feature is one example of binning a continuous variable, as it takes a person's age and places it into one of 130 different year intervals (0 through 129). You'd typically do this as part of the data engineering process before developing the decision tree, but discretization can also be done by algorithms like C4.5 during tree generation.

Bin Determination

Discretization requires selecting the number of bins in which to place the values, as well as the range of values between each of the bins—called a bin's "width." Any amount of discretization will lead to a loss of information, so the goal in configuring these two components is to minimize entropy. The choice of number of bins is dependent on the size of the dataset. Typically, with data examples in the thousands, you should choose five or more bins. So, rather than age in years, you might choose age in decades (13 bins); or, you might choose a smaller number of bins based on some wider age

interval. As the number of bins increases, so too does the complexity of the model, so keep that in mind.

As for determining bin width, there are two primary approaches: equal-width and equal-frequency. In equal-width discretization, each bin spans the same range of values. For example, if you specify 13 bins for ages between 0 and 129, each bin will have a range of 10. The range between minimum (0) and maximum (129) values is simply divided by the number of bins. This approach is sensitive to outliers in the data, but it tends to lead to higher levels of information loss. In the equal-frequency approach, each bin contains the same number of values. The distribution of the variable's values is taken into account. So, one bin might be wider than another if it contains fewer values. This approach tends to be better at minimizing information loss, but it is also less sensitive to outliers.

Discretization of a feature where the classification label is known (supervised) can use that label to make more informed determinations about bin number and width. The idea is to measure the relationship between all of the feature's values and the classification labels. The closer the relationship between a value and a label, the more the complexity of the binning process is justified. A feature that has minimal correlation with the label will be discretized into a small number of bins, even just one bin if there is no significant correlation. In unsupervised tasks, the discretization cannot consider any such relationship, and must use the variable's distribution to determine the number of bins and the width of each bin.

One-Hot Encoding

While decision trees tend not to require as much data preparation as other classification algorithms, this is somewhat dependent on the algorithm's implementation in the programming library you're using. For example, scikit-learn's implementation of decision trees does not handle categorical values all that well. Say you have a feature named "Color" with two possible values: "red" and "blue." Simply converting "red" and "blue" into 0 and 1 will cause that variable to be treated as ordinal data —data that has a natural order. So, 1 will be ranked above 0, which will negatively impact the splitting decisions in the tree, as one color is not more important than another. For cases like this, one-hot encoding is a viable alternative.

One-hot encoding is the process of converting a non-ordinal categorical variable into two or more constituent variables, where, for each example, all of the variables are 0 except for one. Consider the "Color" variable. Rather than having one variable with two possible values, one-hot encoding breaks this variable into two, as follows:

Data Example (Vehicle)	Red?	Blue?
Vehicle 1	1	0
Vehicle 2	0	1
Vehicle 3	1	0

One-hot encoding makes it easier for algorithms like decision trees to work with data, because the data is now in a quasi-ordinal format. In other words, a vehicle is either red or blue, so it can be represented that way. This also works for multi-class labels; "Color" can be split into "Red?" "Blue?" and "Green?" where each example has two 0s and one 1.

Although one-hot encoding may be necessary in certain situations, it can be computationally expensive, slowing down the training process.



Note: The variables that are created from one-hot encoding are also called *dummy variables*.

Decision Tree Algorithm Comparison

The following table compares CART with C4.5 and how they are used in building decision trees.

Licensed For Use Only By: Abdulwahab Alweban dev.abdulwahab@gmail.com Jan 8 202

Algorithm	Splitting Metric	Pruning Method	Supports Classification and Regression?	Supports Multi-Class Splitting?	Supports Missing Values?
CART	Gini index	Cost complexity pruning	Both	No	Yes
C4.5	Information gain ratio	Error-based pruning	Both	Yes	Yes

If your decision tree needs to make decisions based on features with more than two classes, then C4.5 is the obvious choice. If binary splitting is sufficient, then CART may occasionally be better at correctly classifying data examples, though usually not by any significant amount.



Note: Missing values may not be supported in all implementations of CART and C4.5. You may need to manually impute missing values.

Decision Trees Compared to Other Algorithms

Decision trees have some advantages over other machine learning algorithms. One advantage is that decision trees do not require much data preparation to be effective, whereas most other machine learning algorithms do. Decision trees also tend to be simpler than algorithms like logistic regression and SVMs, and can be easier to interpret. They are typically faster than a classification model like k -NN as well. The flexibility of decision trees—the fact that they can handle different types of tasks—is also a reason for their popularity.

However, decision trees may be disadvantageous in certain situations. One of the most common impediments is that they tend to be more prone to overfitting than something like linear regression or logistic regression. This can be somewhat mitigated through pruning and through ensemble methods, but the risk is still there. Likewise, in an effort to improve skill and reduce overfitting, decision trees may end up becoming very complex. Unlike with logistic regression, which can derive the significance of features as they impact classification or prediction, decision trees are unable to derive this significance. When it comes to regression tasks in which a dataset has many features, decision trees may not perform as well as linear regression.

As you might expect, the choice of decision trees comes down to the structure and distribution of your data. There is not necessarily an easily identifiable situation in which decision trees are always the best option. You must apply multiple algorithms on the dataset and evaluate each one's performance.

Guidelines for Building a Decision Tree Model

Follow these guidelines when you are building decision tree models.



Note: All of the Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Build a Decision Tree Model

When building a decision tree model:

- Consider using decision trees for both classification and regression tasks.
- Consider using decision trees if you'd like to keep the model simple, easy to understand, and easy to demonstrate visually—especially to non-technical audiences.
- Consider that, depending on the nature of the dataset and the implementation of the decision tree algorithm, you may not need to prepare that data as much as you would with other algorithms.

- Consider that you may need to discretize continuous variables before using them with decision tree algorithms.
- Consider that you may need to one-hot encode variables before using them with decision tree algorithms.
- Be aware that decision trees are prone to overfitting.
- Perform pre-pruning by tuning various decision tree hyperparameters, like the maximum depth of the tree, to help reduce overfitting.
- Consider performing various pruning methods such as reduced error pruning to further reduce the complexity of trees and minimize overfitting.
- Use C4.5 over CART when multi-class splitting is required.
- Evaluate decision trees using the same metrics you've used for other regression and classification tasks—mean squared error (MSE), accuracy, precision, recall, etc.

Use Python for Decision Trees

The scikit-learn `DecisionTreeClassifier()` and `DecisionTreeRegressor()` classes enable you to construct a classification- or regression-based decision tree model, respectively. The following are some of the objects and functions you can use to build such a model.

- `model = sklearn.tree.DecisionTreeClassifier(criterion = 'gini', max_depth = 5)` —This constructs a decision tree for classification. In this case, the splitting criterion is the Gini index.
- `model = sklearn.tree.DecisionTreeRegressor(max_depth = 5)` —This constructs a decision tree for regression.
- You can use these class objects to call the same `fit()`, `score()`, `predict()`, and `predict_proba()` (classification only) methods as before, as well as any of the applicable `metrics` methods.
- `sklearn.tree.export_graphviz(model, out_file = dot_data, feature_names = X_train.columns.values.tolist(), class_names = ['0', '1'])` —Create a graphical representation of a decision tree's structure.

ACTIVITY 8-1

Building a Decision Tree Model

Data Files

/home/student/CAIP/Decision Trees and Random Forests/DecisionTrees-Titanic.ipynb
/home/student/CAIP/Decision Trees and Random Forests/titanic_data/test.csv
/home/student/CAIP/Decision Trees and Random Forests/titanic_data/train.csv

Before You Begin

Jupyter Notebook is open.

Scenario

The logistic regression model you created and optimized for the *Titanic* dataset has done well in solving the problem assigned to it. However, you're concerned that the model may not be the best possible classifier for the job. Logistic regression isn't the only classification algorithm out there, and in order to truly be sure that you've built the best model, you need to apply your data to other algorithms. So, you'll try to build a competing model using a decision tree algorithm, then compare the results to your earlier logistic regression model. There's no guarantee that the decision tree model will be any better, but you won't know until you actually build and evaluate it.

1. From Jupyter Notebook, select **CAIP/Decision Trees and Random Forests/DecisionTrees-Titanic.ipynb** to open it.
2. Import the relevant libraries and load the dataset.
 - a) View the cell titled **Import software libraries and load the dataset**, and examine the code listing below it.
 - b) Select the cell that contains the code listing, then select **Run**.
 - c) Verify that **test.csv** and **train.csv** are in the project folder, and that the latter was loaded with 891 records.
3. Split the datasets.
 - a) Scroll down and view the cell titled **Split the datasets**, and examine the code listing below it.
 - b) Select the cell that contains the code listing, then select **Run**.

You'll be using cross-validation later during hyperparameter optimization; for now, you'll just split the datasets.
4. Look for categorical features that need to be one-hot encoded.
 - a) Scroll down and view the cell titled **Look for categorical features that need to be one-hot encoded**, and examine the code listing below it.
 - b) Select the cell that contains the code listing, then select **Run**.
 - c) Examine the output.

5. The scikit-learn implementation of decision trees does not work well with non-ordinal categorical values. Such values must be one-hot encoded before being trained. For the following question, keep in mind that PassengerId, Cabin, Ticket, and Name will be dropped from the dataset since they are largely irrelevant to the learning process.

Which of these remaining features do you think need to be one-hot encoded, and why?

6. Perform common preparation on the training and validation sets.

- a) Scroll down and view the cell titled **Perform common preparation on the training and validation sets**, and examine the code listing below it.

```

1 # Perform common cleaning and feature engineering tasks on datasets.
2 def prep_dataset(dataset):
3
4     # PROVIDE MISSING VALUES
5
6     # Fill missing Age values with the median age.
7     dataset['Age'].fillna(dataset['Age'].median(), inplace = True)
8
9     # Fill missing Fare values with the median fare.
10    dataset['Fare'].fillna(dataset['Fare'].median(), inplace = True)
11
12    # Fill missing Embarked values with the mode.
13    dataset['Embarked'].fillna(dataset['Embarked'].mode()[0], inplace = True)
14
15    # ONE-HOT ENCODING
16
17    cols = ['Pclass', 'Sex', 'Embarked']
18
19    for i in cols:
20        dummies = pd.get_dummies(dataset[i], prefix = i, drop_first = False)
21        dataset = pd.concat([dataset, dummies], axis = 1)
22
23    return dataset
24
25 X_train = prep_dataset(X_train.copy())
26
27 X_val = prep_dataset(X_val.copy())

```

This is somewhat different than the data preparation you performed earlier as part of building a logistic regression model.

- On lines 4 to 13, the function is still imputing missing values for Age, Fare, and Embarked. The scikit-learn implementation of decision trees does not support missing values, so this is necessary.
 - On lines 15 to 21, the function is performing one-hot encoding on the categorical features that appear to need it. The `get_dummies()` function from the pandas library automatically performs one-hot encoding for the given features.
 - You're not combining SibSp and Parch into its own feature as before, as a decision tree may be able to generate useful splits from each feature individually.
- b) Select the cell that contains the code listing, then select **Run**.

7. Drop columns that won't be used for training.

- a) Scroll down and view the cell titled **Drop columns that won't be used for training**, and examine the code listing below it.
- b) Select the cell that contains the code listing, then select **Run**.

8. Preview current training data.

- Scroll down and view the cell titled **Preview current training data**, and examine the code listing below it.
- Select the cell that contains the code listing, then select **Run**.
- Examine the output.

Age	SibSp	Parch	Fare	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
439	31.0	0	0	10.5000	0	1	0	0	1	0	0
617	26.0	1	0	16.1000	0	0	1	1	0	0	1
242	29.0	0	0	10.5000	0	1	0	0	1	0	1
82	27.5	0	0	7.7875	0	0	1	1	0	0	1
398	23.0	0	0	10.5000	0	1	0	0	1	0	1

The unnecessary features have been dropped, and the one-hot encoded features have replaced the originals.

9. Create a basic decision tree model.

- Scroll down and view the cell titled **Create a basic decision tree model**, and examine the code listing below it.

```

1 from sklearn.tree import DecisionTreeClassifier
2
3 tree = DecisionTreeClassifier(random_state = 1912)
4 start = time()
5 tree.fit(X_train, np.ravel(y_train))
6 end = time()
7 train_time = (end - start) * 1000
8
9 prediction = tree.predict(X_val)
10
11 # Score using the validation data.
12 score = tree.score(X_val, y_val)
13
14 print('Decision tree model took {:.2f} milliseconds to fit.'.format(train_time))
15 print('Accuracy: {:.0f}%'.format(score * 100))

```

The `DecisionTreeClassifier()` class is scikit-learn's implementation of the CART algorithm. Other than the random seed state, the object being created on line 3 uses all of the default values:

- `criterion` defaults to `gini`, which is the Gini index used by CART. The only other option is `entropy`, or information gain. There is no C4.5 implementation (information gain ratio) in the standard scikit-learn library.
- `max_depth` defaults to `None`; the tree will keep splitting until the Gini index is 0, or until all leaves contain less than the number of samples specified by `min_samples_split`.
- `min_samples_split` defaults to 2.
- `min_samples_leaf` defaults to 1.

- Select the cell that contains the code listing, then select **Run**.
- Examine the output.

```

Decision tree model took 9.44 milliseconds to fit.
Accuracy: 80%

```

Like with other classification methods, you can evaluate the model using the familiar metrics. The default model has an accuracy of 80%.

10. Visualize the decision tree structure.

- a) Scroll down and view the cell titled **Visualize the decision tree structure**, and examine the code listing below it.

```

1 from sklearn.tree import export_graphviz
2 from sklearn.externals.six import StringIO
3 from IPython.display import Image, display
4 import pydotplus as pdotp
5
6 def plot_tree(model, image):
7     dot_data = StringIO()
8     export_graphviz(model, out_file=dot_data,
9                     filled=True,
10                    rounded=True,
11                    special_characters=True,
12                    feature_names=X_train.columns.values.tolist(),
13                    class_names=['0', '1'])
14
15 graph = pdotp.graph_from_dot_data(dot_data.getvalue())
16 graph.write_png(image)
17 Image(graph.create_png())

```

This function, when called, will use the `export_graphviz` library to visually generate the structure of the decision tree. The `pydotplus` library writes the visual structure to an image file on the computer's file system.

- b) Select the cell that contains the code listing, then select **Run**.

11. Compute accuracy, precision, recall, and F_1 score.

- a) Scroll down and view the cell titled **Compute accuracy, precision, recall, and F_1 score**, and examine the code listing below it.
- b) Select the cell that contains the code listing, then select **Run**.

This function, when called, computes the four named statistical measures.

12. Generate a ROC curve and compute the AUC.

- a) Scroll down and view the cell titled **Generate a ROC curve and compute the AUC**, and examine the code listing below it.
- b) Select the cell that contains the code listing, then select **Run**.

This function, when called, generates a ROC curve.

13. Generate a precision–recall curve and compute the average precision.

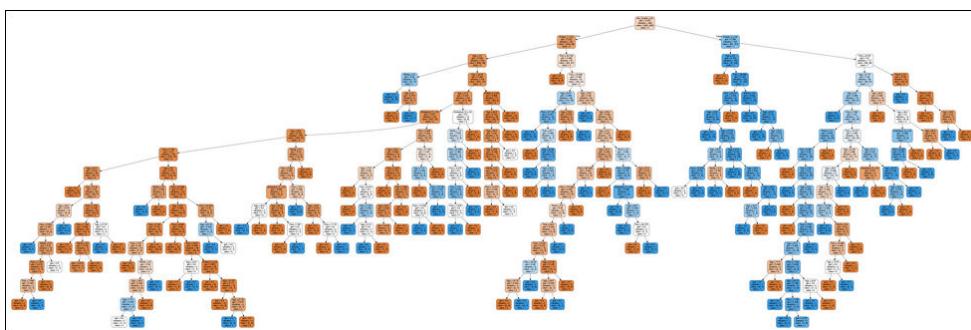
- a) Scroll down and view the cell titled **Generate a precision–recall curve and compute the average precision**, and examine the code listing below it.
- b) Select the cell that contains the code listing, then select **Run**.

This function, when called, generates a precision–recall curve.

14. Evaluate the initial decision tree model.

- a) Scroll down and view the cell titled **Evaluate the initial decision tree model**, and examine the code listing below it.
- b) Select the cell that contains the code listing, then select **Run**.

- c) Examine the output.



By default, the decision tree is quite large and difficult to interpret. You'll address this soon; for now, you can just ignore the specifics.

- d) Scroll down and select the next code listing cell.

```
1 initial_predict = tree.predict(X_val)
2
3 model_scores(y_val, initial_predict)
```

- e) Select Run.
f) Examine the output.

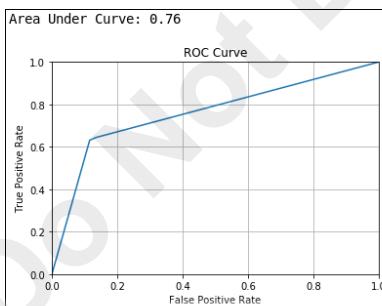
```
Accuracy: 80%
Precision: 74%
Recall: 63%
F1: 68%
```

There's room to improve these scores, which you'll do shortly.

- g) Scroll down and select the next code listing cell.

```
1 initial_predict_proba = tree.predict_proba(X_val)
2
3 roc(y_val, initial_predict_proba[:, 1])
```

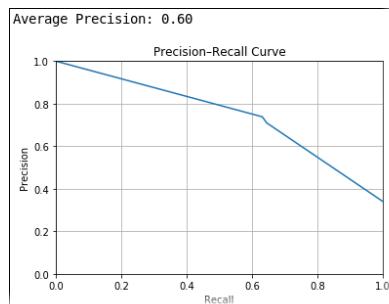
- h) Select Run.
i) Examine the output.



- j) Scroll down and select the next code listing cell.

```
1 prc(y_val, initial_predict_proba[:, 1])
```

- k) Select **Run**.
- l) Examine the output.



Both the AUC and the average precision are relatively low at this point.

15. Perform pre-pruning on the decision tree.

- a) Scroll down and view the cell titled **Perform pre-pruning on the decision tree**, and examine the code listing below it.

```
1 from sklearn.tree import DecisionTreeClassifier
2
3 pruned_tree = DecisionTreeClassifier(max_depth = 4, random_state = 1912)
4 start = time()
5 pruned_tree.fit(X_train, np.ravel(y_train))
6 end = time()
7 train_time = (end - start) * 1000
8
9 prediction = pruned_tree.predict(X_val)
10
11 # Score using the validation data.
12 score = pruned_tree.score(X_val, y_val)
13
14 print('Decision tree model took {:.2f} milliseconds to fit.'.format(train_time))
15 print('Accuracy: {:.0f}%'.format(score * 100))
```

The only difference here is that the decision tree is being constrained to a max depth of 4. This is a pre-pruning technique that can help improve the skill of the model, as too large of trees can lead to overfitting.

- b) Select the cell that contains the code listing, then select **Run**.
- c) Examine the output.

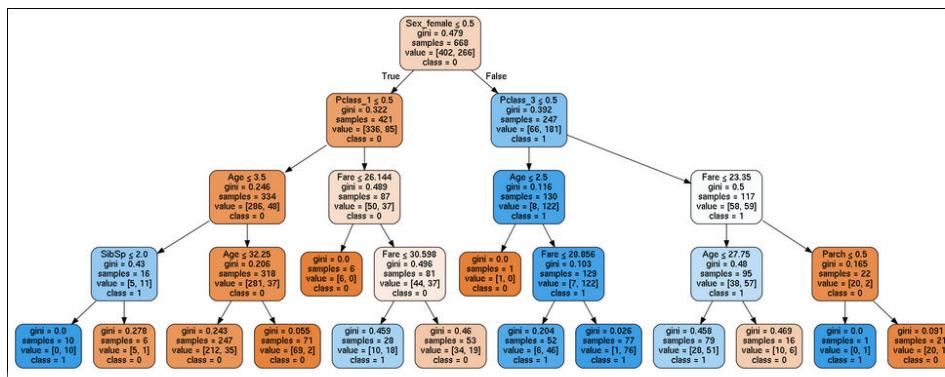
```
Decision tree model took 5.74 milliseconds to fit.
Accuracy: 82%
```

The accuracy has increased slightly.

16. Evaluate the pruned decision tree model.

- a) Scroll down and view the cell titled **Evaluate the pruned decision tree model**, and examine the code listing below it.
- b) Select the cell that contains the code listing, then select **Run**.

- c) Examine the output.



This pruned tree is much more manageable and easier to interpret.



Note: You can double-click the image to zoom in.

Each node includes splitting/decision information at that node. For the root decision node at the top:

- The tree starts by evaluating if the passenger was *not* female (in other words, male).
- The Gini index at this point is 0.479. An index of 0 represents purity, so the number should be decreasing as you descend the branches.
- The number of samples at this node is 668 (the entire training set).
- The `value` shows that 402 examples are true for this decision, and 266 are false. In other words, 402 passengers were male, and 266 passengers were female.
- The class prediction at this point is 0, meaning perished. This prediction will get much more granular as you descend the tree.

For the first "True" branch on the left:

- The decision node here is evaluating if the passenger was *not* in first class.
- The Gini index is 0.322, which, as expected, is becoming more pure.
- The number of samples is 421.
- 336 samples were not in first class; 85 were.
- The class prediction here is still 0.

For the first "False" branch on the right:

- The decision node here is evaluating if the passenger was *not* in third class.
- The Gini index is 0.392.
- The number of samples is 247.
- 66 samples were not in third class; 181 were.
- The class prediction is 1 (survived).

You can continue to descend the tree until you get to the bottom, where there are multiple leaves, each with their own class predictions based on the branching logic of their parent nodes. Note that the Gini indices for these leaves are low, but few of them are totally pure (0). This is because you specified a maximum depth as an early stopping criterion.

- d) Scroll down and select the next code listing cell.

```

1 pruned_predict = pruned_tree.predict(X_val)
2
3 model_scores(y_val, pruned_predict)

```

- e) Select Run.

- f) Examine the output.

```
Accuracy: 82%
Precision: 82%
Recall: 59%
F1: 69%
```

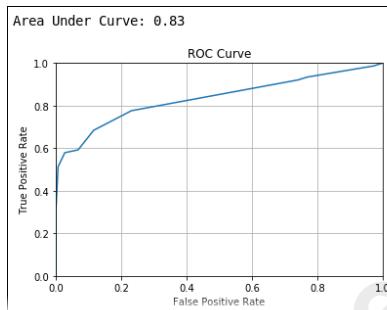
The recall seems to have decreased by a few percent, whereas the other metrics have increased by a few.

- g) Scroll down and select the next code listing cell.

```
1 pruned_predict_proba = pruned_tree.predict_proba(X_val)
2 roc(y_val, pruned_predict_proba[:, 1])
```

- h) Select Run.

- i) Examine the output.

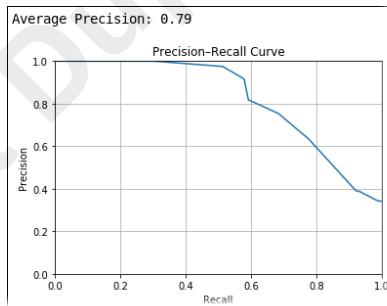


- j) Scroll down and select the next code listing cell.

```
1 prc(y_val, pruned_predict_proba[:, 1])
```

- k) Select Run.

- l) Examine the output.



Both the AUC and the average precision have improved; the latter has experienced a more significant improvement.

17. Fit a decision tree model using randomized search with cross-validation.

- a) Scroll down and view the cell titled **Fit a decision tree model using randomized search with cross-validation**, and examine the code listing below it.

```

1 from sklearn.model_selection import RandomizedSearchCV
2 from scipy.stats import randint as sp_randint
3
4 dist = {'criterion': ['gini', 'entropy'],
5         'splitter': ['best', 'random'],
6         'max_depth': sp_randint(2, 10),
7         'min_samples_split': sp_randint(5, 100),
8         'min_samples_leaf': sp_randint(5, 100)}
9
10 search = RandomizedSearchCV(tree, param_distributions = dist, n_iter = 500,
11                             scoring = 'f1', cv = 5, iid = False, random_state = 1912)
12 search.fit(X_train, np.ravel(y_train));
13 optimized_tree = search.best_estimator_
14
15 print(search.best_params_)

```

The `dist` dictionary holds the hyperparameters and the hyperparameter distributions that the randomized search will use to train and evaluate the model. Because there are so many potential combinations, grid search will take too long, so randomized search is preferable. The dictionary is as follows:

- The `criterion` is either the Gini index or information gain.
- The `splitter` is either "best" (the default Gini index approach) or "random".
- The `max_depth` is a distribution of random integers from 2 to 10.
- The `min_samples_split` is a distribution of random integers from 5 to 100.
- The `min_samples_leaf` is a distribution of random integers from 5 to 100.

When `RandomizedSearchCV()` is called:

- The hyperparameter dictionary is passed in with the `param_distributions` argument.
- The number of times a random combination of hyperparameters will be sampled (`n_iter`) is 500.
- The `scoring` argument is what the randomized search is optimizing for. In this case, you're optimizing for F_1 score. Like with grid search, you could optimize for whichever metric you'd prefer.
- By providing a `cv` value of 5, the dataset will be split using stratified k -fold cross-validation, where k is 5.

- b) Select the cell that contains the code listing, then select **Run**.

This may take a few moments.

- c) Examine the output.

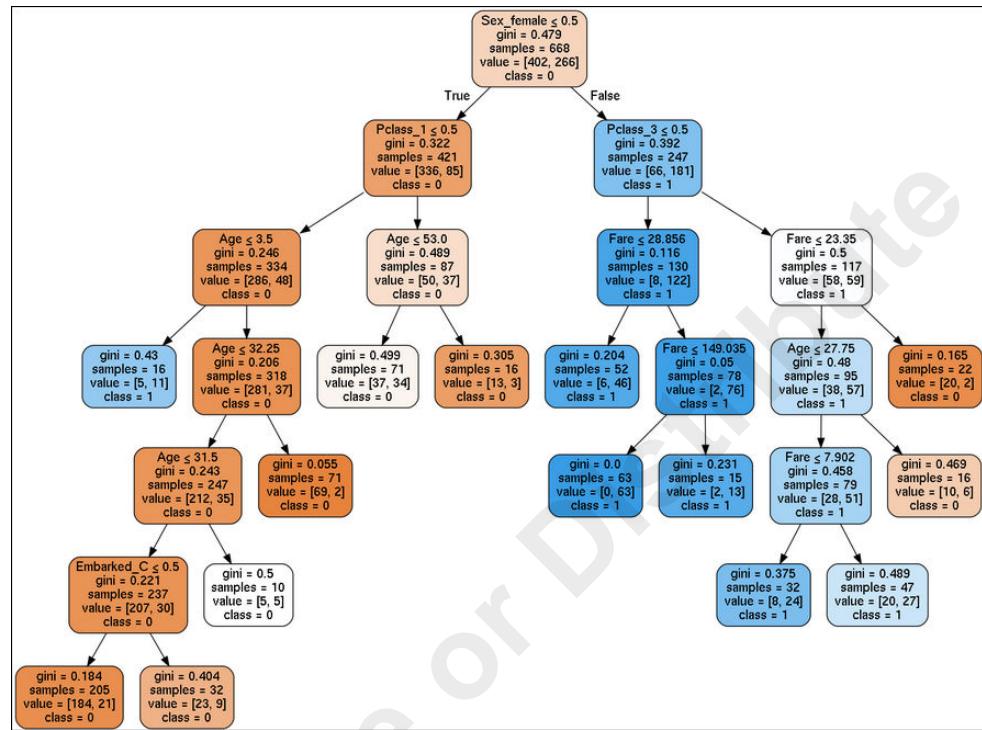
```
{"criterion": "gini", "max_depth": 6, "min_samples_leaf": 10, "min_samples_split": 78, "splitter": "best"}
```

Randomized search identified these hyperparameters as being the best combination for optimizing the F_1 score on the model.

18. Evaluate the optimized model.

- a) Scroll down and view the cell titled **Evaluate the optimized model**, and examine the code listing below it.
- b) Select the cell that contains the code listing, then select **Run**.

- c) Examine the output.



The decision tree has changed based on the selected hyperparameters. The max depth is larger, as are both the minimum number of splits and the minimum number of samples at a leaf.

- d) Scroll down and select the next code listing cell.

```
1 optimized_predict = optimized_tree.predict(X_val)
2
3 model_scores(y_val, optimized_predict)
```

- e) Select Run.
f) Examine the output.

```
Accuracy: 84%
Precision: 92%
Recall: 59%
F1: 72%
```

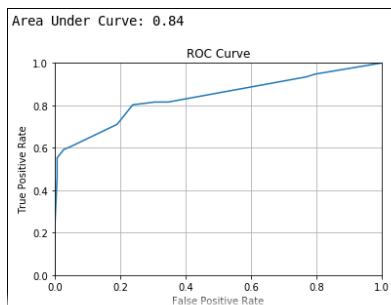
Accuracy and F_1 have improved slightly, precision has improved by a lot, and recall hasn't changed.

- g) Scroll down and select the next code listing cell.

```
1 optimized_predict_proba = optimized_tree.predict_proba(X_val)
2
3 roc(y_val, optimized_predict_proba[:, 1])
```

- h) Select Run.

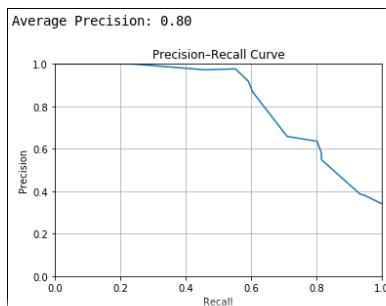
- i) Examine the output.



- j) Scroll down and select the next code listing cell.

```
1 | prc(y_val, optimized_predict_proba[:, 1])
```

- k) Select Run.
l) Examine the output.



Both the AUC and the average precision have experienced a very slight improvement.

19. Compare the results to a model that used a different algorithm.

- a) Compare the optimized decision tree model with the results of the optimized logistic regression model you created earlier.

For reference, here are the results:

Model	Accuracy	Precision	Recall	F_1	ROC AUC	Average Precision
Optimized logistic regression	83%	76%	71%	73%	85%	84%
Optimized decision tree	84%	92%	59%	72%	84%	80%

- b) Use the results table to answer the following question.

20. Are you satisfied with the results of the decision tree as compared to the logistic regression model? Do you think one model is more skillful than the other? If so, which one, and why?

21. Shut down this Jupyter Notebook kernel.

- a) From the menu, select **Kernel**→**Shutdown**.
 - b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
 - c) Close the **DecisionTrees-Titanic** tab in Firefox, but keep a tab open to **CAIP/Decision Trees and Random Forests/** in the file hierarchy.
-

TOPIC B

Build Random Forest Models

Now that you've built individual decision trees, you can begin aggregating multiple trees into a forest, which will often improve the skill of your regression and classification models.

Ensemble Learning

Ensemble learning is an application of machine learning in which the predictions of multiple models are considered in combination. The purpose is to obtain a more skillful prediction than any one model could in isolation. While a single model trained using a specific set of hyperparameters might appear to solve a classification or regression problem, it is not necessarily the optimal way of solving the problem. Studies have shown that aggregating predictions from multiple models, especially a diverse set of models, tends to lead to better predictions.

For example, you might train several models to classify a medical patient as either having heart disease or not having heart disease. One of your models might be trained using a logistic regression; another might be trained using support-vector machines (SVMs); another using a decision tree; and so on. Each model may achieve a certain level of skill, whether you measure it in accuracy, precision, recall, etc. Simply choosing the model that has the best of any one skill measurement is not guaranteed to give you ideal results. Instead, many ensemble learning methods use a majority voting system that selects the class that is predicted by the most models. For example, the following models predict a binary classifier for the heart disease problem:

- Logistic regression model
 - Accuracy: **75%**
 - Prediction: **Class 1**
- SVMs model
 - Accuracy: **83%**
 - Prediction: **Class 0**
- CART model
 - Accuracy: **80%**
 - Prediction: **Class 1**

So, the ensemble method would predict class **1** because it is the majority. Even though the model using SVMs had the highest accuracy, its prediction of 0 is not considered as useful as the majority vote. This might seem counter-intuitive, but consider how probability works: when rolling a six-sided die, the probability of getting any number is roughly 16.66%. However, if you roll the die 100 times, you may not get an even distribution of approximately 16 results for each number. However, the more rolls you perform, the higher your chances are of getting that even distribution. Rolling the die 100,000 times will have a greater chance of getting close to that even 16.66% split. This is similar to why ensemble learning is so effective—more models can lead to better results.

Random Forest

Not all ensemble methods aggregate a diverse set of training models. Some actually aggregate multiple models that use the same algorithm—the only difference is that each model is trained on a different subset of the data. The ensemble method generates these subsets by randomly sampling the overall training data for each model.

A **random forest** is an ensemble method that aggregates multiple decision tree models together and selects either:

- The mode of the classifiers between all decision trees (classification).

- The mean of the predictors between all decision trees (regression).

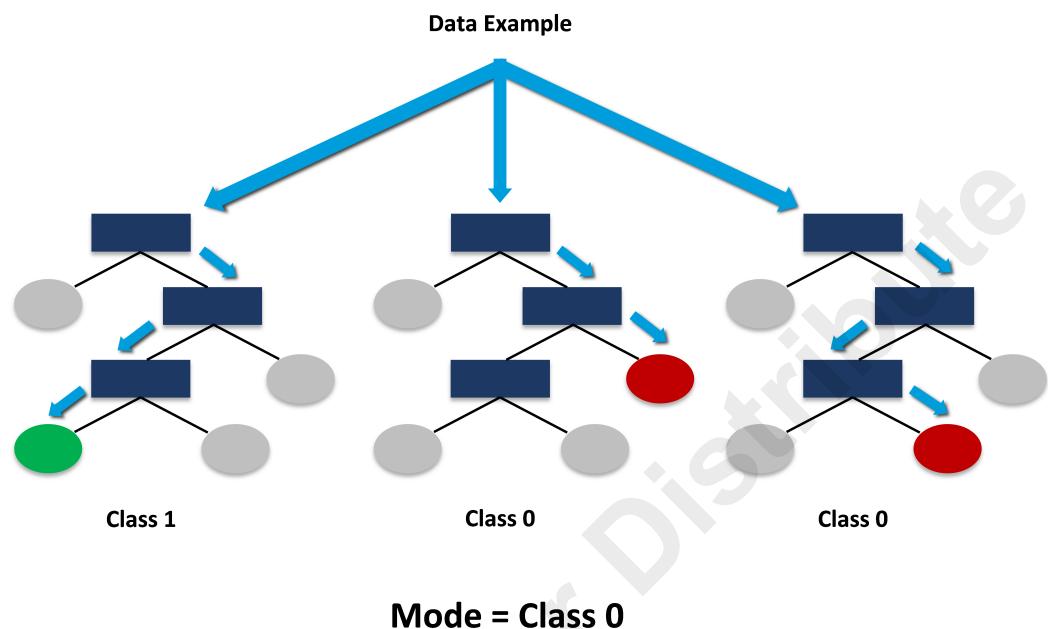


Figure 8-2: A random forest selecting a classifier based on the mode of its constituent trees.

In other words, the classification that gets the most votes among all the decision trees is the classification that the random forest outputs. As with other ensemble methods, this typically results in a more skillful classification than if you had constructed a single tree and used its results, or constructed multiple trees and simply chose the one with the most accuracy. Another way to think about this is that a random forest reduces the tendency of a single model to overfit to the training data because the forest exhibits lower levels of variance.

Most random forests use a data sampling technique called bootstrap aggregating, more commonly shortened to *bagging*. This technique samples the training dataset for each individual tree, *with replacement*. "With replacement" means that a data example can show up in multiple different models. Bagging tends to lead to lower variance, further reducing overfitting. Depending on the library used, bagging may also only use a fraction of the total feature space at each decision node.

Out-of-Bag Error

Using a cross-validation technique like stratified k -fold is not strictly necessary with random forests in order to select samples that are representative of the training data. In fact, the bagging process relies on the assumption that the sampling is representative of the data. The bagging process randomly samples about 2/3 of the data examples for any given tree in the forest, while the other 1/3 of data examples are left out for that tree. These samples are referred to as *out-of-bag*. Even though no one tree sees all of the training data, the entire forest does. You can therefore evaluate the performance of each tree in the forest based on the out-of-bag data it hasn't seen, outputting an error value. Then, you'd take the average of each error value to determine the out-of-bag error for the entire ensemble. This helps the random forest avoid overfitting, much like how k -fold cross-validation helps non-ensemble methods avoid overfitting.

To produce the out-of-bag error, each example that is left out of one or more decision trees is used to validate the training on those particular trees. Each tree predicts a class for these examples, and then the majority class vote is taken. The majority vote is compared to the actual label. So, the out-of-bag error indicates the ratio of incorrect majority vote predictions to correct majority vote

predictions, when considering all out-of-bag data examples. For instance, an out-of-bag error of 0.19 indicates that 19% of the predictions were incorrect.

Random Forest Hyperparameters

Most of the hyperparameters that apply to singular decision trees also apply to random forests. You can define the maximum depth of the trees in the forest, the minimum number of samples required to split a decision node, the minimum number of samples required to be at a leaf node, etc. One hyperparameter you cannot change is `splitter`—by default, it is set to "random". This is because a random forest does not just identify the optimal feature on which to split a decision node, like when using a typical Gini index, but instead selects the optimal feature after a series of random splits.

One additional hyperparameter that is relevant to a random forest is the number of trees in the forest (`n_estimators` in scikit-learn). As stated earlier, the more trees you add to the forest, the more skillful the prediction will be, just like rolling a die 100,000 times is better than rolling it 100. However, as the number of trees increases, so too does the computational cost of the forest. At some point the computational cost will outweigh the slight gains made by adding more trees. While you should test this tradeoff on your own data to determine what works best for your situation, it's usually a good idea to limit the number of trees to the hundreds, or even fewer than one hundred. Newer versions of scikit-learn's `RandomForestClassifier` module will set the default number of trees to 100.

Feature Selection Benefits

Not only are random forests useful as a skillful method for making predictions or decisions, but they are also good at revealing the importance (or weight) of selected feature variables. The model relies more on features with greater importance to make its determinations than features with lesser importance. For example, a person's age may be more important for determining their risk of diabetes than the person's height.

A random forest model examines how well each decision node associated with a feature reduces impurity in the tree. It takes a weighted average of this impurity reduction across all trees in the forest, where the amount of data examples at that node influence the node's weight. The output for each feature is a value between 0 and 1, where all features added together equal 1. This can be expressed as a percentage. As a hypothetical example, the following might be the weighted importance of the features in classifying whether or not a patient has diabetes:

- Body weight: 35%
- Age: 21%
- Blood pressure: 14%
- Weekly exercise: 13%
- Family history: 10%
- Race: 6%
- Others: 1%

So, as you may have guessed, random forests can aid in the feature selection process that goes into reducing the dimensionality of a training dataset. You might choose to eliminate any features that fall under "Others" in order to reduce overfitting and increase model performance.

Guidelines for Building a Random Forest Model

Follow these guidelines when you are building random forest models.

Build a Random Forest Model

When building a random forest model:

- Consider using an ensemble learning method like random forests to improve your model's predictive skill and reduce overfitting.
- Use a bootstrap aggregation (bagging) technique with random forests to perform data sampling.
- Consider that the bagging process makes cross-validation unnecessary with random forests.
- Use out-of-bag error to evaluate the performance of the trees in the forest on the data they haven't seen during the bagging process.
- Use most of the same pre-pruning hyperparameters in a random forest as you would on a single decision tree.
- Consider limiting the number of trees to grow in the forest to around a few hundred, as growing more may not be worth the extra training time.
- Use random forests for feature selection, culling the features that exhibit less importance and thereby reducing the dimensionality of the dataset.
- Evaluate random forests using the same metrics you've used for other regression and classification tasks.

Use Python for Random Forests

The scikit-learn `RandomForestClassifier()` and `RandomForestRegressor()` classes enable you to construct a classification- or regression-based random forest model, respectively. The following are some of the objects and functions you can use to build such a model.

- `model = sklearn.tree.RandomForestClassifier(n_estimators = 100, criterion = 'gini', max_depth = 5)` —This constructs a random forest for classification. In this case, the splitting criterion is the Gini index, and there will be 100 trees in the forest.
- `model = sklearn.tree.RandomForestRegressor(n_estimators = 100, max_depth = 5)` —This constructs a random forest for regression.
- You can use these class objects to call the same `fit()`, `score()`, `predict()`, and `predict_proba()` (classification only) methods as before, as well as any of the applicable `metrics` methods.
- `model.oob_score_` —An attribute that returns the out-of-bag score. Subtract this score from 1 to get the error.
- `model.oob_decision_function_` —An attribute that returns the probability predictions for each out-of-bag sample (classification only).
- `model.feature_importances_` —An attribute that returns the importances of each feature, with all feature importances adding up to 1.

ACTIVITY 8-2

Building a Random Forest Model

Data Files

/home/student/CAIP/Decision Trees and Random Forests/RandomForests-Titanic.ipynb
/home/student/CAIP/Decision Trees and Random Forests/titanic_data/test.csv
/home/student/CAIP/Decision Trees and Random Forests/titanic_data/train.csv

Before You Begin

Jupyter Notebook is open.

Scenario

As part of your quest to optimize the *Titanic* survivors model, you want to keep pushing further and trying out different classification algorithms. The single decision tree you built earlier produced some promising results, but training multiple trees in an ensemble is usually an even better approach. So, you'll build a random forest that incorporates many decision trees in order to maximize the predictive skill of the model.

1. Load the notebook and prepare the dataset.

- From Jupyter Notebook, select **CAIP/Decision Trees and Random Forests/RandomForests-Titanic.ipynb** to open it.
- Select the code cell beneath the **Import software libraries and load the dataset** title, then select **Run**.
- Verify that **test.csv** and **train.csv** are in the project folder, and that the latter was loaded with 891 records.

```
Libraries used in this project:  
- Python 3.7.3 (default, Mar 27 2019, 22:11:17)  
[GCC 7.3.0]  
- NumPy 1.16.2  
- pandas 0.24.2  
- Matplotlib 3.0.3  
- Seaborn 0.9.0  
- scikit-learn 0.20.3  
  
Data files in this project: ['test.csv', 'train.csv']  
Loaded 891 records from ./titanic_data/train.csv.
```

- Scroll down and select the code cell beneath the **Split the datasets** title, then select **Run**.

The datasets and labels have been split.

2. Why is cross-validation typically not necessary when training a random forest model?

3. Perform common preparation on the training and validation sets.

- a) Scroll down and select the code cell beneath the **Perform common preparation on the training and validation sets** title, then select **Run**.

Perform common preparation on the training and validation sets

```

1 # Perform common cleaning and feature engineering tasks on datasets.
2 def prep_dataset(dataset):
3
4     # PROVIDE MISSING VALUES
5
6     # Fill missing Age values with the median age.
7     dataset['Age'].fillna(dataset['Age'].median(), inplace = True)
8
9     # Fill missing Fare values with the median fare.
10    dataset['Fare'].fillna(dataset['Fare'].median(), inplace = True)
11
12    # Fill missing Embarked values with the mode.
13    dataset['Embarked'].fillna(dataset['Embarked'].mode()[0], inplace = True)
14
15    # ONE-HOT ENCODING
16
17    cols = ['Pclass', 'Sex', 'Embarked']
18
19    for i in cols:
20        dummies = pd.get_dummies(dataset[i], prefix = i, drop_first = False)
21        dataset = pd.concat([dataset, dummies], axis = 1)
22
23    return dataset
24
25 X_train = prep_dataset(X_train.copy())
26
27 X_val = prep_dataset(X_val.copy())
28
29 print('The datasets have been cleaned and prepared.')

```

The datasets have been cleaned and prepared.

Since a random forest is just a collection of decision trees, you'll need to prepare the data in the same way (i.e., impute missing values and one-hot encode non-ordinal categorical values). This function, when called, does this same data preparation.

4. Drop columns that won't be used for training.

- a) Scroll down and select the code cell beneath the **Drop columns that won't be used for training** title, then select **Run**.

Drop columns that won't be used for training

```

1 # Drop unused columns from datasets.
2 def drop_unused(dataset):
3
4     dataset = dataset.drop(['PassengerId'], axis = 1)
5     dataset = dataset.drop(['Cabin'], axis = 1)
6     dataset = dataset.drop(['Ticket'], axis = 1)
7     dataset = dataset.drop(['Name'], axis = 1)
8
9     # These have been replaced with one-hot encoding.
10    dataset = dataset.drop(['Pclass'], axis = 1)
11    dataset = dataset.drop(['Sex'], axis = 1)
12    dataset = dataset.drop(['Embarked'], axis = 1)
13
14    return dataset
15
16 X_train = drop_unused(X_train.copy())
17
18 X_val = drop_unused(X_val.copy())
19
20 print('Unused columns have been dropped.')

```

Unused columns have been dropped.

5. Create a random forest model.

- a) Scroll down and view the cell titled **Create a random forest model**, and examine the code listing below it.

