

Certified Artificial Intelligence (AI) Practitioner (Exam AIP-210)

Certified Artificial Intelligence (AI) Practitioner (Exam AIP-210)

Part Number: CNX0016

Course Edition: 1.0

Acknowledgements

PROJECT TEAM

<i>Author</i>	<i>Technical Consultants</i>	<i>Content Editor</i>
Jason Nufryk	Sarah Haq	Michelle Farney
	Ed Griebel	Peter Bauer

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 © 2022 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-210)

Lesson 1: Solving Business Problems Using AI and ML....	1
Topic A: Identify AI and ML Solutions for Business Problems.....	2
Topic B: Formulate a Machine Learning Problem.....	11
Topic C: Select Approaches to Machine Learning.....	18
Lesson 2: Preparing Data.....	33
Topic A: Collect Data.....	34
Topic B: Transform Data.....	52
Topic C: Engineer Features.....	69
Topic D: Work with Unstructured Data.....	91
Lesson 3: Training, Evaluating, and Tuning a Machine Learning Model.....	131
Topic A: Train a Machine Learning Model.....	132
Topic B: Evaluate and Tune a Machine Learning Model.....	146

Lesson 4: Building Linear Regression Models.....	163
Topic A: Build Regression Models Using Linear Algebra.....	164
Topic B: Build Regularized Linear Regression Models.....	187
Topic C: Build Iterative Linear Regression Models.....	200
Lesson 5: Building Forecasting Models.....	213
Topic A: Build Univariate Time Series Models.....	214
Topic B: Build Multivariate Time Series Models.....	230
Lesson 6: Building Classification Models Using Logistic Regression and k–Nearest Neighbor.....	249
Topic A: Train Binary Classification Models Using Logistic Regression..	250
Topic B: Train Binary Classification Models Using k–Nearest Neighbor..	266
Topic C: Train Multi–Class Classification Models.....	274
Topic D: Evaluate Classification Models.....	284
Topic E: Tune Classification Models.....	302
Lesson 7: Building Clustering Models.....	313
Topic A: Build k–Means Clustering Models.....	314
Topic B: Build Hierarchical Clustering Models.....	330
Lesson 8: Building Decision Trees and Random Forests.....	345
Topic A: Build Decision Tree Models.....	346
Topic B: Build Random Forest Models.....	365
Lesson 9: Building Support–Vector Machines.....	381
Topic A: Build SVM Models for Classification.....	382
Topic B: Build SVM Models for Regression.....	400

Lesson 10: Building Artificial Neural Networks..... 409

Topic A: Build Multi-Layer Perceptrons (MLP).....	410
Topic B: Build Convolutional Neural Networks (CNN).....	432
Topic C: Build Recurrent Neural Networks (RNN).....	451

Lesson 11: Operationalizing Machine Learning Models..... 467

Topic A: Deploy Machine Learning Models.....	468
Topic B: Automate the Machine Learning Process with MLOps.....	482
Topic C: Integrate Models into Machine Learning Systems.....	491

Lesson 12: Maintaining Machine Learning Operations..... 507

Topic A: Secure Machine Learning Pipelines.....	508
Topic B: Maintain Models in Production.....	521

Appendix A: Mapping Course Content to CertNexus® Certified Artificial Intelligence (AI) Practitioner (Exam AIP-210)..... 535

Appendix B: Datasets Used in This Course.....	537
Mastery Builders.....	541
Solutions.....	565
Glossary.....	579
Index.....	593

About This Course

Artificial intelligence (AI) and machine learning (ML) have become essential parts 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, all while following a methodical workflow for developing data-driven solutions.

Course Description

Target Student

The skills covered in this course converge on four areas—software development, IT operations, applied math and statistics, and business analysis. Target students for this course should be looking to build upon their knowledge of the data science process so that they can apply AI systems, particularly machine learning models, to business problems.

So, the target student is likely a data science practitioner, software developer, or business analyst looking to expand their knowledge of machine learning algorithms and how they can help create intelligent decision-making products that bring value to the business.

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-210) certification.

Course Prerequisites

To ensure your success in this course, you should be familiar with the concepts that are foundational to data science, including:

- The overall data science and machine learning process from end to end: formulating the problem; collecting and preparing data; analyzing data; engineering and preprocessing data; training, tuning, and evaluating a model; and finalizing a model.
- Statistical concepts such as sampling, hypothesis testing, probability distribution, randomness, etc.
- Summary statistics such as mean, median, mode, interquartile range (IQR), standard deviation, skewness, etc.
- Graphs, plots, charts, and other methods of visual data analysis.

You can obtain this level of skills and knowledge by taking the CertNexus course *Certified Data Science Practitioner (CDSP) (Exam DSP-110)*.

You must also be comfortable writing code in the Python programming language, including the use of fundamental Python data science libraries like NumPy and pandas. The Logical Operations course *Using Data Science Tools in Python®* teaches these skills.

Course Objectives

In this course, you will develop AI solutions for business problems.

You will:

- Solve a given business problem using AI and ML.
- Prepare data for use in machine learning.
- Train, evaluate, and tune a machine learning model.
- Build linear regression models.
- Build forecasting models.
- Build classification models using logistic regression and k -nearest neighbor.
- Build clustering models.
- Build classification and regression models using decision trees and random forests.
- Build classification and regression models using support-vector machines (SVMs).
- Build artificial neural networks for deep learning.
- Put machine learning models into operation using automated processes.
- Maintain machine learning pipelines and models while they are in production.

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.
- **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.
	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

Lesson Introduction

Machine learning 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.
- Formulate a machine learning approach to solve a specific business problem.
- Select approaches to machine learning.