Create a random forest model

```
In [ ]: 1 from sklearn.ensemble import RandomForestClassifier
2
3 forest = RandomForestClassifier(n_estimators = 100,
4                                 criterion = 'gini',
5                                 max_depth = 6,
6                                 min_samples_leaf = 10,
7                                 min_samples_split = 78,
8                                 bootstrap = True,
9                                 oob_score = True,
10                                random_state = 1912)
11
12 forest.fit(X_train, np.ravel(y_train))
13
14 prediction = forest.predict(X_val)
15
16 # Score using the validation data.
17 score = forest.score(X_val, y_val)
18
19 print('Accuracy: {:.0f}%'.format(score * 100))
```

The `RandomForestClassifier()` class is scikit-learn's implementation of a decision tree ensemble. The class uses much of the same hyperparameters as a lone decision tree. In this case, the hyperparameters are set to the same optimized values that were identified earlier. However, there are three new hyperparameters/arguments:

- `n_estimators` specifies how many decision trees will be grown in the forest. Having more than a few hundred trees will usually just slow the training process down without providing any significant gains, so 100 should be adequate.
 - `bootstrap` specifies whether or not bagging will be performed. `True`, the default value, means that it will be. If `False`, the entire training set is used for every tree.
 - `oob_score` set to `True` tells the algorithm to compute accuracy scores using out-of-bag samples.
- b) Select the cell that contains the code listing, then select **Run**.
 c) Examine the output.

Accuracy: 83%

This initial random forest model has an accuracy of 83%. You'll evaluate the model using other metrics shortly.

6. Visualize the structure of one decision tree in the forest.

- a) Scroll down and select the code cell beneath the **Visualize the structure of one decision tree in the forest** title, then select **Run**.

Visualize the structure of one decision tree in the forest

```

1 from sklearn.tree import export_graphviz
2 from sklearn.externals.six import StringIO
3 from IPython.display import Image, display
4 import pydotplus as pdotp
5
6 def plot_tree(model, image):
7     dot_data = StringIO()
8     export_graphviz(model, out_file = dot_data,
9                     filled = True,
10                    rounded = True,
11                    special_characters = True,
12                    feature_names = X_train.columns.values.tolist(),
13                    class_names = ['0', '1'])
14
15 graph = pdotp.graph_from_dot_data(dot_data.getvalue())
16 graph.write_png(image)
17 Image(graph.create_png())
18
19 print('The visualization function has been defined.')

```

The visualization function has been defined.

As before, this function, when called, will generate an image of a decision tree. However, since a random forest is made of many trees, you'll need to specify which trees to visualize.

7. Compute accuracy, precision, recall, and F_1 score.

- a) Scroll down and select the code cell beneath the **Compute accuracy, precision, recall, and F_1 score** title, then select **Run**.

Compute accuracy, precision, recall, and F_1 score

```

1 from sklearn.metrics import accuracy_score
2 from sklearn.metrics import precision_score
3 from sklearn.metrics import recall_score
4 from sklearn.metrics import f1_score
5
6 def model_scores(y, prediction):
7     acc = accuracy_score(y, prediction)
8     print('Accuracy: {:.0f}%'.format(np.round(acc * 100)))
9
10    precision = precision_score(y, prediction)
11    print('Precision: {:.0f}%'.format(np.round(precision * 100)))
12
13    recall = recall_score(y, prediction)
14    print('Recall: {:.0f}%'.format(np.round(recall * 100)))
15
16    f1 = f1_score(y, prediction)
17    print('F1: {:.0f}%'.format(np.round(f1 * 100)))
18
19 print('The function to show the scores has been defined.')

```

The function to show the scores has been defined.

8. Generate a ROC curve and compute the AUC.

- a) Scroll down and select the code cell beneath the **Generate a ROC curve and compute the AUC** title, then select **Run**.

Generate a ROC curve and compute the AUC

```

1 from sklearn.metrics import roc_curve
2 from sklearn.metrics import roc_auc_score
3
4 def roc(y, prediction_proba):
5     fpr, tpr, thresholds = roc_curve(y, prediction_proba)
6
7     plt.plot(fpr, tpr);
8     plt.xlim([0.0, 1.0]);
9     plt.ylim([0.0, 1.0]);
10    plt.title('ROC Curve');
11    plt.xlabel('False Positive Rate');
12    plt.ylabel('True Positive Rate');
13    plt.grid(True);
14
15    auc = roc_auc_score(y, prediction_proba)
16    print('Área Under Curve: {:.2f}'.format(auc))
17
18 print('The function to generate the ROC curve and compute AUC has been defined.')

```

The function to generate the ROC curve and compute AUC has been defined.

9. Generate a precision–recall curve and compute the average precision.

- a) Scroll down and select the code cell beneath the **Generate a precision–recall curve and compute the average precision** title, then select **Run**.

Generate a precision–recall curve and compute the average precision

```

1 from sklearn.metrics import precision_recall_curve
2 from sklearn.metrics import average_precision_score
3
4 def prc(y, prediction_proba):
5     precision, recall, thresholds = precision_recall_curve(y, prediction_proba)
6
7     plt.plot(recall, precision);
8     plt.xlim([0.0, 1.0]);
9     plt.ylim([0.0, 1.0]);
10    plt.title('Precision-Recall Curve');
11    plt.xlabel('Recall');
12    plt.ylabel('Precision');
13    plt.grid(True);
14
15    ap = average_precision_score(y, prediction_proba)
16    print('Average Precision: {:.2f}'.format(ap))
17
18 print('The function to generate a PRC and compute average precision has been defined.')

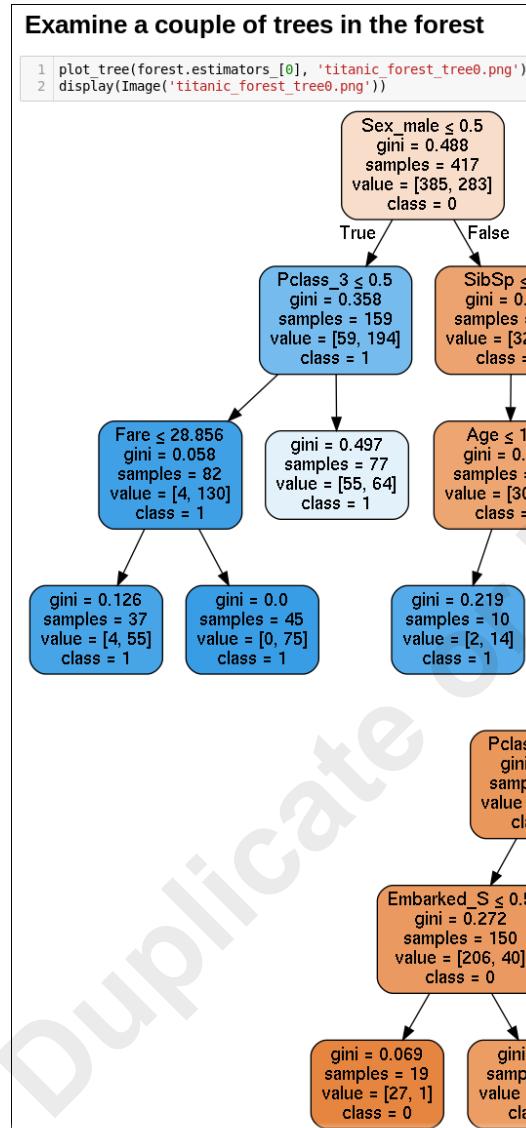
```

The function to generate a PRC and compute average precision has been defined.

10. Examine a couple of trees in the forest.

- a) Scroll down and select the code cell beneath the **Examine a couple of trees in the forest** title, then select **Run**.

- b) Examine the output.



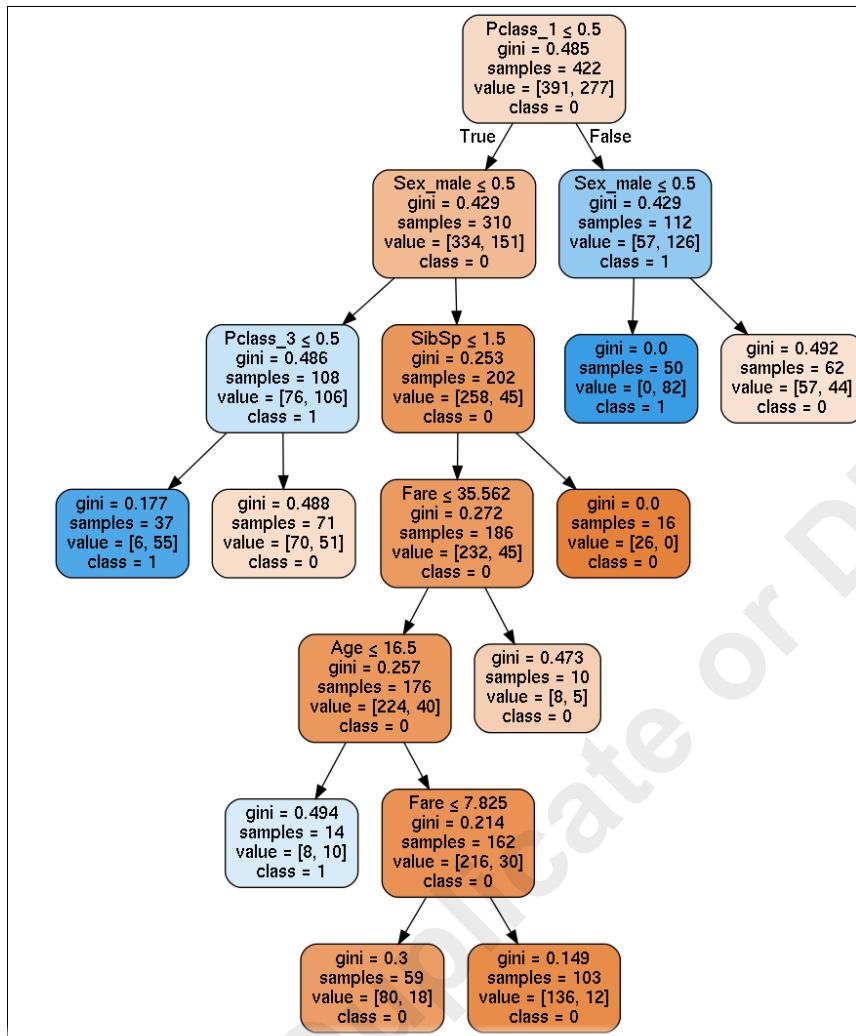
The `estimators_` attribute refers to each tree in the forest, so the tree at index 0 is being shown here.

- c) Scroll down and select the next code listing cell.

```
In [ ]: 1 plot_tree(forest.estimators_[1], 'titanic_forest_tree1.png')
2 display(Image('titanic_forest_tree1.png'))
```

- d) Select Run.

e) Examine the output.



The tree at index 1 is being shown here.

11. Compare the two trees.

How do the first two branches of each tree differ in terms of their splitting criteria?

12. Evaluate the random forest model.

- Scroll down and select the code cell beneath the **Evaluate the random forest model** title, then select **Run**.

- b) Examine the output.

Evaluate the random forest model

```
1 model_scores(y_val, prediction)
```

Accuracy: 83%
Precision: 80%
Recall: 67%
F1: 73%

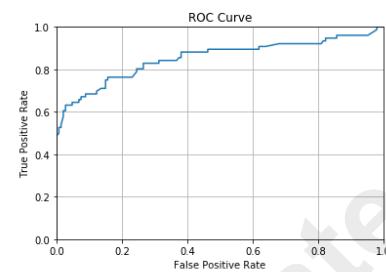
- c) Scroll down and select the next code listing cell.

```
In [ ]: 1 predict_proba = forest.predict_proba(X_val)
          2
          3 roc(y_val, predict_proba[:, 1])
```

- d) Select Run.

- e) Examine the output.

Area Under Curve: 0.85



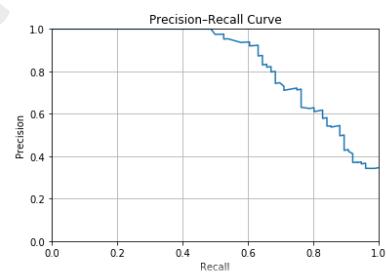
- f) Scroll down and select the next code listing cell.

```
In [ ]: 1 prc(y_val, predict_proba[:, 1])
```

- g) Select Run.

- h) Examine the output.

Average Precision: 0.84



- i) Compare the random forest model with the results of the single optimized decision tree model you created earlier.

For reference, here are the results:

Model	Accuracy	Precision	Recall	F_1	ROC AUC	Average Precision
Single optimized decision tree	84%	92%	59%	72%	84%	80%
Random forest	83%	80%	67%	73%	85%	84%

Some of these metrics changed slightly (e.g., accuracy and F_1), whereas others change more dramatically (e.g., precision and recall). While the hyperparameters worked well for the one decision tree, they may not be optimal for the entire forest. You could conceivably run a randomized search on the forest to find better hyperparameters, but this will take a while depending on how many trees (`n_estimators`) are being grown in the forest.

13.What advantage does a random forest have over a single decision tree?

14.Generate the out-of-bag error and decision function.

- Scroll down and select the code cell beneath the **Generate the out-of-bag error and decision function** title, then select **Run**.
- Examine the output.

```
out-of-bag error: 0.22
```

In this case, 22% of the predictions were incorrect.

- Scroll down and select the next code listing cell.

```
In [ ]: 1 forest.oob_decision_function_
```

- Select **Run**.
- Examine the output.

```
array([[0.83990492, 0.16009508],
       [0.44309796, 0.55690204],
       [0.88436021, 0.11563979],
       ...,
       [0.72455935, 0.27544065],
       [0.43065855, 0.56934145],
       [0.18766646, 0.81233354]])
```

This is an array of probability predictions for each out-of-bag example. The first column indicates the probability for class 0 (perished) and the second column indicates the probability for class 1 (survived). Like with any other binary classifier, the class with the highest probability is used as the prediction.

15.Verify that the random forest took the majority vote of all trees in the forest.

- Scroll down and select the code cell beneath the **Verify that the random forest took the majority vote of all trees in the forest** title, then select **Run**.

- b) Examine the output.

Verify that the random forest took the majority vote of all trees in the forest

```

1 from statistics import mode
2
3 class_list = []
4 class_mode = []
5 i = 0
6
7 for i in range(10): # Get predictions for the first 10 data examples.
8     for tree in forest.estimators_: # Get predictions from all trees in forest.
9         predict = tree.predict(X_val)
10        predict_examples = predict[i]
11        class_list.append(predict_examples)
12
13    # Get mode of predictions to determine majority vote for each example.
14    class_mode.append(int(mode(class_list)))
15    class_list = []
16
17 print('Majority vote classification: {}'.format(np.array(class_mode)))
18 print('Random forest classification: {}'.format(forest.predict(X_val)[0:10]))

```

Majority vote classification: [0 0 0 1 0 1 1 0 0 0]
 Random forest classification: [0 0 0 1 0 1 1 0 0 0]

The purpose of this code block is to demonstrate that the random forest, for each data example, returns the majority vote across all of the trees in the forest. As you can see, manually taking this vote gives you the same results as automatically calling the forest's `predict()` method.

16. Identify important features.

- a) Scroll down and select the code cell beneath the **Identify important features** title, then select **Run**.
 b) Examine the output.

Identify important features

```

1 # Get feature importances.
2 feature_importances_ = list(forest.feature_importances_)
3
4 # Get column names.
5 feature_list = list(X_train)
6
7 importances_list = []
8
9 # Map feature/importance indices and put into a single list.
10 for feature, importance in zip(feature_list, feature_importances_):
11     importances_list.append((feature, round(importance, 2)))
12
13 # Sort list by importance index.
14 importances_list = sorted(importances_list, key = lambda i: i[1], reverse = True)
15
16 for feature, importance in importances_list:
17     print('Feature: {} | Importance: {}'.format(feature, importance))

```

Feature	Importance
Sex_female	0.34
Sex_male	0.25
Pclass_3	0.11
Fare	0.1
Pclass_1	0.06
Age	0.04
SibSp	0.02
Parch	0.02
Pclass_2	0.02
Embarked_C	0.02
Embarked_S	0.01
Embarked_Q	0.0

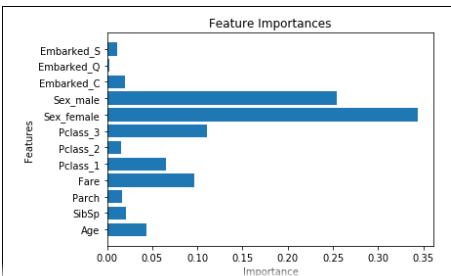
The `feature_importances_` attribute returns a ranking of each feature's contribution to the class prediction in the random forest. All of the features combined add up to 1, and higher values indicate more importance.

- It looks like the sex of the passenger, particularly if they were female, contributed the most to the predictions.
- Whether or not a passenger was in third class, as well as the passenger's fare, also seemed to have some importance.
- Analyzing this data visually might be more helpful.

- c) Scroll down and select the next code listing cell.

```
In [ ]: 1 y_values = list(range(len(feature_importances)))
2 plt.barh(y_values, feature_importances)
3 plt.yticks(y_values, feature_list, rotation='horizontal')
4 plt.xlabel('Importance')
5 plt.ylabel('Features')
6 plt.title('Feature Importances');
```

- d) Select Run.
e) Examine the output.



This visual plot helps to reinforce which features are truly important. Likewise, it can help you decide which features to keep and which to drop when you retrain the forest.

- Sex_male, Sex_female, Pclass_3, and Fare all seem relatively important, so you'll keep them.
- Pclass_1 and maybe even Age still have some importance, though perhaps not to any degree that it would impact the model all that much. You'll drop these, along with the rest of the features not mentioned.

17. Select the most important features to use for a new round of training.

- a) Scroll down and select the code cell beneath the **Select the most important features to use for a new round of training** title, then select Run.
b) Examine the output.

Select the most important features to use for a new round of training

```
1 X_train = X_train[['Sex_female', 'Sex_male', 'Pclass_3', 'Fare']].copy()
2 X_val = X_val[['Sex_female', 'Sex_male', 'Pclass_3', 'Fare']].copy()
3
4 X_train.head()
```

	Sex_female	Sex_male	Pclass_3	Fare
439	0	1	0	10.5000
617	1	0	1	16.1000
242	0	1	0	10.5000
82	1	0	1	7.7875
398	0	1	0	10.5000

The training and validation sets now only include the features that were identified as having high importance, reducing the datasets' dimensionality.

18. Train a new random forest model on the dataset with reduced dimensionality.

- a) Scroll down and view the cell titled **Train a new random forest model on the dataset with reduced dimensionality**, and examine the code listing below it.

Train a new random forest model on the dataset with reduced dimensionality

```
In [ ]: 1 from sklearn.ensemble import RandomForestClassifier
2
3 forest = RandomForestClassifier(n_estimators = 100,
4                                 criterion = 'gini',
5                                 max_depth = 6,
6                                 min_samples_leaf = 10,
7                                 min_samples_split = 78,
8                                 bootstrap = True,
9                                 oob_score = True,
10                                random_state = 1912)
11
12 forest.fit(X_train, np.ravel(y_train))
13
14 prediction = forest.predict(X_val)
15
16 # Score using the validation data.
17 score = forest.score(X_val, y_val)
18
19 print('Accuracy: {:.0f}%'.format(score * 100))
```

This is the same training code as before, but now it's using the newly reduced datasets.

- b) Select the cell that contains the code listing, then select **Run**.
 c) Examine the output.

Accuracy: 84%

The model's accuracy has increased slightly.

19. Evaluate the new random forest model.

- a) Scroll down and select the code cell beneath the **Evaluate the new random forest model** title, then select **Run**.
 b) Examine the output.

Evaluate the new random forest model

```
1 model_scores(y_val, prediction)
Accuracy: 84%
Precision: 82%
Recall: 67%
F1: 74%
```

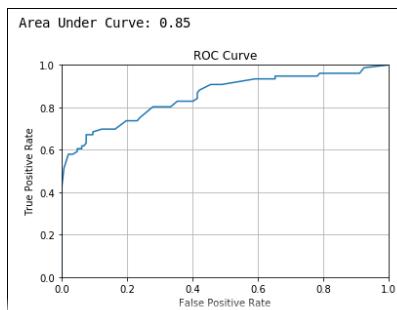
Compared to the previous model, the new model's accuracy, precision, and F_1 score have improved slightly.

- c) Scroll down and select the next code listing cell.

```
In [ ]: 1 predict_proba = forest.predict_proba(X_val)
2
3 roc(y_val, predict_proba[:, 1])
```

- d) Select **Run**.

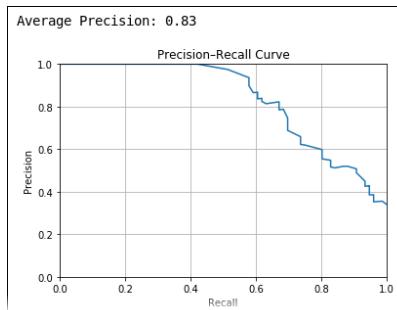
- e) Examine the output.



- f) Scroll down and select the next code listing cell.

```
In [ ]: 1 prc(y_val, predict_proba[:, 1])
```

- g) Select Run.
h) Examine the output.



- The ROC AUC seems to have stayed the same, whereas the average precision decreased slightly.
i) Scroll down and select the next code listing cell.

```
In [ ]: 1 oob_error = 1 - forest.oob_score_
2 print('Out-of-bag error: {:.3f}'.format(round(oob_error, 3)))
```

- j) Select Run.
k) Examine the output.

```
Out-of-bag error: 0.207
```

The out-of-bag error improved on this reduced model.

20. The changes to these scores were largely the result of reducing the datasets' dimensionality.

How might you retrain the model to improve these scores even further?

21. Shut down this Jupyter Notebook kernel.

- a) From the menu, select **Kernel**→**Shutdown**.
 - b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
 - c) Close the **RandomForests-Titanic** tab in Firefox, but keep a tab open to the file hierarchy in Jupyter Notebook.
-

Summary

In this lesson, you built decision trees as an alternative approach to solving regression and classification problems. You then combined multiple trees to create a forest, an ensemble approach that helps minimize overfitting and improve overall model skill. Although these algorithms are not necessarily better than linear regression or logistic regression in all cases, it's important to experiment with different types of algorithms to see how they produce different models.

In your own environment, how might you use decision trees to solve business problems?

In your own environment, how might you use random forests to solve business problems?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Do Not Duplicate Or Distribute

9

Building Support-Vector Machines

Lesson Time: 2 hours

Lesson Introduction

Another alternative approach to regression and classification comes in the form of support-vector machines (SVMs). In this lesson, you'll build SVMs that can do a good job of handling outliers and tackling high-dimensional data in an efficient manner.

Lesson Objectives

In this lesson, you will:

- Build SVM models for classification.
- Build SVM models for regression.

TOPIC A

Build SVM Models for Classification

You'll start by building an SVM classifier to see how well it compares to other classification algorithms.

Support-Vector Machines (SVMs)

Support-vector machines (SVMs) are supervised learning algorithms that can be used to solve both classification and regression problems. SVMs separate data values using a hyperplane. The **hyperplane** includes a decision boundary that is either a line or curve function. On each side of this boundary is a line or curve function parallel and equidistant to the decision boundary. The functions on either edge of the decision boundaries are called the support vectors, and they construct margins around the decision boundary.

It's much easier to grasp the concept of SVMs using visuals, so consider the following figure, in which a hyperplane is plotted on a graph.

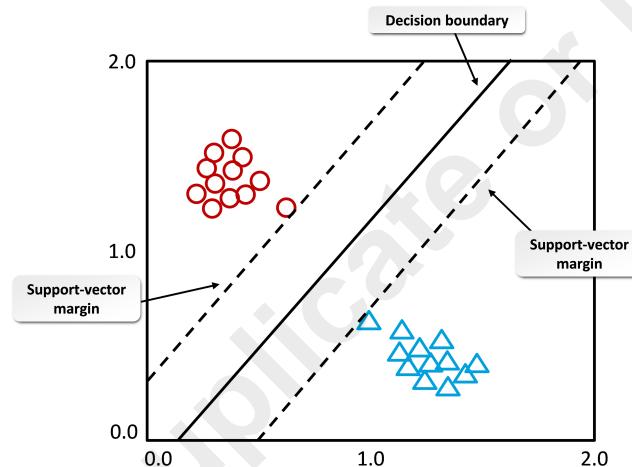


Figure 9–1: An example of a hyperplane used in SVMs.

SVMs for Linear Classification

How a hyperplane is constructed and how it applies to the data are dependent on the type of machine learning problem you're trying to solve. The previous figure demonstrated a hyperplane applied to a linear classification problem; that is, a classification problem whose classes can be reasonably split using a straight line when mapped to a feature space.

In the following figure, the graph on the left demonstrates a decision boundary using a standard linear classification algorithm. The decision boundary in this graph does a good job of separating both classes in the feature space, but it comes very close to the edge data examples. This will likely lead to problems of overfitting, as new test data may not generalize well using this model. Compare this to the model introduced earlier (on the right of this figure), which uses SVMs. The decision boundary is plotted in such a way that it not only splits the classes, but it stays as far away from the edge examples as possible. The dashed lines create the margins of separation by intersecting the edge examples.

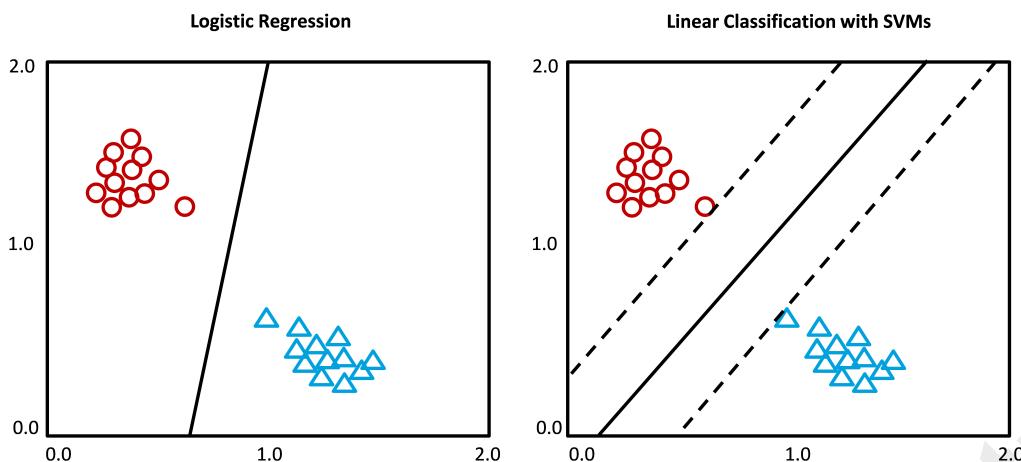


Figure 9-2: A linear classifier without SVMs (left) compared to a linear classifier with SVMs (right).

Now, observe what happens when a new test example is evaluated on the model.

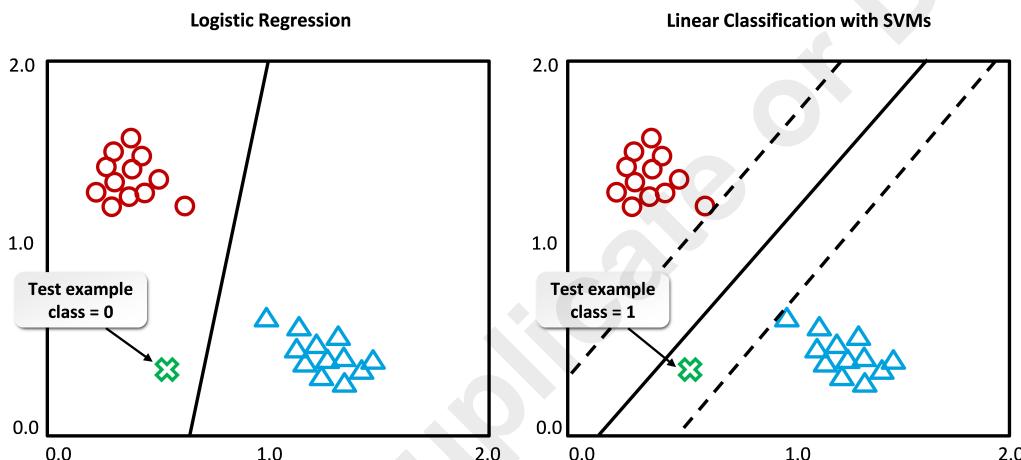


Figure 9-3: Classifying a test example for a model without SVMs (left) and a model with SVMs (right).

Although the test example has the same values when plugged into both models, the model with SVMs classified it differently. SVMs are therefore useful in classification tasks where the training data includes outliers. They tend to outperform logistic regression and other classification algorithms in this regard.

Hard-Margin Classification

The previous figure demonstrated what is known as **hard-margin classification**, or a type of classification in SVMs where all data examples are outside of the margins, and each example is on the "correct" side of the margins. This is sufficient in some situations, but it can cause problems when there are even more extreme outliers. Consider the following figure, in which both Class 0 and Class 1 outliers are plotted close to the data examples in the opposite class.

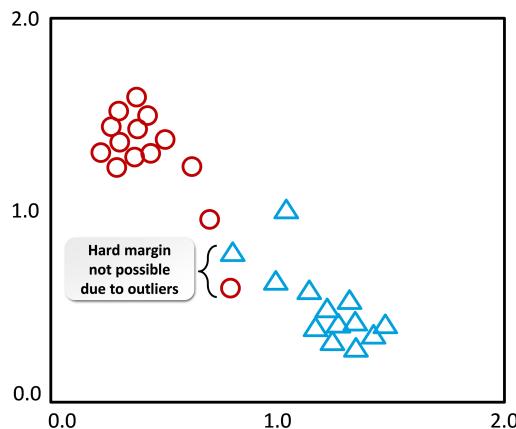


Figure 9-4: Extreme outliers in the training may lead to hard-margin classification failing.

This renders hard-margin classification ineffective; the data examples will either be on the wrong side of the margins, or multiple data examples will be inside the margins. Because there is no way to cleanly separate the data, hard-margin classification fails in this case.

In other cases, an extreme example may lead to a very small distance between the margins, meaning the model will do a poor job at generalizing new test data.

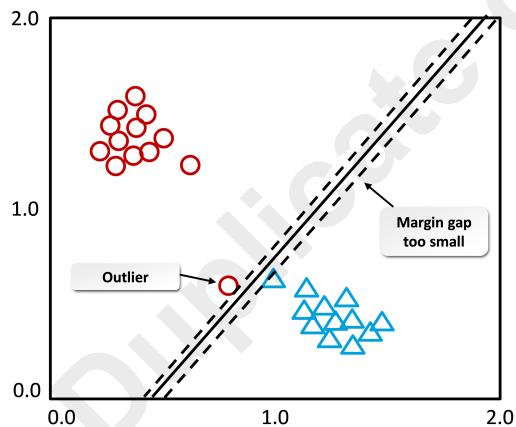


Figure 9-5: A single outlier leading to a poorly fit hyperplane.

Soft-Margin Classification

Soft-margin classification is an approach that strikes a balance between keeping the distance between the margins as large as possible and minimizing the number of examples that end up inside the margins (the support vectors). So, it does not perfectly solve the problem of extreme outliers, but is instead a compromise. Nevertheless, soft-margin classification ends up being more effective than hard-margin classification in such cases. Depending on your toolset, you may be able to tune a hyperparameter to give more weight to one output over another (i.e., wider margins vs. number of examples inside margins).

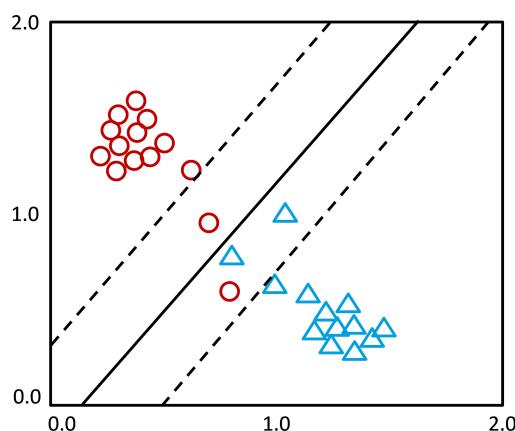


Figure 9–6: A hyperplane plotted using soft-margin classification. The margin gap remains wide at the expense of keeping some outliers within the margins.

SVMs for Non-Linear Classification

The previous examples showed how SVMs can be used for classification tasks where the classes are separable by a straight line. A few extreme outliers can make this challenging, but even in those cases, a straight line is still a reasonably good fit. However, not all datasets are so easily divided. Consider the following figure, in which the data examples plotted on the feature space aren't very accommodating of a straight line.

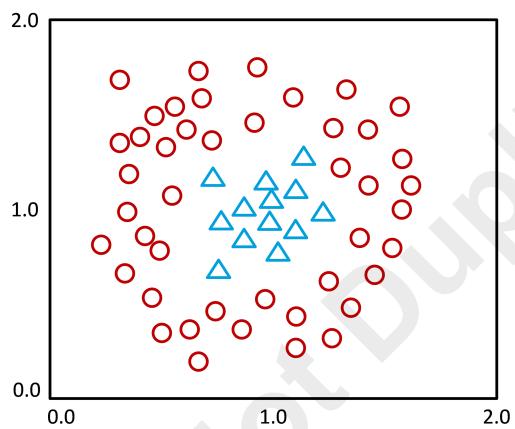


Figure 9–7: A dataset that cannot be fit well using a straight line.

This is therefore a non-linear classification task. A very basic approach to non-linear classification is to simply add more features to the dataset until it becomes linearly separable. Or, you could apply various transformations to each feature to make the data linearly separable. For example, you can add polynomial features (e.g., powers of each initial feature) to the dataset. This may—but is not guaranteed to—lead to classification that supports a straight-line fit. In addition to not guaranteeing linearly separable data, adding features can lead to performance issues, especially if many new features are added. Fortunately, there is a better solution.

Kernel Trick

In SVMs, the **kernel trick** is a group of mathematical methods for efficiently representing non-linearly separable data in higher-dimensional space. Directly mapping features to more dimensions—for example, computing polynomials for every feature—can quickly become computationally expensive. The kernel trick is able to avoid this by representing the data in a feature space that is of infinite possible dimensions, *without* directly computing the transformation of the features into more dimensions.

The actual process of representing the data in higher dimensions is performed by the kernel function. Instead of computing the exact coordinates of the data in the higher dimensions, the kernel function computes a similarity score between data examples by taking the *dot product* of the data in higher dimensions. The dot product is an operation that multiplies vectors together to return a scalar, which is just a single number. Computing this dot product is much easier than computing the actual coordinates.

Consider the following figure. On the left is the feature space for a classification dataset mapped to two dimensions. There is no way to directly separate this data by a straight line. So, the kernel trick takes the data and then represents it in more dimensions—like the three-dimensional graph on the right. Now, the model can draw a hyperplane that separates the data linearly.

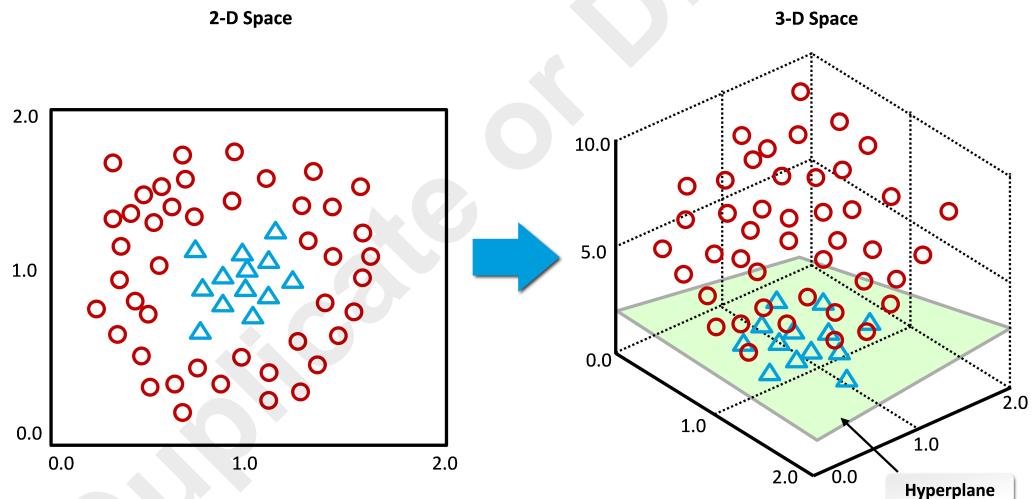


Figure 9-8: The kernel trick representing features from two-dimensional space in three-dimensional space.

Using this hyperplane, the model can make classification decisions much more quickly than if you had just explicitly transformed the features into higher dimensions and computed their coordinates.



Note: The dimension of the hyperplane is the dimension of the feature space minus one. So, in the figure, the hyperplane has two dimensions in three-dimensional space.

Kernel Trick Example

It can be easier to grasp just *how* the kernel trick works, and why it's so useful, by looking at a simple mathematical example. First, it helps to know the basic equation behind the kernel trick:

$$K(x, y) = \langle f(x), f(y) \rangle$$

Where:

Licensed For Use Only By: Abdulwahab Alweban dev.abdulwahab@gmail.com Jan 8 202

- κ refers to the kernel function.
- x and y are the inputs of any n dimension.
- f refers to a function that maps the values from n dimension to any higher dimension.
- $\langle \cdot \rangle$ (angle brackets) represent the dot product of $f(x)$ and $f(y)$.

Start by focusing on the right side of the equation. Suppose you have an x and y of 3 dimensions. So, x is a vector consisting of (x_1, x_2, x_3) . The y vector is likewise (y_1, y_2, y_3) . If you wanted to increase the dimensions of these features, you could square them (3^2), which would be 9. This means that $f(x)$ is equal to:

$$(x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3)$$

And, $f(y)$ would follow this same pattern of multiplying each value by every other value. Using actual numbers, where $x = (0, 1, 2)$ and $y = (3, 4, 5)$, the mapping function would output the following:

$$f(x) = (0, 0, 0, 0, 1, 2, 0, 2, 4)$$

$$f(y) = (9, 12, 15, 12, 16, 20, 15, 20, 25)$$

As you can see, both inputs went from 3 dimensions to 9. Now you can take the dot product of both $f(x)$ and $f(y)$. This is done by looking at the first dimension for both vectors and multiplying the two numbers in that dimension. Then, you'd repeat the same process for dimension 2 all the way to dimension 9. Lastly, you'd add up all of these products to get your ultimate value. So, this would be:

$$\langle f(x), f(y) \rangle = 0 + 0 + 0 + 0 + 16 + 40 + 0 + 40 + 100 = 196$$

The problem here is that mapping the values to higher dimensions first and then doing their dot products is very computationally expensive, especially if your inputs start out in high-dimensional space. When you consider that the end result is simply a one-dimensional value, this type of computation is often not worth the effort. Thankfully, a kernel function acts like a shortcut to arriving at the same answer. It computes the dot product of each initial input value, then maps to higher dimensions (in this case, squaring the inputs to go from 3 to 9):

$$K(x, y) = (0 + 4 + 10)^2 = 196$$

The kernel trick found the same correct answer as the traditional method, only much faster and with much less computational complexity. This is why it is so effective at helping SVMs transform non-linearly separable data in lower dimensions into linearly separable data at higher dimensions.

Kernel Methods

The kernel trick can be implemented in several different ways. The following table describes some of the most popular kernels used with SVMs.

Kernel Method	Description
Linear kernel	<p>This is the simplest type of kernel, and it only applies to data that is linearly separable (unlike other kernels). Linear kernels are useful when the feature space of the training set is already very large, because mapping a large number of features to higher-dimensional space does not always produce performance benefits. Linear kernels tend to be much faster in these circumstances.</p> <p>Linear kernels are popular in text classification due to the high number of features these datasets tend to have. A text document (the data example) can have many different words in it (each word being a feature). Even if your classification problem isn't text-based, it's usually a good idea to start with a linear kernel, assuming your data is already linearly separable.</p>
Gaussian radial basis function (RBF) kernel	<p>This kernel takes the form of a specific type of radial basis function called a Gauss function. This function projects a new feature space in higher dimensions by measuring the distance between all data examples and data examples that are defined as centers.</p> <p>The Gaussian RBF kernel is one of the most popular and effective kernels when using SVMs. It is particularly effective when there are many more data examples than there are features, as this enables projection into higher-dimensional space without much of a performance loss. However, Gaussian RBF may be prone to overfitting, so you must tune the regularization hyperparameter accordingly.</p>
	 <p>Note: The Gaussian RBF kernel is also called the RBF kernel or the Gaussian kernel.</p>
Polynomial kernel	<p>This kernel uses polynomial values as part of its feature space projection. A polynomial kernel takes a hyperparameter that defines the degree by which it projects the feature space. A high polynomial degree might lead to overfitting, and a low degree to underfitting.</p> <p>The polynomial kernel has found some success in natural language processing (NLP) tasks, but it is less popular than Gaussian RBF kernels since it tends to perform worse in many cases.</p>
Sigmoid kernel	<p>This kernel uses a hyperbolic tangent function (\tanh) to create what is essentially equivalent to a type of perceptron neural network. It takes two hyperparameters: the slope and the intercept.</p> <p>Because it approximates a neural network, the sigmoid kernel with SVMs is commonly applied to problems that are often solved by neural networks, such as image classification and object detection.</p>

Guidelines for Building SVM Models for Classification

Follow these guidelines when you are building classification-based support-vector machine (SVM) models.



Note: All of the Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Build an SVM Model for Classification

When building an SVM model for classification:

- Consider using an SVM model when the problem you are trying to solve is sensitive to outliers.
- Consider using an SVM model when working with a high-dimensional dataset.
- In classification, recognize that the goal of an SVM model is to widen the margins as much as is feasible, while at the same time keeping data examples outside of the margins.
- Tune the regularization hyperparameter to adjust the size of the margins.
- Consider that narrowing the margins too much to keep all examples outside of those margins may lead to complications (e.g., overfitting).
- Consider softening the margins to avoid hard-margin overfitting issues.
- Recognize that softening the margins will likely place some examples within those margins, which is often a necessary tradeoff.
- Apply a kernel trick method to SVM models whose training data is not linearly separable.
- Consider the different types of kernel methods and how one might be more applicable to your current problem.

Use Python for Classification-Based SVM Models

The scikit-learn `SVC()` class enables you to construct a classification model with SVMs. The following are some of the objects and functions you can use to build such a model.

- `model = sklearn.svm.SVC(kernel = 'sigmoid', C = 10)` —This constructs an SVM model for classification. In this case, the model is using the sigmoid kernel with 10 as the regularization penalty.
- You can use these class objects to call the same `fit()`, `score()`, and `predict()` methods as before, as well as any of the applicable `metrics` methods.
- `model.support_` —An attribute that returns the indices of the support vectors.
- `model.support_vectors_` —An attribute that returns the support vectors themselves.

ACTIVITY 9–1

Building an SVM Model for Classification

Data File

/home/student/CAIP/SVMs/SVMs-Iris.ipynb

Before You Begin

Jupyter Notebook is open.

Scenario

Another department at the college has expressed interest in your machine learning classification models. A Botany professor wants to demonstrate how the biological components that make up plant life can determine a plant's taxonomic ranks (family, genus, species, etc.). The professor provides you with a dataset of plants and some of their physical attributes. In speaking with this professor, you learn that the dataset may contain several outliers. Rather than create a logistic regression or decision tree-based model, you decide to create a classifier using SVMs. This will hopefully lead to a model with more skill when it comes to classifying data with outliers.

1. Open the notebook and load the dataset.

- a) From Jupyter Notebook, select **CAIP/SVMs/SVMs-Iris.ipynb** to open it.
- b) Select the code cell beneath the **Import software libraries and load the dataset** title, then select **Run**.
- c) Verify that 150 records were loaded.

This dataset, which comes pre-packaged with scikit-learn, is one of the most well-known datasets in machine learning. It includes some physical attributes of three types of flowers within the *Iris* genus.

2. Get acquainted with the dataset.

- a) Scroll down and select the code cell beneath the **Get acquainted with the dataset**, then select **Run**.

- b) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal length (cm)    150 non-null float64
sepal width (cm)     150 non-null float64
petal length (cm)    150 non-null float64
petal width (cm)     150 non-null float64
target               150 non-null int64
dtypes: float64(4), int64(1)
memory usage: 5.9 KB
None

   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  target
0             5.1          3.5            1.4           0.2          0
1             4.9          3.0            1.4           0.2          0
2             4.7          3.2            1.3           0.2          0
3             4.6          3.1            1.5           0.2          0
4             5.0          3.6            1.4           0.2          0
5             5.4          3.9            1.7           0.4          0
6             4.6          3.4            1.4           0.3          0
7             5.0          3.4            1.5           0.2          0
8             4.4          2.9            1.4           0.2          0
9             4.9          3.1            1.5           0.1          0
```

- The training set includes 150 rows and 5 columns.
- All of the columns contain float values, except for the `target` column, which contains integer values. The `target` column is the label of *Iris* species the model must predict, classified as 0, 1, or 2. Each species' label is as follows:
 - Iris setosa* (0)
 - Iris versicolor* (1)
 - Iris virginica* (2)
- There is no missing data; all rows have values for every column.
- The columns describe the length and width of the flower's sepal (a part of the flower that supports the petals) and the length and width of the flower's petals.

3. Examine a general summary of statistics.

- a) Scroll down and select the code cell beneath the **Examine a general summary of statistics** title, then select **Run**.
- b) Examine the output.

```
   sepal length (cm)  sepal width (cm)  petal length (cm) \
count          150.00        150.00        150.00
mean           5.84         3.06         3.76
std            0.83         0.44         1.77
min            4.30         2.00         1.00
25%           5.10         2.80         1.60
50%           5.80         3.00         4.35
75%           6.40         3.30         5.10
max            7.90         4.40         6.90

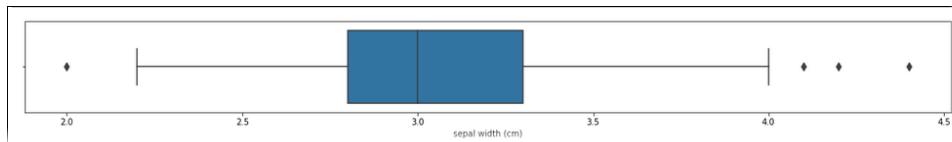
   petal width (cm)  target
count          150.00        150.00
mean           1.20         1.00
std            0.76         0.82
min            0.10         0.00
25%           0.30         0.00
50%           1.30         1.00
75%           1.80         2.00
max            2.50         2.00
```

- This is a very simple and relatively clean dataset. Not much feature engineering needs to be done.
- However, there may be a few outliers that could impact a classification model's skill.

4. Identify outliers.

- a) Scroll down and select the code cell beneath the **Identify outliers** title, then select **Run**.

- b) Examine the output.



There appear to be a few outliers for sepal width.

5. Why are SVMs often better than other algorithms at handling datasets with outliers?

6. Reduce the dimensionality of the dataset.

- a) Scroll down and view the cell titled **Reduce the dimensionality of the dataset**, and examine the code listing below it.

Reduce the dimensionality of the dataset

```
In [ ]: 1 X = iris['data'][:, :2] # Only use first two features (sepal length and sepal width).
2 y = iris['target']
3
4 print("\nBefore reduction:")
5 print("X dataset dimensions are", X.shape)
6 print("y dataset dimensions are", y.shape)
7
8 # Only use labels 0 and 1 (setosa and versicolor).
9 class_labels = (y == 0) | (y == 1)
10 X = X[class_labels]
11 y = y[class_labels]
12
13 print("\nAfter reduction:")
14 print("X dataset dimensions are", X.shape)
15 print("y dataset dimensions are", y.shape)
```

For demonstration purposes, you'll start by looking at only two of the four features (sepal length and sepal width), and considering only two of the three *Iris* species (*setosa* and *versicolor*). Later, you'll train the model on the full dataset.

- b) Select the cell that contains the code listing, then select **Run**.

```
Before reduction:
X dataset dimensions are (150, 2)
y dataset dimensions are (150,)

After reduction:
X dataset dimensions are (100, 2)
y dataset dimensions are (100,)
```

7. Examine the separation between classes using a scatter plot.

- a) Scroll down and select the code cell beneath the **Examine the separation between classes using a scatter plot** title, then select **Run**.

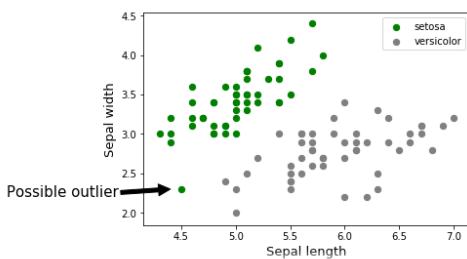
- b) Examine the output.

Examine the separation between classes using a scatter plot

```

1 # Sepal length along x-axis, sepal width along y-axis.
2 scatter_x = X[:, 0]
3 scatter_y = X[:, 1]
4
5 cdict = {0: 'green', 1: 'grey'}
6
7 # Generate scatter plot with legend.
8 for c_label in np.unique(y):
9     if c_label == 0:
10         iris = 'setosa'
11     if c_label == 1:
12         iris = 'versicolor'
13
14     ix = np.where(y == c_label)
15     plt.scatter(scatter_x[ix], scatter_y[ix], c = cdict[c_label], label = iris, s = 40)
16
17 plt.legend()
18 plt.xlabel("Sepal length", fontsize = 13)
19 plt.ylabel("Sepal width", fontsize = 13)
20 plt.annotate('Possible outlier', xy = (4.4, 2.3), xytext = (2.9, 2.2),
               arrowprops = dict(color = 'black'), fontsize = 15);
21

```



This is a basic scatter plot of the two features extracted earlier (sepal length and sepal width).

- The green dots in the top left are the data points for *setosa* flowers.
- The grey dots in the bottom right are the data points for *versicolor* flowers.
- For the most part, these two features seem reasonably separable by a straight decision boundary.
- At least one potential outlier is being called out; this might get misclassified.

8. Plot a decision boundary for a given model.

- a) Scroll down and view the cell titled **Plot a decision boundary for a given model**, and examine the code listing below it.

```

1 def plot_decision_boundary(X, y, model, is_svm):
2     scatter_x = X[:, 0]
3     scatter_y = X[:, 1]
4
5     cdict = {0: 'green', 1: 'grey'}
6
7     for c_label in np.unique(y):
8         if c_label == 0:
9             iris = 'setosa'
10            if c_label == 1:
11                iris = 'versicolor'
12
13            ix = np.where(y == c_label)
14            plt.scatter(scatter_x[ix], scatter_y[ix], c = cdict[c_label], label = iris, s = 40)
15
16    plt.legend()
17    plt.xlabel("Sepal length", fontsize = 13)
18    plt.ylabel("Sepal width", fontsize = 13)
19
20    ax = plt.gca()
21    xlim = ax.get_xlim()
22    ylim = ax.get_ylim()
23
24    # Create grid.
25    xx = np.linspace(xlim[0], xlim[1], 40)
26    yy = np.linspace(ylim[0], ylim[1], 40)
27    YY, XX = np.meshgrid(yy, xx)
28    xy = np.vstack([XX.ravel(), YY.ravel()]).T
29    Z = model.decision_function(xy).reshape(XX.shape) # Use model decision function to plot boundary.
30

```

This function, when called, will plot a decision boundary on the scatter plot shown previously.

- The function requires the training data, label data, and model object as input.
- The `is_svm` argument will determine whether or not to plot the margins (for SVM models).
- Lines 2 through 18 create the scatter plot, same as before.
- Lines 20 through 29 begin creating a grid on which to plot the model's decision function.

```

31     if is_svm == True:
32         # Plot decision boundary and margins.
33         ax.contour(XX, YY, Z, colors = 'r', levels = [-1, 0, 1],
34                    linestyles=['--', '--', '--'])
35
36         # Plot support vectors.
37         ax.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1],
38                    s = 100, linewidth = 1, facecolors = 'none', edgecolors = 'k')
39     else:
40         ax.contour(XX, YY, Z, colors = 'r', levels = [0],
41                    linestyles=['-'])
42
43     plt.show()
44
45 print('Function to plot the decision boundary has been defined.')

```

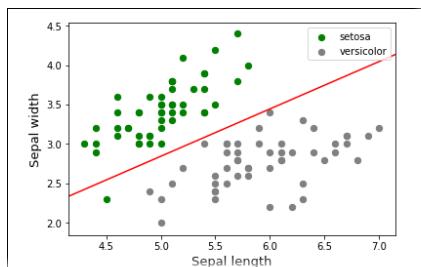
- Lines 31 through 38 plot the decision boundary line, then plot the support-vector margins for SVM models.
 - Lines 39 through 41 just plot the decision boundary for non-SVM models.
 - Line 45 shows a message when the function has been defined.
- b) Select the cell that contains the code listing, then select **Run**.

Function to plot the decision boundary has been defined.

9. Train a basic logistic regression model and plot its decision boundary.

- a) Scroll down and select the code cell beneath the **Train a basic logistic regression model and plot its decision boundary** title, then select **Run**.

- b) Examine the output.



The decision boundary, for the most part, seems to cleanly divide the classes. However:

- The boundary comes pretty close to the edge data examples, potentially leading to overfitting.
- The outlier mentioned earlier has been misclassified.

10. Train an SVM model and plot its decision boundary plus margins.

- a) Scroll down and view the cell titled **Train an SVM model and plot its decision boundary plus margins**, and examine the code listing below it.

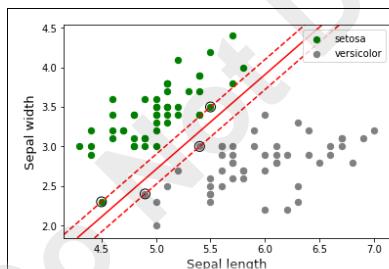
Train an SVM model and plot its decision boundary plus margins

```
In [ ]: 1 from sklearn.svm import SVC
          2
          3 svm = SVC(kernel = 'linear', C = 100, random_state = 1936)
          4 svm.fit(X, y)
          5 plot_decision_boundary(X, y, svm, True)
```

The `SVC()` class implements support-vector classification. In this case, there are two hyperparameters being set:

- `kernel` specifies the kernel method to use; in this case, you'll start with a linear kernel.
- `C` is the regularization penalty that determines the "wideness" of the road (i.e., its margins). A higher penalty leads to narrower margins. You're starting rather high with a value of 100.

- b) Select the cell that contains the code listing, then select **Run**.
c) Examine the output.



- The solid red line is the decision boundary.
- The dashed red lines are the support-vector margins.
- The circled data points are the support vectors.

11. How does this SVM boundary fit to the data as compared to the logistic regression boundary?

12. Reduce the regularization penalty to soften the margin.

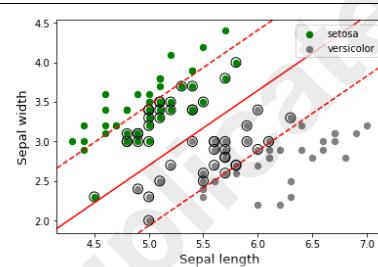
- a) Scroll down and view the cell titled **Reduce the regularization penalty to soften the margin**, and examine the code listing below it.

Reduce the regularization penalty to soften the margin

```
In [ ]: 1 svm = SVC(kernel = 'linear', C = 0.1, random_state = 1936)
          2 svm.fit(X, y)
          3
          4 plot_decision_boundary(X, y, svm, True)
```

While the previous SVM model seemed to do well as compared to logistic regression, the fact that the margins were rather narrow means that one or more outliers may potentially lead to overfitting. So, you'll plot another model where the regularization penalty is much lower, leading to softer margins.

- b) Select the cell that contains the code listing, then select **Run**.
- c) Examine the output.



The margins are much wider now, and more data examples are placed within those margins. This is not necessarily better for this particular case, and may actually lead to underfitting; however, it serves to demonstrate how softening the margins of an SVM model can change its classification decisions. You'll search for the optimal C value shortly, when you train on the full dataset.

13. Split the datasets.

- a) Scroll down and view the cell titled **Split the datasets**, and examine the code listing below it.

Split the datasets

```
In [ ]: 1 from sklearn.model_selection import train_test_split
2
3 label_columns = ['target']
4
5 training_columns = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
6
7 # Split the training and test datasets and their labels.
8 X_train, X_test, y_train, y_test = train_test_split(data_raw[training_columns],
9                                                    data_raw[label_columns],
10                                                   random_state = 1936)
11
12 print('The training and test datasets and their labels have been split.)
```

This splits the datasets in the usual way, incorporating the entire dataset this time (i.e., all four features and all three labels).

- b) Select the cell that contains the code listing, then select **Run**.

The training and test datasets and their labels have been split.

14. Evaluate an SVM model using a holdout test set.

- a) Scroll down and view the cell titled **Evaluate an SVM model using a holdout test set**, and examine the code listing below it.

Evaluate an SVM model using a holdout test set

```
In [ ]: 1 svm = SVC(kernel = 'linear', C = 100, random_state = 1936)
2 svm.fit(X_train, np.ravel(y_train))
3
4 # Score using the test data.
5 score = svm.score(X_test, y_test)
6
7 print('Accuracy: {:.0f}%'.format(score * 100))
```

To start with, you'll just use the arbitrary value of 100 for the regularization penalty.

- b) Select the cell that contains the code listing, then select **Run**.
c) Examine the output.

Accuracy: 92%

- The accuracy for this initial model is 92%.
- You could evaluate the model using all of the usual metrics—precision, recall, F_1 score, etc.—but for this simple dataset, accuracy should suffice.

15. Optimize the SVM model with grid search and cross-validation.

- a) Scroll down and view the cell titled **Optimize the SVM model with grid search and cross-validation**, and examine the code listing below it.

Optimize the SVM model with grid search and cross-validation

```
In [ ]: 1 from sklearn.model_selection import GridSearchCV
2
3 svm = SVC(gamma = 'auto', random_state = 1936)
4
5 grid = [{kernel: ['linear', 'rbf', 'poly', 'sigmoid'],
6           'C': [0.01, 0.1, 1, 5, 10, 25, 50, 100]}]
7
8 search = GridSearchCV(svm, param_grid = grid, scoring = 'accuracy', cv = 5, iid = False)
9 search.fit(X_train, np.ravel(y_train));
10
11 print(search.best_params_)
```

This code performs a grid search to determine the best hyperparameters for an SVM model.

- On line 3, the model will default to using a `gamma` of `auto`. Gamma determines the coefficient to use with non-linear kernels (in this case, 1 divided by the number of features).
- On line 5, the grid search will identify the best kernel to use among the following four:
 - `linear` is the linear kernel.
 - `rbf` is the Gaussian radial basis function (RBF) kernel.
 - `poly` is the polynomial kernel.
 - `sigmoid` is the sigmoid kernel.
- On line 6, the grid search will try these kernels against a list of several `C` values, ranging from 0.01 (very soft margins) to 100 (very hard margins).
- On line 8, the grid search will be optimizing for accuracy and will perform five-fold cross-validation on the training data.

- b) Select the cell that contains the code listing, then select **Run**.

- c) Examine the output.

```
{'C': 0.01, 'kernel': 'poly'}
```

The grid search identified a low `C` value with the polynomial kernel as the optimal hyperparameter combination.

- d) Scroll down and select the next code listing cell.

```
In [ ]: 1 # Score using the test data.
2 score = search.score(X_test, y_test)
3
4 print('Accuracy: {:.0f}%'.format(score * 100))
```

- e) Select **Run**.

- f) Examine the output and confirm that the model's accuracy increased.

```
Accuracy: 95%
```

16. Examine the optimized SVM model's predictions.

- a) Scroll down and select the code cell beneath the **Examine the optimized SVM model's predictions** title, then select **Run**.

- b) Examine the output.

Examine the optimized SVM model's predictions

```

1 # Use test set to evaluate.
2 results_comparison = X_test.copy()
3 results_comparison['Predicted Iris'] = search.predict(X_test)
4 results_comparison['Actual Iris'] = y_test.copy()
5
6 # Map labels to actual Iris names.
7 iris_encode = {0: 'setosa', 1: 'versicolor', 2: 'virginica'}
8
9 results_comparison['Predicted Iris'] = results_comparison['Predicted Iris'].map(iris_encode)
10 results_comparison['Actual Iris'] = results_comparison['Actual Iris'].map(iris_encode)
11
12 # View examples of the predictions compared to actual Iris.
13 results_comparison.head(20)

```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Predicted Iris	Actual Iris
121	5.6	2.8	4.9	2.0	virginica	virginica
67	5.8	2.7	4.1	1.0	versicolor	versicolor
148	6.2	3.4	5.4	2.3	virginica	virginica
77	6.7	3.0	5.0	1.7	versicolor	versicolor
31	5.4	3.4	1.5	0.4	setosa	setosa
7	5.0	3.4	1.5	0.2	setosa	setosa
5	5.4	3.9	1.7	0.4	setosa	setosa
127	6.1	3.0	4.9	1.8	virginica	virginica
146	6.3	2.5	5.0	1.9	virginica	virginica
35	5.0	3.2	1.2	0.2	setosa	setosa
110	6.5	3.2	5.1	2.0	virginica	virginica
29	4.7	3.2	1.6	0.2	setosa	setosa
126	6.2	2.8	4.8	1.8	virginica	virginica
143	6.8	3.2	5.9	2.3	virginica	virginica
58	6.6	2.9	4.6	1.3	versicolor	versicolor
47	4.6	3.2	1.4	0.2	setosa	setosa
90	5.5	2.6	4.4	1.2	versicolor	versicolor
34	4.9	3.1	1.5	0.2	setosa	setosa
92	5.8	2.6	4.0	1.2	versicolor	versicolor
83	6.0	2.7	5.1	1.6	virginica	versicolor

This seems to confirm the high accuracy that was just reported. The first misclassification appears on the last line that was output (example index 83).

17. Shut down this Jupyter Notebook kernel.

- a) From the menu, select **Kernel→Shutdown**.
- b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
- c) Close the **SVMs-Iris** tab in Firefox, but keep a tab open to **CAIP/SVMs/** in the file hierarchy.

TOPIC B

Build SVM Models for Regression

Now that you've built an SVM model for classification, you'll build one to tackle problems of linear regression.

SVMs for Regression

In addition to classification, SVMs can also take on regression tasks. Like with classification, the ideal hyperplane is one that creates the largest possible space between the support-vector margins. However, unlike with classification, you actually want to fit as many examples *within* the margins as possible. Only examples outside of the margins—the support vectors themselves—are incorporated in the cost function's calculations.

The space between the margins is specified using the hyperparameter ε as a threshold. Predictions made by the model must be within the margin space defined by ε . The larger ε is, the more errors there will be in the model's predictions. Likewise, small ε values penalize errors more when calculating the cost function, but may lead to overfitting.

	Note: ε is the Greek letter epsilon.
---	---

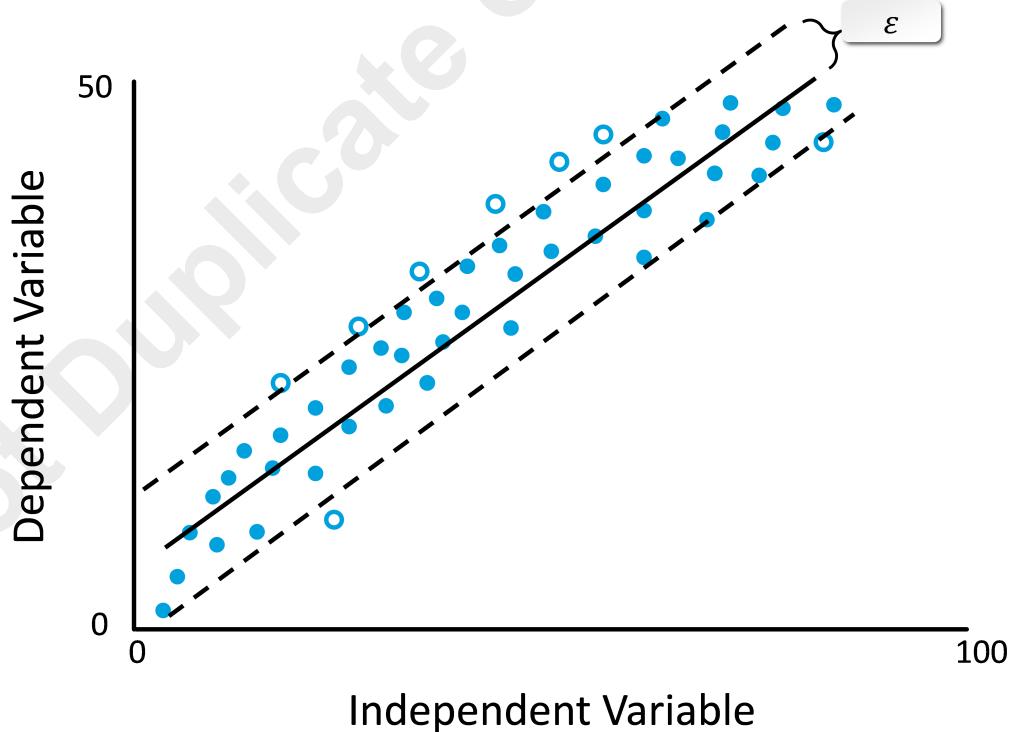


Figure 9–9: SVMs used in linear regression. The hollow circles indicate data examples that are outside of the margins (i.e., support vectors).

As with logistic regression, SVMs tend to handle outliers better than linear regression. Also, SVMs can be applied to non-linear regression tasks. You can use kernel methods like polynomial kernels on non-linear regression tasks to optimize a model's performance when using SVMs.

Guidelines for Building SVM Models for Regression

Follow these guidelines when you are building regression-based support-vector machine (SVM) models.

Build an SVM Model for Regression

When building an SVM model for regression:

- As with classification, consider building an SVM regression model for problems that are sensitive to outliers, and with data that has high dimensionality.
- In regression, recognize that the goal of an SVM model is to widen the margins as much as is feasible, while at the same time keeping data examples inside of the margins.
- Tune the ϵ hyperparameter to adjust the size of the margins.
- Consider that as ϵ increases, the amount of errors increases.
- Consider that as ϵ decreases, the model becomes more prone to overfitting.

Use Python for Regression-Based SVM Models

The scikit-learn `SVR()` class enables you to construct a regression model with SVMs. The following are some of the objects and functions you can use to build such a model.

- `model = sklearn.svm.SVR(kernel = 'rbf', epsilon = 10)` —This constructs an SVM model for regression. In this case, the model is using the Gaussian radial basis function (RBF) kernel with 10 as the space between the margins.
- You can use these class objects to call the same `fit()`, `score()`, and `predict()` methods as before, as well as any of the applicable `metrics` methods.
- You can return the same `support_` and `support_vectors_` attributes, as with classification.

ACTIVITY 9–2

Building an SVM Model for Regression

Data File

/home/student/CAIP/SVMs/SVMs-Boston.ipynb

Before You Begin

Jupyter Notebook is open.

Scenario

Thus far, you've trained the Boston housing dataset on various linear regression models. These models have helped you predict the median value of a house in a particular area, given a number of factors. However, you've noticed that there are some outliers in the dataset that may be negatively affecting the model's predictive skill. There are various ways of addressing outliers, but one of the most effective is using SVMs. So, you'll retrain the dataset, this time on SVM regression models, to see if you can improve your ability to predict housing prices.

1. Load the notebook and dataset.

- From Jupyter Notebook, select **CAIP/SVMs/SVMs-Boston.ipynb** to open it.
- Select the code cell beneath the **Import software libraries and load the dataset** title, then select **Run**.
- Verify that 506 records were loaded.

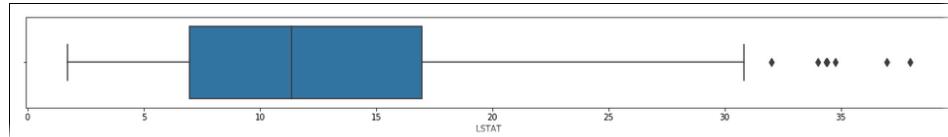
2. Drop columns that won't be used for training.

- Scroll down and select the code cell beneath the **Drop columns that won't be used for training** title, then select **Run**.
- Examine the output.

Recall that, when you trained a linear regression model on this dataset, you dropped the **CHAS** feature ("Charles River dummy variable") because of its weak correlation with the target and because it's a categorical variable. You're doing the same here.

3. Identify outliers.

- Scroll down and select the code cell beneath the **Identify outliers** title, then select **Run**.
- Examine the output.



- LSTAT** is the percentage of the population in an area that qualifies as "low status."
- There appear to be a few outliers toward the higher end of the range of values.
- The percentage of low-status households is usually below 30% for a given area.

4. Reduce the dimensionality of the dataset.

- a) Scroll down and view the cell titled **Reduce the dimensionality of the dataset**, and examine the code listing below it.

Reduce the dimensionality of the dataset

```
In [ ]: 1 X = boston['data'][:, [12]] # Only use one feature (LSTAT).
          2 y = boston['target']
          3
          4 print('Dataset dimensionality reduced.')
```

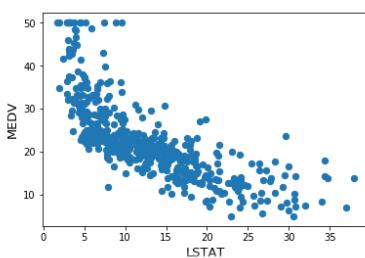
For demonstration purposes, you'll start by looking at only one feature (LSTAT) as compared to the target label (median house value, or MEDV). Later, you'll train the model on the full dataset.

- b) Select the cell that contains the code listing, then select **Run**.

```
Dataset dimensionality reduced.
```

5. Examine a scatter plot of LSTAT and MEDV.

- a) Scroll down and select the code cell beneath the **Examine a scatter plot of LSTAT and MEDV** title, then select **Run**.
 b) Examine the output.



- As you might expect, a house's value tends to decrease as the percentage of low-status households in the area increases.
- You can also see some of the outliers at the right end of the x-axis.

6. Plot a regression line for a given model.

- a) Scroll down and view the cell titled **Plot a regression line for a given model**, and examine the code listing below it.

```

1 def plot_regression(X, y, model, is_svm):
2     plt.scatter(X, y, s = 40)
3     plt.xlabel("LSTAT", fontsize = 13)
4     plt.ylabel("MEDV", fontsize = 13)
5
6     predict = model.predict(X)
7
8     if is_svm == True:
9         # Calculate margins based on epsilon value.
10        high_margin = predict + model.epsilon
11        low_margin = predict - model.epsilon
12
13        # Sort arrays for plotting.
14        X_sort, predict_sort, high_margin_sort, low_margin_sort = \
15        zip(*sorted(zip(X, predict, high_margin, low_margin)))
16
17        plt.plot(X_sort, predict_sort, c = 'r', linestyle = '--') # Plot regression line.
18        plt.plot(X_sort, high_margin_sort, c = 'r', linestyle = '---') # Plot upper margin.
19        plt.plot(X_sort, low_margin_sort, c = 'r', linestyle = '---') # Plot lower margin.
20
21        # Plot support vectors.
22        plt.scatter([model.support_], y[model.support_],
23                    s = 100, linewidth = 1, facecolors = 'none', edgecolors = 'k')
24
25    else:
26        X_sort, predict_sort = \
27        zip(*sorted(zip(X, predict)))
28
29        plt.plot(X_sort, predict_sort, c = 'r', linestyle = '--')
30
31    plt.show()
32
33 print('Function to plot the regression line has been defined.')

```

This function, when called, will plot a regression line on the scatter plot shown previously.

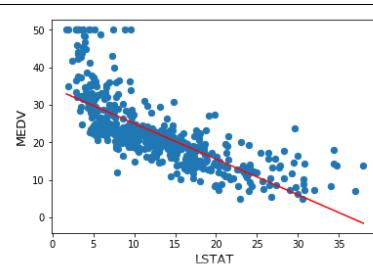
- The function requires the training data, label data, and model object as input.
- The `is_svm` argument will determine whether or not to plot the margins (for SVM models).
- Lines 2 through 4 create the scatter plot, same as before.
- Lines 8 through 23 plot the regression line, then plot the support-vector margins for SVM models.
- Lines 25 through 29 just plot the regression line for non-SVM models.
- Line 33 shows when the function has been defined.

- b) Select the cell that contains the code listing, then select **Run**.

Function to plot the regression line has been defined.

7. Train a basic linear regression model and plot its line of best fit.

- a) Scroll down and select the code cell beneath the **Train a basic linear regression model and plot its line of best fit** title, then select **Run**.
- b) Examine the output.



The regression line doesn't appear to account for the outliers all that well, which can lead to underfitting.

8. Train an SVM model and plot its regression line plus margins.

- a) Scroll down and view the cell titled **Train an SVM model and plot its regression line plus margins**, and examine the code listing below it.

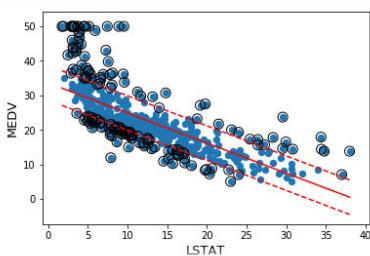
Train an SVM model and plot its regression line plus margins

```
In [ ]: 1 from sklearn.svm import SVR
          2
          3 svm = SVR(kernel = 'linear', epsilon = 5)
          4 svm.fit(X, y)
          5
          6 plot_regression(X, y, svm, True)
```

The `SVR()` class implements support-vector regression. In this case, there are two hyperparameters being set:

- `kernel` specifies the kernel method to use; in this case, you'll start with a linear kernel.
- `epsilon (ε)` determines the space between the margins. A higher value leads to wider margins. You're starting with a value of 5.

- b) Select the cell that contains the code listing, then select **Run**.
 c) Examine the output.



- The solid red line is the regression line.
- The dashed red lines are the support-vector margins.
- The circled data points are the support vectors.
- The regression line with SVMs appears to have shifted upward slightly, centering more on the overall spread of the dataset. This may help the model generalize better with the presence of outliers in the training.

9. How does SVM regression differ from SVM classification in terms of how the data examples are included or not included within the margins?

10. Adjust the margins using a different `epsilon` value.

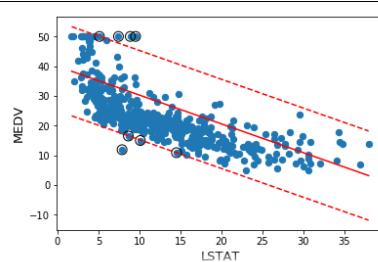
- a) Scroll down and view the cell titled **Adjust the margins using a different epsilon value**, and examine the code listing below it.

Adjust the margins using a different epsilon value

```
In [ ]: 1 svm = SVR(kernel = 'linear', epsilon = 15)
2 svm.fit(X, y)
3
4 plot_regression(X, y, svm, True)
```

While the previous SVM model seemed to do well in incorporating outliers as compared to linear regression, the margins may still be too narrow, leading to underfitting. So, you'll plot another model where the `epsilon` value is larger, leading to wider margins.

- b) Select the cell that contains the code listing, then select **Run**.
 c) Examine the output.



The margins are much wider now, and more data examples are placed within those margins. This is not necessarily better for this particular case, and may actually lead to overfitting. You'll search for the optimal `epsilon` value shortly, when you train on the full dataset.

11. Split the datasets.

- a) Scroll down and view the cell titled **Split the datasets**, and examine the code listing below it.

Split the datasets

```
In [ ]: 1 from sklearn.model_selection import train_test_split
2
3 label_columns = ['target']
4
5 # Split the training and test datasets and their labels.
6 X_train, X_test, y_train, y_test = train_test_split(data_raw.loc[:, 'CRIM': 'LSTAT'],
7 data_raw[label_columns],
8 random_state = 2)
9
10 print('Training and test datasets and their labels have been split.')
```

This splits the datasets in the usual way, incorporating the entire dataset this time.

- b) Select the cell that contains the code listing, then select **Run**.

```
Training and test datasets and their labels have been split.
```

12. Standardize the features.

- a) Scroll down and view the cell titled **Standardize the features**, and examine the code listing below it.

Standardize the features

```
In [ ]: 1 def standardize(X):
2     result = X.copy()
3
4     for feature in X.columns:
5         result[feature] = (X[feature] - X[feature].mean()) / X[feature].std() # z-score formula.
6
7     return result
8
9 X_train = standardize(X_train)
10 X_test = standardize(X_test)
11
12 print('The features have been standardized.')
```

Recall that you're scaling the features on this dataset so that the regularization penalty is applied equally.

- b) Select the cell that contains the code listing, then select **Run**.

The features have been standardized.

13. Evaluate an SVM model using a holdout test set.

- a) Scroll down and view the cell titled **Evaluate an SVM model using a holdout test set**, and examine the code listing below it.

Evaluate an SVM model using a holdout test set

```
In [ ]: 1 from sklearn.metrics import mean_squared_error as mse
2
3 svm = SVR(kernel = 'linear', epsilon = 5)
4 svm.fit(X_train, np.ravel(y_train))
5
6 predict = svm.predict(X_test)
7
8 # Calculate cost using the test data.
9 cost = mse(y_test, predict)
10
11 print('Cost (mean squared error): {:.2f}'.format(cost))
```

To start with, you'll just use the arbitrary value of 5 for `epsilon`.

- b) Select the cell that contains the code listing, then select **Run**.
c) Examine the output.

Cost (mean squared error): 21.17

The mean squared error (MSE) for this model is 21.17. This is somewhat improved over your previous linear regression models, but there's still more opportunity to improve it further.

14. Optimize the SVM model with grid search and cross-validation.

- a) Scroll down and view the cell titled **Optimize the SVM model with grid search and cross-validation**, and examine the code listing below it.

Optimize the SVM model with grid search and cross-validation

```
In [ ]: 1 from sklearn.model_selection import GridSearchCV
2
3 svm = SVR(gamma = 'auto')
4
5 grid = [{kernel: ['linear', 'rbf', 'poly', 'sigmoid'],
6 'C': [0.01, 0.1, 1, 5, 10, 25, 50, 100],
7 'epsilon': [0.01, 0.1, 1, 5, 10, 25, 50, 100]}]
8
9 search = GridSearchCV(svm, param_grid = grid, scoring = 'neg_mean_squared_error', cv = 5, iid = False)
10 search.fit(X_train, np.ravel(y_train));
11
12 print(search.best_params_)
```

This code performs a grid search to determine the best hyperparameters for an SVM model.

- On line 3, the model will default to using a `gamma` of `auto`. Like with the `SVC()` class, `gamma` determines the coefficient to use with non-linear kernels (in this case, 1 divided by the number of features).
- On line 5, the grid search will identify the best kernel to use among the same four kernels as before.
- On line 6, the grid search will try these kernels against a list of several `C` values, ranging from 0.01 to 100. Unlike with classification, the `C` in SVM regression does not determine the margin width, but rather just applies a regularization penalty.
- On line 7, the grid search will try the same range of values for `epsilon`, which determines the width of the margins.
- On line 9, the grid search will be optimizing for MSE and will perform five-fold cross-validation on the training data.

- b) Select the cell that contains the code listing, then select **Run**.

This will take a few moments to complete.

- c) Examine the output.

```
{'C': 100, 'epsilon': 1, 'kernel': 'rbf'}
```

The grid search identified a `C` value of 100 and an `epsilon` value of 1, along with the RBF kernel, as the optimal hyperparameter combination.

- d) Scroll down and select the next code listing cell.

```
In [ ]: 1 predict = search.predict(X_test)
2
3 cost = mse(y_test, predict)
4
5 print('Cost (mean squared error): {:.2f}'.format(cost))
```

- e) Select **Run**.

- f) Examine the output and confirm that the model's error decreased.