TOPIC A

Identify AI and ML Solutions for Business Problems

Before you dive into the technical details of artificial intelligence and machine learning, you need to understand how they fit within a larger business context. On the job, you'll be applying your skills to achieve one or more business goals, so it's important to keep those goals in mind all throughout the project.

AI and Machine Learning

Machine learning is a discipline of **artificial intelligence (AI)**. In machine learning, computers make predictions and other decisions based on sets of data, and do so without any explicit instructions provided by a human. This differentiates machine learning techniques from traditional software solutions that are fully programmed to perform some intended function. Independence from human intervention enables machine learning techniques to automate decision-making processes in ways that are faster and more efficient than any traditional software solution. In addition, the other major strength of machine learning is in the second word of its name—its ability to *learn*. Machine learning processes build on themselves over time through experience, becoming more and more effective at solving a problem or set of problems. A subset of machine learning is **deep learning**, which involves the use of complex artificial neural networks that are even more effective at solving problems.

The most fundamental concept behind machine learning is the algorithm. An **algorithm** is a set of rules for solving problems. At its core, an algorithm is a mathematical formula or group of formulas that take some input and then output some result. Algorithms also support the "learning" aspect of machine learning since they are designed to update the beliefs and assumptions about the data they're working with over time.

You may also hear of data science spoken alongside machine learning and AI. In practice, the tasks involved in data science and machine learning often overlap. It's common to think of a data science practitioner as preparing and analyzing data, but they may also use that data to train machine learning models. Data science encompasses more than just machine learning, but machine learning is a significant part of the data science process.

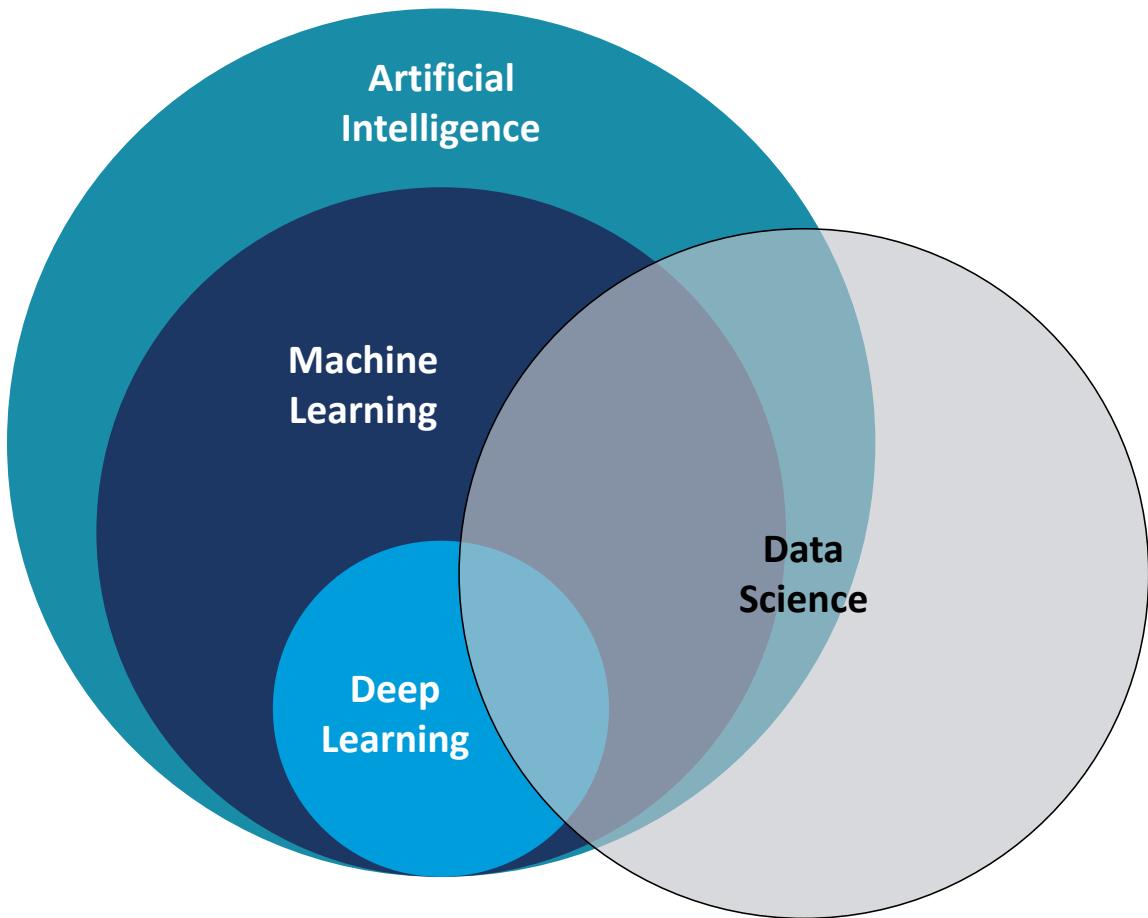


Figure 1–1: A Venn diagram that approximates the relationship between AI, machine learning, deep learning, and data science.



Note: Some sources consider AI and machine learning to be a subset of data science, rather than a related field with overlapping concerns.

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, and 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 suggest 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.



Note: When depicted as a pyramid, the DIK hierarchy usually places data at the base and knowledge or wisdom as the capstone. This demonstrates that the higher levels are built upon the lower levels (e.g., information is generated from data).

Applied AI and ML

Artificial intelligence is an academic discipline, and a great deal of research goes into understanding and improving that field of study. The theory behind AI is certainly interesting, but the job of a business professional involves applying AI and machine learning principles to solve practical problems. So, although understanding some basic theory is important, much of the machine learning practitioner's daily responsibilities will focus on creating a product that meets the needs of an organization and/or its customers.