```
Cost (mean squared error): 8.15
```

15. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **SVMs-Boston** tab in Firefox, but keep a tab open to the file hierarchy in Jupyter Notebook.

Summary

In this lesson, you built SVM models for both classification and regression purposes. Like other alternative algorithms, SVMs are not necessarily better in all circumstances, but they do offer another approach that might be ideal in datasets with a large feature space and many outliers.

In your own business environment, how might you use SVMs to solve classification problems?

In your own business environment, how might you use SVMs to solve regression problems?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Do Not Duplicate Or Distribute

10

Building Artificial Neural Networks

Lesson Time: 4 hours, 30 minutes

Lesson Introduction

All of the algorithms discussed thus far fall under the general umbrella of machine learning. While they are powerful and complex in their own right, the algorithms that make up the subdomain of deep learning—called artificial neural networks (ANNs)—are even more so. In this lesson, you'll build ANNs that can tackle the same basic types of tasks (regression, classification, etc.), while being better suited to solving more complicated and data-rich problems. These include problems in the domains of computer vision and natural language processing (NLP).

Lesson Objectives

In this lesson, you will:

- Build multi-layer perceptrons (MLP).
- Build convolutional neural networks (CNN).
- Build recurrent neural networks (RNN).

TOPIC A

Build Multi-Layer Perceptrons (MLP)

Before you build neural networks that can tackle computer vision and NLP problems, you need to first build the foundational type of ANN—the multi-layer perceptron (MLP).

Artificial Neural Network (ANN)

An **artificial neural network (ANN)** is a machine approximation of biological neural networks—like the connective structure of the human brain—for the purpose of learning. Like the brain, ANNs include nodes that can transmit information to other nodes, called neurons. Artificial neurons are connected to one another similar to how biological neurons are connected via synapses. These neurons are grouped into various layers. Each layer can perform a specific function and share its output with other layers to create a highly integrated platform for making intelligent decisions.

Although each individual neuron may be relatively simple, a network comprising millions upon millions of interconnected neurons is able to perform very complex calculations that are not always possible with other types of machine learning models. This has made ANNs a popular choice for solving complex problems in the areas of natural language processing (NLP) and computer vision, as well as other domains where large volumes of training data are available.

Perceptron

A **perceptron** is an algorithm used in ANNs to solve binary classification problems. It is, in fact, one of the simplest implementations of an ANN. A perceptron is composed of two layers of neurons: an input layer and an output layer. A neuron in the input layer feeds a number into the neurons of the output layer, which uses that input to make a classification decision.

How the neurons make this classification decision is as follows: The input neurons are each given numerical weights w_n where n is the total number of inputs. This is combined into \mathbf{w} , a vector of the weights. The output neuron then calculates the weighted sum of the inputs. This weighted sum is then applied to a *step function*, a function of horizontal lines that look like a series of steps when graphed. The step function most often used is called the Heaviside step function or the unit step function. It simply outputs to 0 if the weighted sum is less than 0, and outputs to 1 if the weighted sum is greater than or equal to 1. This, in effect, outputs the binary classification you're trying to solve for. The output neuron that calculates this weighted sum and then implements a step function is called a **threshold logic unit (TLU)**.

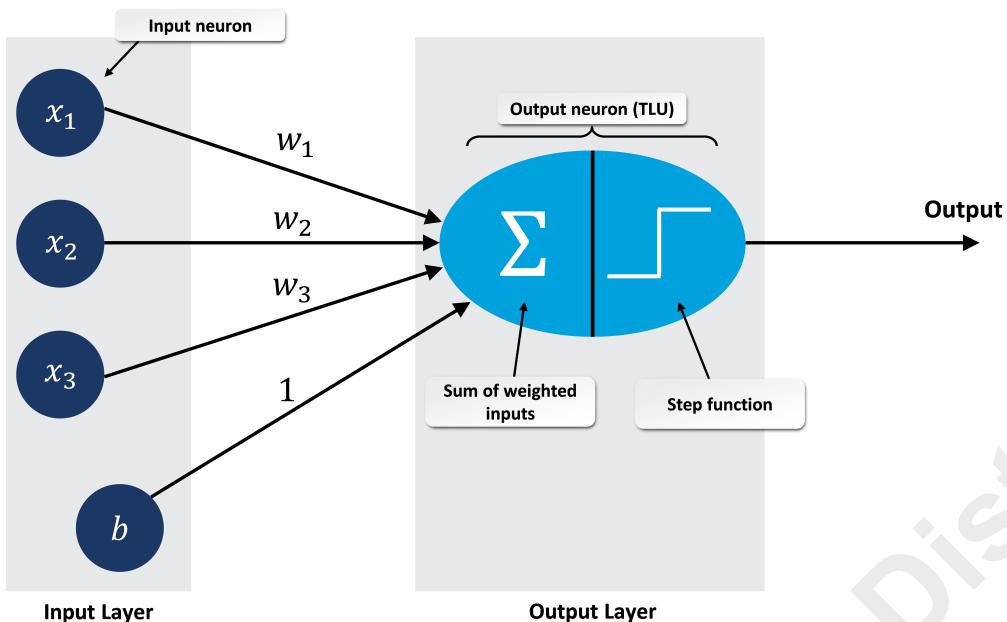


Figure 10–1: A simple perceptron with several inputs and one TLU output neuron.



Note: In the figure, b refers to the bias term. The bias term is a neuron that always outputs to 1. This is very similar in purpose to the intercept (b) in linear algebra—it helps create a better fit by shifting the function curve along the horizontal axis.

Multi-Label Classification Perceptron

The previous example showed a simple perceptron with only one output neuron (TLU), but perceptrons can have multiple output neurons in one layer. For example, you might have a multi-label classification problem where each label is an output neuron. Each label is still a binary classifier—outputting 0 or 1—but in this case, an input can be labeled in multiple ways. All input neurons are connected to all output neurons in the proceeding layer, as in the following figure.

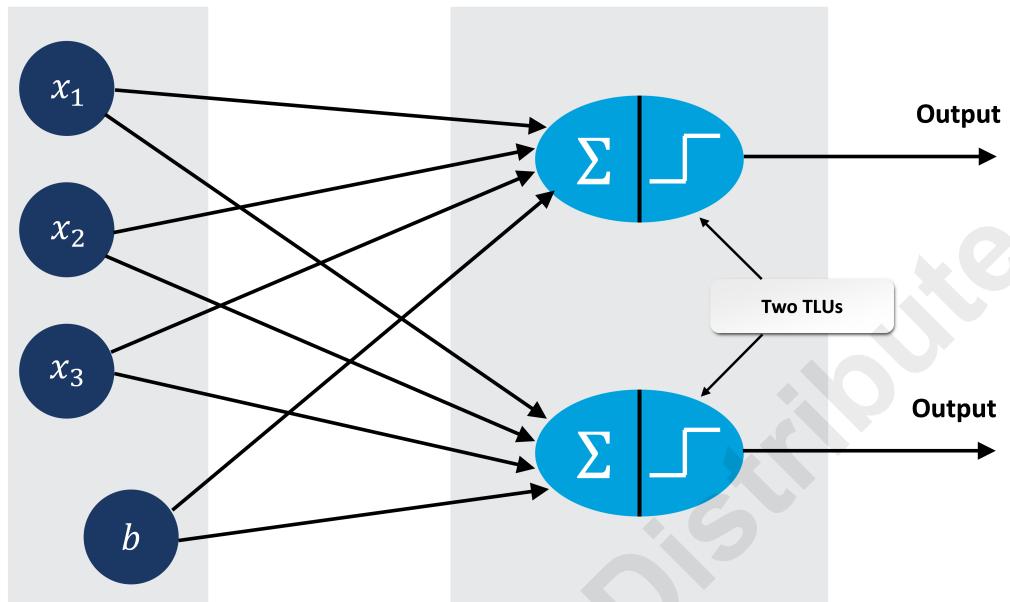


Figure 10–2: A perceptron that solves a multi-label classification problem.

Perceptron Training

The process of training a perceptron is based on the idea that the weights between input and output neurons are increased in magnitude when those inputs lead to the correct output predictions. This effectively strengthens the connection between neurons. When inputs lead to the wrong prediction —i.e., there are errors in the network—the connection between the relevant neurons is not strengthened.

Training a perceptron therefore involves the following basic process:

1. A data example from the training dataset is fed into the neural network.
2. A prediction is made for this data example.
3. For an output neuron that made an incorrect prediction, the input neurons that would have led to the correct prediction have their weights increased with that output neuron.



Note: The weights that a perceptron estimates during training are equivalent to the θ model parameters of a linear model.

The process of updating the neurons' weights can be expressed as an equation:

$$w'_i = w_i + \eta(y_j - \hat{y}_j)x_i$$

Where:

- w_i is the weight between the i^{th} input neuron and an output neuron.
- w'_i is the updated weight value (i.e., the next step).
- η is the learning rate.
- y_j is the actual label of the j^{th} output neuron.
- \hat{y}_j is the predicted value of the j^{th} output neuron.
- x_i is the value of the i^{th} training example input.



Note: This equation is very similar to the equation for stochastic gradient descent.



Note: Unlike with logistic regression classifiers, a perceptron classifier will directly produce either a 0 or a 1, rather than a probability value between 0 and 1.

Perceptron Shortcomings

One of the shortcomings of the standard single-layer perceptron is that it cannot adequately solve problems that aren't linearly separable. One such example is exclusive OR (XOR) logic—an operation that outputs to true *only* if one input is true and the other input is false. The following figure demonstrates this.

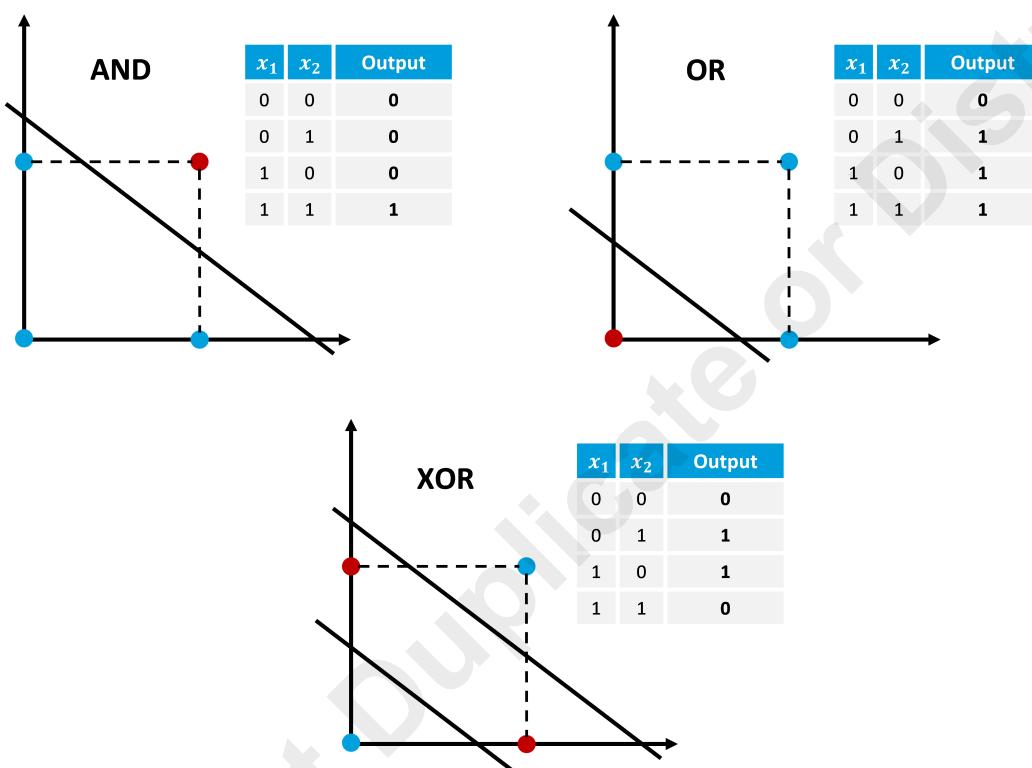


Figure 10-3: The XOR operation is not linearly separable.

The graphs at the top represent AND and OR operations. For both operations, there are three cases that produce a specific outcome, and one case that produces the opposite outcome. When graphed, this makes them linearly separable; for example, in the AND operation, you can draw a single straight line through the graph to separate the single true statement (1, 1) from the three remaining false statements. The XOR operation, on the other hand, cannot be separated like this. The outcomes are split evenly—two true, two false. You'd need two lines to separate them, which won't work for a single perceptron.

Thankfully, problems like these are solvable through multi-layer perceptrons.

Multi-Layer Perceptron (MLP)

A **multi-layer perceptron (MLP)** is a neural network algorithm that has multiple distinct layers of TLUs, rather than just one layer. Both input layers and output layers still exist, but there is one new

Licensed For Use Only By: Abdulwahab Alweban dev.abdulwahab@gmail.com Jan 8 202

type of layer: the hidden layer. The hidden layer is just a layer of TLUs that sits between the input and output layers. There may only be one hidden layer, or there may be many; there is no theoretical limit. A hidden layer's purpose is to add more complexity and sophistication to the neural network; otherwise, its neurons are not much different than those in the output layer.

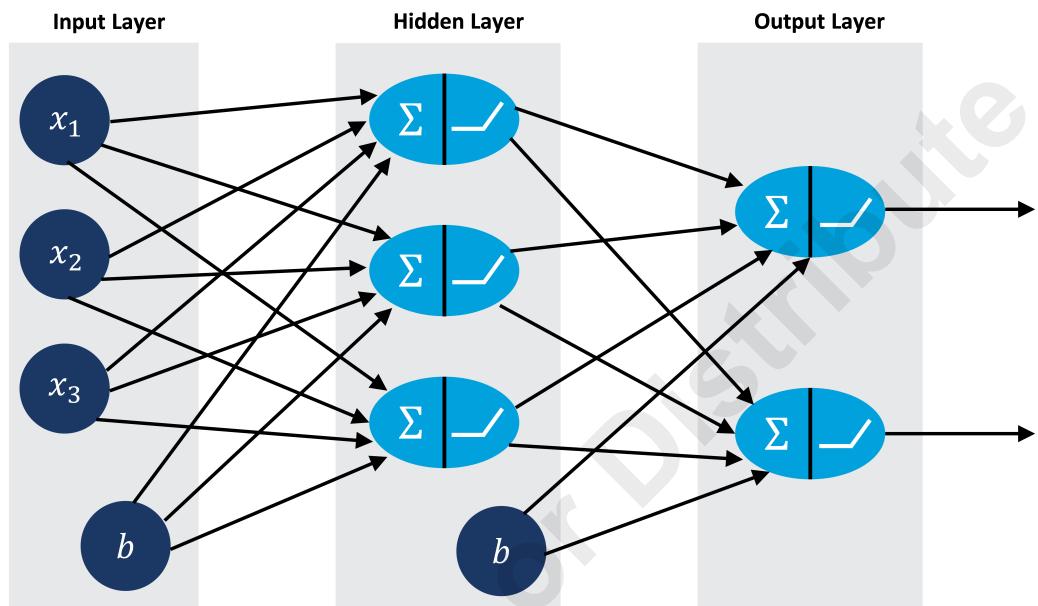


Figure 10–4: The presence of at least one hidden layer defines a multi-layer perceptron.

Note that each neuron in a hidden layer is connected to all of the neurons in the output layer. Also, the hidden layer has a bias term.

	Note: The step function symbol has been replaced by an activation function symbol. Activation functions will be discussed shortly.
	Note: A neural network with multiple hidden layers is also referred to as a deep neural network (DNN).

ANN Layers

To recap, there are three main types of layers within an MLP neural network. The following list describes each layer using an example of software that detects faces within images:

- **Input layers**—these layers deal with information that is directly exposed to the input. For example, the raw pixels in an image are easily visible to the network, but they are not particularly useful by themselves.
- **Hidden layers**—these layers are not directly exposed to the input and require additional analysis. For example, determining the edges of shapes can occur here, as can higher-level activities like identifying eyes, noses, etc.
- **Output layers**—these layers format and output data that is relevant to the problem. For example, this could be the determination that "this is a face" or "this is not a face."

Backpropagation

Backward propagation, commonly called **backpropagation**, is the method by which an MLP neural network is trained. Its basic process is as follows:

1. A data example from the training dataset is fed into the neural network.
2. A prediction is made for this data example.
3. The error between actual and predicted values is computed.
4. Starting from the last hidden layer, it measures how much each neuron in this hidden layer contributed to the error in each output layer neuron (i.e., the error gradient).
5. The previous step is repeated for the second-to-last hidden layer, then the hidden layer before that, etc., until the input layer is all that remains.
6. The connection weights are updated as necessary.

So, in effect, the error calculations start at the end of the network and then work backward. This enables the algorithm to more efficiently calculate the error gradients, as the calculations of one layer are used in the calculations of the layer before it.

Activation Functions

Unlike with single-layer perceptrons, MLPs using backpropagation do not use a step function. Step functions produce flat surfaces, not error gradients. So, the step function in an MLP is replaced with one of several **activation functions**. An activation function computes the output of a neuron to solve non-linear tasks. There are several different types of activation functions, and you are able to mix and match them at different points in the network. For example, some activation functions are better applied to the output layer, whereas others are better for hidden layers.

Three commonly used activation functions are described in the following table.

Activation Function	Description
Sigmoid function	This is the original activation function used with MLPs. Like when used in logistic regression, this outputs an S-shaped curve to account for non-linear data. The output range of the sigmoid function is between 0 and 1 to help constrain the values. This function is a good choice for the output layer of a classifier. If applied to hidden layers, the backpropagation of the gradient error may become too slow.
Hyperbolic tangent (tanh) function	This is essentially a scaled version of the sigmoid function. Instead of being constrained to output from 0 to 1, it outputs from -1 to 1. The tanh function therefore has a more normalized output than the sigmoid function due to centering on 0. It can also do a better job of reducing bias in the error gradients.
Rectified linear unit (ReLU) function	This function calculates a linear function of the inputs. If the result is positive, it outputs that result. If it is negative, it outputs 0. So, the ReLU function does not have a maximum value constraint like the others. Because it outputs to 0 if the input function is negative, it helps make the network "sparse"—in other words, only around 50% of neurons in hidden layers are activated, increasing training performance. However, ReLU is susceptible to the vanishing gradients problem, in which some neurons become inactive. This prevents the error gradient from propagating backward, reducing the model's skill. Several variants of ReLU, like <i>leaky ReLU</i> , can mitigate this issue. These ReLU variants are usually the recommended functions to use for the network's hidden layers.

Guidelines for Building MLPs

Follow these guidelines when you are building multi-layer perceptrons (MLPs).



Note: All of the Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Build an MLP

When building an MLP:

- Consider that an MLP can be used to solve multiple types of machine learning tasks, including classification and regression.
- Consider using neural networks when you have extremely large datasets—datasets with upward of hundreds of thousands, and even millions of data examples.
- Scale your dataset's features before training them on the network.
- Consider selecting the tanh or rectified linear unit (ReLU) activation functions for the hidden layer(s).
- Consider using the sigmoid activation function for the output layer of a classification problem.
- Consider that a ReLU variant like leaky ReLU can help overcome the vanishing gradients problem.
- Consider starting with just a couple hidden layers, as this will be sufficient for many types of tasks and will require much less training time.
- For complex tasks where time is less of a concern, consider gradually increasing the number of hidden layers until you start to overfit the training data.
- Consider building each hidden layer with the same or similar number of neurons.
- As with the number of hidden layers, start with a small number of neurons per layer and then gradually increase that number until the network starts overfitting.

Use Python for MLPs

The scikit-learn `MLPClassifier()` and `MLPRegressor()` classes enable you to construct an MLP neural network. The following are some of the objects and functions you can use to build such a network.

- `model = sklearn.neural_network.MLPClassifier(hidden_layer_sizes = (5, 5, 5), activation = 'relu')` —This constructs an MLP to use for classification. In this case, three hidden layers will be constructed, each with five neurons. The ReLU activation function is being applied to the hidden layer neurons.
- `model = sklearn.neural_network.MLPRegressor(hidden_layer_sizes = (2, 2), activation = 'tanh')` —This constructs an MLP to use for regression. In this case, two hidden layers will be constructed, each with two neurons. The tanh activation function is being applied to the hidden layer neurons.
- You can use these class objects to call the same `fit()`, `score()`, `predict()`, and `predict_proba()` (classification only) methods as before, as well as any of the applicable `metrics` methods.
- `model.coefs_` —An attribute that returns the weights between the neurons of each layer.
- `model.intercepts_` —An attribute that returns the bias terms for each layer.
- `model.loss_curve_` —An attribute that returns the change in loss values for each iteration of a gradient descent solver.

ACTIVITY 10-1

Building an MLP

Data Files

/home/student/CAIP/Neural Networks/NeuralNetworks-Occupancy.ipynb
 /home/student/CAIP/Neural Networks/VisualizeNN.py
 /home/student/CAIP/Neural Networks/occupancy_data/test.csv
 /home/student/CAIP/Neural Networks/occupancy_data/train.csv

Before You Begin

Jupyter Notebook is open.

Scenario

You work for IOT Company, which sells building automation systems. Sensors are placed all throughout a building that measure various physical attributes of the immediate room. In order to make the system smarter, you want it to be able to detect the presence of people in a room, and react accordingly. For example, if there is someone in the room, the system might turn up the heat to more comfortable levels; and when no one is in the room, the system turns down the heat to conserve energy. Or, the system might have a virtual assistant that will offer its services when the room is occupied, and then turn off when it is not.

You've been given a labeled dataset that has sensor measurements for an office room, taken every minute for several weeks. The room is classified as either occupied or not occupied. Since there are thousands of data examples—one for each minute of observation—you feel you have a large enough dataset to make use of a neural network. So, you'll create a classification model using a multi-layer perceptron (MLP).

1. Load the notebook and dataset.

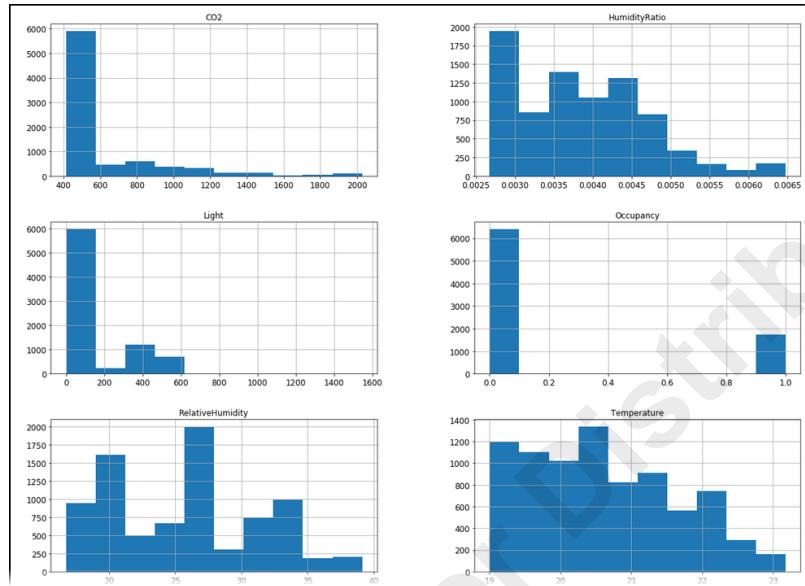
- From Jupyter Notebook, select **CAIP/Neural Networks/NeuralNetworks-Occupancy.ipynb** to open it.
- Select the code cell beneath the **Import software libraries and load the dataset** title, then select **Run**.
- Verify that **train.csv** and **test.csv** are in the project folder, and that they were loaded with 8,143 and 2,665 records, respectively. This dataset already comes pre-split for training and testing.

2. Get acquainted with the dataset.

- Scroll down and select the cell beneath the **Get acquainted with the dataset** title, then select **Run**.
- Examine the output.
 - The training set includes 8,143 rows and 7 columns.
 - Most of the columns contain float values and are measurements of some physical aspect of the room.
 - The columns that don't contain floats are `Date` (string objects) and `Occupancy` (integers). The `Occupancy` column is the label that specifies if a room is not occupied (0) or if it is occupied (1).
 - Each row is a point in time specified by the `Date` column. The room's measurements were taken in intervals of one minute. Because this is a string object, you'll need to find some way to handle these time values. You could drop the `Date` column entirely, but it's highly likely that time has an effect on whether or not a room is occupied.
 - There is no missing data; all rows have values for every column.

3. Examine the distribution of various features.

- Scroll down and select the code cell beneath the **Examine the distribution of various features** title, then select **Run**.
- Examine the output.



- Several features are right-skewed, especially CO2 and Light.
- The distribution for RelativeHumidity seems to have alternating peaks.

4. Examine a general summary of statistics.

- Scroll down and select the code cell beneath the **Examine a general summary of statistics** title, then select **Run**.
- Examine the output.

	Temperature	RelativeHumidity	Light	CO2	HumidityRatio	\
count	8143.000	8143.000	8143.000	8143.000	8143.000	
mean	20.619	25.732	119.519	606.546	0.004	
std	1.017	5.531	194.756	314.321	0.001	
min	19.000	16.745	0.000	412.750	0.003	
25%	19.700	20.200	0.000	439.000	0.003	
50%	20.390	26.223	0.000	453.500	0.004	
75%	21.390	30.533	256.375	638.833	0.004	
max	23.180	39.117	1546.333	2028.500	0.006	
 Occupancy						
count	8143.000					
mean	0.212					
std	0.409					
min	0.000					
25%	0.000					
50%	0.000					
75%	0.000					
max	1.000					

- This dataset has uneven scaling.
- Light and CO2 have comparatively high values.
- Temperature and RelativeHumidity have comparatively low values.
- HumidityRatio has an especially tiny scale; its max value is ~0.006.
- MLPs are highly sensitive to scaling, so you'll need to transform these features before training the model.

5. Split the label from the datasets.

- Scroll down and select the code cell beneath the **Split the label from the datasets**, then select **Run**.

- b) Verify that the labels for both training and test sets were split from the rest of the data.

6. Convert the Date column to datetime format for processing.

- a) Scroll down and view the cell titled **Convert the Date column to datetime format for processing**, and examine the code listing below it.

Convert the Date column to datetime format for processing

```
In [ ]: 1 X_train['Date'] = pd.to_datetime(X_train['Date'])
          2 X_test['Date'] = pd.to_datetime(X_test['Date'])
          3
          4 X_train.head()
```

The pandas `to_datetime()` function converts a loosely defined object with date and time values into a rigidly defined datetime format. It's quite smart at parsing the many different ways to represent date and time.

- b) Select the cell that contains the code listing, then select **Run**.
 c) Examine the output.

	Date	Temperature	RelativeHumidity	Light	CO2	HumidityRatio
0	2015-02-05 19:37:00	21.200	19.840	0.0	525.333333	0.003082
1	2015-02-04 22:11:00	21.390	25.700	0.0	475.000000	0.004046
2	2015-02-09 13:51:00	21.245	32.925	474.5	1126.500000	0.005146
3	2015-02-05 20:36:00	21.200	19.390	0.0	472.500000	0.003012
4	2015-02-10 02:39:00	20.290	32.900	0.0	460.000000	0.004846

- The datetime format makes it much easier to work with date and time values.
- Your next step is to extract each relevant date and time component (year, month, day, hour, etc.) and place it in its own feature column. That way, each component will have some significance on the output.
- However, some date and time components may stay the same for all examples. If every measurement was taken in the same year, there's not much point in making it a feature.

7. Determine which datetime components have unique values.

- a) Scroll down and select the code cell beneath the **Determine which datetime components have unique values** title, then select **Run**.
 b) Examine the output.

```
Unique years: [2015]
Unique months: [2]
Unique days: [ 5  4  9 10  6  8  7]
Unique hours: [19 22 13 20  2 21  7  6  9 18 16 15  4  0 23 17  3  8  5 12 11 10  1 14]
Unique minutes: [37 11 51 36 39 20 17 44  6 54 49 52 14 56 46 34  8 10 38 45 30 58  7 13
                12 42 31 23 28 16 33 43 53 47  9  1 21  4  5  0 59 22 50  3 26 24 29  2
                19 25 48 35 57 32 15 27 18 40 55 41]
Unique seconds: [0]
```

- Since there is only one unique value for year and month, that means all of the measurements were taken in the same month of the same year. You won't use these as features.
- Days, hours, and minutes have multiple unique values. You'll use these as features.
- The only unique value for seconds is 0. Either each measurement was taken precisely on the minute, or the time measurement wasn't precise enough to capture seconds. In either case, there's no need to use this as a feature.

8. Perform common preparation on the training and test sets.

- a) Scroll down and view the cell titled **Perform common preparation on the training and test sets**, and examine the code listing below it.

Perform common preparation on the training and test sets

```
In [ ]: 1 # Perform common cleaning and feature engineering tasks on datasets.
2 def prep_dataset(X):
3
4     # FEATURE ENGINEERING
5
6     # Extract days, hours, and minutes from timestamp.
7     day = X['Date'].dt.day
8     X['Day'] = day.astype('float64')
9
10    hour = X['Date'].dt.hour
11    X['Hour'] = hour.astype('float64')
12
13    minute = X['Date'].dt.minute
14    X['Minute'] = minute.astype('float64')
15
16    return X
17
18 X_train = prep_dataset(X_train.copy())
19
20 X_test = prep_dataset(X_test.copy())
21
22 X_train.head()
```

- On lines 7 through 14, a new column is being created for days, hours, and minutes.
 - Each time component has a method that enables this extraction, like `dt.day()` to extract the day from a full datetime object.
- b) Select the cell that contains the code listing, then select **Run**.
 c) Examine the output.

	Date	Temperature	RelativeHumidity	Light	CO2	HumidityRatio	Day	Hour	Minute
0	2015-02-05 19:37:00	21.200	19.840	0.0	525.333333	0.003082	5.0	19.0	37.0
1	2015-02-04 22:11:00	21.390	25.700	0.0	475.000000	0.004046	4.0	22.0	11.0
2	2015-02-09 13:51:00	21.245	32.925	474.5	1126.500000	0.005146	9.0	13.0	51.0
3	2015-02-05 20:36:00	21.200	19.390	0.0	472.500000	0.003012	5.0	20.0	36.0
4	2015-02-10 02:39:00	20.290	32.900	0.0	460.000000	0.004846	10.0	2.0	39.0

Now that you've extracted the relevant date and time components, you can drop the `Date` column.

9. Drop columns that won't be used for training.

- a) Scroll down and select the code cell beneath the **Drop columns that won't be used for training** title, then select **Run**.
 b) Examine the output and verify that the `Date` column was dropped.

10. Standardize the features.

- a) Scroll down and view the cell titled **Standardize the features**, and examine the code listing below it.

Standardize the features

```
In [ ]: 1 def standardize(X):
          2     result = X.copy()
          3
          4     for feature in X.columns:
          5         result[feature] = (X[feature] - X[feature].mean()) / X[feature].std() # z-score formula.
          6
          7     return result
          8
          9 X_train = standardize(X_train)
         10 X_test = standardize(X_test)
         11
         12 print('The features have been standardized.')
```

As you've seen before, this function applies the *z*-score formula to the features for scaling purposes.

- b) Select the cell that contains the code listing, then select **Run**.

```
The features have been standardized.
```

- c) Scroll down and select the next code listing cell.

```
In [ ]: 1 with pd.option_context('float_format', '{:.2f}'.format):
          2     print(X_train.describe())
```

- d) Select **Run**.
e) Examine the output and verify that the features have been scaled.

11. Train an MLP model.

- a) Scroll down and view the cell titled **Train an MLP model**, and examine the code listing below it.

```
In [ ]: 1 from sklearn.neural_network import MLPClassifier
2
3 mlp = MLPClassifier(hidden_layer_sizes = (2),
4                      activation = 'relu',
5                      solver = 'adam',
6                      alpha = 0.0001,
7                      learning_rate_init = 0.001,
8                      max_iter = 500,
9                      tol = 1e-4,
10                     n_iter_no_change = 10,
11                     verbose = True,
12                     random_state = 87)
13
14 mlp.fit(X_train, np.ravel(y_train))
15
16 score = mlp.score(X_test, y_test)
17
18 print('Accuracy: {:.0f}%'.format(score * 100))
```

As its name implies, the `MLPClassifier()` class in scikit-learn trains an MLP neural network for classification purposes. It has many hyperparameters/arguments. The ones being specified here include:

- `hidden_layer_sizes` specifies the number of hidden layers to include in the network, as well as the number of neurons in each hidden layer. The argument takes a tuple, where the first value is the number of neurons in the first hidden layer, the second value is the number of neurons in the second hidden layer, and so on. So, if you wanted three hidden layers where each layer has ten neurons, you'd input `(10, 10, 10)` as the tuple. Here, because there is only one value, there will only be one hidden layer, and it will include two neurons.
- `activation` specifies the activation function to use. ReLU is being used here.
- `solver` specifies the method used to minimize cost and optimize the connection weights. The `adam` solver is similar to stochastic gradient descent (SGD).
- `alpha` is the ℓ_2 regularization penalty to apply. Here, the model is using the default value.
- `learning_rate_init` is the initial learning rate to use in gradient descent solvers.
- `max_iter` is the maximum number of iterations that the solver will perform if it doesn't converge first.
- `tol` defines a tolerance threshold for the solver to exceed when minimizing cost. If the cost is not minimized by more than `tol` for a specified number of iterations, then the solver will stop. The value here is the default.
- `n_iter_no_change` is the number of iterations for which the solver can fail to exceed `tol` before it stops.
- `verbose`, when set to `True`, will print out the loss at each iteration.

- b) Select the cell that contains the code listing, then select **Run**.

- c) Examine the output.

```

Iteration 142, loss = 0.05191888
Iteration 143, loss = 0.05184846
Iteration 144, loss = 0.05175415
Iteration 145, loss = 0.05176166
Iteration 146, loss = 0.05165864
Iteration 147, loss = 0.05161039
Iteration 148, loss = 0.05154295
Iteration 149, loss = 0.05148317
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Accuracy: 88%
```

- The loss at each iteration is output.
- Each iteration minimizes the loss more than the previous iteration.
- Iteration 149 was the 10th iteration for which the loss did not improve by more than `tol`, so the solver stopped there.
- You'll plot this loss minimization in the next step to get a better look.
- The accuracy of this model is 88%. As with any other classifier, you can use many different evaluation metrics. In this case, you'll just try to optimize accuracy.

12. Visualize the loss minimization through gradient descent.

- a) Scroll down and view the cell titled **Visualize the loss minimization through gradient descent**, and examine the code listing below it.

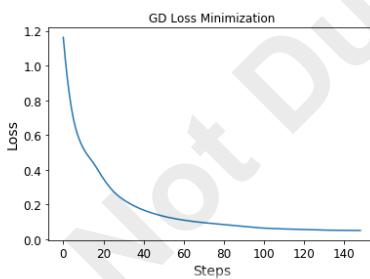
Visualize the loss minimization through gradient descent

```

In [ ]: 1 def plot_loss(model):
          2     plt.plot(model.loss_curve_)
          3     plt.title('GD Loss Minimization')
          4     plt.xlabel('Steps')
          5     plt.ylabel('Loss')
          6
          7 plot_loss(mlp)
```

The `loss_curve_` attribute conveniently returns an array of the loss value at each iteration.

- b) Select the cell that contains the code listing, then select **Run**.
c) Examine the output.



The loss decreases dramatically for the first 20 or so iterations, but after that, the change is minor. If time were more of a factor, you might want to increase the value of `tol` so that the solver isn't wasting time on more iterations for little gain.

13. Visualize the neural network architecture.

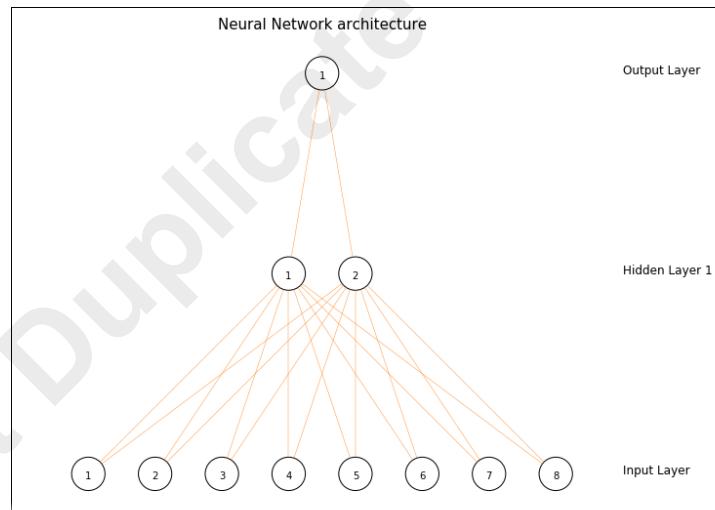
- a) Scroll down and view the cell titled **Visualize the neural network architecture**, and examine the code listing below it.

Visualize the neural network architecture

```
In [ ]: 1 def nn_diagram(X, y, model, show_weights):
2
3     # Create structure of network from dataset shapes and hidden layer sizes.
4     nn_struct = np.hstack((X.shape[1], np.asarray(model.hidden_layer_sizes), [y.shape[1]]))
5
6     # Only plot weights if specified.
7     if show_weights == True:
8         network = VisNN.DrawNN(nn_struct, model.coefs_)
9     else:
10        network = VisNN.DrawNN(nn_struct)
11
12    network.draw()
13
14 nn_diagram(X_train, y_train, mlp, False)
```

This function uses the `VisualizeNN.py` module to draw a visual representation of the neural network.

- Line 4 creates a structure object using the shape of the training data and the labels, along with the size of the hidden layers. This object will be passed in to the class that creates the diagram.
 - Lines 7 through 10 determine whether to draw the diagram with or without weights, according to the user's preference.
 - Line 14 calls the function without weights.
- b) Select the cell that contains the code listing, then select **Run**.
 c) Examine the output.



The network architecture, as expected, is broken down into input, hidden, and output layers.

- The input layer includes eight neurons, each of which maps to a feature in the dataset.
- The hidden layer has two neurons. This is the number of neurons you arbitrarily specified when creating the `MLPClassifier()` object.
- The output layer has one neuron—the label classification of a data example (0 or 1).
- Each neuron in one layer is connected to all neurons in the next layer.

14. Retrieve the neuron weights and bias terms and redraw the network architecture.

- Scroll down and select the code cell beneath the **Retrieve the neuron weights and bias terms and redraw the network architecture** title, then select **Run**.
- Examine the output.

```
Weights between input layer and hidden layer:
[[-0.3438876  0.254666]
 [ 0.07710875 -0.35894019]
 [ 1.27571406 -1.05065641]
 [ 0.62432824 -1.29862377]
 [-0.42358403 -0.35945967]
 [ 0.09719079  0.53563612]
 [-0.1719664   0.10169442]
 [-0.09943472 -0.00328182]]

Weights between hidden layer and output layer:
[[ 2.06152438]
 [-2.24583236]]

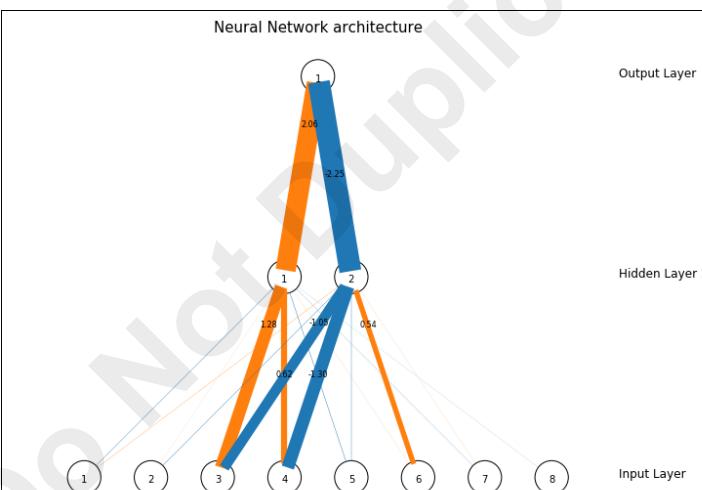
Bias terms between input layer and hidden layer:
[0.48037193 1.30762661]

Bias terms between hidden layer and output layer:
[-2.74116309]
```

- The first array lists the weights of each neuron in the connections between the input layer and the hidden layer.
 - The second array does likewise for the connections between the hidden and output layer.
 - The third array lists the bias terms for the connections between the input layer and the hidden layer.
 - The fourth array does likewise for the connection between the hidden and output layer.
- Scroll down and select the next code listing cell.

```
In [ ]: 1 nn_diagram(X_train, y_train, mlp, True)
```

- Select **Run**.
- Examine the output.



This time, the network diagram outputs with the neuron weights.

- The thicker the line is, the stronger the weight.
- Orange lines indicate positive weights.
- Blue lines indicate negative weights.
- The actual weight value is displayed for any weight that is above 0.5 or below -0.5 (i.e., the weights of most significance).

15. How does backpropagation generate the weights between the neurons of different layers in an MLP neural network?

16. What can you tell about the weights of this particular network structure?

17. Fit an MLP model using grid search with cross-validation.

- Scroll down and view the cell titled **Fit an MLP model using grid search with cross-validation**, and examine the code listing below it.

Fit an MLP model using grid search with cross-validation

```

1 from sklearn.model_selection import GridSearchCV
2
3 mlp = MLPClassifier(alpha = 0.0001,
4                      learning_rate_init = 0.001,
5                      max_iter = 500,
6                      tol = 1e-4,
7                      n_iter_no_change = 10,
8                      random_state = 87)
9
10 grid = {'hidden_layer_sizes': [(5), (6)],
11          'activation': ['logistic', 'tanh', 'relu'],
12          'solver': ['sgd', 'adam']}
13
14 search = GridSearchCV(mlp, param_grid = grid, scoring = 'accuracy', cv = 5, iid = False)
15
16 start = time()
17 search.fit(X_train, np.ravel(y_train))
18 end = time()
19 train_time = (end - start)
20
21 print('Grid search took {:.2f} seconds to find an optimal fit.'.format(train_time))
22 print(search.best_params_)

```

This code performs a grid search to determine optimal hyperparameters for the MLP model.

- On lines 3 through 8, the algorithm will use several of the same numeric values for the hyperparameters that were used before.
- On line 10, the grid begins by alternating between having one hidden layer with five neurons, and one hidden layer with six neurons.
- On line 11, each of the major activation functions is tried: the logistic (sigmoid) function, the tanh function, and the ReLU function.
- On line 12, two SGD-like weight optimization techniques are tried. These tend to be most useful in large datasets with thousands of data examples.
- To save time during class, the search field is relatively sparse. In a real-world situation, where time is less of a factor, you'd include many more possible combinations of hyperparameters. Also, grid search is being used here so that the outcome is more deterministic, but in a real-world situation, you'd use randomized search.
- On line 14, the grid search will be optimizing for accuracy and will perform five-fold cross-validation on the training data.

- Select the cell that contains the code listing, then select **Run**.



Note: It may take up to 10 minutes for the search to complete.

- c) Examine the output.

```
Grid search took 124.61 seconds to find an optimal fit.
{'activation': 'logistic', 'hidden_layer_sizes': 6, 'solver': 'sgd'}
```

- The optimal activation function is the logistic (sigmoid) function.
- The optimal number of neurons in the hidden layer is six.
- The optimal weight optimization technique is SGD.

- d) Scroll down and select the next code listing cell.

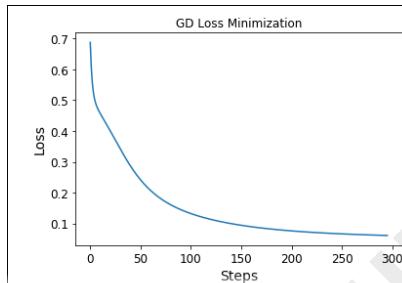
```
In [ ]: 1 score = search.score(X_test, y_test)
          2
          3 print('Accuracy: {:.0f}%'.format(score * 100))
```

- e) Select **Run**.
- f) Examine the output and confirm that the model's accuracy increased.

```
Accuracy: 94%
```

18. Visualize the loss minimization of the optimized model.

- a) Scroll down and select the code cell beneath the **Visualize the loss minimization of the optimized model** title, then select **Run**.
- b) Examine the output.

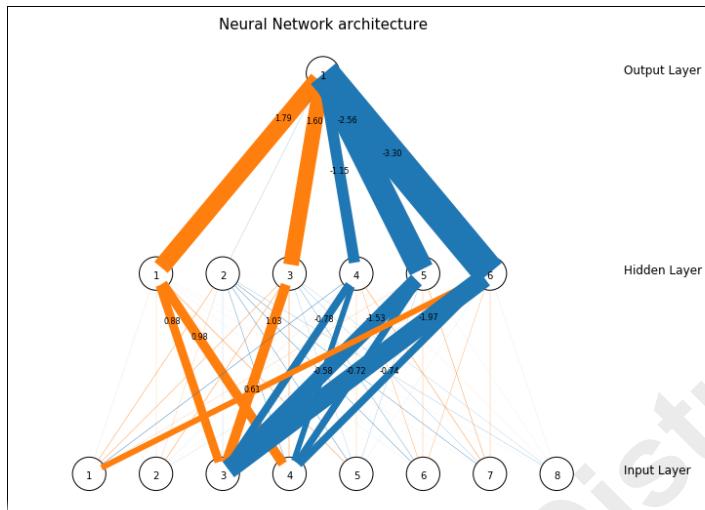


It takes about 300 steps for the solver to converge before failing to minimize more than the tolerance threshold.

19. Visualize the network structure of the optimized model.

- a) Scroll down and select the code cell beneath the **Visualize the network structure of the optimized model** title, then select **Run**.

- b) Examine the output.



- The input layer and output layer have the same number of neurons, but the hidden layer now has six.
- The weights between neurons have changed. While some of the weights are relatively weak, there are also plenty of stronger weights.
- Features 3 and 4 (Light and CO₂) seem to have strong positive weight with hidden neurons 1 and 3, while also having strong negative weight with hidden neurons 4, 5, and 6.
- Feature 1 (Temperature) seems to have a relatively strong positive weight with hidden neuron 6.
- Hidden neurons 1 and 3 have strong positive weight with the output neuron.
- Hidden neurons 4, 5, and 6 have strong negative weight with the output neuron.
- Hidden neuron 2 seems to not have strong weight with any layer, input or output.

20. Examine the model's predictions on the test set.

- Scroll down and select the code cell beneath the **Examine the model's predictions on the test set** title, then select **Run**.
- Examine the output.

	Date	Temperature	RelativeHumidity	Light	CO2	HumidityRatio	ActualOccupancy	PredictedOccupancy
0	2/2/2015 23:49	20.650000	22.245000	0.000000	443.000000	0.003342	0	0
1	2/2/2015 21:10	20.890000	23.000000	0.000000	491.666667	0.003508	0	0
2	2/3/2015 19:56	21.245000	27.745000	0.000000	770.750000	0.004331	0	0
3	2/3/2015 5:51	20.290000	22.650000	0.000000	431.000000	0.003328	0	0
4	2/3/2015 2:12	20.525000	22.267500	0.000000	442.750000	0.003320	0	0
5	2/4/2015 8:50	21.200000	25.180000	454.000000	740.200000	0.003917	1	0
6	2/2/2015 17:19	22.500000	24.865000	433.000000	816.500000	0.004189	1	1
7	2/4/2015 9:56	23.200000	25.500000	722.000000	1011.400000	0.004485	1	1
8	2/4/2015 8:44	21.083333	25.200000	453.000000	719.500000	0.003892	1	0

21. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **NeuralNetworks-Occupancy** tab in Firefox, but keep a tab open to **CAIP/NeuralNetworks/** in the file hierarchy.

TOPIC B

Build Convolutional Neural Networks (CNN)

Now that you've built MLP neural networks, you can incorporate them into a wider architecture called a convolutional neural network (CNN). This type of network excels at solving computer vision problems.

Traditional ANN Shortcomings

A traditional MLP neural network in which all neurons in a layer are connected to the neurons of adjacent layers excels at solving many different types of tasks using many different types of input. However, this neural network architecture is at a disadvantage when it comes to processing images. Images are input in the form of pixels, and even small images can contain tens of thousands of pixels; this would therefore require an extremely high number of connections between neurons, increasing the network's density and reducing its performance.

In addition, a traditional neural network would require the 2-D image to be flattened into a 1-D vector before being fed as input, which removes some crucial information, particularly the spatial relationships between pixels.

Convolutional Neural Network (CNN)

A ***convolutional neural network (CNN)*** is a type of neural network that forms partial connections between neurons using a component called a ***convolutional layer***. A convolutional layer comes from research into human vision indicating that each neuron in the brain only reacts to a small portion of stimuli in the entire receptive field. For example, one neuron may react to simple lines, another may react to more complex shapes, another may react to a different orientation of those lines or shapes, and so on, with many more possible combinations. What's more, each neuron appears to build on the neurons near it, filling out a more complex image as the visual information gets to higher and higher levels. Ultimately, these neurons combine to form the entire receptive field.

The same basic idea is behind the convolutional layer. The first convolutional layer is only connected to input neurons whose pixels are in the appropriate portion of the receptive field. The second convolutional layer's neurons are only connected to the neurons in the first convolutional layer that are appropriate for that portion of the receptive field, and so on. This results in a neural network that is only partially connected, making them ideal for processing pixel-dense images.



Note: CNNs are also applied to tasks like natural language processing (NLP), but they tend to be more commonly thought of as efficient solvers of computer vision tasks.

CNN Filters

Filters are the portion of the receptive field that a convolutional layer neuron uses to scan the image at prior layers. Each filter will scan the image for a particular feature, like a horizontal line, an oval shape, etc., and then outputs a feature map. A ***feature map*** is a representation of the image that focuses on whatever feature the filter searches for. So, if the filter is a horizontal line, it will highlight the areas of the image that most strongly exhibit horizontal lines. While defining the number of filters at each layer is possible through a hyperparameter, you don't actually define the filter itself; the CNN learns the appropriate filters to use through training.

In the following figure, the square on the bottom is an input layer, and the square above it is a convolutional layer. Each small square inside the larger squares is a pixel, and also a neuron of that

layer. This highlighted neuron is scanning a portion of the input image to see if it contains the feature that the filter is looking for.

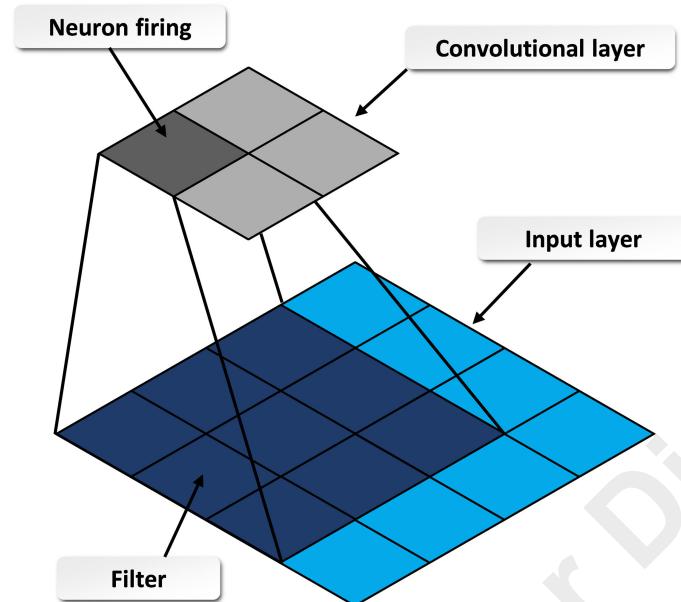


Figure 10–5: A convolutional layer using a filter to scan an input image.

All of the neurons in this convolutional layer are looking for the same thing, just in different spots. The following figure shows another convolutional neuron looking at that same image.

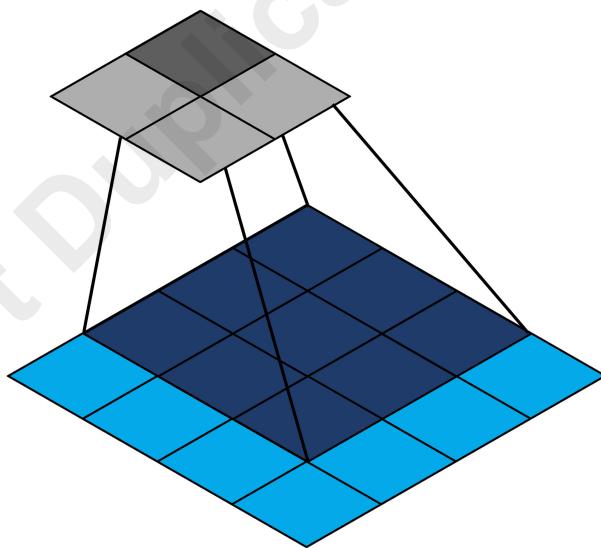


Figure 10–6: Another neuron in the convolutional layer scans the image.

Once all of the neurons in the convolutional layer have finished, that layer can produce a feature map. This feature map can be used in the next convolutional layer higher in the network to learn even more complex patterns.

CNN Filter Example

Consider the following figure, where each sub-square represents a pixel, and each pixel is either "on" (1) or "off" (0). On the left is the input image, which forms the capital letter "T". On the right is a simple filter that looks for vertical lines.

Input Layer

1	1	1	1	1
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0

Filter

0	1	0
0	1	0
0	1	0

Figure 10-7: An input image (left) and a vertical line filter (right).

Now, consider how this filter would be projected on top of the input image.

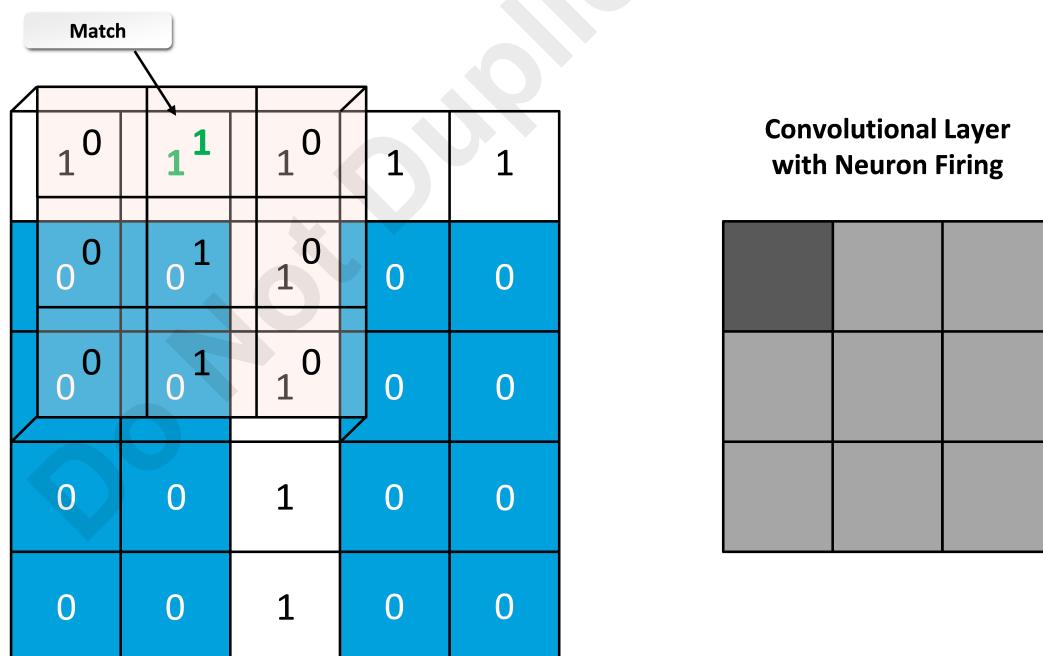


Figure 10-8: The vertical line filter applied to the image for the first neuron.

Licensed For Use Only By: Abdulwahab Alweban dev.abdulwahab@gmail.com Jan 8 202

As you can see, the number 1 matched a pixel in the input once. Now consider how the filter is projected from the next neuron in the convolutional layer.

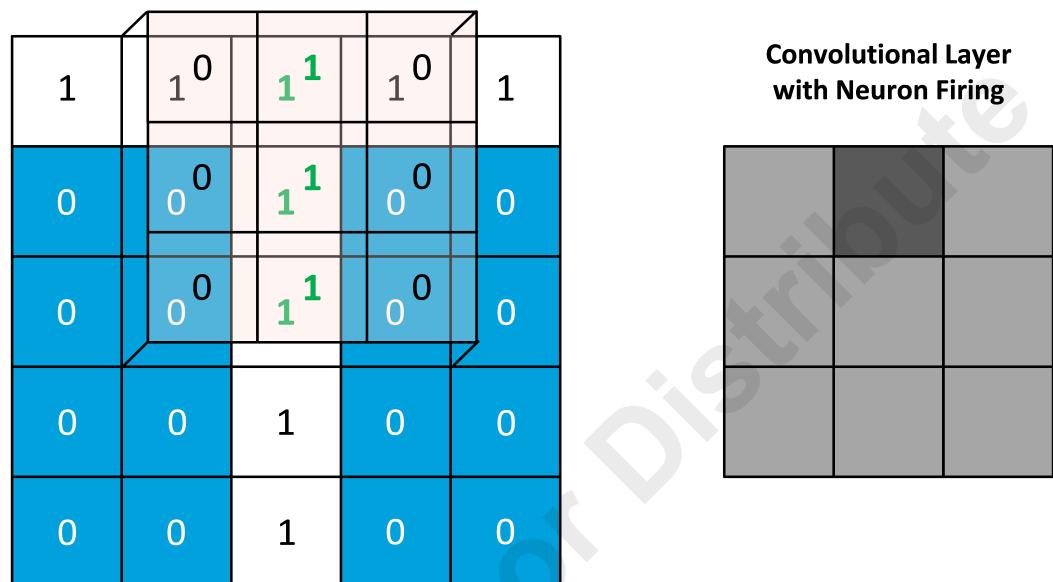


Figure 10-9: The next neuron applying the filter.

In this case, the filter found 3 matches. Once the filter is applied by the entire convolutional layer, the values of each pixel across the input and the filter are combined and then run through an activation function to calculate the feature weights. So, using the same example, each neuron in the convolutional layer would have the following weights.

1	3	1
0	3	0
0	3	0

Figure 10-10: Convolutional neuron weights based on the input image. The darker the neuron, the more weight it has.

Padding

Padding is the practice of adding pixels to the input in order to preserve the dimensions of the image, while enabling the convolutional layer to be the same size as the actual input. When both are the same size, the convolutional neurons can scan the image completely with their filters. In the earlier "T" image example, the vertical filter was unable to detect the top-left and top-right pixels of the "T" because the filter's vertical line wasn't applied to those pixels. This can be solved by padding.

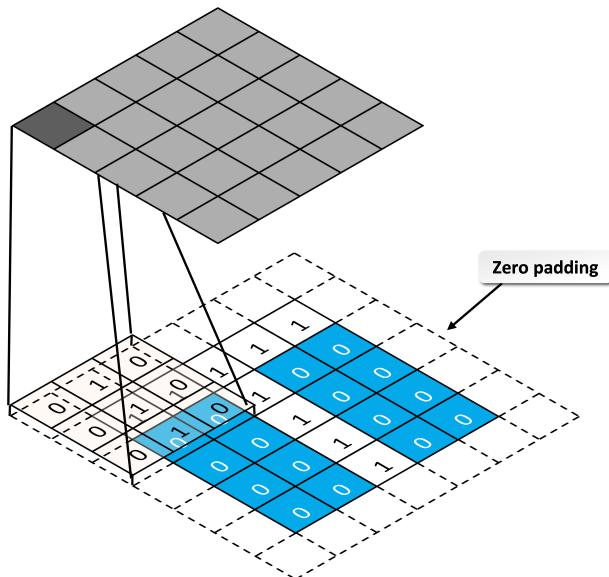


Figure 10-11: Padding an input image with zeros so that the convolutional neurons' filter can cover every part of the image.

Stride

Another way to modify how convolutional neurons scan the input image is by changing the stride. The **stride** is just the distance between the filters as they scan the image. Changing the stride enables a smaller convolutional layer to scan a large input layer. This effectively downsamples the image, decreasing computation time. In the previous examples, the stride is the default value of 1. In the following figure, the stride is 2.

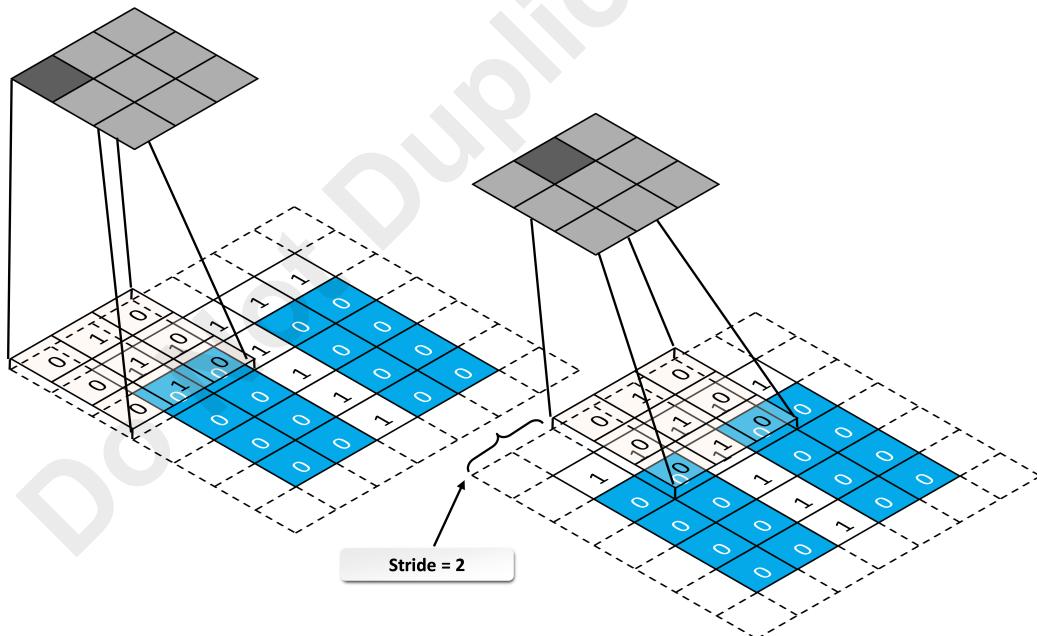


Figure 10-12: A stride of 2 enables this smaller convolutional layer to perform a downsampled scan of a large input layer.

Pooling Layer

A **pooling layer** is another component of CNNs. It is very similar to a convolutional layer, except that the neurons in a pooling layer are not given weights. Instead, some aggregation function is applied to the features to make a more efficient selection. For example, taking the maximum value of the filter is commonly used in pooling layers. The pooling layer will only pass on the highest value to the next layer. Consider the feature map generated earlier in the "T" example. You now have another filter on top of it that scans this feature map.

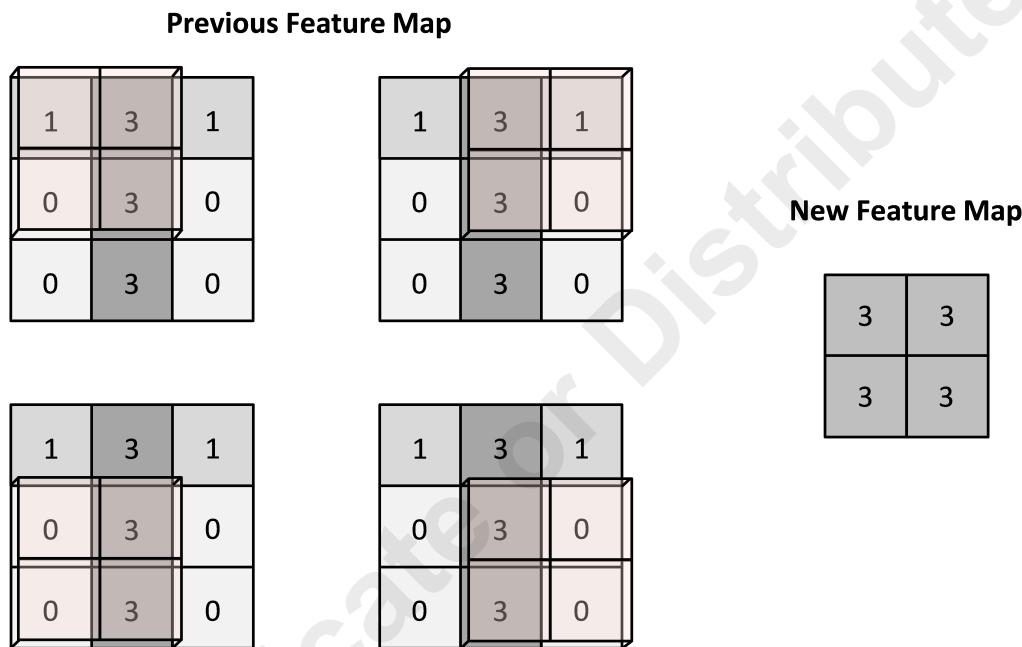


Figure 10-13: A max pooling layer. Note that the 3x3 squares on the left are the same feature map being scanned four separate times.

In this case, each neuron in the top filter just retrieved the maximum value from its scanning field. None of the other values in the field made it to the next layer's neuron.

Pooling is intended to reduce performance issues by only sampling a portion of the input. Not only can this save time and memory, but it can also reduce overfitting.



Note: Padding is not commonly used in pooling layers.

CNN Architecture

A true CNN is more complex than the simple examples shown previously. Rather than using just one filter/feature map per convolutional layer, CNNs typically stack multiple feature maps on top of each other. This enables the neural network to generate new feature maps based on previous feature maps, improving pattern recognition across all areas of the network. So, using the "T" example, a convolutional layer (also simply called a "convolution") may apply a vertical line filter, a horizontal line filter, and a diagonal line filter all in one main layer. The same stacking concept applies to pooling layers.

In addition, a CNN usually places a convolutional layer on top of the input, then places a pooling layer on top of that, then another convolutional layer, then another pooling layer, and so on. The top of the network is usually just a fully connected MLP like you've seen before. The feature maps

are fed into these fully connected layers as input. There are one or more hidden layers that use an activation function like ReLU, and a final output layer that, for example, outputs a classification.

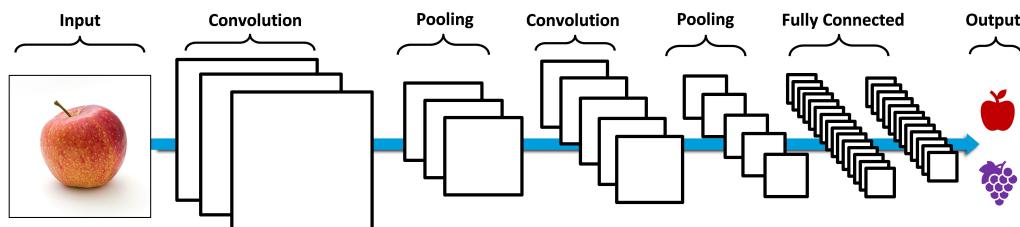


Figure 10–14: An example CNN architecture. Note that the image gets smaller as it goes through the network, while also generating more feature maps.

Generative Adversarial Network (GAN)

A **generative adversarial network (GAN)** is a neural network architecture that pits two different neural networks against each other. One network is a discriminative model, the other is a generative model. The act of training these networks in an oppositional way leads to highly effective models whose skill surpasses those of standard neural networks.

A discriminative model, also simply called a **discriminator**, is not much different than the neural networks that have been discussed thus far. This type of model predicts a label given a set of features. A discriminator classifies a medical patient as having heart disease (1) or not having heart disease (0) when trained to look for correlations between features. A **generator**, on the other hand, does the opposite—it predicts features given a label. For a patient that has heart disease, a generator determines the probability that the patient's weight, height, age, etc., play a role. More specifically, it maps the distribution of a feature's values, and then generates new values that fit well within this distribution.

GANs are adversarial because the discriminator and generator are in a constant state of conflict. The generator creates new data examples and sends them to the discriminator. The generator's intent is to maximize the discriminator's error rate. In other words, the generator attempts to "fool" the discriminator into thinking the new data is authentic data from the actual training set. The discriminator, meanwhile, attempts to spot any "forgeries" that the generator gives it. This leads to a tug-of-war; each network is working toward minimizing an opposing cost function, pushing back on its opponent as long as it can. Ultimately, this strengthens the GAN's ability to generate highly convincing data, as less convincing forgeries will simply be rejected.

GAN Architecture

In the following example, a GAN is tasked with generating new images of dogs.

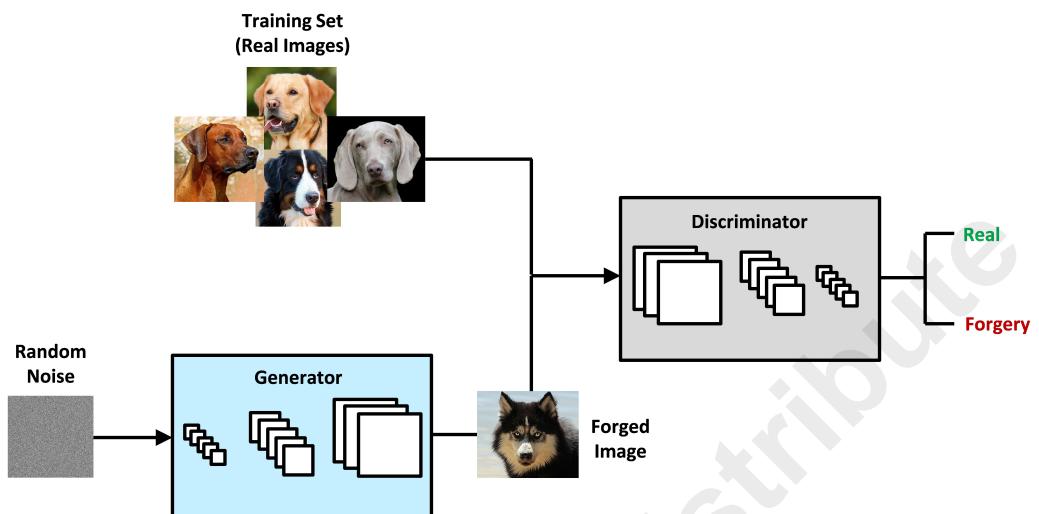


Figure 10–15: A general GAN architecture used for image generation.

Each network is a CNN. The discriminator is a typical CNN as described earlier, whereas the generator is something akin to an inverse CNN; in other words, it starts with random noise and continually upsamples image data to eventually create a fully rendered image. Contrast this with a typical CNN used by the discriminator, in which the input image is downsampled through pooling layers.

The basic process is as follows:

1. The generator is initialized with random noise and gradually builds a forged image based on its training.
2. The forged image is fed into the discriminator along with actual examples provided by the training set.
3. The discriminator evaluates all of these images and outputs a probability between 0 and 1. A probability of 0 means the discriminator thinks it's a forgery, whereas 1 means it thinks it's real.
4. This process repeats until some stopping criterion is reached. Usually, this is when the generator's failed attempts at convincing the discriminator are equal in number to the discriminator's failed attempts at spotting the generator's forgeries.

Most GANs are trained to work on image data. Skillful GANs excel at generating images, and have even been able to generate faces of non-existent people that look entirely real to any human observer. They are also adept at creating higher quality versions of existing images, like upscaling older pictures that may have been originally taken at a low resolution. One downside to GANs is that they tend to require a large amount of time and computational power during training.

Guidelines for Building CNNs

Follow these guidelines when you are building convolutional neural networks (CNNs).

Build a CNN

When building a CNN:

- Use CNNs for classifying images, tracking video, and performing other computer vision-related tasks.
- Use convolutional layers to scan image inputs using filters.
- Consider using padding to enable a convolutional filter to scan an entire input image.
- Consider using stride to effectively downsample an image, saving on training time.

- Use pooling layers to sample only portions of the input, reducing training time and minimizing overfitting.
- Construct a CNN architecture that alternates between convolutional and pooling layers, terminating with a fully connected layer for the output.
- Use generative adversarial networks (GANs) that incorporate CNNs to generate convincing artificial images or upscaled versions of existing images.
- Consider that GANs have a very high computational cost.

Use Python for CNNs

Keras, a frontend to TensorFlow, has a `Sequential()` class that you can use to sequentially build an artificial neural network. Using the `layers` module, you can define specific types of layers to add to this network. The following are some of the objects and functions you can use to build a CNN.

- `network = keras.models.Sequential()` —This constructs an object that you can use to start building network layers sequentially.
- `network.add(keras.layers.Conv2D(filters = 64, kernel_size = (2, 2), input_shape = (50, 50, 1), padding = 'same', activation = 'relu'))` —This adds a convolutional layer that can work on two-dimensional images. In this case, the number of output filters is 64, the shape of those filters is 2×2 , the shape of the input is 50×50 , the layer will use padding, and the layer will use the standard ReLU activation function.
- `network.add(keras.layers.MaxPooling2D((2, 2)))` —This adds a pooling layer that will downscale the image by half its original height and width.
- `network.add(keras.layers.Flatten())` —This adds a flattening layer to reduce the dimensionality of the previous layer's output.
- `network.add(keras.layers.Dense(3, activation = 'softmax'))` —This adds a dense (fully connected) layer to use as the output layer. In this case, there are three possible classes, and the activation function being used is softmax.
- `network.compile(optimizer = 'sgd', loss = 'categorical_crossentropy', metrics = ['accuracy'])` —This compiles a sequentially built network. In this case, the network will use a stochastic gradient descent (SGD) solver, a loss function that is commonly used for multi-class classification, and accuracy as the evaluation metric.
- `network.fit(X_train, y_train, validation_data = (X_val, y_val), epochs = 10)` —Fit the training data to the neural network for the specified number of epochs. Optionally, you can supply data to use for validation.
- `network.summary()` —Print the general structure of a sequential network you've built.
- `keras.utils.plot_model(network, to_file = 'network.png')` —Creates a more visual representation of the network structure and saves it to a file.
- `network.evaluate(X_test, y_test)` —Evaluate the network's performance on test data. This returns both the loss value and the value of the chosen metric.
- `network.predict(X_test)` —Use the network to make predictions on test data.

ACTIVITY 10-2

Building a CNN

Data File

/home/student/CAIP/Neural Networks/NeuralNetworks-Fashion.ipynb

Before You Begin

Jupyter Notebook is open.

Scenario

You work for an online retailer that sells various articles of clothing from many different brands. Each brand provides you with different suggestions for how to categorize each article to facilitate user searches. Rather than use the brands' suggestions, which often conflict with one another or aren't particularly useful, the storefront uses a categorization scheme that was developed in house. Currently, each new article must be manually categorized for searching by several employees. An employee visually examines a product and determine whether or not it is a shoe, a shirt, a hat, etc. This is tedious work that can be automated using computer vision.

Thankfully, you have a rather large database of existing product images that have already been categorized. So, you'll use this dataset to train a convolutional neural network (CNN) to classify new product images. Once the model has achieved enough success, you'll be able to push it to production.

1. Load the notebook and dataset.

- From Jupyter Notebook, select **CAIP/Neural Networks/NeuralNetworks-Fashion.ipynb** to open it.
- Select the code cell beneath the **Import software libraries and load the dataset** title, then select **Run**.

```
Using TensorFlow backend.

Libraries used in this project:
- Python 3.7.3 (default, Mar 27 2019, 22:11:17)
[GCC 7.3.0]
- NumPy 1.16.2
- Matplotlib 3.0.3
- scikit-learn 0.20.3
- TensorFlow 2.0.0
- Keras 2.3.1

Loaded 60000 training records.
Loaded 10000 test records.
```

- Verify that the training and test sets were loaded with 60,000 and 10,000 records, respectively.

This dataset—called Fashion-MNIST—includes numerical values that can be used to construct small grayscale images of different types of clothing. The dataset was loaded using Keras, a frontend to the TensorFlow deep learning library. You'll use Keras to build and train a CNN.



Note: Fashion-MNIST is another take on the MNIST dataset, an image dataset of handwritten numbers that is very commonly used to teach machine learning concepts. To learn more about Fashion-MNIST, following this link: <https://github.com/zalandoresearch/fashion-mnist>.

2. Get acquainted with the dataset.

- Scroll down and select the code cell beneath the **Get acquainted with the dataset** title, then select **Run**.

- b) Examine the output.

```
Shape of feature space: (28, 28)
A few examples:
[[[ 0  0  0 ... 0  0  0]
 [ 0  0  0 ... 0  0  0]
 [ 0  0  0 ... 0  0  0]
 ...
 [ 0  0  0 ... 180 0  0]
 [ 0  0  0 ... 72  0  0]
 [ 0  0  0 ... 70  0  0]]
 [[ 0  0  0 ... 0  0  0]
 [ 0  0  0 ... 0  0  0]
 [ 0  0  0 ... 39  1  0]
 ...
 [ 0  0  0 ... 238 0  0]
 [ 0  0  0 ... 131 0  0]
 [ 0  0  0 ... 0  0  0]]
 [[ 0  0  0 ... 0  0  0]
 [ 0  0  0 ... 0  0  0]
 [ 0  0  0 ... 7  0  0]
 ...
 [ 0  0  0 ... 0  9  0]
 [ 0  0  0 ... 0  3  0]
 [ 0  0  0 ... 0  0  0]]
```

- The shape of the feature space shows that the training set is multi-dimensional; rather than an image having 28 features, it has 28×28 features. This corresponds to the dimensions of the image—it is 28 pixels wide and 28 pixels high.
- Example images 7, 8, and 9 have their features printed. The features are truncated, but you can see that most of their values are 0, though some have actual positive values. Each number represents that particular pixel's intensity in grayscale. In other words, a 0 is completely black, whereas 255 is completely white.

- c) Scroll down and select the next code listing cell.

```
In [ ]: 1 class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
   'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
2
3
4 for i in range(10):
5     print('{} ({})'.format(class_names[i], np.unique(y_train)[i]))
```

- d) Select Run.
e) Examine the output.

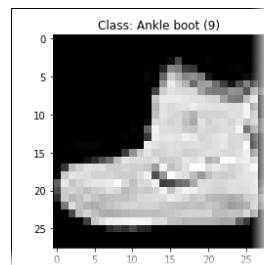
```
T-shirt/top (0)
Trouser (1)
Pullover (2)
Dress (3)
Coat (4)
Sandal (5)
Shirt (6)
Sneaker (7)
Bag (8)
Ankle boot (9)
```

Each class label is mapped to its actual class name (i.e., the type of clothing). There are 10 total classes.

3. Visualize the data examples.

- a) Scroll down and select the code cell beneath the **Visualize the data examples** title, then select Run.

- b) Examine the output.



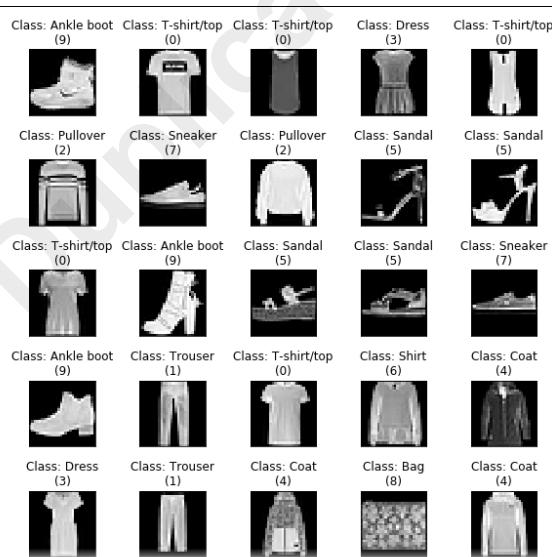
Using the image's grayscale features, Matplotlib was able to plot what the first image in the training set actually looks like. This image is classified as an ankle boot (9).

- c) Scroll down and select the next code listing cell.

```
In [ ]: 1 fig, axes = plt.subplots(nrows = 5, ncols = 5, figsize = (8, 8))
2
3 for i, ax in zip(range(25), axes.flatten()):
4     ax.imshow(X_train[i,:,:], cmap = 'gray') # Plot training example.
5     ax.title.set_text('Class: {} \n({})'.format(class_names[y_train[i]], y_train[i]))
6
7 # Turn off axis ticks for readability.
8 for ax in axes.flatten():
9     ax.set_xticks([])
10    ax.set_yticks([])
11
12 fig.tight_layout()
```

- d) Select Run.

- e) Examine the output.



The first 25 images from the training set are plotted in a grid.

4. Prepare the data for training with Keras.

- a) Scroll down and view the cell titled **Prepare the data for training with Keras**, and examine the code listing below it.

Prepare the data for training with Keras

```
In [ ]: 1 # Reshape arrays to add greyscale flag.
2 X_train = X_train.reshape(-1, 28, 28, 1)
3 X_test = X_test.reshape(-1, 28, 28, 1)
4
5 from keras.utils import to_categorical
6
7 # One-hot encode the data for each label.
8 y_train = to_categorical(y_train)
9 y_test = to_categorical(y_test)
10
11 print('One-hot encoding for first image: {}'.format(y_train[0]))
```

Because the data is rather simple and uniform, not much data preparation needs to be done. However, in order for the CNN to predict a classification, the label needs to be one-hot encoded.

- Lines 2 and 3 reshape the data to a format that is supported by Keras. The first argument (-1) tells the function to reshape the dataset according to its total length (number of examples), which you want to preserve. You also need to preserve the 28×28 feature space in the next two arguments. The last argument (1) indicates to Keras that these images are in grayscale.
- On lines 8 and 9, the `to_categorical()` method is an easy way to one-hot encode values using the Keras library.

- b) Select the cell that contains the code listing, then select **Run**.
 c) Examine the output.

```
One-hot encoding for first image: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
```

Each example image is now one-hot encoded, where there is only one "activated" value (1) for an image, depending on its class. In this case, the first image has a 1 in the last column, indicating that it is labeled as class 9 (ankle boot). You can scroll up to the images you displayed earlier to verify that this is correct.

5. Split the datasets.

- a) Scroll down and select the code cell beneath the **Split the datasets** title, then select **Run**.
 b) Examine the output.

```
Training features: (45000, 28, 28, 1)
Validation features: (15000, 28, 28, 1)
Training labels: (45000, 10)
Validation labels: (15000, 10)
```

You're splitting the training dataset in order to have a validation holdout. Although the test set you loaded at the start is labeled, you'll treat it as the ultimate test case.

6. Build the CNN structure.

- a) Scroll down and view the cell titled **Build the CNN structure**, and examine the code listing below it.

Build the CNN structure

```
In [ ]: 1 from keras.models import Sequential
2 from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
3 from keras.layers.advanced_activations import LeakyReLU
4
5 cnn = Sequential()
6
7 # Start stacking layers one-by-one.
8 cnn.add(Conv2D(filters = 32,
9                 kernel_size = (3, 3), # First convolutional layer (32 output filters, 3x3 filter size
10                input_shape = (28, 28, 1),
11                padding = 'same',
12                activation = 'linear')) # Will add leaky ReLU layer next.
13 cnn.add(LeakyReLU(alpha = 0.1))
14 cnn.add(MaxPooling2D((2, 2), padding = 'same')) # First pooling layer with 2x2 size.
15
16 cnn.add(Conv2D(64, (3, 3), padding = 'same', activation = 'linear'))
17 cnn.add(LeakyReLU(alpha = 0.1))
18 cnn.add(MaxPooling2D((2, 2), padding = 'same'))
19
20 cnn.add(Conv2D(128, (3, 3), padding = 'same', activation = 'linear'))
21 cnn.add(LeakyReLU(alpha = 0.1))
22 cnn.add(MaxPooling2D((2, 2), padding = 'same'))
23
24 cnn.add(Flatten()) # Connect convolution and dense layer.
25 cnn.add(Dense(10, activation = 'softmax')) # Dense output layer with softmax activation.
26
27 print('The CNN structure has been built.')
```

This code builds the actual structure of the CNN with Keras.

- The `Sequential()` class indicates that you'll build the structure as a sequence of layers, which is an easy way to go about building a relatively simple network.
- Lines 8 through 12 add the first layer in the stack—the layer closest to the input. The `Conv2D()` object builds a convolutional layer. This particular layer has the following hyperparameters/arguments:
 - `filters` specifies the number of output filters in the convolutional layer. In this case, there will be 32 filters. This number is somewhat arbitrary, as there is not necessarily a "best" number of filters to choose for any layer. However, the more filters there are, the longer it will take to train the network.
 - `kernel_size` specifies the dimensions of the filter itself. In this case, it will be 3×3 pixels.
 - `input_shape`, as the name suggests, is the shape of the input features the network will be training on. The last number indicates grayscale.
 - `padding` determines the padding to use, if any. A padding of `same` means that the layer will be padded in such a way that the output of the layer has the same dimensions as the layer's input.
 - `activation` is the activation function to use at the layer. Right now, this is a simple `linear` function. You could specify `relu` here, but this just uses the standard ReLU function that is susceptible to the vanishing gradients problem. You need to actually specify more advanced activation functions as their own layer. So, `linear` is acting as a placeholder until then.
- Line 13 adds an advanced activation function layer—in this case, leaky ReLU. The `alpha` argument is the slope coefficient.
- Line 14 adds a pooling layer after the convolution. The first argument defines the pooling size—in other words, the factor by which the image will be downsampled. So, a pooling layer of $(2, 2)$ means that the image will be downsampled to half its initial size across both width and height. Also, padding is used.
- Lines 16 through 22 repeat this process, adding two more groups of convolutional and pooling layers. The only change is that the size of the convolution's output filter is being increased.
- Line 24 adds a "flattening" layer in order to reduce the dimensionality of the preceding output to just one. This is necessary in order to feed the multi-dimensional output of a convolution into a one-dimensional vector that is supported by the next fully connected layer.
- Line 25 adds the final layer, the fully connected (dense) layer that is similar to a traditional MLP. This layer uses the softmax activation function to generate a multi-class classification decision from the flattened input. The first argument specifies the number of possible outputs (class labels).

- b) Select the cell that contains the code listing, then select **Run**.

The CNN structure has been built.

7. Compile the model and examine the layers.

- a) Scroll down and view the cell titled **Compile the model and examine the layers**, and examine the code listing below it.

Compile the model and examine the layers

```
In [ ]: 1 cnn.compile(optimizer = 'adam',
2                 loss = 'categorical_crossentropy',
3                 metrics = ['accuracy'])
4
5 cnn.summary()
```

The `compile()` method takes the Keras CNN object built in the previous code block and configures it for training.

- `optimizer` specifies the loss optimization method to use. The `adam` method is similar to stochastic gradient descent (SGD).
- `loss` is the actual loss function to use. The `categorical_crossentropy` function is used with multi-class classification; it measures how much the predicted probability for a class diverges from the actual class label. The lower the value, the better. For example, if the network predicted a 0.97 for an image that was actually a 0, this would be a very large discrepancy and therefore lead to a higher loss value.
- `metrics` specifies a scoring metric to use besides just the loss. To keep things simple, you'll be evaluating accuracy.

- b) Select the cell that contains the code listing, then select **Run**.
c) Examine the output.

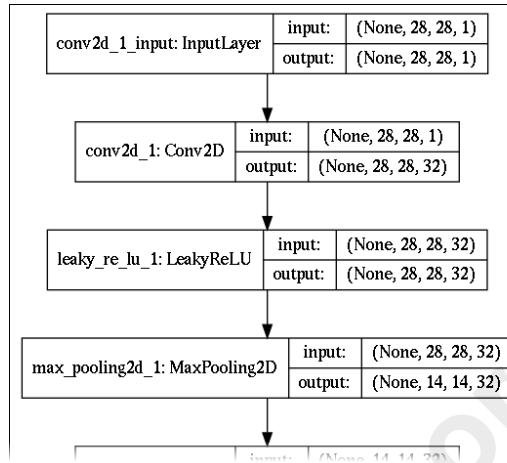
Model: "sequential_1"	Layer (type)	Output Shape	Param #
	conv2d_1 (Conv2D)	(None, 28, 28, 32)	320
	leaky_re_lu_1 (LeakyReLU)	(None, 28, 28, 32)	0
	max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 32)	0
	conv2d_2 (Conv2D)	(None, 14, 14, 64)	18496
	leaky_re_lu_2 (LeakyReLU)	(None, 14, 14, 64)	0
	max_pooling2d_2 (MaxPooling2D)	(None, 7, 7, 64)	0
	conv2d_3 (Conv2D)	(None, 7, 7, 128)	73856
	leaky_re_lu_3 (LeakyReLU)	(None, 7, 7, 128)	0
	max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 128)	0
	flatten_1 (Flatten)	(None, 2048)	0
	dense_1 (Dense)	(None, 10)	20490
Total params:	113,162		
Trainable params:	113,162		
Non-trainable params:	0		

This provides an overview of the CNN's structure. It lists each layer's type, its output shape, and the number of parameters at each convolution. The number of parameters is defined as: number of output filters \times (number of input filters \times filter size + 1).

- d) Scroll down and select the next code listing cell.

```
In [ ]: 1 from keras.utils import plot_model
2 plot_model(cnn, show_shapes = True, to_file = 'model.png')
```

- e) Select **Run**.
f) Examine the output.



This is a more visual way of displaying the different layers of the CNN. Note that the input and output sizes change between layers. The size of each layer appears to decrease as you get closer and closer to the final output layer (dense network).

8. Train the model.

- a) Scroll down and view the cell titled **Train the model**, and examine the code listing below it.

Train the model

```
In [ ]: 1 cnn_trained = cnn.fit(X_train, y_train,
2                           validation_data = (X_val, y_val),
3                           epochs = 1,
4                           verbose = 1)
```

You're using the training dataset along with the validation set to fit the CNN model. Due to classroom time constraints, you'll only train for one epoch. In a real-world scenario, you'd want to train for several epochs.

- b) Select the cell that contains the code listing, then select **Run**.
c) Examine the output.

```
Train on 45000 samples, validate on 15000 samples
Epoch 1/1
14272/45000 [=====.....] - ETA: 52s - loss: 0.9581 - accuracy: 0.7890
```

Keras provides a way to observe the progress of the training while it is underway.



Note: It may take up to 10 minutes for training to complete.

- d) While you wait, observe how the training loss decreases over time, while the accuracy gradually increases.
- e) When training is complete, examine the final scores on the validation set.

```
Train on 45000 samples, validate on 15000 samples
Epoch 1/1
45000/45000 [=====] - 81s 2ms/step - loss: 0.5645 - accuracy: 0.8431 - val_loss: 0.4933 - val_accuracy: 0.8305
```



Note: Since there is some degree of randomness, your results may not align exactly with the results in the screenshot.

9. Evaluate the model on the test data.

- a) Scroll down and view the cell titled **Evaluate the model on the test data**, and examine the code listing below it.

Evaluate the model on the test data

```
In [ ]: 1 eval_test = cnn.evaluate(X_test, y_test, verbose = 0)
          2
          3 print('Loss: {}'.format(round(eval_test[0], 2)))
          4 print('Accuracy: {:.0f}%'.format(eval_test[1] * 100))
```

Since your test set is labeled, you'll evaluate the model on that test set as well.

- b) Select the cell that contains the code listing, then select **Run**.
- c) Examine the output.

```
Loss: 0.52
Accuracy: 83%
```

The scores on the test set should be fairly close to the scores on the validation set.

10. Make predictions on the test data.

- a) Scroll down and select the code cell beneath the **Make predictions on the test data** title, then select **Run**.
- b) Examine the output.

```
Actual class: [9 2 1 1 6 1 4 6 5 7]
Predicted class: [9 2 1 1 6 1 2 6 5 7]
```

For the first 10 images, most of the predictions align with the actual class labels.



Note: As before, your results may differ due to randomness.

11. Visualize the predictions for several examples.

- a) Scroll down and view the cell titled **Visualize the predictions for several examples**, and examine the code listing below it.

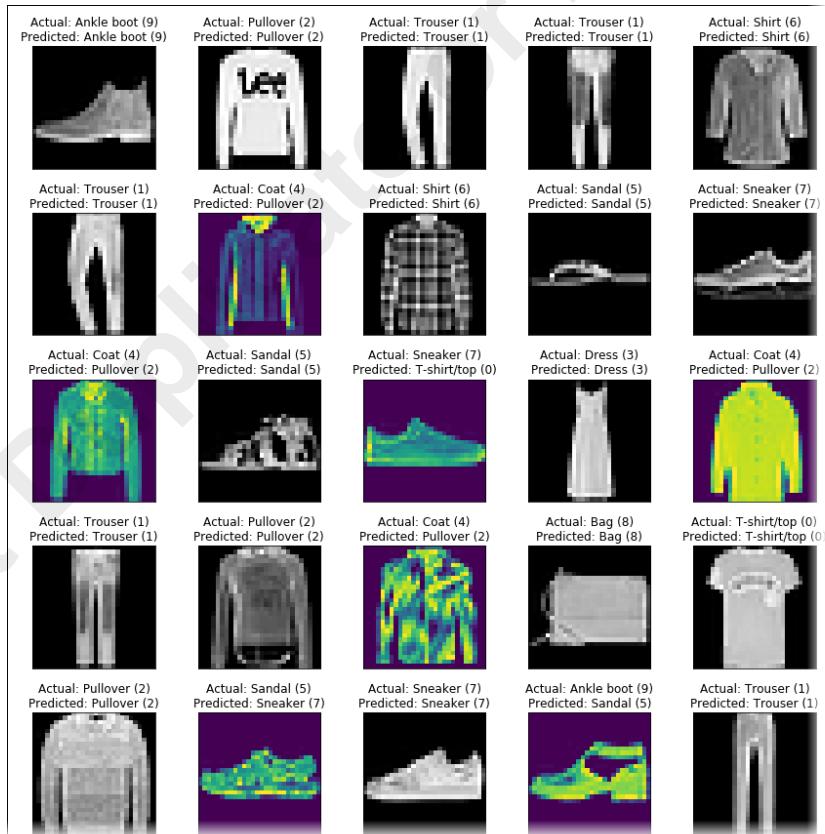
```