You'll likely develop AI/ML solutions as part of a project, so it helps to contextualize your work as part of an overall effort to achieve a goal or set of goals. You may be sharing that effort with other members of a team, so communication is an important skill for a machine learning practitioner to have.

It's important that you get a taste of a multitude of practical applications of AI/ML solutions, even if these applications do not apply to your specific industry. After all, having a well-rounded skillset is a quality that attracts employers, particularly in fields like AI/ML that are in high demand. And, having a breadth of knowledge can spur creative problem solving, whereas a narrow perspective can hinder it.

Solutions for Commercial Problems

The following table lists some common problems in commercial organizations, with examples of how AI/ML can solve those problems.

Problem	Example Solutions
Poor sales growth	<ul style="list-style-type: none"> • Identify new leads and prospects that will maximize engagement and stimulate sales. • Automate sales management processes so salespeople can focus more on content creation and less on perfunctory administration tasks.
Low customer retention	<ul style="list-style-type: none"> • Make the most effective use of marketing and sales by focusing contact based on a customer's interests, browsing, and buying history. • Recommend additional/other products based on those the customer has already shown interest in.

Problem	Example Solutions
Lack or excess of available inventory	<ul style="list-style-type: none"> Predict when sales for a product are likely to increase so that enough inventory is available. Identify product inventory that can be reduced to lower inventory carry costs.
Faulty product being released to vendors	<ul style="list-style-type: none"> Scan parts automatically as they emerge from an injection mold, and eject flawed parts into a rework bin. Inspect products in a manufacturing center for damage as they pass down an assembly line.
Operational interruptions due to employee turnover	<ul style="list-style-type: none"> Determine what factors are most likely to cause an employee to leave the company so that they can be rectified. Identify what operational areas are most affected by employee turnover so more resources can be assigned to them in such an event.

Solutions for Governmental Problems

Next are some problems common to government institutions, along with some example solutions.

Problem	Example Solutions
Public health crises	<ul style="list-style-type: none"> Predict areas or populations most likely to see a surge in health issues so that safety measures can be implemented. Optimize the logistics of rolling out a vaccine or other treatment plan.
Security breaches	<ul style="list-style-type: none"> Use heuristic analysis systems to detect and prevent novel malware and network intrusion vectors. Use real-time security camera analysis to track the movement of people within a building to perform surveillance and identify unusual traffic patterns or behavior.
Terrorism	<ul style="list-style-type: none"> Monitor known communication channels and use language analysis techniques to determine if an attack is being planned. Use facial recognition systems to identify persons of interest as they move through airports and other public places.
Economic downturn	<ul style="list-style-type: none"> Use market factors to anticipate an impending recession or other economic downturn so that mitigation efforts can begin early. Assess the behaviors that contribute the most to economic downturn so that policies or legislation can be enacted to prevent such problems.
Tax fraud	<ul style="list-style-type: none"> Identify anomalous entries in tax filings that could prompt an audit. Predict the tax gap for next year to determine if anti-fraud efforts are effective or need to be re-assessed.

Solutions for Public Interest Problems

Some issues are related to the public interest and may be addressed by government agencies, private advocacy groups, or both.

Problem	Example Solutions
Poverty	<ul style="list-style-type: none"> Determine how to grow high-yield crops in arid regions to provide a food source to impoverished communities. Use analytics to identify the factors that contribute the most to individual and communal poverty so that resources can be allocated properly.
Human rights abuses	<ul style="list-style-type: none"> Analyze available resources to determine how best to provide support to citizens who are subject to human rights abuses. Use an intelligent scoring system to evaluate organizations or governments that engage in human rights abuses.
Lack of education	<ul style="list-style-type: none"> Design learning plans that are tailored to the needs of individual students based on the students' profiles. Assess the effectiveness of current teaching and testing programs to identify areas for improvement.
Crime	<ul style="list-style-type: none"> Predict surges in criminal activity in certain areas to better prepare law enforcement and crime-watch groups. Identify activities and resources that are the most effective at reducing criminal behavior.
Climate change	<ul style="list-style-type: none"> Predict the effects of climate change on people, infrastructure, and the environment. Determine best practices for individuals and organizations to help mitigate the effects of climate change.

Solutions for Research Problems

AI can also help solve problems that hinder the acquisition of knowledge, whether scientific, cultural, historical, or otherwise.

Problem	Example Solutions
Lack of peer review	<ul style="list-style-type: none"> Develop intelligent systems that can assess research literature for deficiencies, rather than requiring a human expert to perform a review. Determine the most effective and time-efficient strategies for reviewing a work so that human reviewers are not overburdened.
Bias	<ul style="list-style-type: none"> Evaluate data collection and surveying methods used by researchers to identify any that contribute to biased results and conclusions. Perform a meta-analysis of research literature to identify trends that could reveal bias.

Problem	Example Solutions
Non-reproducible studies	<ul style="list-style-type: none"> Identify and acquire more data sources that would increase sample sizes in scientific studies, thus making it easier to reproduce the studies' stated effects. Extract useful metascientific data—research about research—to identify where improvements to the process can be made.
Gaps in research	 <p>Note: The term "replication crisis" is also used to describe the ongoing problem in scientific research whereby the results of published studies are difficult to reproduce.</p>
Unethical practices	<ul style="list-style-type: none"> Use analytics to determine where the largest gaps in research are for a particular academic discipline or field of study. Predict where gaps in research may begin to form in certain disciplines over time. Recognize patterns in research methodology to uncover ethical and integrity violations. Develop intelligent frameworks for ethically conducting research in subject areas that involve sensitive information or safety issues.