1 fig, axes = plt.subplots(nrows = 5, ncols = 5, figsize = (12, 12))
2
3 for i, ax in zip(range(25), axes.flatten()):
4
5     if actual[i] == prediction[i]:
6         ax.imshow(X_test[i].reshape(28, 28), cmap = 'gray')
7     else:
8         ax.imshow(X_test[i].reshape(28, 28)) # Highlight wrong predictions.
9
10    ax.title.set_text('Actual: {} ({})\nPredicted: {} ({})'.format(class_names[actual[i]], actual[i],
11                                                               class_names[prediction[i]], prediction[i]))
12
13 # Turn off axis ticks for readability.
14 for ax in axes.flatten():
15     ax.set_xticks([])
16     ax.set_yticks([])
17
18 fig.tight_layout()

```

This will create a grid of images, where each image has its actual class label and the label that the model predicted. The `if` loop from lines 5 through 8 detects incorrect predictions and "highlights" any by plotting that image with a color palette. All correct predictions will be displayed as grayscale images.

- b) Select the cell that contains the code listing, then select **Run**.
c) Examine the output.



Of the first 25 images, a few were incorrectly classified.



Note: As before, your results may differ due to randomness.

12. Examine the incorrect predictions in the preceding screenshot (rather than your own results). Focus on the actual label vs. the predicted label for each image.

What can you tell about these incorrect predictions? From the perspective of your own human judgment, does it make sense that these images might be misclassified in the way that they were?

13. What are some ways you might retrain this CNN model to improve its skill?

14. Shut down this Jupyter Notebook kernel.

- a) From the menu, select **Kernel→Shutdown**.
 - b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
 - c) Close the **NeuralNetworks-Fashion** tab in Firefox, but keep a tab open to **CAIP/NeuralNetworks/** in the file hierarchy.
-

ACTIVITY 10-3

(Optional) Building a CNN to Classify Handwritten Characters

Data File

/home/student/CAIP/Optional/CNN.ipynb

Before You Begin

Jupyter Notebook is open.

Scenario

A company that provides online ancestry search services has contracted with you to develop tools that they can use to convert historical documents scanned as images into content stored in searchable databases. Many of the original documents are handwritten, making the task more challenging.

As a proof-of-concept for how a CNN might be used to process scanned images, you will create a model to identify scanned characters in the MNIST database, which contains 60,000 scanned characters in the training set and 10,000 scanned characters in the test set.



Note: Sometimes, when you run code containing logic errors or bugs, you may corrupt the data contained in variables you created in previous code cells. To clear out such problems, you can select **Kernel→Restart & Clear Output**, then run each code cell again.

1. Open the CNN notebook.

- From Jupyter Notebook, select **CAIP/Optional/CNN.ipynb** to open it.



Note: Be careful *not* to open the solution file.

- Observe the notebook.

Placeholder code cells have been provided in which you can add your own code. Comments provide hints on tasks you might perform in each code cell. The first code cell has already been completely programmed for you.

2. Import software libraries and load the dataset.

- Select the code listing under **Import software libraries and load the dataset**.
- Run the code and examine the results.

The dataset is loaded for you. There are 60,000 training records and 10,000 test records.

3. Get acquainted with the dataset.

- In the code block under **Get acquainted with the dataset**, write statements to show dimensions of the training and testing sets and their labels.
- Run the code cell and verify that the shape of the training data, training labels, testing data, and testing labels are shown.

4. Visualize the data examples.

- In the code block under **Visualize the data examples**, write statements to show a preview of the first 20 images in the training dataset. Label each image with its class, which you can obtain from the training label set.
- Run the code cell and verify that the first twenty images are shown along with their labels.

5. Prepare the data for training with Keras.

- In the code block under **Prepare the data for training with Keras**, write statements to reshape arrays to add the grayscale flag, and use one-hot encoding to encode the data for each label. Print the one-hot encoding data for the first image to confirm that it's being generated properly.
- Run the code cell and verify that one-hot encoding for the first image is shown.

6. Split the datasets.

- In the code block under **Split the datasets**, write statements to split the training and validation datasets and their labels. To confirm the split, print the shape of the training set, validation set, training labels, and validation labels.
- Run the code cell and verify that the data is being split into a training set, validation set, training label set, and validation label set.

7. Build the CNN structure.

- In the code block under **Build the CNN structure**, write statements to import the required libraries, create the model, add model layers, and print a message to confirm the structure has been built. The layers you will use for this model include:
 - Conv2D, 64 nodes, kernel size of 3, with ReLU activation, and an input shape of (28, 28, 1)
 - Conv2D, 32 nodes, kernel size of 3, with ReLU activation
 - Flatten (to connect the convolution and dense layers)
 - Dense, 10 nodes, with softmax activation
- Run the code cell and verify that the structure has been built.

8. Compile the model and summarize the layers.

- In the code block under **Compile the model and summarize the layers**, write statements to compile the model and summarize its layers once it's been compiled. Use the following options when you compile:
 - Use the `adam` optimizer.
 - Use the `categorical_crossentropy` loss function.
 - Use the `accuracy` metric.
- Run the code cell and verify that the model has been compiled using the layers you specified.

9. Plot a graph of the model.

- In the code block under **Plot a graph of the model**, write statements to plot a graph of the model.
- Run the code cell and verify that the graph is shown.

10. Train the model.

- In the code block under **Train the model**, write statements to train the model over 1 epoch using the training data and labels.
- Run the code cell and wait for the model to be trained.

11. Evaluate the model on the test data.

- In the code block under **Evaluate the model on the test data**, write statements to evaluate the model on the test data, printing messages to show the loss and accuracy.
- Run the code cell and observe the loss and accuracy values.

12. Make predictions on the test data.

- In the code block under **Make predictions on the test data**, write statements to make predictions on the test data, showing the first 30 examples of actual values compared to predictions.

- b) Run the code cell and compare the actual values to the predicted values.

13. Visualize the predictions for 30 examples.

- a) In the code block under **Visualize the predictions for 30 examples**, write statements to show the first 30 predictions along with the image, highlighting any incorrect predictions in color.
b) Run the code cell and compare the actual values to the predicted values.

14. In Jupyter Notebook, open CAIP/Optional/CNN-Solution.ipynb and compare it to the code you wrote.



Note: Since there are many ways to write code to do the same basic thing, don't expect your code to match exactly. The important thing is that your code accomplishes the same basic goals, and returns similar results.

15. Shut down the Jupyter Notebook kernels.

- a) From the menu, select **Kernel→Shutdown**.
b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
c) Close the **CNN** tab in Firefox.
d) Repeat this process to shut down and close the **CNN-Solution** kernel.
e) Return to **CAIP/Neural Networks/** in the file hierarchy.
-

TOPIC C

Build Recurrent Neural Networks (RNN)

The last type of neural network you'll build is a recurrent neural network (RNN), which is most often used to process natural languages.

Recurrent Neural Network (RNN)

The neural networks discussed thus far have been ***feedforward neural networks (FNNs)***—information flows to and from neurons of different layers in a single direction. A ***recurrent neural network (RNN)*** enables information to flow in two directions through looped connections in the network. RNNs work with sequences of information, and therefore incorporate time as an important component. This makes them ideal in many different circumstances, especially when you're interested in predicting the next link in a series. For example, the primary application of RNNs is in natural language processing (NLP); words, letters, and sentences are treated as separate instances in time, each one occurring either before or after some other word, letter, or sentence. A traditional FNN is not ideal for this, because it would be unable to account for this sequential context.

In an RNN, a recurrent neuron is a neuron that takes an input vector \mathbf{x} at time step t , while also taking the output scalar y of the previous time step ($t - 1$). When applied to a single neuron, this would look like the following figure.

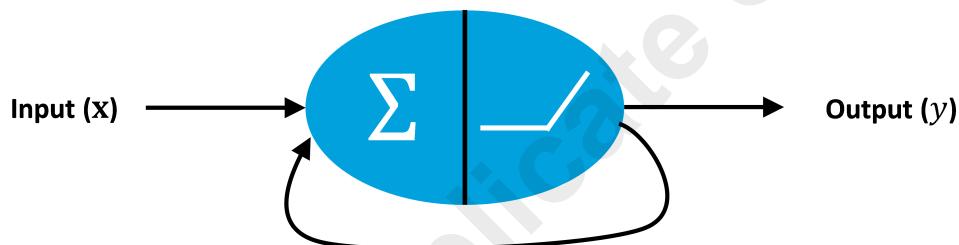


Figure 10-16: A single recurrent neuron.

However, most RNNs will place several recurrent neurons in one layer. So, the output scalar y becomes a vector \mathbf{y} , which incorporates the output of all neurons in the layer.

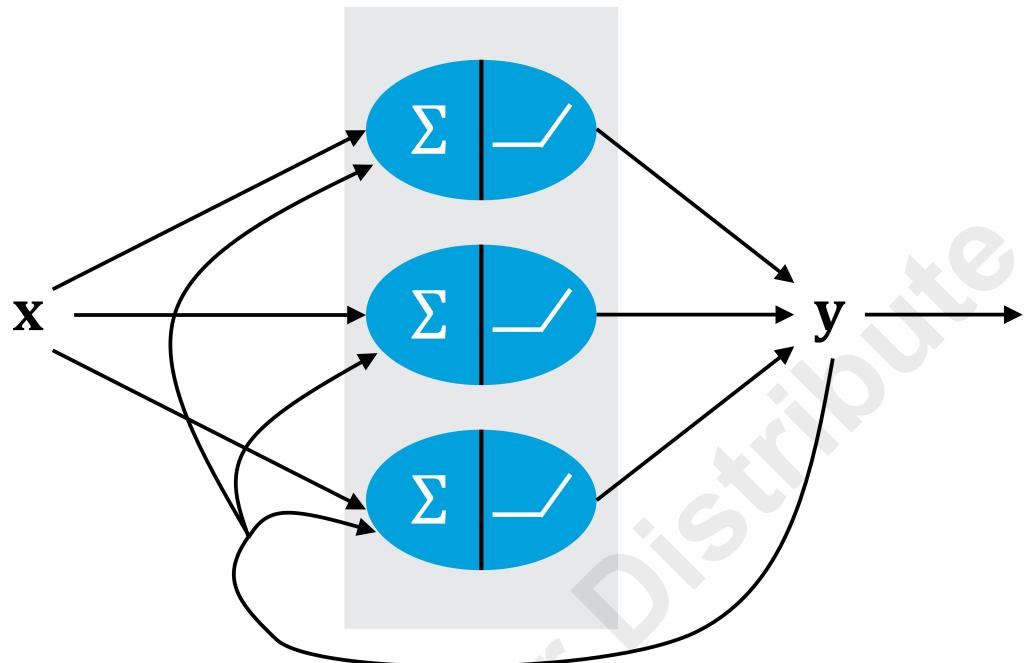


Figure 10–17: Several recurrent neurons in a layer.

When multiple RNN layers are represented in a time sequence—a process that is also called *unrolling*—you get something like the following figure.

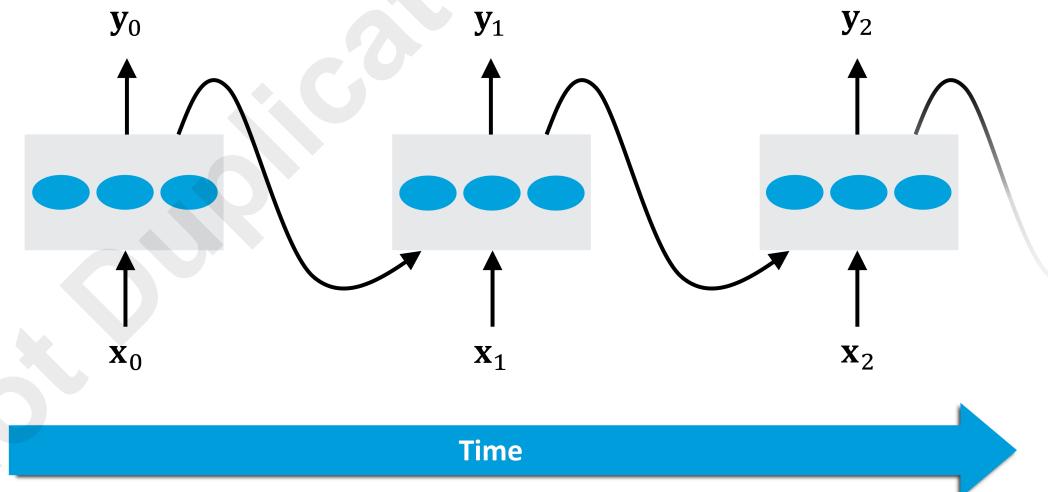


Figure 10–18: Unrolling several layers of recurrent neurons through time.

Ultimately, the layer of recurrent neurons has a matrix of weighted input values (w_x) and a matrix of weighted output values (w_y) based on the whole time sequence. An RNN can use an activation function like ReLU to calculate an output for each layer based on these weights.

Memory Cell

A recurrent neuron or a layer of recurrent neurons has something akin to memory—it is able to incorporate the output from all of the neurons/layers that came before it in time. This means that

such neurons and layers can be described as ***memory cells***, or components of the network that maintain a certain state in time. In the most basic memory cells, the state of the cell (also called its *hidden state*) is the same as its output. In other words, a cell's state and its output are the functions of the previous state/output (y_{t-1}) and the current input (x_t). The hidden state is often represented as the vector h_t .

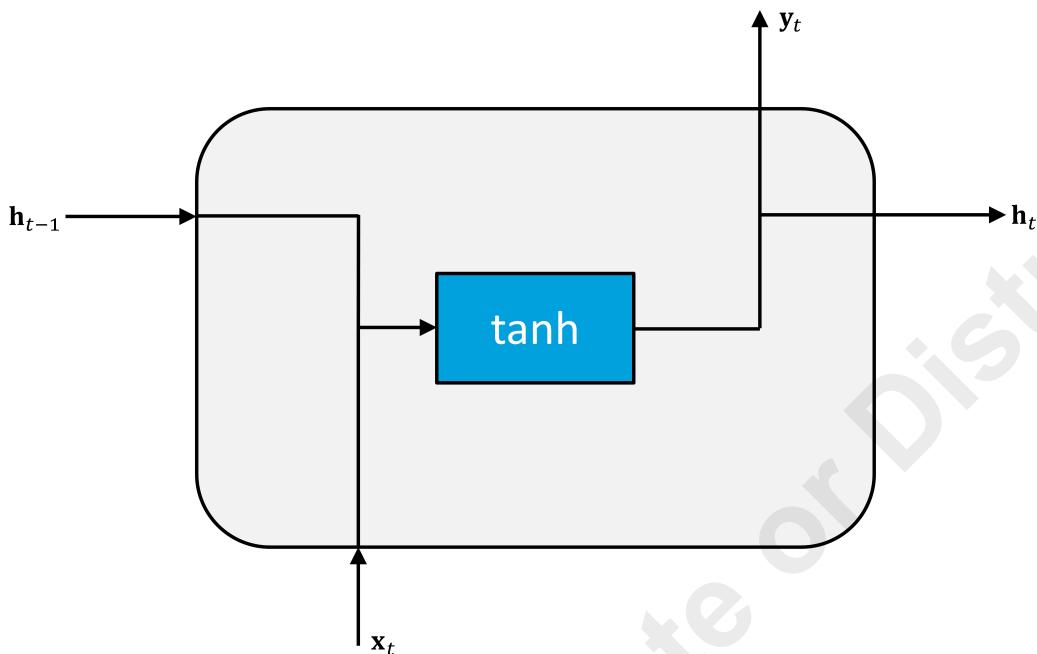


Figure 10-19: A basic memory cell with input, output, and hidden state values. The tanh activation function is applied.

RNN Training

RNN training is performed through a process called ***backpropagation through time (BPTT)***. In BPTT, the time sequence of RNN layers is first unrolled, and then backpropagation is performed just like it is using a traditional ANN. The error gradients between actual and predicted values are identified starting at the last time step (t), then these gradients are propagated backward for the next layer's error calculation, and so on, until reaching the first time step. The weights between neurons are updated as a result.

RNN Shortcomings

RNNs can suffer performance issues if they are fed long sequences of inputs. The unrolled layers may lead to computational inefficiency, slowing down the training process. This can be mitigated somewhat by reducing the number of time steps along which the network is unrolled. Skipping some of the time steps will likely reduce the number of computations that are necessary to train the model. However, this puts the model at risk of not being able to learn patterns in the input that only manifest over the long term. Skipping time steps can therefore lead to a reduction of the model's skill.

Another major issue with RNNs involves the use of the basic memory cells mentioned earlier. As data traverses a neural network, and inputs lead to outputs throughout the multiple layers and neurons, that data is subjected to many different calculations. Eventually, some of the initial data is lost in the network, and later memory cells start missing key pieces of information from the earlier inputs. For example, let's say you feed your RNN a customer review of a restaurant. The model's

goal is to determine the customer's overall feelings about the restaurant. The review starts with the customer stating, "I really like this place, but ..." The customer then spends the rest of the review offering suggestions for improvement. A traditional RNN may end up "forgetting" the first few words and incorrectly determine that the review was negative.

These issues led to the development of more complex and effective memory cells.

Long Short-Term Memory (LSTM) Cell

Earlier, the structure of a basic memory cell was described as the cell's state being the same as its output. In a **long short-term memory (LSTM) cell**, this is not always the case. LSTM cells have a hidden state that is not just one vector, but two: a short-term state (\mathbf{h}_t) and a long-term state (\mathbf{c}_t). They were developed to overcome the limitations mentioned previously, namely that they are able to preserve input that is significant to the training process, while "forgetting" input that is not. This also has the effect of speeding up the training process.

Like a basic memory cell, LSTM cells are very useful in NLP. Their ability to retain important information, while disposing of unimportant information, makes them highly suitable to solving issues that require the sequential processing of input. This also makes them a powerful component of RNNs used in forecasting time series. They can predict future trends and data values based on past information, such as the total sales a business can expect to generate over a certain period of time, or how much bandwidth a server farm will consume within a certain time period, and many more such examples.



Note: LSTM cells can also work with input sequences of variable lengths.

LSTM Cell Architecture

An LSTM cell consists of four fully connected layers. All of these layers are fed information from the input vector \mathbf{x}_t and the previous cell's short-term memory vector \mathbf{h}_{t-1} . The layers process this information and contribute to the cell's decisions to preserve or drop certain information.

The tanh layer, represented as \mathbf{g}_t , is like the basic memory cell shown earlier—it applies the tanh activation function to \mathbf{x}_t and \mathbf{h}_{t-1} . However, unlike the basic cell, its results are transformed in several ways before being added to memory.

Aside from this tanh layer, the other three layers are referred to as *gates*. Each gate applies the sigmoid activation function (σ) on \mathbf{x}_t and \mathbf{h}_{t-1} to constrain values between 0 and 1. Values close to 0 are forgotten; values close to 1 are kept. These gates are:

- \mathbf{f}_t , the *forget* gate. This gate determines what should be removed from the long-term cell.
- \mathbf{i}_t , the *input* gate. This gate identifies the important information to keep in long-term memory.
- \mathbf{o}_t , the *output* gate. This gate determines what should be included in the next cell's short-term memory.

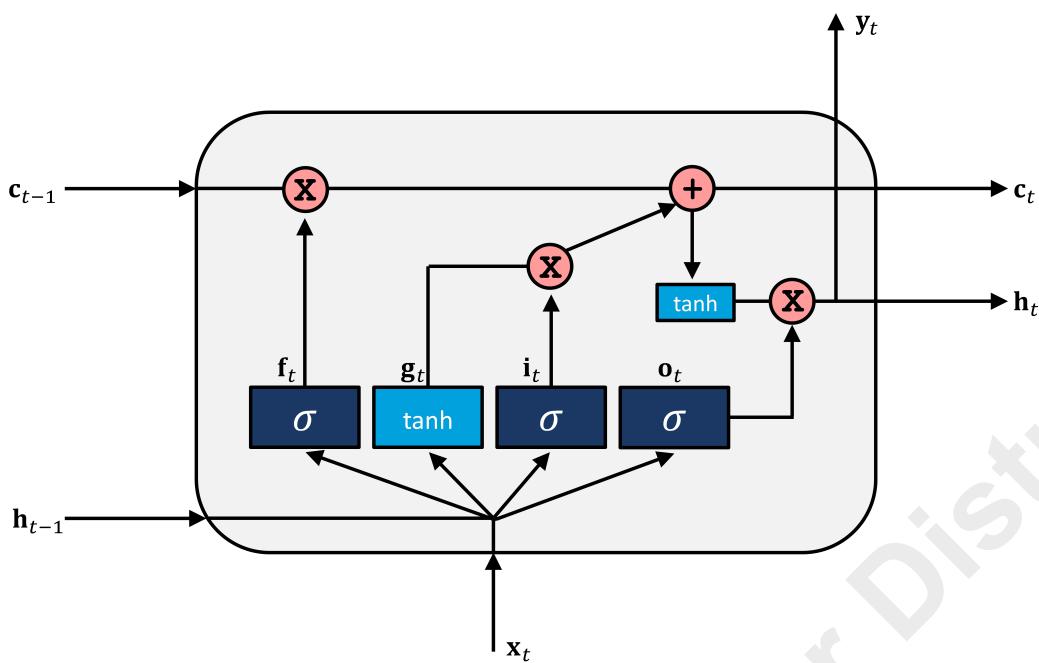


Figure 10–20: The architecture of an LSTM cell.

LSTM Cell Process

Now that you know the roles played by the different layers/gates, the flow of information in an LSTM cell can be described in the following process:

1. Starting from the left side of the diagram, the previous cell's long-term state (c_{t-1}) is pointwise multiplied by the result of the forget gate (f_t). A pointwise operation is one that is applied separately to each value within a function.
2. The result of the input gate (i_t) is pointwise multiplied by the result of the tanh function layer (g_t).
3. The result from step 1 is pointwise added to the result from step 2. This produces the next cell's long-term state, c_t .
4. On the right of the cell diagram, the result from step 3 is run through a tanh function, then this is pointwise multiplied by the result of the output gate (o_t). The result of this operation is the next cell's short-term state, h_t . The result may also be output as y_t .

Gated Recurrent Unit (GRU) Cell

The **gated recurrent unit (GRU) cell** is a simplified version of the LSTM cell that has seen some success. In a GRU cell, both short-term and long-term states are merged into one. The forget and input gates are also merged into one, where a result of 0 closes the forget gate and opens the input gate, and a result of 1 opens the forget gate and closes the input gate. The last difference is that there is a reset gate rather than an output gate. The reset gate determines how much of the previous state to forget. Ultimately, the simplification of GRU cells may improve training speed over LSTM cells in certain circumstances.

Embedding

In the context of RNNs, **embedding** is the process of condensing a language vocabulary into vectors of relatively small dimensions. These vectors end up being much more efficient and easier to

process than the traditional method of one-hot encoding words in a vector of large dimensions. For example, the average adult American knows the meaning of about 40,000 words in the English language. In a one-hot encoding scheme, there'd be 40,000 dimensions in a single vector, with each word in an element. If the RNN needed to identify a single word—"blue," for instance—39,999 of those elements would be 0, and only one would have a value of 1. This presents a significant performance problem to neural networks that must work with very large language-based inputs. Embedding helps solve this problem.

Embedding represents each word as its own vector, usually no more than a few hundred dimensions. These vectors constitute the overall embedded space. A word's vector occupies a certain point within this space. These embedding vectors are trained on a fully connected neural network. At first the vectors are initialized to random values, but as they are trained on some language input, the similarities between the words they represent are identified by the network. Eventually, words with similar meanings are placed close to each other within the embedded space. This enables the network to group together words in meaningful ways without any direct guidance; for example, words might be placed in groups that represent the part of speech they're most often used in: subjects, objects, verbs, adjectives, etc. So, the RNN will be better at recognizing that the sentence "His shirt is blue" is valid, whereas the sentence "His shirt is computer" is not.

Many machine learning practitioners before you have already trained useful word embeddings, so you should consider leveraging these pre-trained embeddings in your own projects. However, you may wish to train your own embeddings. There are several tools that you can use to do this. Some of the most popular include:

- **Word2vec**, which takes a large corpus of text as input and produces an embedded space in which each word is a vector. This is the traditional approach.
- **Doc2vec**, an extension of Word2vec that generates vectors from entire documents, rather than single words. This can increase the speed of training in some circumstances.
- **fastText**, another extension of Word2vec, which partitions words into *n*-grams. For example, the *n*-grams for the word "phone" might be "pho," "hon," and "one." The vector for "phone" is the sum of these *n*-grams. The *n* determines the length of each partition, e.g., the example just given has partitions of length three—also called *trigrams*. If the partitions were two letters in length, they'd be *bigrams*, and so on. The *n*-gram approach has the advantage of generating more effective embeddings for rare words, as these words may have some *n*-grams in common with other, more common words.

Bag-of-Words

Embedded vectors are typically not created directly from textual content. Instead, some degree of pre-processing is involved. One of the most common ways to represent text for training is the **bag-of-words** approach. A bag of words is simply a list of each individual word without respect to grammar, punctuation, or any other language component that may be extraneous to the word itself. For example, consider the sentence, "I really enjoyed this movie—more than I enjoyed the first one." As a bag of words, this could be represented as:

```
['I', 'really', 'enjoyed', 'this', 'movie', 'more', 'than', 'I', 'enjoyed',
'the', 'first', 'one']
```

From a processing standpoint, this can be condensed further by assigning it to a dictionary, where each key is a word and each value is its frequency:

```
{'I': 2, 'really': 1, 'enjoyed': 2, 'this': 1, 'movie': 1, 'more': 1, 'than': 1,
'the': 1, 'first': 1, 'one': 1}
```

A similar approach is to use a bag of *n*-grams, in which each element of the list/dictionary is an *n*-gram rather than a word. This is a common approach to representing features for classifying text into different languages.

Guidelines for Building RNNs

Follow these guidelines when you are building recurrent neural networks (RNNs).

Build an RNN

When building an RNN:

- Use RNNs for classifying text, recognizing speech, or performing other natural language processing-related tasks.
- Use RNNs for predicting time sequence data.
- Rather than standard memory cells, use a long short-term memory (LSTM) in RNNs to optimize the memorization of important input and reduce training time.
- Consider using a gated recurrent unit (GRU) cell for a possible improvement over LSTMs in terms of training time.
- Rather than use a typical encoding scheme to represent individual words in a text, use word embeddings to condense the language vocabulary into relatively small vectors.
- To save on training time, consider relying on pre-trained word embeddings that other machine learning practitioners have made available to the public.
- If you wish to train your own embeddings, consider using tools like Word2vec or fastText.

Use Python for RNNs

Like with CNNs, you can use the Keras `Sequential()` class and the `layers` module to build an RNN. The following are some of the objects and functions you can use to build an RNN.

- `network = keras.models.Sequential()` —This constructs the sequential network object.
- `network.add(keras.layers.Embedding(input_dim = 5000, output_dim = 200, input_length = 1000))` —This constructs an embedding layer. In this case, the vocabulary size is 5,000 words, the word vector is 200 dimensions, and the length of the input text is 1,000 words.
- `network.add(keras.layers.LSTM(units = 128))` —This creates an LSTM cell of the specified dimensions.
- You can use the same `Flatten()` and `Dense()` layers where necessary.
- You can use the same object methods as with a Keras CNN to compile the model, fit the training data, evaluate the model, predict test data, etc.

ACTIVITY 10-4

Building an RNN

Data File

/home/student/CAIP/Neural Networks/NeuralNetworks-IMDb.ipynb

Before You Begin

Jupyter Notebook is open.

Scenario

Another major component of your online storefront is user reviews. Users can leave written reviews for any product they purchase. These are typically more insightful than the normal five-star rating system, as such a system does not allow for nuanced and well-formed opinions. The business can learn a lot about which products are doing well in the public eye and which are doing poorly by considering these reviews. Up until now, human personnel have been reading through each review to determine whether that review is positive, negative, or neutral. This is tedious and an unnecessary waste of time, so you want to automate the process.

A machine should be able to quickly "read" through each review and, based on its language usage, determine whether the user did or did not like a product. This is sentiment analysis—a natural language processing (NLP) task—and one that can be dealt with by creating a recurrent neural network (RNN). So, you'll create a review classifier using an RNN.

1. Load the notebook and dataset.

- a) From Jupyter Notebook, select **CAIP/Neural Networks/NeuralNetworks-IMDb.ipynb** to open it.
- b) View the code cell beneath the **Import software libraries and load the dataset** title, then select **Run**.
- c) Verify that the training and test sets were both loaded with 25,000 records.
 - This is a dataset of movie reviews from IMDb, an online database that catalogues various information about movies, TV shows, and other video content. The dataset was loaded with Keras, which you'll use to build and train an RNN.
 - Rather than load every value of the dataset, the `num_words` argument is limiting the values to only the 10,000 most common words. Any word not within these parameters is represented with an out-of-value character. Constraining the dataset in this manner will help reduce training time.

2. Get acquainted with the dataset.

- a) Scroll down and select the code cell beneath the **Get acquainted with the dataset** title, then select **Run**.

- b) Examine the output.

```
First example features:
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 109, 43, 838, 112, 50, 670, 2
, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 236, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16,
6, 147, 2025, 19, 44, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17,
12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5,
25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256
, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 409, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 20
71, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 486, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104
, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]

Label: 1
```

- This output shows the features from just the first data example (i.e., a movie review).
 - Each feature represents, in order, a word in the review.
 - The numerical value of each feature represents the "rank" of the word in terms of its frequency within the dataset. Very common words are assigned lower numbers, whereas uncommon words are assigned very high numbers.
 - The label of this example is 1. This dataset poses a binary classification problem, in which 1 represents a review that offers a positive sentiment about a movie, and 0 represents a review that offers a negative sentiment about a movie. Neutral reviews are not included in this dataset.
- c) Scroll down and select the next code listing cell.

```
In [ ]: 1 # Decode sequence values into actual text.
2 index = datasets.imdb.get_word_index()
3 index_dict = dict([(value, key) for (key, value) in index.items()])
4 decode = ' '.join([index_dict.get(i - 3, '?') for i in X_train[0]]) # Replace unknown words with '?'
5 print(decode)
```

- d) Select Run.
e) Examine the output.

```
? this film was just brilliant casting location scenery story direction everyone's really suited the part they played an
d you could just imagine being there robert ? is an amazing actor and now the same being director ? father came from the
same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks througho
ut the film were great it was just brilliant so much that i bought the film as soon as it was released for ? and would r
recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what
they say if you cry at a film it must have been good and this definitely was ? to the two little boy's that played
the ? of norman and paul they were just brilliant children are often left out of the ? list i think because the stars th
at play them all grown up are such a big profile for the whole film but these children are amazing and should be praised
for what they have done don't you think the whole story was so lovely because it was true and was someone's life after a
ll that was shared with us all
```

- Each numerical feature value has been converted to a word. This is done by mapping the values to an existing dictionary for the dataset.
- A question mark (?) indicates a word that is not known.
- Comparing the actual words to their numerical values, you can see that the rankings make sense. For example, the second word ("this") has a rank of 14, which is rather common. The eighth word ("location") has a rank of 1,622, indicating it is much less common.
- The text of the review is streamlined; there is no punctuation, nor are there any capital letters. This helps to simplify the input for the neural network.
- Despite this, the text of the review is still legible, and you can see why it was labeled as being positive.

3. Examine some statistics about the reviews.

- a) Scroll down and select the code cell beneath the **Examine some statistics about the reviews**, then select Run.

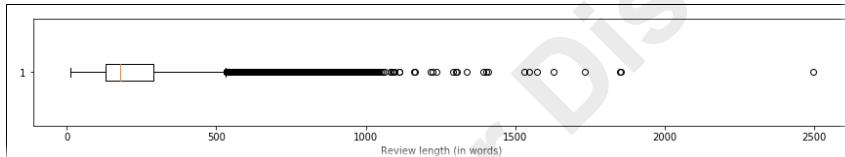
- b) Examine the output.

```
Mean review length (in words): 239
Standard deviation (in words): 176
```

- The average length of a review is around 239 words.
 - The standard deviation is around 176 words.
- c) Scroll down and select the next code listing cell.

```
In [ ]: 1 plt.figure(figsize = (15, 2))
2 plt.boxplot(result, vert = False)
3 plt.xlabel('Review length (in words)')
4 plt.show()
```

- d) Select **Run**.
- e) Examine the output.



- This box plot confirms the mean and standard deviation figures.
- Looking at the upper whisker, it appears that the vast majority of reviews are under 500 words long. In an effort to simplify the model, you'll use this number as a guide when feeding these reviews as input.

4. Add padding to the data.

- a) Scroll down and view the cell titled **Add padding to the data**, and examine the code listing below it.

Add padding to the data

```
In [ ]: 1 from keras.preprocessing import sequence
2
3 X_train = sequence.pad_sequences(X_train, maxlen = 500)
4 X_test = sequence.pad_sequences(X_test, maxlen = 500)
5
6 print('Number of features: {}'.format(X_train.shape[1]))
```

- Because the majority of the reviews are 500 words long, you'll truncate all of the reviews to that length. This is part of what the `pad_sequences()` function does.
 - For reviews that are under 500 words long, `pad_sequences()` adds padding so that they are all the same size. This will make all of the input the same shape, simplifying the model.
- b) Select the cell that contains the code listing, then select **Run**.
- c) Examine the output.

```
Number of features: 500
```

This confirms that the number of features in each data example is 500. In other words, there are 500 words (known or unknown) for each review.

5. Split the datasets.

- a) Scroll down and select the code cell beneath the **Split the datasets** title, then select **Run**.

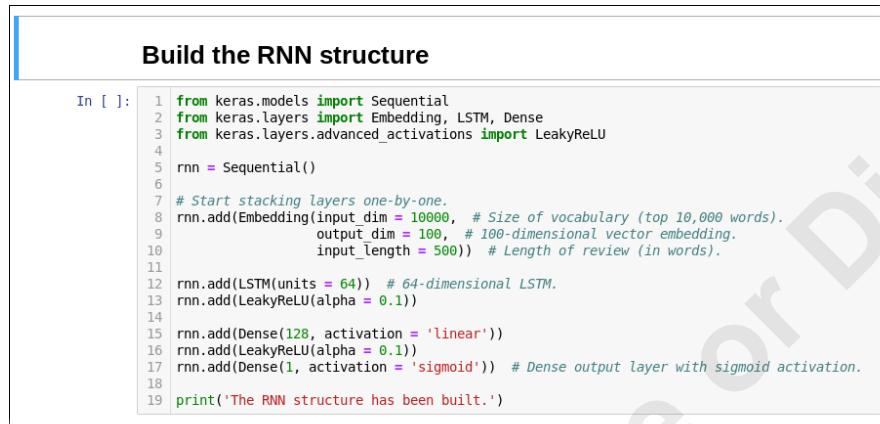
- b) Examine the output.

```
Training features:      (18750, 500)
Validation features:   (6250, 500)
Training labels:       (18750,)
Validation labels:     (6250,)
```

You're splitting the training dataset in order to have a validation holdout. Although the test set you loaded at the start is labeled, you'll treat it as the ultimate test case.

6. Build the RNN structure.

- a) Scroll down and view the cell titled **Build the RNN structure**, and examine the code listing below it.



Build the RNN structure

```
In [ ]:
 1 from keras.models import Sequential
 2 from keras.layers import Embedding, LSTM, Dense
 3 from keras.layers.advanced_activations import LeakyReLU
 4
 5 rnn = Sequential()
 6
 7 # Start stacking layers one-by-one.
 8 rnn.add(Embedding(input_dim = 10000, # Size of vocabulary (top 10,000 words).
 9                   output_dim = 100, # 100-dimensional vector embedding.
10                   input_length = 500)) # Length of review (in words).
11
12 rnn.add(LSTM(units = 64)) # 64-dimensional LSTM.
13 rnn.add(LeakyReLU(alpha = 0.1))
14
15 rnn.add(Dense(128, activation = 'linear'))
16 rnn.add(LeakyReLU(alpha = 0.1))
17 rnn.add(Dense(1, activation = 'sigmoid')) # Dense output layer with sigmoid activation.
18
19 print('The RNN structure has been built.')
```

This code builds the actual structure of the RNN with Keras.

- You're using the `Sequential()` class to build the network structure layer-by-layer.
- Lines 8 through 10 add the first layer in the stack. The `Embedding()` object builds an embedded input layer. This particular layer has the following hyperparameters/arguments:
 - `input_dim` specifies the size of the input vocabulary. Recall that you constrained the input to the most common 10,000 words, so those are the dimensions of the input.
 - `output_dim` specifies the size of the embedding vector that the model will train. You could load pre-trained word embeddings, but in this case, the model will train its own. This model will create an embedding vector of 100 dimensions.
 - `input_length` specifies the total length of the input, which, in this case, is the number of words in a review. Recall that you truncated/padded each review so that they are all 500 words long.
- Line 12 adds a long short-term memory (LSTM) cell as the next layer. In this case, the cell's output will have 64 dimensions. This is an arbitrary size and can be tuned to improve performance, if necessary.
- Line 13 adds a leaky ReLU layer for the activation function.
- Lines 15 and 16 add a fully connected (dense) layer with leaky ReLU activation. In this case, the dense layer has an arbitrary size of 128.
- Line 17 is the final output layer. It has a single output and uses the sigmoid activation function to generate a binary classification decision.

- b) Select the cell that contains the code listing, then select **Run**.

```
The RNN structure has been built.
```

7. What is word embedding, and why might it be beneficial to use in this case?

8. In an LSTM cell, which of the following activation functions is used by the forget gate (f_t)?

- Hyperbolic tangent (tanh)
- Sigmoid
- Rectified linear unit (ReLU)
- Leaky ReLU

9. Compile the model and examine the layers.

- a) Scroll down and view the cell titled **Compile the model and examine the layers**, and examine the code listing below it.

Compile the model and examine the layers

```
In [ ]: 1 rnn.compile(optimizer = 'adam',
2                      loss = 'binary_crossentropy',
3                      metrics = ['accuracy'])
4
5 rnn.summary()
```

The `compile()` method takes the Keras RNN object built in the previous code block and configures it for training. As with the CNN you built earlier, you'll use the `adam` optimizer and will return the model's accuracy score. You'll use the `binary_crossentropy` loss function, which is similar to `categorical_crossentropy`, except that it is more suited to binary classification problems.

- b) Select the cell that contains the code listing, then select **Run**.
c) Examine the output.

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 500, 100)	1000000
lstm_1 (LSTM)	(None, 64)	42240
leaky_re_lu_1 (LeakyReLU)	(None, 64)	0
dense_1 (Dense)	(None, 128)	8320
leaky_re_lu_2 (LeakyReLU)	(None, 128)	0
dense_2 (Dense)	(None, 1)	129

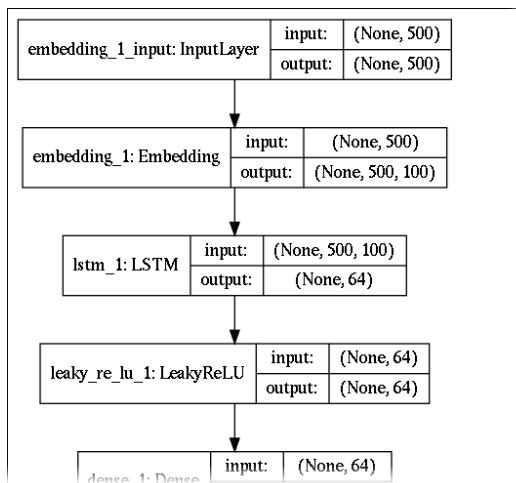
Total params: 1,050,689
Trainable params: 1,050,689
Non-trainable params: 0

This provides an overview of the RNN's structure. As with the CNN, each layer is listed, along with its output shape and number of parameters.

- d) Scroll down and select the next code listing cell.

```
In [ ]: 1 from keras.utils import plot_model
2 plot_model(rnn, show_shapes = True, to_file = 'model2.png')
```

- e) Select **Run**.
- f) Examine the output.



10. Train the model.

- a) Scroll down and view the cell titled **Train the model**, and examine the code listing below it.

Train the model

```
In [ ]: 1 rnn_trained = rnn.fit(X_train, y_train,
 2 validation_data = (X_val, y_val),
 3 epochs = 1,
 4 verbose = 1)
```

You're using the training dataset along with the validation set to fit the RNN model. Due to classroom time constraints, you'll only train for one epoch. In a real-world scenario, you'd want to train for several epochs.

- b) Select the cell that contains the code listing, then select **Run**.
- c) Examine the output.

```
Train on 18750 samples, validate on 6250 samples
Epoch 1/1
768/18750 [>.....] - ETA: 3:45 - loss: 0.6929 - accuracy: 0.5286
```



Note: You can ignore the warning about converting to a dense tensor of unknown shape.



Note: It may take up to 20 minutes for training to complete.

- d) While you wait, observe how the training loss decreases over time, while the accuracy gradually increases.

- e) When training is complete, examine the final scores on the validation set.

```
Train on 18750 samples, validate on 6250 samples
Epoch 1/1
18750/18750 [=====] - 232s 12ms/step - loss: 0.4475 - accuracy: 0.7834 - val_loss: 0.3935 - val_accuracy: 0.8522
```



Note: Since there is some degree of randomness, your results may not align exactly with the results in the screenshot.

11. Evaluate the model on the test data.

- a) Scroll down and view the cell titled **Evaluate the model on the test data**, and examine the code listing below it.

Evaluate the model on the test data

```
In [ ]: 1 eval_test = rnn.evaluate(X_test, y_test, verbose = 0)
          2
          3 print('Loss: {}'.format(round(eval_test[0], 2)))
          4 print('Accuracy: {:.0f}%'.format(eval_test[1] * 100))
```

Since your test set is labeled, you'll evaluate the model on that test set as well.

- b) Select the cell that contains the code listing, then select **Run**.
c) Examine the output.

```
Loss: 0.4
Accuracy: 85%
```

The scores on the test set should be fairly close to the scores on the validation set.



Note: It may take a few minutes for the evaluation to finish.

12. Make predictions on the test data.

- a) Scroll down and select the code cell beneath the **Make predictions on the test data** title, then select **Run**.
b) Examine the output.