Stakeholder Influence

A project **stakeholder** is a person who has a vested interest in the outcome of a project or who is actively involved in its work. Stakeholders take on various roles and responsibilities; their participation in the project will have an effect on its outcome and its chance of success. There are many different types of stakeholders, both internal to the organization and external.

Common stakeholders include:

- Customers/end users.** These are the people who will use the product or service generated by the project. They typically provide payment for the project output and can also provide feedback from a usability perspective.
- Sponsors/champions.** These are individuals or groups that provide finances, management support, and overall control of the project. They are integral to ensuring that the project outcomes are on track and within budget.
- Program/project managers.** These are the people who manage the overall project or an aspect of a project. They typically provide guidance as far as timelines and resource allocation.
- Team members.** These people are the practitioners who work directly to develop the project. They can provide insight into what tools, technologies, and other resources are needed in order for the project to be successful.
- Business partners.** These organizations might sell the product or service, or they might provide resources during development. Since they're external, they tend to bring a fresh "outsider" perspective to the project.
- Governments.** Various government organizations might take interest in projects that are large in scope or have legal ramifications. They provide rules and frameworks for legal issues.
- Societies.** Some projects can have a profound impact on society at large, so people throughout a society will take notice. As a collective, people might express concerns about a project and its impact on everyday life.

Some or all of these stakeholders might be relevant to the specific project you're working on. You should consult with your project team leaders to get an accurate picture of who is involved.

Stakeholder Requirements

Stakeholders have competing interests, needs, priorities, and opinions. They may have conflicting visions for the project's successful outcome. Project leaders must identify internal and external stakeholders as early as possible, learn what their needs are, and secure their participation in defining the project's parameters and success criteria. As a practitioner, you need to at least understand the requirements that stakeholders have for the project so that you don't just work in a vacuum. After all, the whole purpose of the project is to bring value to the business, and stakeholder requirements define the form that value takes.

Communication Strategy

Most projects, machine learning or otherwise, will benefit from a stakeholder communication strategy. The strategy outlines the ways in which the organization should interact with the project's many stakeholders. This includes everything from soliciting requirements at the beginning stages of a project, all the way to communicating results at the end of the project.

Even if you're not responsible for creating this strategy, you may be responsible for carrying out at least part of it. For example, you may be called on to:

- **Investigate user requirements feasibility.** Your solution will take different forms depending on your users' requirements for that solution. For example, you may need to ensure that a machine learning model can be incorporated into a larger cloud-based app ecosystem that interfaces with end users. So, you could be expected to solicit user feedback, or perhaps respond to it.
- **Consult with industry experts.** As much as you may learn from a course like this, there's really no substitute for experience. Machine learning experts can help you determine the best course of action in complex and unique circumstances. Or, you may need to consult with experts in the problem domain itself. For example, if you want to predict failures in manufacturing equipment, you might consult with the engineers who designed this equipment so that you have a deeper understanding of how it operates.
- **Participate in a community.** Some organizations, in an effort to promote transparency, provide updates on their projects to wider communities. As a practitioner with specialized knowledge, you may be asked to provide high-level insight into the machine learning process. Or, you may simply be expected to keep up with the AI/ML industry by participating in online forums, attending conferences, and so on.
- **Communicate results to an audience.** At any stage of the project where you have some deliverable, you may be asked to explain that deliverable to an audience. The audience might be your immediate manager, the organization's upper management, business partners, or the public. For example, you may need to convince stakeholders that the solution you developed is able to effectively solve the problem. Therefore, you create a presentation that incorporates statistical evidence and visuals supporting your solution.
- **Address ethical risks.** Some risks are only really apparent in the technical application of AI/ML. So, you may need to explain what those risks are, and identify ways to avoid or address them. For example, developing a self-driving car has obvious safety risks, so you'll likely need to outline how you believe your solution minimizes the chance of harm.

Ethical Risks in AI and ML

The ethics of AI/ML are important for all stakeholders to consider, even if not every stakeholder has a hand in addressing those issues. As a practitioner, you are responsible for developing solutions that minimize harm and align with the ethical principles established early on in the project. For now, you should be aware of the high-level risks that are inherent in all AI/ML projects.

- **Privacy**—Protects sensitive data about people from unauthorized or unwanted access. Powerful data-driven technologies like AI/ML raise concerns about how personal data is being used.

- **Accountability**—Ensures people are held responsible for their actions. AI/ML can make accountability a challenge, since decisions of great consequence are made by machines and not people.
- **Transparency and explainability**—Enable people to see into the inner workings of the technology and its effects on people and society. The complex nature of AI/ML can make supporting these principles difficult.
- **Fairness and non-discrimination**—Ensure people are not treated improperly because of who they are. AI/ML can contribute to issues of inequality.
- **Safety and security**—Minimize the chance that people are physically harmed and that property (whether physical or virtual) is compromised. Emerging technologies like AI/ML can cause harm directly or indirectly.

ACTIVITY 1–1

Identifying AI and ML Solutions for Business Problems

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're working with a major online clothing retailer that has been having difficulty retaining customers after an initial purchase.

How might AI/ML be used to solve this problem?

2. What stakeholders might be involved in the decision to use AI/ML to solve this customer retention problem?

3. You're working with a government organization that operates a major power plant. Because it supplies power to so many people, the power plant is considered critical infrastructure and must be secured against a cyberattack. A recent breach has exposed significant gaps in the plant's security.

How might AI/ML be used to solve this problem?

4. What kind of ethical risks might there be in the power plant implementing AI/ML?

TOPIC B

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

Problem formulation is the process of identifying an issue that should be addressed and putting that issue in terms that are understandable and actionable.

To make a problem "understandable" is to ensure it can be easily defined in terms of business needs. To make a problem "actionable" is to give high-level direction as to how the project members can approach solving the problem. Most machine learning projects will benefit greatly from some kind of problem formulation. The following outlines a general approach to problem formulation that you might use.

Frame the problem.

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:

- **Task:** What the solution should accomplish—for example, "Predict the sale price for a house."
- **Experience:** What dataset the solution uses to learn—for example, "Records of real estate transactions for King County for the year 2022."
- **Performance:** How you will evaluate the performance of the solution. How you measure this depends on the type of task the solution performs.

Identify why the problem must be solved.

Aspects of this include:

- **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'll evaluate your success.
- **Lifetime and use**—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.

Being 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.

Provide background information that will help solve the problem.

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:

- **Assumptions**—A list of assumptions about the problem, such as:
 - 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 timelines, 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. While AI/ML may always seem like the more attractive option, it is not without its risk. It can be difficult to go through the entire process effectively, especially if the quality or quantity of data is insufficient. The process can also become expensive and time consuming, potentially to a degree that it outweighs its benefits.

So, you should always ask yourself and your colleagues if the solution to the problem can be developed using simpler methods and strategies. AI/ML in the business world must serve the business first and foremost, and if it cannot be justified for a particular scenario, then it shouldn't be chosen.

Consider a help desk system that needs to route tickets to the appropriate IT personnel. You might be able to program an algorithm that parses the text of a ticket and then decides where to route the request based on a set of rules (e.g., if the ticket contains the term "VPN," then it would get routed to the network specialist). This could be done using traditional programming logic, without the aid of AI/ML. Of course, that's not to say that AI/ML *can't* be used in this scenario, but that it may not be worth the extra effort and expense.

Machine Learning Outcomes

The models produced by the machine learning process can fit into one of several categories. Each category describes a particular task that the model performs. You can also think of these tasks as outcomes, since the model is applied to a real-world issue in order to return some kind of result. Dividing models into these outcomes is not just an academic exercise—it will have a significant impact on the approach you use to develop the model.

Outcome	Description
Regression	<p><i>Given a new instance of data, estimate the value of a numeric variable.</i></p> <p>Regression models are used to determine the relationships between variables, generating an outcome that is numeric (rather than categorical, like in classification). For example, based on patterns inherent in already measured data, a regression model 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>
Classification	<p><i>Given a new instance of data, identify the class it belongs in.</i></p> <p>The goal of a classification model may be to simply place the data instance into a predefined 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>

Outcome	Description
Clustering	<p><i>Without any knowledge of a target variable, identify components that belong grouped together.</i></p> <p>Clustering models are similar to classification models in that they can categorize data. But, clustering is used when the input data does not already have predefined classes. For example, you may want to segment your customers into groups for targeted marketing purposes, but you don't necessarily know what those groups are. Clustering enables you to discover those groups, as well as place a new data instance into one of those groups.</p>

Randomness and Uncertainty

Machine learning is based on the mathematical fields of statistics and probability. Although statistics and probability are often mentioned together and sometimes used interchangeably, they in fact 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.

Randomness suggests a lack of identifiable pattern. When it comes to data, 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 creating and testing the model, and so on. There is also randomness in the algorithms and other mathematical formulas that create statistical models. Two different algorithms may produce similar outcomes, but they may also produce slightly different results simply because they take different steps to obtain those results.

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 determined. 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, the model can make reasonably good decisions about individual data samples. This is why, despite the element of randomness in machine learning models, they are still useful for many tasks.

There are some techniques that can help you mitigate the effects of randomness in machine learning models. At the moment, you just need to be aware that machine learning does not offer guarantees or promises when it comes to solving problems—only estimations. Still, if properly devised, these estimations can be extremely powerful.

Design of Experiments (DOE)

Experimentation is an important part of the machine learning 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 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 (DOE)** approach (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 effectiveness of the model.
- **Adjust settings of the learning algorithm to optimize its performance.** You can experiment to see how you can adjust various configuration parameters to tune the performance of the model, and choose parameters that produce the best model for your needs.

Probability of Success

As part of determining whether or not an AI/ML solution is appropriate in solving a business problem, you can consider the solution's probability of success. As powerful as AI/ML might be, it will not necessarily be able to address every issue to the level you need it to. If properly developed and implemented, chances are it won't fail outright, but it may still come up short. This might be because the problem itself is complicated, or because there is a lack of useful, available data. For example, developing a model to predict long-term changes in cultural beliefs and attitudes is not guaranteed to succeed because the problem domain has so many variables and nuances that are difficult to account for. Or, perhaps you're just trying to predict customer interest for a small business, but you have very little historical data to draw from, so the solution is too feeble to make any worthwhile predictions.

To determine the probability of this type of success, you should consider factors like hardware and software requirements, personnel skill requirements, time requirements, stakeholder requirements, weighing the potential benefits of success against the potential costs/risks of failure, and other metrics that are typically used to identify the viability of a project.

Even if you've determined in a general sense that AI/ML has a good chance of succeeding, you may also want to investigate the probability of success for more specific algorithms and implementations. You might rank each algorithm according to how useful it may be in solving your particular problem. This way, you can prioritize the top-ranking algorithms so that you don't waste time and resources on algorithms that have little chance of success.

To generate these rankings, you can consider the results of other machine learning projects in similar fields, as well as the overall effectiveness of an algorithm as compared to its alternatives. You may also generate a preliminary *proof of concept (POC)* solution on a small amount of data (whether real or artificial) and then evaluate its performance using different algorithms. The performance of each algorithm in the POC can suggest that some are likely to be more successful than others at solving your business problem.

Guidelines for Formulating a Machine Learning Problem



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

Follow these guidelines when formulating a machine learning problem.