```
Actual class: [0 1 1 0 1 1 1 0 0 1]
Predicted class: [0 1 1 0 1 1 1 0 1 1]
```

The predictions align with the actual class labels for most of the reviews.



Note: As before, your results may differ due to randomness.

13. Examine a review that was correctly classified.

- a) Scroll down and select the code cell beneath the **Examine a review that was correctly classified** title, then select **Run**.

- b) Examine the output.



Note: The following summarizes the review in the preceding screenshot. The review that was pulled from your model may be different due to randomness.

- The model correctly predicted that this review is negative.
 - Reading the review seems to confirm this—the user obviously didn't like the movie.
 - Because this review is under 500 words long, it was padded at the beginning with unknown word values (question marks).

14. Examine a review that was incorrectly classified.

- a) Scroll down and select the code cell beneath the **Examine a review that was incorrectly classified** title, then select **Run**.
 - b) Examine the output.



Note: The following summarizes the review in the preceding screenshot. The review that was pulled from your model may be different.

- The model incorrectly predicted this review as being positive, when it is in fact negative.
 - Reading the review, you can tell that it is negative, but you may be able to identify reasons why the model thought it was positive.
 - As before, the review is padded to be 500 words long.

15. Examine the incorrect prediction in the preceding screenshot (rather than your own results).

Why do you think the model incorrectly classified this review as positive?

16.What are some ways you might retrain this RNN model to improve its skill?

17.Shut down Jupyter Notebook.

- a) From the menu, select **Kernel**→**Shutdown**.
 - b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
 - c) Close all tabs in Firefox, then close the terminal window that is running the program.
-

Summary

In this lesson, you built several different kinds of artificial neural networks (ANNs). You built a basic multi-layer perceptron (MLP) to solve tasks with large volumes of data; you built a convolutional neural network (CNN) for image processing; and you built a recurrent neural network (RNN) for natural language processing. These types of ANNs have different applications, but all of them are powerful methods for revealing insights into voluminous and complex datasets.

In your own environment, how might you use convolutional neural networks (CNNs) to solve business problems?

In your own environment, how might you use recurrent neural networks (RNNs) to solve business problems?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Do Not Duplicate Or Distribute

11

Promoting Data Privacy and Ethical Practices

Lesson Time: 1 hour, 30 minutes

Lesson Introduction

With access to broad datasets and the capability of extracting meaningful insights from them, you have the responsibility to protect the privacy of users and organizations that you serve.

Lesson Objectives

In this lesson, you will:

- Protect data privacy.
- Promote ethical practices in AI and machine learning projects.
- Comply with policies to promote data privacy and ethical practices.

TOPIC A

Protect Data Privacy

When you work with other people's data, you are obligated—often by law—to protect their privacy.

Protected Data

AI/ML projects are heavily based on data, and often that data involves **personally identifiable information (PII)**, which must be protected to ensure the privacy of the people described by that data. PII is associated with an individual person, such as an employee, customer, or patient. PII can be used to uniquely identify, contact, or locate an individual. Examples include a person's name, email address, home address, Social Security number (even if it's just the last 4 digits), and so forth.

PII that has been anonymized (handled in a way that eliminates any association with a specific person) is not considered sensitive. In more than 80 countries, government laws and regulations require you to protect any PII that is not publicly available information, and to inform the individual of the types of data being collected, its reason for collection, and planned uses of the data. This includes information stored or transmitted in various forms.

Obligation to Protect PII

Workers within an organization are obligated to protect PII to meet the company's policies, standards, and management directives—as well as numerous laws, regulations, and mandates the company is subject to. They must control access to PII so they can meet these obligations.

The more PII is accessed by people and systems, the greater the opportunity for confidentiality to be compromised. Organizations need to consider what levels of access to PII will enable them to maintain their legal obligations, and may choose to put more rigorous restrictions in place than the law requires. For example, if PII is accessed or stored on devices outside the direct control of the organization, they might opt to be more restrictive regarding who can access the data, in order to reduce the increased risk associated with the way the information is accessed.

Quasi-identifiers (QI) are other data values that do not directly contain PII, but may be used in combination with other data values to identify an individual. For example, a database query based on a combination of values such as ZIP/postal code, age, and gender might be used to identify a particular individual, and should therefore be considered when handling data to ensure that privacy is protected.

Relevant Data Privacy Laws

There are many reasons why an organization using PII in AI/ML solutions should implement well-defined processes to ensure data privacy. The need to comply with regulations is one of them. Regulations such as the EU General Data Protection Regulation (GDPR), the California Consumer Privacy Act (CCPA), and California's AB 1950 provide specific requirements that organizations are legally bound to comply with.

GDPR requires that organizations obtain users' permission to process data through a clear affirmation by the user, follow the rules of **Privacy by Design**, and report any data breaches. Special care must be taken when handling the personal data of children. Consumer IoT devices (such as smart speakers, personal fitness trackers, smart home devices, and connected cars) are especially affected by this law.

The CCPA and California's AB 1950 require that organizations provide "reasonable security" to protect consumers' data privacy.

While GDPR protects citizens of the European Union, and the California regulations apply to California residents, they apply to organizations worldwide who handle their data.

Failure to meet requirements can expose organizations to a negligence suit, much as a medical practice can be sued for negligence. In a negligence suit, AI and ML practitioners may be treated like other professionals because they market specialized skills that presume an understanding of issues related to data privacy protection.

The regulations identify measures that an organization should take. To protect themselves and organizations that they work for, practitioners should provide a well-documented plan and maintain records to prove that these measures have been taken.



Note: There are many other laws that address data privacy and security, such as the Health Insurance Portability and Accountability Act (HIPAA) and Sarbanes–Oxley. Make sure you know what laws affect your operations so you can ensure you adhere to them.

Privacy by Design

GDPR requires that organizations adhere to the rules of Privacy by Design. Privacy by Design is an approach to software development that takes privacy into account throughout every phase of development. The underlying premise of Privacy by Design is not simply protecting data, but as much as possible designing systems so data doesn't need protection—for example, minimizing data collected in the first place.

Item	Description
1. Proactive not Reactive; Preventative not Remedial	Data privacy should be considered early and often (throughout the entire development lifecycle), not just after there is a problem.
2. Privacy as the Default Setting	As initially installed, the application should be as private as possible. The user must opt in to decrease privacy to less private settings. By default, restrictions are placed on sharing, data collection, and data retention.
3. Privacy Embedded into Design	Privacy should be built into the design of the software. It should be explicitly included in processes like requirements identification, threat modeling, user interface design, testing, and so forth.
4. Full Functionality—Positive-Sum, Not Zero-Sum	Customers value privacy. It is part of the value that customers pay for.
5. End-to-End Security—Full Lifecycle Protection	Privacy protections should follow the data wherever it goes—when it is first created, shared with others, archived, and deleted.
6. Visibility and Transparency—Keep it Open	Privacy practices should be clear and overt, so users can have confidence in their privacy expectations. Policies and mechanisms should be in place to ensure users can address problems and have them resolved efficiently.
7. Respect for User Privacy—Keep it User-Centric	Users own their data. Data held in the software should be accurate, and the user must have the power to correct errors. The user can grant and revoke consent on the use of the data.

The seven principles of Privacy by Design were initially proposed by Ann Cavoukian, the Information & Privacy Commissioner of Ontario, Canada.

Data Privacy Principles at Odds with Machine Learning

Some of the fundamental principles of GDPR and machine learning seem to be at odds with each other, as shown here.

GDPR Principle	Machine Learning Principle
Less data is better. Companies must minimize the amount of data they collect, limit it to what is strictly necessary for stated purposes, and put limits on how long they hold data.	More data is better. Machine learning is data driven. More data helps to produce more skillful models.
Decisions made about people must be transparent. If personal data is used to make automated decisions that affect people, the organization must be able to clearly explain the logic behind those decisions.	It may be hard to completely know how decisions are made. The model evolves its inner workings as it learns. Details of the model's inner workings may not even be completely clear to the person who created the model.
Data usage must be revealed upfront. Individuals who are the target or subject of the data must be told upfront what data will be collected, and how it will be used.	Data usage emerges through the training process. What data values will actually be used and how they will be used is not known upfront. These things emerge through the training process, and even after the model is completed it may not be completely known how the data is being used. It may be necessary to retain sensitive data within datasets to ensure accurate and fair results, and to properly analyze results for signs of discrimination or bias.
The user is king. The privacy rights of users must be honored, even if it results in reduced system capabilities or performance.	The model is king. The end goal is often to create an optimal model, and a liberal use of data tends to support that goal. In practice, performance and capabilities of the model may need to be compromised to protect data privacy.

In some machine learning scenarios, meeting the transparency requirements of GDPR and other regulations may seem to require compromises that lead to less effective machine learning models.

Guidelines for Complying with Data Privacy Laws and Standards

It is the responsibility of you and your organization to handle data in a way that complies with applicable laws and standards. Follow these guidelines when handling data.



Note: All of the Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Identify Personally Identifiable Information (PII)

Some information may require special care and handling to protect users. Appropriate controls should be applied to any sensitive information to ensure it remains private. A good place to start is to always consider all Personally Identifiable Information (PII) sensitive, as it can be used to establish a person's identity and might be used to cause them substantial harm, embarrassment, inconvenience, or unfairness. Refer to privacy guidelines for your country, municipality, or organization for specific lists of PII you may be legally required to protect. A typical list is provided here.

- User name
- Email address
- Home address
- Phone number
- Social Security number (even if it's just the last 4 digits)
- Driver's license or state ID number
- Passport number
- Alien registration number
- Financial account number
- Biometric identifiers
- Citizenship or immigration status
- Medical information
- Ethnic or religious association
- Sexual orientation
- Account passwords
- Date of birth
- Criminal history
- Mother's maiden name

Identify Personal Data as Defined by GDPR

GDPR defines personal data as "any information relating to an identified or identifiable natural person (data subject). An identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier" such as:

- A name
- An identification number
- Location data
- An online identifier
- One or more factors specific to the data subject's physical, physiological, genetic, mental, economic, cultural, or social identity

Ensure Privacy Policies, Terms, and Conditions Are Clear

Software may not make it clear to users what it will do with their data so users can make good decisions about how to manage their data within the software. Take steps to follow these best practices if your organization develops software. As an end user of software, consider these features when you evaluate and select software.

- Release notes should be included with software updates to clearly and simply explain how terms and conditions change over time.
- Users should be informed of all tracking mechanisms used in the software, explaining how and by whom the information is used.
- Users should be informed (through a clear and well-written terms and conditions page, for example) how data is processed, including collection, storage, processing, and deletion.
- A Do Not Track feature should be implemented on the server side, so users can disable tracking, with an opt-out capability provided for users.

Ensure Software Does Not Share Data Without Consent

The software should not provide data to a third party without obtaining the user's consent. Software should:

- Acquire and document the user's consent for any data collected before the data is actually collected.
- Acquire and document the user's consent for any additional data that is collected later (due to software feature updates or new compliance requirements, for example).
- Anonymize personal data before sharing it with a third party.

- Work with developers to ensure that applications that handle data are secure, protecting data from leakage to unauthorized users.

Do Not Duplicate or Distribute

ACTIVITY 11-1

Complying with Applicable Laws and Standards

Scenario

In 2018 it was revealed that there had been a major violation of data privacy on the Facebook social media platform. This wasn't due to a cyberattack or security breach, but rather a deliberate sharing of user data.

The data was obtained through a third-party Facebook app named *This is your digital life*, which was created by a researcher at Cambridge University's Psychometrics Center. Approximately 300,000 users downloaded the app. While the app obtained the consent of its users to obtain some data, the app collected far more personal data than it had obtained consent for.

The researcher who developed the app also did consulting work for Cambridge Analytica, a company that specialized in consulting for election campaigns. The researcher sold the data to Cambridge Analytica for about 200 million dollars. While there were only 300,000 consenting users of the app, it had collected personal data belonging to others they were socially connected to on Facebook. Ultimately, data belonging to 87 million users of the Facebook social media platform was obtained without their consent or any notification of how their data was being used. The data was subsequently used for targeted ads.

The contact data was obtained through Facebook's Events API (application programming interface), which allowed developers of Facebook apps to access the guest list for events or gatherings posted on Facebook. To prevent further abuse of this and other APIs, Facebook has imposed strict access control limitations on who can access these APIs.

For Facebook, there have been numerous repercussions of this scandal, with Facebook paying significant fines, plunging prices of their stock shares, and extensive bad press.

Facebook, saying that it had been deceived, banned Cambridge Analytica from its platform. For its part, Cambridge Analytica closed operations in 2018, although related firms are still in operation.

Based on this scandal and numerous other data privacy concerns that appear all too frequently in the news, various governments continue to effect new regulations that hold organizations legally responsible to protect data privacy.

1. What fundamental principle of data privacy was violated in this scenario?

2. What steps might Facebook have taken to prevent this controversy?

3. In this case study, what *data security* vulnerability contributed to a *data privacy* problem?

Open Source Data Sharing and Privacy

The public exposure of datasets on open source data sharing sites leads to concerns for the privacy of the subjects who are described by the data. Detailed information about a large population's health, wealth, lifestyles, behavior patterns, and so forth can be invaluable to researchers, but the individuals represented by that data might not want their private information made public. So researchers must be very careful to ensure that data is anonymized so the individuals who are the target or subject of that data cannot be associated with their private or sensitive information.



Note: When sharing datasets, it is also important for researchers to consider how the data they share might be used by other researchers. They must be careful to ensure that datasets they share do not contain biased or misrepresentative data since others may build on that data and come to false conclusions.

Data Anonymization

To maintain privacy, personally identifiable information (PII) may need to be anonymized before it can be processed and analyzed. This means that the identity associated with personal data has been masked somehow so the data can be processed and analyzed without revealing the person associated with that data.

Guidelines for Data Anonymization

Follow these guidelines when anonymizing data to protect users' data privacy.

Anonymize Personal Data

To anonymize personal data:

- Use one of the following techniques to mask the identifying data:
 - **Replacement**—Substitute any values that could be used to identify the user with different values.
 - **Suppression**—Omit (all or in part) any values that could be used to identify the user.
 - **Generalization**—Substitute specific values that could be used to identify the user with something less specific. For example, generalize the date of birth to the year or decade in which the user was born.
 - **Perturbation**—Make random changes to the data to corrupt values that could be used to identify the user.
- Anonymize non-sensitive data as well, if it could be used for the reverse anonymization of sensitive data.
- Make sure that the masking process is not reversible.
- Make sure that the same masking process will produce the same results each time.
- Make sure that data types remain compliant with the schema.
- Preserve the meaning of the data.

The Big Data Challenge

Strategies such as data anonymization may become more difficult to employ in scenarios involving big data, AI, and ML. The more data there is about a subject, the more clues there are as to the subject's identity. With the vast amounts of data available in a big data scenario, it becomes much harder to effectively anonymize the subjects of that data.

The challenge is not simply that more data means there is more data to anonymize, but also that larger volumes of data tend to provide more clues. Given enough clues, the prodigious capabilities of AI and ML make it possible to extract meaningful information from otherwise meaningless data. For example, individuals can be reliably identified through deep analysis of facial features and other

biometric data, patterns of movement, habits, and behaviors. Private personal traits and attributes can be extrapolated from a user's data. Analysis of a user's network of personal relationships may give clues regarding their race, religion, income, political affiliation, and so forth.

Even relatively simple data mining techniques such as ***inference attacks*** enable an attacker to illegitimately gain knowledge about a subject or database by inferring a data subject's identity and other data values from clues provided in the recordset—for example, in the way data has been collected, ordered, or sequenced. If data for a village's residents has been organized by name or street address, then even if the residents' name, house number, and street have been removed, it may be possible to reidentify the records because of the order the records are in.

Over time, the ability of AI and ML to perform such deep analysis will only improve, and privacy will become harder to ensure.

Guidelines for Protecting Data Privacy

When you collect data from users for AI/ML projects and applications, you are responsible to protect data privacy

Protect Privacy

To ensure that personal information and privacy are protected:

- Minimize data collection.
- Consult with data scientists and legal and compliance teams to determine risk of data collection and storage.
- Provide end users the option to specify what data will be collected.
- Use encryption to protect all collected personal data at rest and in transit.
- Ensure that collected personal information is accessible only by authorized users.
- Ensure that a data retention policy is in place.
- Anonymize collected data.

Prevent Operator-Sided Data Leakage

The system ***operator*** responsible for managing the software or platform you host your machine learning systems on (e.g., cloud services provider) may expose data. It might be an intentional malicious breach or an unintentional mistake. Defects in the software may enable such leakage, such as weak or poorly implemented access management controls, encryption, and so forth.

- Implement effective and secure access management controls.
- Encrypt data at rest and in transit.
- Implement effective and secure appropriate identity and access management.
- Provide awareness training for all employees on the handling of personal data.
- Implement a data classification and information handling policy.
- Implement tools like data leakage prevention and SIEMs to monitor and detect classified data leaked from endpoints, web portals, and cloud services.
- Implement privacy by design.
- Anonymize personal data.

Respond Appropriately to Data Breaches

When you collect private data from users, you have a responsibility to protect it. When there is a security breach that compromises the privacy of user data, you must quickly follow up with an appropriate response. You must limit the extent of the leak, inform those who are affected, and remedy any defects or problems that made the breach possible.

- Provide continuous monitoring and logging features to monitor for situations that might indicate personal data leakage and loss.
- Provide features to warn users of possible suspicious activity in their accounts.
- Create, maintain, and periodically test an incident response plan.

- Continuously monitor for personal data leakage and loss.
- When a breach occurs:
 - Validate that the breach occurred.
 - Determine the most effective way to prevent further leakage and implement it.
 - Assign an incident manager to be responsible for the investigation.
 - Decide how to investigate and respond to the data breach to ensure that evidence is appropriately handled.
 - Assemble an incident response team.
 - Notify affected people as appropriate.
 - Determine whether to notify the authorities as appropriate.
 - Remedy any defects or problems that made the breach possible.

Do Not Duplicate or Distribute

ACTIVITY 11–2

Protecting Data Privacy

Scenario

You are continuing your consulting work with the CapitalR Real Estate company. They provide services to member real estate companies in exchange for an annual membership fee. One of the services they provide is the real estate website.

This site enables customers to directly search for a house among the many houses listed by agents in the affiliated companies. They can search based on various criteria such as number of bedrooms, lot size, and so forth, and they can view information about the houses they find, including photos, a map of the house's location, property taxes, and utility costs. If customers are interested in a house, they can click a link to send a message directly to the listing agent to set up a viewing.

Customers can use these features without setting up an account. However, if they set up a free user account, they gain access to additional tools. For example, they can collect a list of "favorite" houses, organize them in order by those they are most interested in, pre-qualify for a mortgage to determine how much they can afford to spend, and so on.

The website is a valuable tool for the various affiliate real estate companies that participate in the service. Not only does the site provide a service to customers, but it also channels customers who may not already have selected a buyer's agent to agents who have listed the houses, potentially driving more business to the agents. By empowering customers to perform searches directly, it enables agents to be more productive, as many customers come to them with a list of homes in hand.

Furthermore, the site is valuable to the real estate board itself. The board mines data on the site based on the searches that customers perform, information customers provide on the site, sales records, and other data. Information gained from this data mining is sold back to affiliated real estate companies in the form of reports, which are provided to members as a benefit of their annual fees.

- 1. When should the site prompt users to obtain their consent to collect their data and inform them how their data will be used—when they create an account, or as soon as they start using the site?**

- 2. What should happen to the private data in CapitalR Real Estate Company's system if CapitalR goes out of business or is sold to another company?**

TOPIC B

Promote Ethical Practices

A variety of ethical concerns are presented with the use of AI and machine learning.

Preconceived Notions

People construct meaning based on what we observe about the present, building on our experiences from the past. Through our experiences, we naturally bring preconceived notions to any problem set. This building on experience helps us to quickly construct meaning of new things we encounter and to find quick solutions to new problems. While building on experience is valuable, it can present a problem when we treat preconceived notions as experience and allow them to lead us to see what we expect to see or want to see. This phenomenon is called *observer bias*.

There are numerous examples in which alleged observer bias resulted in decisions that affected people's lives significantly. For example, the work of Cyril Burt, whose research on intelligence and genetics was largely discredited due to observer bias at best (or falsification of data at worst), influenced the creation of a two-tier educational system in schools based on social class.



Note: For more information, see <https://www.intelltheory.com/burtaffair.shtml>.

When any researcher approaches a problem set, they bring prior knowledge, and may have preconceived notions or subjective feelings about the group they are studying. It is essential that you evaluate your conscious or unconscious prejudices, and ensure they do not influence the training or outcomes of the project.

The Black Box Challenge

To some extent, machine learning models function like a *black box*. Rather than simply following instructions provided by a programmer, machine learning models make predictions and recommendations based on patterns detected in training datasets. The person who creates the model provides training data, makes various decisions such as which algorithms to use, and may supervise aspects of the learning process. The model improves itself with practice, and as it learns, to some extent it may evolve past the understanding of the person who created it. You provide the model with inputs and you get outputs, but what goes on inside the model may not be completely clear. Particularly in the case of deep learning and automated feature extraction, it can be very difficult to determine how the machine actually accomplishes its evaluation or the extent to which it uses various data points, since these things were not directly programmed.

Machine learning models may be used for the purpose of *profiling* people or performing other tasks that may have a direct impact on people's lives, such as analyzing or predicting aspects of someone's work performance, reliability, behavior, personal preferences, interests, health, economic situation, location, or movements. Traditional approaches to computer programming might accomplish such a task through well-defined rules that have been directly coded by a programmer. Since these rules were directly defined by the programmer, they are relatively easy to list, describe, and explain, making it possible to provide a reasonable level of transparency. Machine learning models, on the other hand, function more like a black box, making it harder to pinpoint the actual rules the model has developed to perform the task. This can make it very difficult to meet the transparency requirements put forth by regulations such as those in GDPR.

AI and ML have been very successful, as attested by the increasing use of these technologies to solve business problems, but their use can present a challenge when dealing with regulations such as those put forth in GDPR, which require that systems that process personal data be demonstrably

fair and transparent. Regulations require clarity in regard to the data that will be used, and obtaining the consent of data subjects for specific uses of that data.

Prejudice Bias

Reducing different types of bias (such as selection bias, reporting bias, and attrition bias) is necessary to create models that generalize well and are not sensitive to noise in the training dataset. However, there is another aspect of bias to consider in regard to ethics and fairness.

Prejudice bias is introduced when training data is influenced by cultural or other stereotypes. For instance, suppose you provide the model with a set of images intended to train the model about patterns of work. The image dataset includes thousand of images showing men and women in various job roles. If the images reflect gender stereotypes (men performing work of one type and women performing work of another type), the model is at risk of learning gender stereotypes.

Prejudice bias may become a significant problem when the model takes the biased data into account when making critical decisions that affect people's lives, such as screening job applicants or loan applications, detecting fraud or suspicious activity, making medical diagnoses, and so forth. In these cases, bias may not only lead to an unskillful model, but if the model is skewed based on systematic prejudice, it can also cause harm to a particular class of people.

As a practitioner of AI and ML, it is important to recognize that your selection, labeling, and annotation of training data may consciously or unconsciously convey social stereotypes. When assembling a training dataset, you should take time to ensure that the data presents an even-handed distribution of various classes of people, so the focus will be on other, more relevant attributes.

Proxies for Larger Social Discriminations

Using certain attributes when training a model, such as those that represent protected classes of people, could result in prejudice bias. For example:

- Race
- Gender
- Color
- Religion
- National origin
- Marital status
- Sexual orientation
- Education background
- Source of income
- Age

It is important to bear in mind that attributes other than those listed above may function as a proxy for those attributes. For example, certain geographic locations (represented by ZIP/postal codes, for example) may serve as a proxy for race, color, religion, national origin, or even age—if members of those categories are predominant in those locations.

Common sense can be helpful when deciding what attributes are fair and appropriate to use when training a model. For example, it seems fair to consider income as an attribute when training a model to evaluate a loan applicant's ability to repay a loan.

However, it may be unfair to use ZIP/postal codes, even though ZIP codes may correlate somewhat to income levels. An applicant who happens to live in a ZIP/postal code characterized by low income levels may well have the means to repay the loan and a personal history of repaying loans. Training the model to discriminate based on ZIP/postal code may not produce as skillful a model as one trained more directly on income level. Since ZIP/postal codes may in fact serve as a proxy for other attributes, such as race, color, religion, national origin, or age, discriminating based on ZIP/postal codes may effectively discriminate against those categories of people.

Ethics in NLP

There are numerous ethical concerns related to the use of natural language processing (NLP). For example, you should make sure that NLP doesn't inadvertently exclude users—for example, those who speak a different language than your application supports, or users who are unable to hear or speak.

In some cases, developers might incorporate features intended to improve an application's ability to communicate, while inadvertently violating a user's privacy. For example, developers of voice-activated home automation devices might occasionally "listen in" on conversations between users and the application to assess how well the device is responding to voice commands. Or developers might peruse messages logged in application logs. Both of these approaches may violate the user's privacy.

NLP technologies such as chat-bots have been used to influence online dialogues in forums such as Twitter. In addition to spreading misinformation, bots can manipulate the direction of discussions or can create noise and distractions that make legitimate discussions difficult. It may be necessary to create laws that require chat-bots to identify themselves as automated entities, or ban them outright, and put into place effective legal repercussions for those who violate the law. Such laws would likely have to require that online services be able to demonstrate due diligence in policing their services for the presence of bots. Although AI and ML can be used to create such bots, they may also be useful in helping online services to monitor their systems for the presence of bots.

Use of Data for Unintended Purposes

Given enough data over time, machine learning can glean significant information from seemingly insignificant data. For example, researchers demonstrated that data collected by smart electric meters, which monitor electricity usage in homes, could reveal which programs customers watch.

Every two seconds, the smart meters they tested took detailed measures of energy consumption used by appliances, transmitting those measurements to a server at the utility company. The data collected by the meters included the amount of electricity used, the type of appliance used, and in the case of digital appliances such as some televisions, even more detailed data that the appliance may reveal to the meter.

This data enables the power company to provide customers with very detailed information about their power usage, to help customers monitor their consumption of power. However, in the wrong hands, the researchers demonstrated that the detailed information, collected every two seconds, could be used for purposes the power company never intended. They determined that the data could actually be used to determine which channels and programs the customers were watching at various times of the day. With enough data samples, the patterns of power consumption necessary to display a particular show as the image and lighting changes on-screen would essentially serve as a signature of a particular channel at a particular time.

There are many other examples of how collected data can be used for unintended purposes, if you have the means to interpret the data. As more people gain access to machine learning technologies that can be used for these questionable purposes, it becomes even more important for organizations collecting data to ensure they've taken every precaution to protect that data.



Note: For more information on the example described here, see <https://www.cnet.com/news/researchers-find-smart-meters-could-reveal-favorite-tv-shows/>.

Guidelines for Promoting Ethical Practices

Follow these guidelines to promote ethical practices within AI and ML projects.

Identify Business Scenarios Adversely Affected by Bias Propagation

Examine your own business scenarios and consider ways in which intentional or unintentional discrimination might unfairly select a particular class of people and make unfair decisions based on prejudice bias. Following are some examples.

- **Application Processing**—Model unfairly rejects an application for a job, loan, educational program, medical benefits, and so forth due to prejudice bias.
- **Crime (Fraud, Terrorism Detection)**—Model incorrectly classifies the subject as a potential offender, possibly leading to pain and suffering, detention/loss of time, legal expenses, embarrassment, and possible loss of reputation.
- **Government Benefits**—Model incorrectly classifies potential recipients of government benefits, resulting in eligible people not receiving benefits and ineligible people receiving benefits.
- **Housing**—Model makes an unfair assessment of a property based on prejudice bias (related to location, community, geography, and so forth), resulting in unfair high tax assessments or unfair low sale prices.
- **Insurance**—Model assesses unfair, high premiums for a customer based on prejudice bias such as race or gender.

ACTIVITY 11–3

Promoting Ethical Practices

Scenario

Develetech Industries receives thousands of applications for job openings they post. To enable human screeners to spend more time focusing on the most viable candidates, they have hired you to help them develop a machine learning model to screen résumés for job candidates. They feel they have generally hired well in the past, so they have provided you with résumés of past "good hires" to use for training the model.

1. Is there any danger in using the historical hiring data to train a résumé-screening model?

2. How can you evaluate whether the machine learning models you develop are free of prejudice bias?

TOPIC C

Establish Data Privacy and Ethics Policies

When you work with data, AI, and machine learning capabilities, you and the organization you work for may be accountable to comply with laws regarding data privacy and ethical practices.

Privacy and Data Governance for AI and ML

When it comes to data privacy, different people within an organization have different expertise and motivations. Some (the organization's legal team, for example) may have a clear understanding of the applicable laws governing data privacy. Others (such as the IT team) may know how to implement technical protections for the data. And yet others (such as the intelligence or data strategies team) may be more focused on how data must be used to accomplish the organization's business objectives. Cooperation among these different functions is essential to find an acceptable balance between data use and data protection.

The various functions of the organization should work together to identify applicable laws and standards and to translate them into policies, practices, procedures, and tools that will ensure the entire organization is handling data privacy in a unified manner.

Intellectual Property

Intellectual property (IP) rights are an issue of creativity. When an individual produces a creative work, that individual or the organization the individual is working for generally owns the rights to the created work. Legal protections for such creative works may take the form of copyrights, patents, and so forth. These laws were created based on the premise that the entities actually performing the creative work are humans.

The capabilities of AI have progressed to the extent that now autonomous machines may perform creative work. Of course, this complicates the business of protecting IP since it creates some ambiguity around who owns the intellectual property rights. There may be room to debate whether a machine can have property rights. It may be argued that the creative work is essentially "made for hire" by the machine, so the person or organization who owns the machine would then own the rights to creative works.

The concept of creative works made by machines introduces a variety of ethical and legal concerns for organizations. Some examples are described here.

Issue	Description
Duration of IP protections	Some laws protecting IP rights are in effect for a certain period, based on the lifetime of the author. For example, in U.S. copyright law, works created after 1997 are protected for the entire life of the author plus an additional 70 years. If the author of a protected work is an immortal machine, then it may not be clear how long the copyright protection lasts.
IP rights violated by AI	If an AI system produces creative works that violate someone else's IP rights, it may not be clear who is liable for infringement.

Issue	Description
Protecting the AI or ML itself	<p>Organizations might invest significant creative effort in the development of such systems and may develop innovative approaches to algorithms, training, tuning, architecture, and so forth—which an organization may want to protect as a trade secret, or through a patent.</p> <p>Protecting the approach as a trade secret (keeping its inner workings hidden and protected) may violate principles of data privacy, since it conflicts with the idea of transparency. On the other hand, there may be risk in filing for a patent. While a patent application is transparent and public (exposing the approaches' inner workings and attempting to protect it through legal action), the application may be rejected, leaving the approach both exposed and unprotected.</p>
	<p>Note: How governments will determine these issues may vary. Some may focus on IP rights as a matter of human rights, determining that machines can't have rights. Any rights or responsibilities might flow back to those who own or create the system. Others may focus instead on the utility of IP protections to produce economic incentives for organizations to invest in innovation. Just as laws governing human-created IP vary from one government to another, it is likely that any laws applying to creative works made by machines will vary from one government to another. It will be important for organizations to understand the IP laws that apply to them as creative works are increasingly produced by machines.</p>

Humanitarian Principles

AI technologies will increase the power and influence of those nations who have access to them. A race among world powers to develop new capabilities is underway. While science fiction movies demonstrate the potential for AI to be used in ways that create more disparity, increase war-making capabilities, and subjugate people, there is also much potential to benefit humanity.

There have already been many examples of international cooperation, sharing, and moves toward governance to promote the common good, and advances in AI are being applied to improve healthcare, agriculture, and other fields that benefit humanity. It will be increasingly important for humanitarian principles to guide the development of AI that is safe and beneficial, while reducing the associated risks to humanity.

Following are some of the concerns that have been raised in regard to AI.

Concerns	Description
Transparency	Because of the "black box" nature of many AI algorithms, it can be difficult to hold developers of an AI system accountable. It is not always possible to identify precisely how an algorithm produced its result. As decisions are increasingly delegated to AI systems, there will be some debate regarding who can be held responsible for any harm the systems may cause.
Privacy	In addition to data privacy, AI can present challenges to freedom and civil liberties. AI is already being used to provide real-time monitoring and analysis of video and other data streams. Individuals can be located using their smartphone and other connected devices, and located visually using facial recognition features.

Concerns	Description
Job Elimination	<p>Industrial robotics and workforce automation have already resulted in job elimination in many industries. While it is believed that the disruption caused by this particular technological revolution will even out over time as workers retrain and find other types of jobs, many predict that the disruption will continue as new ways are found to replace human workers with robots.</p> <p>To ease social and political tensions and reduce disruption to people's lives, some have proposed comprehensive retraining programs and economic policies such as a robot tax and universal basic income.</p>
Disinformation	<p>AI can be used to increase the effectiveness of disinformation. Bots can inject misleading traffic and communications into social media sites. This can undermine democracy, incite terror and social unrest, and diminish the benefits of a free press.</p>
Cybersecurity	<p>AI provides new opportunities for cyberattackers. Not only do new AI implementations add to the potential attack surface that an attacker can target, but it can also function as a tool to help carry out their attacks. AI can be leveraged to automate attacks, perform them faster, with greater precision, and on a greater scale.</p>
Disparity	<p>AI technologies may be used to augment people's natural abilities, giving them super-human capabilities through nanobots, implants, and so forth. These types of applications may bring safety and ethical challenges, and also have the potential to dramatically increase disparity between those who have access to the technology and those who do not.</p>
Weaponization	<p>Remote controlled robotic weapons and drones are already in significant use. It may soon be possible to develop lethal autonomous weapon systems (LAWS), which can function without human control. It is argued that LAWS would violate international humanitarian law since no human would be involved in making life-and-death decisions.</p> <p>There is an ongoing international debate regarding how much autonomy would be acceptable in weapon systems. Numerous organizations and technology leaders are promoting national and international bans on LAWS and have promised they will not participate in nor support their development and use.</p> <p>An AI arms race could lower an aggressor's perceived self-risks, making wars more likely, and magnifying the scope of war, similar to nuclear weapons. However, barriers to the development of weaponized AI would be much less than for nuclear weapons, providing potential for increased use by non-state actors and significant disruption.</p>
Unintended Consequences	<p>AI technologies are under rapid development, and in some cases, AI is even being used to help develop new AI. Any technology under such rapid development is subject to "growing pains." New AI systems must be carefully tested for safety and unintended consequences.</p> <p>Many have raised the concern that artificial intelligence can eventually surpass human intelligence, making it hard to contain and control, eventually leading to the "rise of the robots" scenario played out in many science fiction movies.</p>

Guidelines for Establishing Policies Covering Data Privacy and Ethics

Follow these guidelines when establishing policies covering data privacy and ethics.

Asilomar AI Principles

The Asilomar AI Principles were identified at the Asilomar Conference on Beneficial AI, a conference hosted by the Future of Life Institute in Pacific Grove California in 2017. The principles provide guidelines for the research and development of AI. Thousands of AI researchers and organizations have indicated they would support and adhere to these principles. For more information, see <https://futureoflife.org/ai-principles/>.

- **Research**

- The goal of AI research should be to create not undirected intelligence, but beneficial intelligence.
- Investments in AI should be accompanied by funding for research on ensuring its beneficial use.
- There should be constructive and healthy exchange between AI researchers and policy-makers.
- A culture of cooperation, trust, and transparency should be fostered among researchers and developers of AI.
- Teams developing AI systems should actively cooperate to avoid corner-cutting on safety standards.

- **Ethics and Values**

- AI systems should be safe and secure throughout their operational lifetime, and verifiably so where applicable and feasible.
- If an AI system causes harm, it should be possible to ascertain why.
- Any involvement by an autonomous system in judicial decision-making should provide a satisfactory explanation auditible by a competent human authority.
- Designers and builders of advanced AI systems are stakeholders in the moral implications of their use, misuse, and actions, with a responsibility and opportunity to shape those implications.
- Highly autonomous AI systems should be designed so that their goals and behaviors can be assured to align with human values throughout their operation.
- AI systems should be designed and operated so as to be compatible with ideals of human dignity, rights, freedoms, and cultural diversity.
- People should have the right to access, manage, and control the data they generate, given AI systems' power to analyze and utilize that data.
- The application of AI to personal data must not unreasonably curtail people's real or perceived liberty.
- AI technologies should benefit and empower as many people as possible.
- The economic prosperity created by AI should be shared broadly, to benefit all of humanity.
- Humans should choose how and whether to delegate decisions to AI systems, to accomplish human-chosen objectives.
- The power conferred by control of highly advanced AI systems should respect and improve, rather than subvert, the social and civic processes on which the health of society depends.
- An arms race in lethal autonomous weapons should be avoided.

- **Longer Term Issues**

- Unless there is consensus, avoid strong assumptions regarding upper limits on future AI capabilities.
- Advanced AI could represent a profound change in the history of life on Earth, and should be planned for and managed with commensurate care and resources.

- Risks posed by AI systems, especially catastrophic or existential risks, must be subject to planning and mitigation efforts commensurate with their expected impact.
- AI systems designed to recursively self-improve or self-replicate in a manner that could lead to rapidly increasing quality or quantity must be subject to strict safety and control measures.
- Superintelligence should only be developed in the service of widely shared ethical ideals, and for the benefit of all humanity rather than one state or organization.

Make Sure Privacy Policies, Terms, and Conditions Are Clear

You may collect data for machine learning through web, mobile, and desktop applications. When you do so, clearly inform users what you will do with their data. For example:

- Inform users (through a clear and well-written terms and conditions page, for example) how their data is processed, including collection, storage, processing, and deletion.
- If the application or terms of data use change, provide release notes or notifications to clearly and simply explain what has changed.
- Track which users have consented to the terms and conditions, including the version if terms and conditions have changed over time.
- Implement a Do Not Track feature on the server side, so users can disable tracking, and provide an opt-out capability for users.
- Provide users with a list of all tracking mechanisms used in the software, explaining how and by whom the information is used.

Do Not Collect Non-Essential Data

When you collect data that is not needed to meet requirements, you needlessly put privacy at risk. To avoid this:

- Do not collect descriptive, demographic, or any other user-related data that are not needed for the purposes of the system.
- Enable users to opt out of providing additional data to improve the service.

Delete Private or Sensitive Data that Is No Longer Needed

When you retain data that is no longer needed to meet requirements, you needlessly put privacy at risk. To avoid this:

- Minimize data you collect in the first place.
- Promptly delete data that is no longer needed.
- Properly delete data when a user issues a rightful request.
- Securely lock the data from any access until deletion is possible, if prompt deletion is not possible due to technical restrictions.
- Ensure prompt deletion of data in backups, copies, cloud storage, or data shared with third-party sources.
- Clearly inform users when backups must be kept, as required by law.
- Provide evidence (such as logging and messaging to the user) to verify deletion according to policy.
- Identify deletion policies (circumstances under which data must be deleted, and the timeframe for deletion), and implement automation and/or manual procedures to ensure that happens.

ACTIVITY 11–4

Establishing Policies Covering Data Privacy and Ethics

Scenario

Recently, you worked with a major online clothing retailer to enhance their online Search features, which customers use to find articles of clothing they want to buy. Since you played a key role in designing how data is used on the site, you are now helping to draft the data privacy and ethics policies for the site. This includes drafting a statement that will be provided to end users stating the company's privacy and ethics policies. New users will see this statement and will need to confirm that they have read it when they set up a user account. Current users will also have to confirm it to continue using the site.

1. What sorts of information should be shared with customers in this notice?

2. Other than ensuring the company complies with regulations, are there other benefits to the company establishing clear policies regarding data privacy and ethics?

Summary

In this lesson, you learned about your responsibilities to protect the privacy of users and organizations that you serve, including legal obligations and ethical concerns.

In your machine learning tasks, what sorts of private data will you work with?

What will guide your organization's data privacy and ethics policies?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Do Not Duplicate Or Distribute

Course Follow-Up

Congratulations! You have completed the *Certified Artificial Intelligence (AI) Practitioner (Exam AIP-110)* course. You have gained the practical skills and information you will need to plan machine learning projects, develop problem-solving models, evaluate and improve those models, and present your findings to a business audience. All of these skills combined will help you bring value to the organization by efficiently and intelligently solving key business problems.

You've also gained the knowledge you will need to prepare for the Certified Artificial Intelligence (AI) Practitioner (Exam AIP-110) certification examination. If you combine this class experience with review, private study, and hands-on experience, you will be well prepared to demonstrate your artificial intelligence and machine learning expertise both through professional certification and with solid technical competence on the job.

What's Next?

Your next step after completing this course will probably be to prepare for and obtain your Certified Artificial Intelligence (AI) Practitioner certification. If you want to learn more about the ethical and security implications of AI and other emerging technologies, consider taking the related CertNexus® course that discusses these topics. Since presentation of a machine learning project is an important step in the overall workflow, consider taking the Logical Operations course *Effective Presentations (Second Edition)* if you want to further hone that skillset. Also, you may want to pursue the CertNexus *Certified Internet of Things (IoT) Practitioner (Exam ITP-110)* course if you're interested in the application of data science principles to the world of IoT.

You are encouraged to explore AI further by actively participating in any of the social media forums set up by your instructor or training administrator through the **Social Media** tile on the CHOICE Course screen.

Do Not Duplicate Or Distribute

A | Mapping Course Content to CertNexus® Certified Artificial Intelligence (AI) Practitioner (Exam AIP-100)

Obtaining the Certified Artificial Intelligence (AI) Practitioner certification requires candidates to pass CertNexus® Certified Artificial Intelligence (AI) Practitioner Exam AIP-110.

To assist you in your preparation for the exam, CertNexus has provided a reference document that indicates where the exam objectives are covered in the CertNexus *Certified Artificial Intelligence (AI) Practitioner (Exam AIP-100)* courseware.

The exam-mapping document is available from the **Course** page on CHOICE. Log on to your CHOICE account, select the tile for this course, select the **Files** tile, and download and unzip the course files. The mapping reference will be in a subfolder named **Mappings**.

Best of luck in your exam preparation!

Do Not Duplicate Or Distribute

Solutions

ACTIVITY 1–1: Identifying Appropriate Business Applications for AI and ML

1. What might lead her to such a conclusion from looking at the data?

A: There could be any number of problems, but her concerns may have to do with how well the sample data will represent the business problem you're trying to model. Because machine learning is based on probability, it generally functions better with larger datasets. With very small datasets, a single error or "oddball" (outlier) data value will have greater influence over the results produced by the machine learning solution—to the extent that the results are unreliable.

2. Why is this situation acceptable for a machine learning project?

A: Again, because machine learning is based on probability, the huge amount of data makes the problems in the data less statistically significant. With an adequately sized dataset, there are ways to acceptably deal with missing data.

3. What types of improvements might be added to the Search feature through machine learning technology?

A: Machine learning introduces many new possibilities, so answers to this question may vary widely. Machine learning enables the search capability to improve and adapt over time as customers use the site. For example, search terms entered by customers can be analyzed and related to the products that customers actually "click through," view, and buy. And the system can learn new keywords based on evolving fashion trends as customers start submitting them in their searches. The system can learn the preferences and communication patterns of an individual customer, and can tailor search results accordingly. Over time, learning algorithms can learn to interpret different search words differently based on their context, such as other words they are used in combination with. By involving machine learning algorithms, programmers don't have to write code to retrain the system on new search patterns; the system essentially retrains itself as customers use it.

ACTIVITY 1–2: Planning the Machine Learning Workflow

1. How might you translate a statement like "We need to reduce our cost to manufacture Product X" into a problem that you can solve through AI/ML?

A: As an AI practitioner, you will develop the knowledge and experience needed to determine what kinds of problems can be solved through AI/ML. But you may not always have knowledge of the business processes where AI/ML are going to be applied. Start by enlisting the help of those most familiar with the relevant business processes. As business process owners share what they know about the process, you will recognize situations in which AI/ML can play a role. For example, production costs are increased by waste such as products that are too defective to be sold, or production equipment that must be shut down for repairs when it should be busily manufacturing products. As production machinery is used, parts wear out, eventually to the point where the products produced by the machinery are "out of spec." When this happens, the defective parts may need to be scrapped, and the line must be shut down for unscheduled maintenance. Production workers may remain idle until the machinery can be repaired. Problems like these increase the cost of manufacturing and may be addressed through an AI/ML solution. For example, sensors can be added to the assembly line to measure how closely products are adhering to specifications. As they start approaching the point where they will be "out of spec," AI/ML could be used to predict the most convenient time when maintenance can be scheduled (when the assembly line is not operating), and a batch of defective products can be avoided.

2. How should you proceed?

A: Machine learning depends on data. Poor performance in an ML project is often caused by a lack of data. In some cases, you may be able to cope with missing or spotty data. For example, you may be able to augment data or fabricate values to replace missing data if there are a very small percentage of them, but you need to have a reasonable amount of good data to begin with. This may compromise the quality of your results, leading to a less accurate prediction, for example. But to some extent, you may find that you can allow some compromise on the quality of the output if spotty data is a problem. In some cases, you might be able to use a simpler approach that is not quite as accurate as a more complex approach that requires more data. Also, there are ways to employ multiple machine learning algorithms ("ensemble method") to find one that best compensates small amounts of data or spotty datasets. Another option is to come up with ways to obtain the data that you're missing. For example, if you need to solve a generic problem (not specific to your own organization's data), you might be able to use someone else's data (such as a public, open source dataset). You may be able to form a partnership with another organization that has relevant data you can use. Finally, the machine learning workflow is iterative. In some cases, the data you have may not enable you to solve the problem exactly as you originally formulated it, but may enable you to solve a similar problem that would produce the same end result for the organization. By reconsidering the way you define the problem, you may be able to come up with a solution that can be supported by the data you have.

ACTIVITY 1–3: Framing a Machine Learning Problem

1. What sort of task should the model perform?

A: The model would need to be able to predict a reasonable price for the home based on various input parameters.

2. What sort of experience (training dataset) would you need to provide so the model could learn how to price a home?

A: The machine learning model would need access to historical sales data, such as the price the home sold for, how long the home was on the market before it sold, and various features of the home, such as its size, number of bedrooms, bathrooms, location, and so forth. Since prices, customer preferences, current market conditions, and other trends change, the solution would need access to updated information over time to avoid model obsolescence.

3. Once you've created a prototype machine learning model, how might you evaluate the model's performance (that is, its ability to identify an optimum sales price)?

A: You might train the model using one set of data that shows how much homes actually sold for. Once the model has been trained, you might use another dataset (different from the set you used to train the model) to then test how closely the model comes to accurately predicting the prices those homes actually sold for.

4. Over time, after the real estate company has started using the tool, how might you evaluate whether the new tool has benefited the business?

A: You could use data analysis tools to examine how closely the actual sale prices matched the listing price. You could perform the same comparison using historic sales data from before the company started using the tool to determine how things have changed since the tool was deployed. You might also examine how long homes remained on the market before they sold. You could also analyze customer feedback data to determine how the perceptions of customers (sellers) have changed since you deployed the tool.

5. Is a machine learning solution appropriate for this problem?

A: Yes. This seems to be a statistical problem, well suited for machine learning. The patterns and trends that determine an appropriate price will change over time. While a programmer might be able to program a traditional software application to provide good pricing estimates, the program will have to be updated over time to deal with shifting trends. A machine learning model can automatically update as historical data is added and new patterns emerge.

ACTIVITY 1–4: Selecting a Machine Learning Outcome

1. What type of outcome would you look for in this example?

A: Classification algorithms can be used to classify whether parts are acceptable or defective.

2. What type of outcome are you looking for in this example?

A: Regression might be used to make a prediction based on a history of the production quality measurements. On the other hand, classification might also be used to identify parts that have almost reached the defect threshold. Both of these approaches might be used in combination to improve the quality of predictions.

ACTIVITY 1–5: Selecting a Machine Learning Toolset

1. Which of these tools would enable you to do this?
 - Matplotlib
 - TensorFlow
 - pandas
 - Seaborn
2. Which software library implements machine learning algorithms?
 - scikit-learn
 - Seaborn
 - Matplotlib
 - SciPy
3. What capabilities does Anaconda provide for a machine learning project?
 - The ability to process natural language
 - The ability to load and process data organized in columns and rows
 - The ability to plot charts
 - All of these capabilities

ACTIVITY 2–3: Exploring the General Structure of the Dataset

2. Which attributes do you think might have an influence on price?

A: Some attributes might seem important from a commonsense perspective—such as the lot size (`sqft_lot`), size of the living space, and whether the property is waterfront or has a view. Others such as location (`zipcode`, `lat`, and `long`) might be significant if they correspond to expensive neighborhoods. Other attributes might have a surprising influence on the price. Performing some statistical analysis will help to reveal which values actually correlate with price.

ACTIVITY 2–5: Analyzing a Dataset Using Visualizations

4. What patterns seem to exist regarding house prices and location?

A: There seems to be a cluster of expensive homes located around the lakes in Seattle and Bellevue (Lake Washington and Lake Sammamish). But there are also some expensive homes away from the lakes. Less expensive homes seem to be located in areas away from Seattle and Bellevue, such as the southern part of the county (Federal Way, Auburn, Covington, Maple Valley, and Enumclaw).

ACTIVITY 3-1: Setting Up a Machine Learning Model

1. Why would you use a linear regression algorithm to produce a real estate price estimator?

A: The purpose of the model is to predict the price at which a house with particular attributes will sell. This is a regression type of outcome. While other algorithms can also be used to produce a regression outcome, linear regression is simple, relatively easy to implement, and can produce a good result in many cases.

ACTIVITY 4-1: Translating Results into Business Actions

1. How can you present your findings to the real estate board so they not only understand how your pricing model is different from what they've done in the past, but so they are also convinced they should use it?

A: Answers may vary, as there are many ways this could be presented. Since the real estate board is likely not a technical audience, or doesn't care that much about the technical detail, you'll want to focus your presentation on ideas that address key business concerns. Incorporating visuals into the presentation is especially effective. For example, you could show a map of King County with several houses and their prices as data points that reinforce the old, location-based way of thinking. Houses in one location sell for more than houses in another location. Then, you could show data points of houses that don't fit this way of thinking—houses that sell for relatively cheap in areas where houses are usually expensive, and vice versa. You could then pivot to showing how the other factors (e.g., size and grade) make the issue more complex than just where the house happens to be located. Ultimately, you could show the success of the new pricing model as compared to the old pricing model by providing a few examples of houses that require a more nuanced approach to calculating value.