Formulate a Machine Learning Problem

When formulating a machine learning problem:

- **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 technical solution you come up with, identify what *business* outcome you are trying to achieve.
- **Identify where the data will come from.** You must be able to obtain historic data representing the outputs you want to generate. 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 outputs 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 months-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 machine learning will help you focus on the business logic and may lead you to simpler, more elegant solutions, even if you do end up using a machine learning solution.
- **Frame the business issue 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 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 or clustering).

ACTIVITY 1–2

Formulating a Machine Learning Problem

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 inspection cameras to scan parts coming out of molds to identify cracks, voids, and other defects that render the parts unusable.

Currently, human personnel monitor the inspection cameras and look for the aforementioned issues in the cast metal parts. This is tedious work and is prone to errors, so IOT Company wants to explore ways to automate the process.

In this activity, you'll consider how AI/ML can be used to enhance IOT Company's manufacturing process.

1. The COO of IOT Company issues the following mission statement: "We need to reduce errors in our cast-metal manufacturing process by 20%."

How might you formulate this statement as a problem to be solved by AI/ML?

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

What type of outcome—regression, classification, or clustering—would you look for in this example?

3. Over time, parts coming out of a mold tend to show increasingly more defects, to the point where eventually a threshold is passed. 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 would you look for in this example?

4. You find a source of data that you could use for the project. It includes a historic database of hundreds of thousands of metal parts. Some of the specific data you need is 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, your colleague does not seem to think that the incomplete data is a problem and thinks that the dataset will actually work quite well for machine learning.

Why is this situation acceptable for a machine learning project?

TOPIC C

Select Approaches to Machine Learning

In the previous topics, you got a taste of how machine learning might be used to solve general problems. In this topic, you'll take a deeper dive into the many ways in which machine learning can be applied to practical business scenarios across a variety of industries and fields.

Machine Learning Systems and Techniques

There are various techniques common to applied machine learning, as well as machine learning systems that fulfill specific purposes. Although you may only end up using one or a few on the job, it's good to get a sense of the many ways machine learning is applied to business.

Common machine learning systems and techniques include:

- Prediction
- Recommendation
- Diagnosis
- Natural language processing (NLP)
- Computer vision
- Robotics

Prediction

A ***prediction*** system, or a predictive system, uses machine learning techniques to estimate the state of something in the future based on past or current data. Prediction is perhaps the most common approach in machine learning, or at least the one that comes to many people's minds when they think about AI/ML.

What it means to predict the "state" of something can vary widely, as can the "something" itself. Systems based on machine learning can predict the magnitude of a numeric value for a new instance of data, whether that instance is a newborn's projected height when they're an adult; the expected lifespan (in years) of a computer storage device based on its technology and operational conditions; the level of contaminants in a water source based on its location and climate factors; and so on.

Prediction can also involve estimating the magnitude of some number based on the change in some event over time—a process called forecasting. For example, a machine learning system can forecast the rise or fall of a stock price the next day; the change in coolant levels for a nuclear reactor over the next month; the number of user logins a cloud service expects to see over the next year; and so on.

Machine learning can also predict the nature of someone or something for which this information is not already known. For example, a system may be able to predict that a new customer is likely to be a returning customer based on their initial purchasing behavior, as well as their personal attributes. Or, a system might be able to predict that the business would benefit the most if a customer was placed in Group A rather than Group B.

Recommendation

A ***recommendation system***, or recommender system, suggests items, services, and other things of interest to users. Recommendation systems powered by machine learning are able to more accurately determine what most interests the targeted user(s), or what recommendations users will respond most positively to. Ideally, this makes the system more useful to both the user (who is more likely to see something they like) and the business (who is more likely to generate sales or interest in their brand).

Recommendation systems typically work on the principle that, faced with an overwhelming selection to choose from, most users would rather have those selections filtered to show only the few that are most relevant to them. This can be everything from what product the user might want to buy next from a store, to what movie a user might want to watch next on a streaming platform, and much more.

The two main types of recommendation systems are collaborative filtering and content filtering. In **collaborative filtering**, the system finds similar users who "like" the same things. It assumes that these users will continue to like what they did in the past. So, if User A and User B both like the same 10 movies, the system can find a movie that User A likes that User B hasn't yet seen, and recommend that movie to User B. One of the advantages of collaborative filtering is that it is agnostic to the content of the thing being recommended. There's no need for the system to learn the attributes of the movie—its genre, actors, director, and so on.

In **content filtering**, the system *does* learn the attributes of the thing it is considering for recommendation. It analyzes these attributes and compares them to a profile of the user and their attributes. For example, if a user watches a lot of horror movies and tends to watch movies that have the same few directors or actors, the content filtering system can recommend an appropriate movie that they haven't yet seen. This approach is more commonly applied to machine learning since it requires that deeper analysis of both content and user.



Note: Some recommendation engines adopt a hybrid approach, employing both collaborative filtering and content filtering.

Diagnosis

A **diagnostic** system is one that determines the cause and nature of anomalous behavior, activity, or conditions in an environment. The "environment" can be the human body, a physical device, computing software, etc. With machine learning–capable systems, the speed and accuracy of a diagnosis can increase significantly as compared to that same diagnosis performed by a person.

Medical diagnosis is the most popular application of diagnostic systems. Medical professionals can use machine learning to identify the presence of a disease or other ailment in a patient. Just like a human doctor would, the machine uses information about the patient's signs and symptoms, as well as the patient's profile (e.g., age, weight, ethnicity, etc.) to determine the most likely cause of the problem. The system can use raw data about the patient as well as images of their body to make a diagnosis.