2. How might you make it easy for agents to get pricing for a house based on the new model?

A: There are various ways this could be addressed. Some agents may resist change. To ensure that agents embrace the new model, try to make it very easy to use, preferably even easier than the old worksheet. This way, they will view it as something they want to use—not something that makes their work more complicated or tedious. Given the complexity of the model, it would probably make the most sense to deploy it in software. Identify where in the agent's workflow you can position the tool so it doesn't require agents to go out of their way to use it. For example, if they already use online software tools to list houses for sale, consider implementing the model as a pricing calculator feature within the listing tool. If the tool is ready and convenient at the point where agents need to use it, they are more likely to welcome it as something that improves their job, rather than viewing it as an obstacle.

ACTIVITY 4–2: Incorporating a Model into a Long-Term Solution

1. What market factors might affect the real estate pricing model, and how frequently should the model be updated to account for them?

A: A wide array of factors affect real estate prices. For example, the economy, interest rates, tax rates, and other economic factors affect prices overall, as consumers become more or less able to take on a mortgage. Over time, some neighborhoods may become more or less popular, affecting prices. A particular area may be developed, bringing in new stores, restaurants, greenways, attractions, and conveniences that may increase the appeal of adjoining neighborhoods. New patterns of traffic and crime may diminish the appeal of adjoining neighborhoods. The market also experiences seasonal variations. For example, many families opt to move during summer months, when their children are not in school. Home sales tend to decrease during months with many holidays. These sorts of factors affect supply and demand, which in turn affects pricing. Machine learning is well suited for this type of complexity, but it also makes models susceptible to change. Factors such as those listed here can change quickly, and need to be accounted for in the solution's design.

2. What software requirements might you include to ensure the pricing calculator remains accurate, considering the various market factors you identified in the previous question?

A: Various software functions might be added to account for variability in the model. For example, seasonal variations could be factored into the model based on the date the property is listed. Periodically, pricing predictions made by the tool should be compared to prices the houses actually sold for to determine if the model continues to perform well over time.

ACTIVITY 5–3: Building a Regularized Linear Regression Model

12. Are you satisfied with this MSE value? In other words, would you stop there and finalize the model? Why or why not?

A: Answers may vary. Since there is not one truly "correct" MSE value to shoot for, the decision to stop may come down to what's "good enough," or when the results no longer change significantly. Even though the second round of training produced a lower MSE, there may still be plenty of opportunity to continue tuning the hyperparameters to see if you can get an even better result.

ACTIVITY 5–4: Building an Iterative Linear Regression Model

6. Why is it important to scale down features, such as through standardization, when using an iterative cost minimization technique like gradient descent?

A: Scaling down features helps the model converge on the cost minimum faster, saving on training time.

9. Which of the following describes stochastic gradient descent (SGD)?

- The model has a "memory" of previously computed gradients.
- A data example is selected at random and its gradient is computed for every step.
- A group of data examples is selected at random, and the model steps in the direction of the average gradient from all examples in the group.
- All data examples have their gradients computed for every step.

11. Why is this?

A: The nature of a closed-form solution means that it will determine the model parameters that *best* minimize the cost function. Iterative approaches can get close to the best error value, but they can't do better.

ACTIVITY 6-1: Training Binary Classification Models

5. Given what you know about the dataset thus far, what features do you think might influence the survival rates?

A: Answers may vary. A passenger's socioeconomic status (`Pclass` and `Fare`) may correlate with how that passenger's rescue was prioritized compared to others. `Age` might also be a factor, as older passengers may have been slower to flee danger. `SibSp` and `Parch` might also influence survival rates, as passengers who traveled alone may not have received the same amount of help during rescue attempts as those who traveled with loved ones. Given the policy of "women and children first," `Sex` could also influence survival rate.

6. Why is such a correlation not relevant in classification problems like this one?

A: A correlation indicates how values increase or decrease in relation to one another. Since a classification label like `Survived` is categorical and not a continuous numeric variable, a correlation will not reveal useful information.

ACTIVITY 6-4: Evaluating a Classification Model

13. What does each quadrant indicate in terms of predicting survivors of the *Titanic*?

A: The top-left quadrant indicates that there were 132 instances where the model predicted a passenger would perish, and was correct in its prediction. The top-right quadrant indicates that there were 15 instances where the model predicted a passenger would survive, but was incorrect—those passengers died. The bottom-left quadrant indicates that there were 35 instances where the model predicted a passenger would perish, but was incorrect—those passengers survived. The bottom-right quadrant indicates that there were 41 instances where the model predicted a passenger would survive, and was correct in its prediction.

15. In what situation are precision and recall a better measure of a model's skill than accuracy?

A: Accuracy tends to only be useful in datasets where the class label values are balanced. In datasets with a class imbalance, accuracy may end up being high and yet entirely useless. So, precision and recall are a better summary of a model's skill when a class imbalance is present.

16. Is there any one of these measures you'd be more interested in optimizing than the others? Why or why not?

A: There is no "right" answer, necessarily, because it all comes down to the nature of the dataset and the problem you're trying to solve, which requires a somewhat subjective assessment. Because the class imbalance is rather small, you could argue that high accuracy is an acceptable target to shoot for. You could also argue that precision is important if you're trying to avoid false positives; for example, you want to avoid giving a passenger's relatives false hope by telling them the passenger survived when he or she did not. Recall might be more important to you if you're more concerned about minimizing false negatives; for example, you may want to avoid pronouncing someone as having perished when they actually survived, as that would cause problems for someone who has already suffered a great deal. Or, you may have no clear preference between precision and recall; in which case, the F_1 score is a good way to optimize for both. Ultimately, because this is a mostly academic exercise that isn't going to be applied for any serious purposes, it may be best to try and optimize all of these metrics.

18. What are the advantages of a ROC curve over a precision–recall curve, and vice versa? Given your domain knowledge, are you more interested in improving one of these curves over the other? Why or why not?

A: Once again, there is no "right" answer. ROC and its AUC are good for evaluating classifications for all possible thresholds, which can help you optimize multiple types of errors. However, the precision–recall curve tends to be better at minimizing either one of these errors, which is often more useful in cases of class imbalance. For the *Titanic* dataset, the ROC curve may be sufficient, but you could also argue that the point of this model is to predict examples of the minority class (survived), which a precision–recall curve is best at summarizing.

ACTIVITY 6–5: Tuning a Classification Model

3. Compared to the initial model, how has the confusion matrix changed for the optimized model?

A: There are slightly fewer true negatives; slightly more false positives; fewer false negatives; and more true positives.

5. What else might you do to continue improving your classification performance for this dataset?

A: Answers will vary. Tuning is an iterative process, and you'll often not be done after just the first iteration. You could continue to test your model's performance by optimizing for a metric other than F_1 score, such as optimizing for precision, recall, AUC, etc. You might also want to revisit your data preparation tasks to see if you can do more to optimize the data itself before training. In addition, rather than comparing models that use the same algorithm but different hyperparameters, you could try training a model using a different classification algorithm to see if it performs better than logistic regression.

ACTIVITY 7-1: Building a k-Means Clustering Model

7. How did the clusters form with regard to location?

A: Each cluster appears to be its own quadrant on the map, with cluster 0 representing the southeast; cluster 1 representing the southwest; cluster 2 representing the northeast; and cluster 3 representing the northwest.

9. How did the clusters form with regard to price per square foot?

A: Cluster 1 includes the lowest-priced homes; cluster 3 includes the second-lowest-priced homes; cluster 0 includes the second-highest-priced homes; and cluster 2 includes the highest-priced homes. Other than the heat map, this is also exhibited in the fact that the size of each data point corresponds to its total price.

14. Do you think this model is adequate in solving the problem of recommending similar houses to buyers who have expressed interest in a specific house?

Why or why not?

A: Answers will vary. Given its unsupervised nature, it's very difficult to evaluate the performance of a clustering model. Your best weapons are knowing what you want out of the model, and performing clustering analysis. In this case, the number of clusters isn't supported much by domain knowledge; in other words, there's no set number of groups that houses need to fall into. However, you might argue that more clusters will be helpful to the real estate agents, as they will narrow down the houses more. Still, you may have to trust in analysis methods like silhouette analysis and elbow plots. Although the former was chosen to influence the final model, you could choose the latter and it would be no less valid. You might also find value in doing further analysis by plotting the clusters in more than just the latitude and longitude dimensions. Going back to domain knowledge and what you want out of the model, you may determine that some features are more important than others, which might influence your clustering decisions.

15. What are some reasons why this model may need to be retrained over time?

A: Answers may vary. There are several factors that could potentially require a retrain of this clustering model, most of which have to do with the dataset. In particular, this dataset was captured at a certain point in time, and therefore may not reflect future changes. For example, housing trends come and go, and what may have been a desirable trait when this data was collected may not be as desirable in the future. Likewise, changing economic factors can greatly affect the value of a house from year to year, so you'll likely need to update the model to account for this.

ACTIVITY 7-3: Building a Hierarchical Clustering Model

9. If this were a real-world problem rather than an artificial dataset, how might domain knowledge of that dataset help you determine the optimal number of clusters?

A: Answers may vary. Domain knowledge can inform the optimal number of clusters in several ways. The most obvious is if each data example needs to be placed into one of n groups, with n being known by you. Even if you don't have a specific number of groups in mind, the dataset might place certain constraints on the outcome, like bounding the number of clusters within a range. For example, if your objective is to create a recommendation system for customers based on their shared attributes, you might want to have less than five or so clusters, as it can be difficult to manage many different groups of customers. Alternatively, you might want to have a large number of clusters so that your recommendations are more targeted and therefore more valuable to smaller groups of customers.

ACTIVITY 8–1: Building a Decision Tree Model

5. Which of these remaining features do you think need to be one-hot encoded, and why?

A: One obvious candidate is `Sex`, as not only are the values categorical, they are currently in strings. The same goes for `Embarked`—there are only three options, each one a string. In both cases, directly converting the strings to numbers will be problematic, as the decision tree algorithm will think that 1 is ranked higher than 0. This is why one-hot encoding these features is important. Another candidate for one-hot encoding that might not be so obvious is `Pclass`. While the concept of passenger class implies some sort of order, the order is actually deceptive; a class of 1 (first class) is actually ranked higher than a class of 2 (second class). However, the decision tree algorithm will see this as the exact opposite, and class 3 will be ranked above the others. Therefore, it's safest to just one-hot encode each class type.

20. Are you satisfied with the results of the decision tree as compared to the logistic regression model? Do you think one model is more skillful than the other? If so, which one, and why?

A: Answers will vary greatly. Model evaluation and selection does not produce an objectively correct answer; much of it is up to the machine learning practitioner's judgment and expectations. If you happen to value precision more than recall for this particular problem, you might be more willing to depend on the decision tree model, as it produced a higher degree of decision. If you're focused on recall, then you may prefer the logistic regression model. When it comes to accuracy, F_1 score, and even ROC AUC and average precision, the results were not all that different. Likewise, you may be unsatisfied in the sense that you'd like to do more hyperparameter optimization to potentially get even better results from either model. After all, both searches only had narrow fields of hyperparameters to select from. You may also wish to withhold judgment until you've put the data through even more classification algorithms, as the optimal model may still be out there.

ACTIVITY 8–2: Building a Random Forest Model

2. Why is cross-validation typically not necessary when training a random forest model?

A: The bagging technique used in random forests randomly samples the training dataset for each individual tree in the forest; this ensures that the entire forest sees a representative sampling of the data. In other words, bagging helps to reduce overfitting in a way similar to cross-validation, so the latter is not strictly necessary.

11. How do the first two branches of each tree differ in terms of their splitting criteria?

A: The first tree (index 0) starts by splitting male and female passengers at the root decision node. It then splits based on whether or not female passengers are in third class. At the same time, it splits male passengers based on whether or not they have fewer than three siblings plus a spouse. The second tree (index 1) starts by splitting based on whether or not the passengers are in first class. It then splits based on whether passengers not in first class are male or female. At the same time, it splits first class passengers into male or female as well.

13.What advantage does a random forest have over a single decision tree?

A: Decision trees are prone to overfitting to the training data, whereas random forests reduce variance and therefore help mitigate overfitting issues.

20.How might you retrain the model to improve these scores even further?

A: Answers may vary, but the model might be improved by dropping another feature—for example, you could try dropping `Fare` since it had the least importance of the four features that were kept. This might be successful if `Fare` only adds noise to the data and doesn't truly contribute to the model's predictive power. Alternatively, you might add a feature back to the training—for example, `Pclass_1`, since it had the highest importance of the features that were dropped. This might improve the model if including `Pclass_1` contributes to the model's predictive power. Ultimately, you may need to try several different approaches and see which one works the best for your goals.

ACTIVITY 9–1: Building an SVM Model for Classification

5. Why are SVMs often better than other algorithms at handling datasets with outliers?

A: SVMs create margins of separation that help its decision boundary stay as far away from edge cases as possible, which helps reduce the effect of outliers on the model. Other algorithms may fail to handle outliers in this way, potentially overfitting the model.

11.How does this SVM boundary fit to the data as compared to the logistic regression boundary?

A: The SVM boundary seems to stay away from the edge data examples more successfully than the logistic regression model. It also incorporates the support-vector margins, where a few of the outliers are at or near those margins. Also, the outlier mentioned before appears to be classified correctly this time.

ACTIVITY 9–2: Building an SVM Model for Regression

9. How does SVM regression differ from SVM classification in terms of how the data examples are included or not included within the margins?

A: In classification, the ideal is to separate each data example so that none of the examples are placed within the margins. The support vectors are the data examples on the edge of the margins. In regression, the ideal is to include as many data examples as possible *within* the margins. All of the data examples outside of the margins are the support vectors.

ACTIVITY 10–1: Building an MLP

15.How does backpropagation generate the weights between the neurons of different layers in an MLP neural network?

A: A prediction is made for an example, and then the error between the prediction and values is calculated. Starting from the last hidden layer, the network computes how much each neuron in the hidden layer contributed to the error in each output layer neuron. This process is repeated for the next-to-last hidden layer, and so on, until reaching the input layer. The weights that were just returned were updated to account for the error gradients between neurons.

16.What can you tell about the weights of this particular network structure?

A: From the input to the hidden layer, the highest weights appear to be from features 3, 4, and 6 (Light, CO₂, and Day), to both neurons in the hidden layer. Features 3 and 4 have relatively high positive weights with hidden neuron 1, whereas they tend to have relatively high negative weights with hidden neuron 2. The rest of the input features have much less significant weights that vary between positive and negative. From the hidden layer to the output layer, hidden neuron 1 has a high positive weight, whereas hidden neuron 2 has a high negative weight.

ACTIVITY 10-2: Building a CNN

12.What can you tell about these incorrect predictions? From the perspective of your own human judgment, does it make sense that these images might be misclassified in the way that they were?

A: Answers may vary. In most of these incorrect cases, the predicted class of clothing seemed to be visually similar to the actual class. For example, the image in row 2, column 2 was classified as a pullover by the model, but is actually a coat. A pullover and a coat are pretty similar, and by looking at the image itself, you might not even be able to determine the difference. There is at least one instance of an incorrect prediction being pretty far off: the image in row 3, column 3. The model classified this image as a t-shirt, but it is very clearly some kind of shoe (a sneaker, to be precise).

13.What are some ways you might retrain this CNN model to improve its skill?

A: Answers may vary. The model will very likely improve both its loss score and its accuracy by training over multiple epochs, rather than just one. Also, changing the size of the convolutions, as well as the size of their filters, may lead to a more skillful model. You won't always know the exact values to use for these hyperparameters, so it may be worth experimenting with different combinations to see if you can get better results.

ACTIVITY 10-4: Building an RNN

7. What is word embedding, and why might it be beneficial to use in this case?

A: Embedding is the process of representing a word in its own vector of n dimensions. Each vector combines into the network's overall embedded space. Words with similarities are placed closer within this space, improving the model's ability to recognize patterns. Embedding helps minimize the dimensionality of language-based inputs, which would otherwise require many thousands of features for each word in a vocabulary.

8. In an LSTM cell, which of the following activation functions is used by the forget gate (f_t)?

- Hyperbolic tangent (tanh)
- Sigmoid
- Rectified linear unit (ReLU)
- Leaky ReLU

15. Why do you think the model incorrectly classified this review as positive?

A: There is not necessarily a correct answer, as you can't easily interrogate the model to find out the specifics of *why* it made a decision. However, it's possible that the model picked up on some key words and phrases that typically indicate positive sentiment. For example, the review opens by stating, "I generally *love this type of movie*, however ..." The reviewer also praises one aspect of the movie as being "very cool." Later, were it not for a particularly unpleasant character, the reviewer states that he "would have otherwise *enjoyed the flick*." The network may have picked up on these words and phrases without understanding how context may have changed their meaning.

16. What are some ways you might retrain this RNN model to improve its skill?

A: Answers may vary. As with the CNN, the RNN model will very likely improve both its loss score and its accuracy by training over multiple epochs, rather than just one. Expanding the vocabulary above 10,000 words, or removing that constraint altogether, may also provide the model with more useful data. Lastly, reconfiguring the LSTM and dense layers to use different output sizes, as well as adding more layers, may also improve the model's skill. Once again, your best bet is to experiment with these hyperparameter configurations as much as you can.

ACTIVITY 11–1: Complying with Applicable Laws and Standards

1. What fundamental principle of data privacy was violated in this scenario?

A: While numerous issues led to this problem, at the root of the scandal was the fact that user data was shared without the consent of the data subjects (the 87 million "other" users, who did not use the *This is your digital life* app), and no attempt was made to inform them how their data was being used.

2. What steps might Facebook have taken to prevent this controversy?

A: Following Privacy by Design guidelines such as those put forth by GDPR would have prevented this situation. By default, users' data should remain private and they must personally consent for it to be shared.

3. In this case study, what *data security* vulnerability contributed to a *data privacy* problem?

A: The Events API enabled contact info and other user data to be scraped from the Facebook site, even though the users the data belonged to did not consent for that data to be used. Facebook addressed this security problem by tightening their access controls over the Events API and other APIs that were vulnerable.

ACTIVITY 11–2: Protecting Data Privacy

1. When should the site prompt users to obtain their consent to collect their data and inform them how their data will be used—when they create an account, or as soon as they start using the site?

A: This depends to some extent on which data collection starts and when data is associated with a user's identity. Even if data is only being stored in a cookie, it is important to let users know. In a scenario like this, users may give up more information over time, progressing from an anonymous user only identified by a session cookie, to a user with an account holding their name, address, and so forth, to a user who has provided financial data to prequalify for a mortgage. In a scenario like this, you might provide progressive notifications to users, warning them each time they are about to give up more data, revealing how their data will be used, and obtaining additional consent each time.

2. What should happen to the private data in CapitalR Real Estate Company's system if CapitalR goes out of business or is sold to another company?

A: It depends on a variety of factors. Users trust the company with their data and that duty does not go away if the company's assets are transferred to another company. The company may have included a business transfer clause in the consent agreement, which defines how the company will deal with the data in such a case. Typically users must be notified when a transfer of the business is about to take place.

ACTIVITY 11–3: Promoting Ethical Practices

1. Is there any danger in using the historical hiring data to train a résumé-screening model?

A: The machine may learn unintended patterns. AI is not fundamentally biased, but it can learn from patterns of historical bias. For example, if previous candidates in the company were chosen through human bias (for example, favoring male candidates), the algorithm may learn to favor similar candidates, decreasing diversity and potentially missing out on excellent candidates who don't fit the general profile of current employees. There is also potential selection bias in the process of determining which résumés of current employees represent "good hires."

2. How can you evaluate whether the machine learning models you develop are free of prejudice bias?

A: This requires an understanding of machine learning as well as of unethical and prejudicial practices you are trying to prevent. While the inner workings of the model may not be completely transparent, you can focus on the inputs and the outputs. Examine the data you're using to train the model (the inputs), and test the outcomes the model produces (the outputs). For example, racial steering is a practice in which an agent directs buyers towards or away from certain neighborhoods based on their race. To evaluate whether a machine learning model might somehow incorporate racial steering, examine the features in the model to determine if any of them are indicators of race, ethnicity, and so forth—as well as any features that might serve as a proxy for them. Ensure that it is not possible to determine the race or ethnicity of a buyer or the demographic profile of a house's neighborhood. Once you've produced the model, test it to see if it seems to produce biased or prejudiced outcomes.

ACTIVITY 11–4: Establishing Policies Covering Data Privacy and Ethics

1. What sorts of information should be shared with customers in this notice?

A: Information commonly included in such a statement might include: how cookies are used, how non-identifiable data collected on the site will be used and who it will be shared with, how personal data will be used and who it will be shared with, how the user can opt out of data collection, special policies for certain groups, such as children, how long the data will be kept, and how to obtain a copy of your data and how to provide corrections to the company if any of the collected data is incorrect.

2. Other than ensuring the company complies with regulations, are there other benefits to the company establishing clear policies regarding data privacy and ethics?

A: Making your policies known to users and carefully adhering to them helps to establish that the company is trustworthy. Going above and beyond to protect customers' data can provide competitive differentiation. It is easy to lose the trust of customers. It is important to maintain trust by complying with all applicable compliance mandates and ensuring good decisions are made about personal data. Digital privacy and ethics encompass many facets of how enterprises conduct their business, but without the ability to marry policies to personal data—and specific insights into whose data it is—even the best-laid plans can fall short.

Do Not Duplicate Or Distribute

Glossary

A/B test

A type of hypothesis test that compares two different values of the same variable in order to determine which value is most effective.

accuracy

A measure of how frequently each prediction is correctly deemed positive or negative.

activation function

A function that computes the output of an artificial neuron to solve non-linear tasks.

AI

(artificial intelligence) The ability of machines to exhibit human-like intelligence, as well as the scientific discipline concerned with this idea.

algorithm

A set of rules that defines how a problem-solving operation is performed.

ANN

(artificial neural network) A machine approximation of biological neural networks. Used in deep learning.

anomaly

A record that doesn't seem to belong because it doesn't follow patterns established by other records in the dataset.

ANOVA

(analysis of variance) A type of hypothesis test that compares the mean of three or more distributions.

arithmetic mean

The average of all numbers in a set.

attribute

A column of data that will act as input for a machine learning model. It contains the independent variables a prediction model evaluates to make its predictions.

attrition bias

A type of bias that occurs when the training data excludes participants that dropped out over time.

AUC

(area under curve) The total space that is under a learning model's ROC curve.

backpropagation

A method of training a neural network that starts by computing the error gradient of neurons in the last hidden layer, then the next-to-last hidden layer, and so on, until reaching the input layer. The connection weights between neurons is then updated.

bag-of-words

An approach to representing textual content as a list of individual words, irrespective of other language components like grammar and punctuation.

bagging

(bootstrap aggregating) An ensemble learning technique for data sampling with replacement.

Bayesian optimization

A hyperparameter optimization method that determines the next optimal hyperparameter space to sample from by using past samples to influence where sampling is conducted in subsequent iterations.

BCSS

(between-cluster sum of squares) A clustering model evaluation metric that measures the separation between clusters.

Bessel's Correction

A value of 1 subtracted from the number of items in a sample set in order to compensate for the lack of information that would be provided by the larger population.

BGD

(batch gradient descent) An approach to gradient descent that uses the entire training dataset to calculate the step-wise gradients.

bias

A systematic error in data analysis that results in a faulty model.

big data

Collections of data that are so large and complex that they require advanced tools to process and analyze them.

bimodal

See *multimodal*.

black box

A mechanism to which you provide one or more inputs to produce one or more outputs, without knowledge or visibility into its inner workings.

BPTT

(backpropagation through time) A method of training a recurrent neural network (RNN) in which the time sequence of

RNN layers is first unrolled, and then backpropagation is performed.

C4.5

A machine learning decision tree algorithm that uses information gain ratio for data splitting to solve classification or regression problems.

CACE

Stands for "changing anything changes everything." In other words, changes to the feature set, transformations done on data, learning rate and other parameters, and many other factors can affect a machine learning model's performance.

CART

(classification and regression tree) A machine learning decision tree algorithm that uses the Gini index for data splitting to solve classification or regression problems.

central tendency

A measure such as the mean, median, and mode intended to identify the typical value in a dataset.

centroid

In a clustering algorithm, the mean (average) of all of the data points that the cluster contains, across all features.

chi-squared test

A type of hypothesis test that compares the effect of categorical variables.

cluster analysis

A class of techniques that are used to classify objects or cases into relative groups called clusters.

CNN

(convolutional neural network) A type of artificial neural network (ANN) most commonly used to process pixel data. This approach owes its name to convolution, a mathematical operation that enables it to perceive images by assembling small, simple patterns into larger, more complex patterns. This approach was inspired by the

way natural vision is processed in the visual cortex of animals.

coefficient of determination

A statistical measure that indicates how much of a dependent variable's variance is explainable by a statistical model.

collinearity

The property that describes multiple variables as exhibiting a linear relationship.

computer vision

The discipline that involves computers understanding images and video at a high level.

concept drift

A process through which the patterns initially used to train a machine learning model change over time such that the model no longer performs well with new data.

confidence interval

A measurement that returns a range of values for which there is some specified probability that the true value lies within that range.

confusion matrix

A method of visualizing the truth results of a classification problem.

continuous variable

A variable whose values are uncountable and can extend infinitely.

convolutional layer

A type of layer in a convolutional neural network (CNN) in which the neurons scan a portion of the input image for data that is within the neurons' filter. Also called a convolution.

cost function

A function that attempts to quantify the error between the predicted values and the actual labeled training values.

CPU

(central processing unit) The computer chip that functions as the core component in a general purpose computer.

cross-entropy

A cost function used to evaluate the performance of a softmax function by penalizing low probability scores for a particular class.

cross-validation

A set of methods for measuring how well a machine learning model is able to generalize to new test data.

data binning

The process of discretizing a continuous variable by placing its values within specific intervals.

data cleaning

The process of locating and removing errors and inconsistencies in data.

data mining

(also called knowledge discovery) The process of using statistical analysis methods to find meaningful patterns in data.

data munging

See [data wrangling](#).

data science

The discipline that involves accumulating data, analyzing the data, extracting value from the data, and presenting the value of the data in a meaningful way.

data wrangling

The process of transforming data into a usable form.

Davies-Bouldin index

A clustering evaluation metric that calculates the average ratio of the within-cluster distance and the between-cluster distance for each cluster as compared to its most similar cluster.

decision boundary

The division line that separates negative classes and positive classes in a classification problem.

decision tree

An arrangement of conditional statements and their conclusions in a branch-leaf structure.

deep learning

A type of machine learning that makes complex decisions using multiple layers of information.

dendrogram

A diagram that represents a tree-like hierarchy, commonly used to visualize hierarchical clustering tasks.

dependencies

Records that depend on each other for related information.

dependent variable

In an experiment, the variable that is being studied and that is affected by one or more independent variables.

DevOps

An IT approach in which software development practices help automate systems operations.

dimension

The number of attributes used in a machine learning model.

dimensionality reduction

The process of simplifying a dataset by eliminating redundant or irrelevant features.

discrete variable

A variable whose values are countable and limited, because there is a definite gap between each value in a range of values.

discretization

The process of converting a continuous variable into a discrete variable to simplify and improve a decision tree model.

discriminator

One half of a generative adversarial network (GAN) that predicts a label given a set of features. It tries to determine whether the image created by the generator is real or fake.

DOE

(Design of Experiments) See [Experimental Design](#).

Dunn index

A clustering evaluation metric that calculates the ratio of the smallest distance between two data examples in different clusters, and the largest distance between data examples in the same cluster.

elastic net regression

A regularization technique that uses a weighted average of both ridge regression and lasso regression when training a model.

elbow point

In clustering, the point at which the mean distance between each data example and its associated centroid no longer decreases in a significant way.

embedding

In a recurrent neural network (RNN), the process of condensing a language vocabulary into vectors of relatively small dimensions.

ensemble learning

An application of machine learning in which the predictions of multiple models are considered in combination.

entropy

In the context of decision trees, the property that describes how "ordered" a dataset is by multiplying each feature's class probability by a binary logarithm and multiplying that product by -1 .

error

Incorrect or missing values in data that hinder the training of a machine learning model.

ETL

(extract, transform, load) The process of combining data from multiple sources.

example

In the field of machine learning, an individual instance of data in a training set.

Experimental Design

An approach to identifying, analyzing, and controlling variables used in an experiment. Also referred to as Design of Experiments (DOE or DOX).

F₁ score

The weighted average (harmonic mean) of both precision and recall.

feature

In the field of machine learning, a measurable property of an example in a training set.

feature engineering

The process of identifying what features are needed to support a machine learning algorithm, manipulating existing data as needed to create the required features, and improving the way features are encoded as needed to improve the performance of a machine learning model.

feature extraction

A type of dimensionality reduction in which you derive new features from the original features.

feature map

A representation of an image that focuses on whatever feature a convolution filter searches for in a convolutional neural network (CNN).

feature selection

A type of dimensionality reduction in which you select a subset of the original features.

feature transformation

The process of applying functions to data items to convert them to values that are more suitable for processing by machine learning algorithms—for example, scaling values so they range from 0 to 1, and removing skew from the distribution.

filter

The portion of the receptive field that a convolutional layer neuron uses to scan the image at prior layers.

fit

How well a target function has been approximated.

FNN

(feedforward neural network) A type of artificial neural network (ANN) in which information flows to and from artificial neurons in a single direction.

FPR

(false positive rate) A measure of how frequently the learning model incorrectly predicted positive values.

GAN

(generative adversarial network) A neural network architecture that pits two different neural networks against each other, typically for the purpose of generating images.

Gaussian

Having the shape of a normal curve or a normal distribution.

Gaussian RBF kernel

(Gaussian radial basis function kernel) A kernel trick method that projects a new feature space in higher dimensions by measuring the distance between all data examples and data examples that are defined as centers.

generalization

A model's ability to adapt properly to new, previously unseen data.

generator

One half of a generative adversarial network (GAN) that predicts features given a label. It creates an image and tries to "fool" the discriminator into believing that it is real.

genetic algorithm

An approach to optimization that is inspired by the theory of natural selection formulated by Charles Darwin.

Gini index

A decision tree splitting metric that splits trees based on the "purity" of decision nodes by squaring each feature's class probability.

GPU

(graphics processing unit) The computer chip typically used as the core component in a graphics adapter. These chips are optimized for matrix operations, making them well suited for performing many machine learning tasks.

gradient descent

A method of minimizing a cost function in which a model's parameters are tuned over

several iterations by taking gradual "steps" down a slope, toward a minimum error value.

grid search

A hyperparameter optimization method that takes a set (or grid) of parameter combinations, trains a model using each of those combinations, and then returns the combination that best optimizes a specified evaluation metric.

GRU cell

(gated recurrent unit cell) A simplified version of the long short-term memory (LSTM) cell used in recurrent neural networks (RNNs).

HAC

(hierarchical agglomerative clustering) A type of clustering algorithm that initializes each data example in its own cluster, then gradually merges the closest examples and clusters.

hard-margin classification

A type of classification in SVMs where all data examples are outside of the margins, and each example is on the "correct" side of the margins.

HDC

(hierarchical derivative clustering) A type of clustering algorithm that initializes all data examples in a single cluster, then gradually splits the data into more and more clusters.

heat map

A chart or other visualization that provides color coding (different intensity or shades of color) to label values in a particular area of a map, diagram, or grid containing data values. This approach has been commonly used in weather maps to show temperature distributions in various regions (hence the name "heat map"), but the general color-coding approach can be used to show many types of values (not just temperatures).

heteroscedasticity

(also known as non-constant variance) A specific type of pattern in which the amount of variability for one feature is consistently larger than for another, revealing a cone shape when

instances of the values are shown in a scatterplot.

hidden layer

A layer of neurons in a neural network that is not directly exposed to the input and requires additional analysis.

histogram

A frequently used data visualization technique in machine learning that represents the distribution of a continuous variable over a given interval or period of time.

holdout

A cross-validation method in which the dataset is split into two: the training dataset and the test dataset.

hyperparameter

A parameter that is external to a machine learning model; i.e., set on the algorithm itself and not the learning model.

hyperparameter optimization

The process of repeatedly altering the hyperparameters that an algorithm uses to train a model in order to determine the set of hyperparameters that lead to the best or the desired level of model performance.

hyperplane

In SVMs, a decision boundary that has parallel and equidistant lines or curves on either side of the boundary.

hypothesis

A candidate machine learning model that you create to test its performance, particularly whether it is able to produce the outcome that you require. You program the hypothesis to employ a particular function that you hope will produce the intended outcome. Like any hypothesis, you use it as the basis for additional investigation, experimentation, and evaluation.

ID3

A machine learning decision tree algorithm that uses information gain for data splitting to solve classification problems.

identity matrix

A matrix of all zeros except for the main diagonal, which consists of all 1s.

imputation

A process of filling in missing data values that consists of using statistical calculations to determine what the missing values should be, assuming they follow the general patterns established by other records in the dataset. The intention is to provide values that will not end up introducing significant bias.

independent variable

In an experiment, a variable that can have an effect on the dependent variable.

inference attack

A data mining technique that enables an attacker to illegitimately gain knowledge about a subject or database by inferring a data subject's identity and other data values from clues provided in the recordset such as the way data has been collected, ordered, or sequenced.

information gain

A decision tree splitting metric that splits trees by subtracting the entropy of child decision nodes from the entropy of their parent node.

input layer

A layer of neurons in a neural network that deals with information that is directly exposed to the input.

IQR

(interquartile range) When a distribution is divided at the median, with each resulting set being further divided at their medians, the two inner sets are the interquartile range, where the middle half of data values lie.

irreducible error

Errors that cannot be reduced any further when fitting a machine learning model, due to the way the problem was framed, and caused by factors such as unused or unknown features that would have an effect on the output had they been used.

***k*-fold cross-validation**

A cross-validation method in which the dataset is split into k groups (folds). One group is the

test set, and the remaining groups form the training set.

***k*-means clustering**

An algorithm for unsupervised machine learning that groups like data examples together for the purpose of revealing patterns in the data.

***k*-NN**

(k -nearest neighbor) An algorithm commonly used to classify data examples based on their similarities to other data examples within the feature space.

kernel trick

A group of mathematical methods for efficiently representing non-linearly separable data into higher-dimensional space.

kurtosis

A measure of the shape of the tails of a distribution, representing the combined weight of the tails relative to the center of the distribution.

label

In the field of machine learning, the variable in a training set that you are trying to predict for new samples of data.

lasso regression

A regularization technique that uses an ℓ_1 norm to reduce irrelevant features to 0 when training a model.

learning rate

In gradient descent, the hyperparameter that defines the size of each "step" down the slope.

leptokurtic

Used to describe a distribution curve with narrow tails on the right and left sides.

linear kernel

A simple, fast kernel trick method that applies only to data that is linearly separable.

linear regression

A type of regression analysis in which there is a linear relationship between one independent variable and one dependent variable.

logistic function

The value between 0 and 1 that a logistic regression algorithm outputs, taking an *S* shape. Also used as an activation function in artificial neural networks (ANNs).

logistic regression

A type of linear regression in which the output is a prediction between 0 and 1.

LOOCV

(leave-one-out cross-validation) A leave- p -out cross-validation method in which p is set to 1 to minimize performance issues.

LPOCV

(leave- p -out cross-validation) A k -fold cross-validation method in which k (folds) is equal to all data points in the dataset (n), with $n - p$ being the training set and p being the test set.

LSTM cell

(long short-term memory cell) A type of memory cell in a recurrent neural network (RNN) that preserves input that is significant to the training process, while "forgetting" input that is not.

machine learning workflow

The sequence of steps or phases necessary to produce a machine learning model.

MAE

(mean absolute error) A cost function that calculates the average difference between predicted and actual values without considering the sign of those values.

matrix operation

A type of mathematical procedure that produces a matrix (multidimensional array) as a result of performing operations across elements of other matrices.

maximum

Largest value in the dataset.

MBGD

(mini-batch gradient descent) An approach to gradient descent that selects a group of examples at random from the dataset, then uses it to calculate the step-wise gradients.

memory cell

A component of a recurrent neural network (RNN) that maintains a certain state in time.

mesokurtic

A distribution with average or normal shaped tails on the right and left.

minimum

Smallest value in the dataset.

ML

(machine learning) An AI discipline in which a machine is able to gradually improve its predictive and/or decision-making capabilities without being given explicit instructions. Occasionally abbreviated as ML.

MLP

(multi-layer perceptron) A neural network algorithm that has multiple distinct layers of threshold logic units (TLUs).

model

In machine learning, a specific implementation of an algorithm that is used to generate predictions and other decision-making outcomes based on some training data.

model parameter

A parameter that is internal to a machine learning model; i.e., derived from the model as it undergoes the training process.

moment

A set of four statistical parameters commonly used to measure a distribution, including mean, variance, skewness, and kurtosis.

MSE

(mean squared error) A cost function that squares the error between predicted and actual values, then calculates the average of all squares.

multi-class classification

A classification problem in which a data example can be placed into one of three or more classes.

multi-label classification

A classification problem in which a data example can be given multiple labels.

multimodal

A probability distribution with more than one peak, or mode.

multinomial logistic regression

An algorithm commonly used to solve multi-class classification problems.

munging

See [data wrangling](#).

NLP

(natural language processing) The discipline that involves computers analyzing and applying human languages.

noise

Data values, columns, or records that lead a machine learning algorithm to miss important patterns in the data by making it difficult for the algorithm to "hear" patterns revealed by other data that is actually relevant.

normal distribution

A function that represents the distribution of many random variables as a symmetrical bell-shaped graph.

normal equation

A closed-form solution to linear regression problems.

normalization

A technique in which features are adjusted to a common scale so that some features do not exert undue influence over others.

null hypothesis

The assumption that there is no statistically significant difference between two models under comparison.

observer bias

A problem when we treat preconceived notions as experience and allow them to lead us to see what we expect to see or want to see.

one-hot encoding

The process of converting a non-ordinal categorical variable into two or more constituent variables, where, for each example, all of the variables are 0 except for one.

operator

The person or organization responsible for managing the software or platform you host your machine learning systems on (e.g., cloud services provider).

ordinal data

Data that can be placed in an order.

outlier

A value outside the normal distribution, deviating significantly from the rest of the values in the dataset. This may be caused by errors in measurement or execution.

output layer

A layer of neurons in a neural network that formats and outputs data that is relevant to the problem.

overfitting

A problem in machine learning in which a model's predictions/decisions fit well to the training data, but fail to generalize well to other data. An overfit model exhibits high variance and low bias.

p-value

The probability of obtaining a result from the test given that the null hypothesis is true.

padding

The practice of adding pixels around an input image to preserve its dimensions, while enabling a convolutional layer to be the same size as the actual input.

parallelization

Dividing up processing tasks among multiple processors to scale up the performance of a machine learning environment.

PCC

(Pearson correlation coefficient) Also called Pearson's r , the Pearson product-moment correlation coefficient (PPMCC), or the bivariate correlation. A measure of how closely one value (y) changes in relation to changes in another value (x).

perceptron

An algorithm used in ANNs to solve binary classification problems.

PII

(personally identifiable information) Data that must be protected to ensure the privacy of the people described by that data.

pipeline

A sequential set of processes that automate the machine learning process by feeding the output of one process into the input of the next process.

platykurtic

Used to describe a distribution curve with flat tails; the tapered ends on the right and left sides.

polynomial kernel

A kernel trick method that uses polynomial values as part of its feature space projection.

pooling layer

A type of layer in a convolutional neural network (CNN) that applies an aggregation function to input features in order to make a more efficient selection.

PRC

(precision–recall curve) A method of visualizing the tradeoff between precision and recall.

precision

A measure of how often the positives identified by the learning model are true positives.

prejudice bias

An error in data analysis, introduced when training data is influenced by cultural or other stereotypes, resulting in a faulty model.

Privacy by Design

An approach to software development that takes privacy into account throughout every phase of development.

profiling

The analysis of a person's psychological and behavioral characteristics, so as to assess or predict their capabilities in a certain sphere or to assist in identifying a particular subgroup of people.

pruning

The process of reducing the overall size of a decision tree by eliminating nodes, branches, and leaves that provide little value for the classification or regression problem at hand.

QI

(quasi-identifiers) Data values that do not directly contain PII, but may be used in combination with other data values to identify an individual.

qualitative data

Data that holds categorical values.

quantitative data

Data that holds numerical values.

R²

See *coefficient of determination*.

random forest

An ensemble learning method that aggregates multiple decision tree models together and selects the optimal classifier or predictor.

randomized search

A hyperparameter optimization method that takes a distribution of parameter combinations, trains a model using a random sampling of those combinations, and then returns the combination that best optimizes a specified evaluation metric.

recall

A measure of the percentage of positive instances that are found by a machine learning model as compared to all relevant instances.

regularization

The technique of simplifying a machine learning model by constraining the model parameters, which helps the model avoid overfitting to the training data. This typically involves forcing one or more model parameters to only include values within a small range, or forcing parameters to 0. This helps to minimize the effect of outliers on the model.

reinforcement learning

A type of machine learning in which a software agent acts in an environment in order to obtain a reward.

ReLU function

(rectified linear unit function) An activation function that calculates a linear function of the inputs. If the result is positive, it outputs that result. If it is negative, it outputs 0.

reporting bias

A type of bias that occurs when the training data is missing observations that were not reported.

ridge regression

A regularization technique that uses an ℓ_2 norm to constrain features used to train a model.

RNN

(recurrent neural network) A type of artificial neural network (ANN) in which information can flow to and from artificial neurons in a loop, rather than just a single direction.

robotics

The discipline that involves studying, designing, and operating robots.

ROC curve

(receiver operating characteristic curve) A method of plotting the relationship between predicted hits (true positive rate) versus false alarms (false positive rate).

SAG

(stochastic average gradient) An approach to gradient descent that is similar to stochastic gradient descent (SGD), but which also has a "memory" of past gradient computations for faster convergence.

sample set

A smaller group than the entire population.

scatterplot

Also known as a scatter graph, scatter chart, scattergram, point graph, or $x-y$ plot. This plot draws each data point at an intersection of the x - and y -axis. This can be useful for showing

the relationship of x and y data pairs within a dataset.

search engine

Software that takes a query from the user as input, identifies resources that match the query, and returns these resources to the user.

selection bias

A type of bias that occurs when the training dataset doesn't truly represent the population the model will ultimately be applied to.

semi-structured data

Data that is in a format that facilitates searching, filtering, or extracting some elements of that data, whereas other elements are not so easy to work with.

semi-supervised learning

A type of machine learning in which some label values are provided as input, whereas the rest are unlabeled.

SGD

(stochastic gradient descent) An approach to gradient descent that selects an example at random from the dataset, then uses it to calculate the step-wise gradients.

sigmoid kernel

A kernel trick method that uses a hyperbolic tangent function (\tanh) to create an equivalent of a perceptron neural network.

silhouette analysis

A method of calculating how well a particular data example fits within a k -means cluster as compared to its neighboring clusters.

skew

Used to describe a distribution with a high density of values distributed toward the lower or higher end of the x -axis.

skillful

Used to describe a model that is useful for its intended task. There are degrees of skill; some models are more useful than others. Improving a model's skill is the ultimate goal of the iterative tuning process.

snooping bias

A misuse of data analysis to find patterns that can be presented as statistically significant. In a machine learning workflow, the practitioner may get familiar enough with test and validation datasets as to influence the model, possibly overfitting.

soft-margin classification

An approach to classification with SVMs that keeps the distance between the margins as large as possible while minimizing the number of examples that end up inside the margins.

standard deviation

A measure of variability; the square root of variance.

standardization

A technique in which features are converted into statistical measures so that some features do not exert undue influence over others.

stochastic

A randomly determined process in which individual events or data points cannot be perfectly predicted, but can be shown to follow a general pattern common to the entire set of data.

stratified k -fold cross-validation

A k -fold cross-validation method in which each fold has a representative sample of data in datasets that exhibit class imbalance.

stride

The distance between filters in a convolution as they scan an image.

structured data

Data that is in a format that facilitates searching, filtering, or extracting that data.

supervised learning

A type of machine learning in which known label values are provided as input so that a model can predict these values in future datasets.

SVMs

(support-vector machines) Supervised learning algorithms that can be used to solve

classification and regression problems by separating data values using a hyperplane.

t-test

A type of hypothesis test that compares the mean of two distributions in which the population standard deviation is not known.

tanh function

(hyperbolic tangent function) An activation function whose output values are constraint between -1 and 1.

target function

The formula that an algorithm feeds data in order to calculate predictions.

threshold

A value used by a classification model to classify anything higher than the threshold as positive, and anything lower than the threshold as negative.

TLU

(threshold logic unit) An output neuron that calculates the weighted sum of input neurons and then implements a step function.

TPR

(true positive rate) See [recall](#).

training

In the field of machine learning, the process by which a model learns from input data.

training sample

A dataset of examples used to generate a machine learning model.

transfer learning

A machine learning method in which a model developed for a task is reused as the starting point for a model on a second task.

underfitting

A problem in machine learning in which a model cannot make effective predictions/ decisions due to an inability to identify the underlying patterns in the data. An underfit model exhibits low variance and high bias.

unimodal

A probability distribution with one peak, or mode.

unstructured data

Data that is in a format that makes it difficult to search, filter, or extract that data.

unsupervised learning

A type of machine learning in which label values are not provided as input, so the model does not have an explicit variable that it is predicting.

variance

A measurement of the spread between numbers in a data set.

WCSS

(within-cluster sum of squares) A clustering model evaluation metric that measures the compactness of clusters.

z-score

The number of standard deviations that a sample is above or below the mean of all values in the sample.

z-test

A type of hypothesis test that compares the mean of two distributions when the standard deviation of a population is known.

Do Not Duplicate Or Distribute

Index

A

A/B test [99](#)
accuracy [226](#)
activation functions [369](#)
algorithms [18](#)
analysis of variance test, *See* ANOVA test
ANNs [364](#)
anomalies [4](#)
ANOVA test [100](#)
area under curve, *See* AUC
arithmetic mean [58](#)
artificial neural networks, *See* ANNs
attributes [39](#)
attrition bias [108](#)
AUC [233](#)

B

backpropagation [368](#)
backpropagation through time, *See* BPTT
bagging [314](#)
bag-of-words [410](#)
batch gradient descent, *See* BGD
Bayesian optimization [243](#)
BCSS [256](#)
Bessel's Correction [60](#)
between-cluster sum of squares, *See* BCSS
BGD [185](#)
bias [108](#)
big data [2](#)
bimodal [57](#)
black box [434](#)
box-and-whisker plots [72](#)
box plots [72](#)
See also box-and-whisker plots

BPTT [407](#)

C

C4.5 [296](#)
CACE principle [138](#)
CARTs
 hyperparameters [294](#)
 overview [293](#)
categorical data [86](#)
central processing units, *See* CPUs
centroid [22](#)
changing anything changes everything
principle, *See* CACE principle
Chi-squared test [100](#)
classification and regression trees, *See*
CARTs
clusters [4](#)
CNN [383](#)
coefficient of determination [155](#)
 See also R²
collinearity [169](#)
concept drift [11](#)
confidence interval [101](#)
confusion matrix [225](#)
continuous variables [86](#)
convolutional layer [383](#)
convolutional neural network, *See* CNN
cost function [154](#)
CPUs [30](#)
cross-entropy [217](#)
cross-validation [109](#)

D

data anonymization [430](#)

data binning 297
 data cleaning 85
 data encoding 87
 data mining 4
See also knowledge discovery
 data munging 85
 data snooping bias 108
 data sources 40
 data wrangling 85
 Davies–Bouldin index 258
 decision boundary 196
 decision trees 292
 dendrograms 279
 dependencies 4
 dependent variables 98
 Design of Experiments, *See* DOE
 DevOps 11
 dimensionality reduction 88
 dimensions 39
 discrete variables 87
 discretization 297
 discriminator 389
 DOE 98
 Dunn index 257

E

elastic net regression 169
 elbow point 256
 embedding 409
 ensemble learning 313
 entropy 296
 errors 40
 ETL 47
 Experimental Design 98
 extract, transform, load, *See* ETL

F

false positive rate, *See* FPR
 feature engineering 85
 feature extraction 88
 feature map 383
 feature selection 88
 feature transformations
 functions 114
 in general 114
 feedforward neural networks, *See* FNNs
 filters 383
 fit 90
 FNNs 405
 FPR 230

F1 score 230

G

GAN
 architecture 389
 gated recurrent unit cell, *See* GRU cell
 Gaussian curve 57
 Gaussian radial basis function kernel, *See*
 Gaussian RBF kernel
 Gaussian RBF kernel 340
 generalization 109
 generative adversarial network, *See* GAN
 generator 389
 genetic algorithms 244
 Gini index 293
 GPUs 30
 gradient descent 181
 graphics processing units, *See* GPUs
 grid search 242
 GRU cell 409

H

HAC 277
 hard-margin classification 335
 HDC 278
 heat map 74
 hidden layers 368
 hierarchical agglomerative clustering, *See*
 HAC
 hierarchical divisive cluster, *See* HDC
 histograms 71
 holdout 90
 hyperbolic tangent function, *See* tanh
 function
 hyperparameter optimization 242
 hyperparameters 98, 120
 hyperplanes 334
 hypothesis
 testing 99
 testing methods 99

I

ID3 297
 identity matrix 150
 imputation 89
 independent variables 98
 inference attacks 431
 information gain 296
 input layers 368

intellectual property, *See* IP
 interquartile range, *See* IQR
 IP 439
 IQR 59
 irreducible error 119
 iterative models 181

K

kernel trick
 example 338
 k-fold cross-validation 109
 k-means clustering
 shortcomings 276
 k-nearest neighbor, *See* k-NN
 k-NN 199
 knowledge discovery 4
 kurtosis
 calculation of 64
 types of 63

L

labels 19
 lasso regression 169
 learning rate 183
 leave-one-out cross-validation, *See* LOOCV
 leave-p-out cross-validation, *See* LPOCV
 leptokurtic 63
 linear equation
 data example 145
 shortcomings 148
 linear kernel 340
 linear regression
 in machine learning 148
 matrices in 149
 logistic function 195
 logistic regression 195
 long short-term memory cell, *See* LSTM cell
 LOOCV 110
 LPOCV 110
 LSTM cell
 architecture 408
 process 409

M

machine learning algorithms 101
 machine learning workflow 10
 MAE 155
 matrix operations 30
 maximum 72

MBGD 186
 mean absolute error, *See* MAE
 mean squared error, *See* MSE
 measures of central tendency 58
 memory cells 406
 mesokurtic 63
 mini-batch gradient descent, *See* MBGD
 minimum 72
 MLP 367
 model 10
 model parameters 120
 moments
 statistical 64
 MSE 154
 multi-class classification 216
 multi-label classification 216
 multi-layer perceptron, *See* MLP
 multimodal 57
 multinomial logistic regression 216

N

natural language processing, *See* NLP
 NLP 436
 noise 40
 non-normal distribution 57
 normal distribution 56
 normal equation 150
 normalization 89
 null hypothesis 99
 numerical data 86

O

observer bias 434
 one-hot encoding 298
 open datasets 41
 operator 431
 ordinal data 86
 outliers 40
 out-of-bag error 314
 output layers 368
 overfitting 109

P

padding 386
 parallelization 30
 PCC
 calculation of 65
 Pearson correlation coefficient, *See* PCC
 perceptron

- shortcomings 367
 training 366
personally identifiable information, See PII
PII 424
 pipelines 47
 platykurtic 63
 polynomial kernel 340
 pooling layer 388
PRC 234
 precision 227
precision–recall curve, See PRC
 prejudice bias 435
Privacy by Design 424
 profiling 434
 pruning 295
 p-value 100
- Q**
- QI** 424
 qualitative data 86
See also categorical data
 quantitative data 86
See also numerical data
quasi-identifier, See QI
- R**
- random forest
 hyperparameters 315
 randomized search 243
 range 59
 recall 228
 receiver operating characteristic, *See ROC*
 rectified linear unit function, *See ReLU*
 function
 recurrent neural networks, *See RNNs*
 regularization 120, 168
 reinforcement learning 28
 ReLU function 369
 reporting bias 108
 ridge regression 168
 RNNs
 shortcomings 407
 robotics 441
ROC 230
 R² 155
- S**
- SAG** 186
 samples 98
- sample set 60
 scatterplots 72
 selection bias 108
 semi-structured data 38
 semi-supervised learning 19
SGD 186
 sigmoid function 369
 sigmoid kernel 340
 silhouette analysis 256
 skew 57
 skewness
 calculation of 63
 skillful 108
 soft-margin classification 336
 standard deviation
 calculation of 61
 standardization 89
 standard score 90
 stochastic 20
 stochastic average gradient, *See SAG*
 stochastic gradient descent, *See SGD*
 stratified k-fold cross-validation 110
 stride 387
 structured data 38
 summarization 90
 supervised learning 19
 support-vector machines, *See SVMs*
 SVMs
 for regression 352
- T**
- tanh function 369
 target function 98
threshold logic unit, See TLU
 thresholds 231
TLU 364
TPR 230
 transfer learning 12
 true positive rate, *See TPR*
 t-test 100
- U**
- underfitting 295
 unimodal 57
 unstructured data 38
 unsupervised learning 19
- V**
- variability 59

variance

calculation of [60](#)

in a sample set [61](#)

visualizations [71](#)

W

WCSS [256](#)

within-cluster sum of squares, *See* WCSS

Z

z-score [90](#)

See also standard score

z-test [100](#)

Do Not Duplicate or Distribute

CNX0008S rev 1.0
ISBN-13 978-1-4246-4021-8
ISBN-10 1-4246-4021-0



9 781424 640218



Licensed For Use Only By: Abdulwahab Alweban dev.abdulwahab.alweban@gmail.com Jan 8 202