Aside from medical diagnosis, automated systems can also perform diagnosis in an engineering context. For example, an operating system could be built to analyze the behavior of a computer and/or its user to determine why a network connection has failed, rather than just give generic troubleshooting tips. Or, a system that monitors manufacturing equipment can analyze equipment failures and determine why they are most likely occurring.

The common theme of intelligent diagnostic systems is that they are able to correlate what they know about the target environment with what they know about problems that affect those environments.

Drug Discovery

Beyond diagnosis, machine learning can also aid medical tasks like drug discovery, in which chemical substances are determined to be viable as a new form of medication. Such systems can more efficiently identify how novel chemical compounds interact with the targets of clinical trials. They can also aid the development of clinical trials by quantifying the successes and failures of a trial.

Natural Language Processing (NLP)

Natural language processing (NLP) is the general term for the task by which computers work with human languages using machine learning tactics. It encompasses multiple techniques, including:

- Speech recognition
- Text analysis
- Natural language understanding
- Natural language generation
- Natural language translation

Speech recognition, as an example, has been in use long before machine learning became prominent. Such systems could even exhibit high levels of accuracy. But for most users, even a small margin of error in speech recognition is enough to make such a system frustrating and unusable, so those older systems had little application in the business world. With machine learning, speech recognition accuracy has increased to such a degree that not only is such a task viable, but it has become incredibly popular. After all, millions of people use speech recognition to interact with their smartphones or home assistant devices.

NLP tends to be most successful in a deep learning context, where human language is analyzed using complex artificial neural networks.

Computer Vision

Computer vision, like NLP, is an umbrella term for several related, but distinct techniques. These techniques attempt to process image and other visual data at a level that outpaces human analysis. Some common computer vision techniques include:

- Object recognition
- Object classification
- Image generation
- Motion detection
- Trajectory estimation
- Video tracking
- Navigation

To use object recognition as an example, a common application of computer vision is to detect the presence of something in an image. For instance, a camera could take images of food delivered to a grocery store from a supplier and quickly spot any signs of mold or other spoilage. Although a human could do this, the object recognition system would likely be faster, and may even be more accurate.

Computer vision has found most success in deep learning using artificial neural networks. These networks are able to extract useful patterns from image data that traditional machine learning approaches cannot.

Robotics

Robotics is the discipline that involves studying, designing, and operating robots. There is no universally agreed-upon definition of "robot," however. In general, it can be said that a robot is a machine that can act in the physical world with some degree of autonomy. In other words, a robot can act independent of human control. These are also called autonomous systems.

Robots are typically designed to perform repetitive, strenuous, or dangerous tasks that humans would rather not, or cannot, do. Some examples of robots include robotic welding arms in industrial settings, robots that clean floors in a home, robots that are used in the military, robots that care for the sick and elderly, robotic rovers that collect scientific data on the deep ocean floor and on other planets, and so on.

The autonomous aspect of a robot is commonly constructed using reinforcement learning techniques. These techniques enable the robot to obtain up-to-date data about its environment, as well as use that data to make informed decisions about how to interact with the environment.

Learning Modes

In the realm of machine learning, the outcomes described earlier (regression, classification, and clustering) are actually subsets of broader **learning modes**. These learning modes describe *how* a model learns from the data, and they have an effect on the content and form of the data itself. There are four primary learning modes:

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning

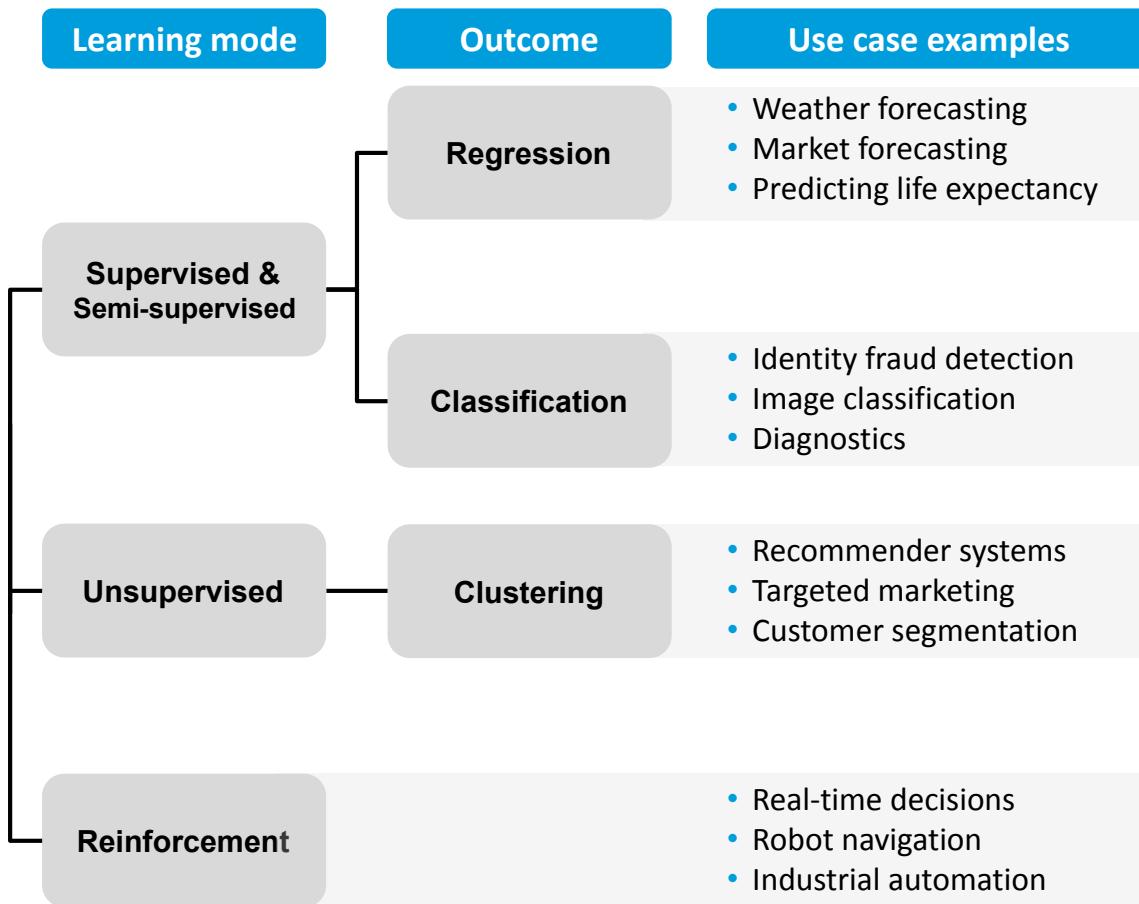


Figure 1-2: Learning modes mapped to outcomes and use cases.

Supervised Learning

In **supervised learning**, you create a machine learning model by providing a dataset for which you already have a set of correct answers, called **labels**. Labels are also referred to as the "ground truth." The model 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 estimations based on patterns in the data or must correctly place items into an appropriate category. Regression and classification are supervised learning outcomes.

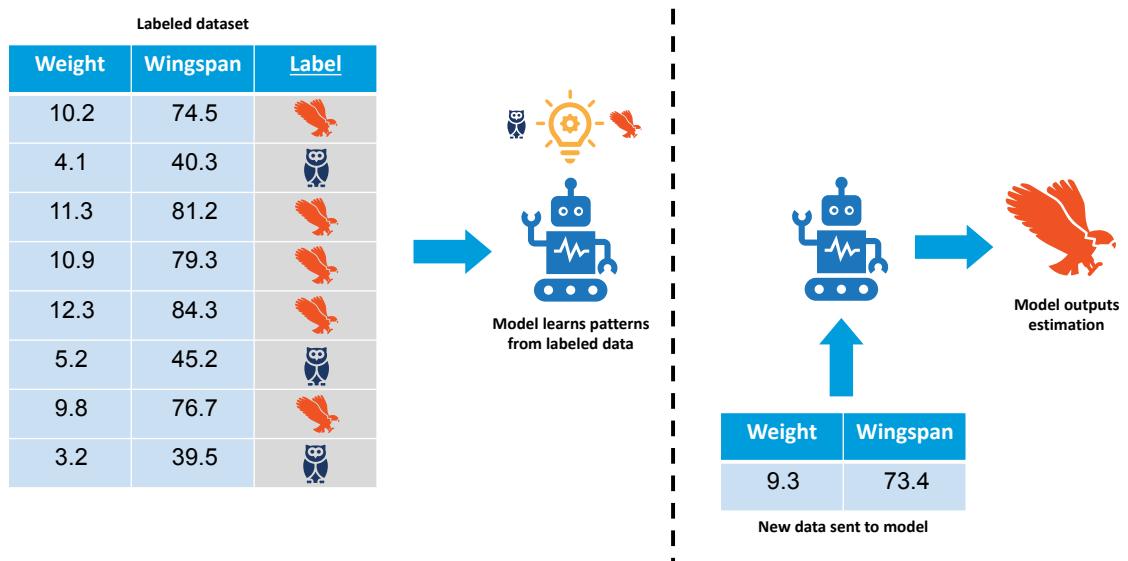


Figure 1–3: The supervised learning process applied to a labeled dataset.

Examples of Supervised Learning Systems

The following are some examples of systems that apply supervised learning to solve real-world business problems.

- **Spam filtering**—Modern anti-spam systems can learn the characteristics of email or text message spam from labeled datasets and then use this information to classify new messages as either spam or not spam. The system can learn from historic datasets where the textual content or metadata of messages (or both) have been labeled by humans who can easily determine whether the message is or is not spam. The system can also learn from new data; for example, a message gets by the filter and the user manually marks it as spam, which gets fed to the system as new data to learn from.
- **Fraud detection**—Banks use fraud detection systems to identify transactions and other activity that is likely to indicate malicious actions taken against users or the bank itself. The system can draw from historical records of accounts or transactions that are known to engage in fraud, which it uses to classify activity as being legitimate or fraudulent. Or, it can rank activity on a sliding scale, where the activity has some *likelihood* of being fraudulent.
- **Language detection**—Some systems process audio or textual data that can automatically detect the language that is spoken or written based on the presence of known, labeled words or sentences. The work can be classified and then sent to the proper channels for analyzing or translating the work. For example, in multilingual countries, a citizen might fill out a government form in their native language; the government agency can automate the sorting of this form depending on the detected language.
- **Image detection**—There are many prominent systems that can classify an image based on a corpus of millions of labeled images. These systems can identify specific characteristics of an image; identify the presence of objects or people; identify the style or origin of an image; and so on. For example, optical character recognition (OCR) systems have learned from many examples of labeled handwritten and typed characters to be able to accurately categorize text.
- **Bioinformatics**—Bioinformatics combines biology and data science to better understand biological processes. For example, a supervised learning system can take a labeled dataset of bacterium genomes and determine whether or not a newly discovered genome belongs to a certain phylum of bacteria. The complexity of genomes, even in simpler organisms, makes this a difficult task for humans to do.
- **Customer churn prediction**—A customer who "churns" does not return to purchase more products. Likewise, a retained customer is one who does. To minimize churn or maximize retention, a business might have a dataset of customer information that is labeled as to whether

or not the customer did return to make more purchases over a time period. A supervised system can use this information to predict new customer churn, enabling the organization to take steps to prevent this (e.g., enticing the customer with a coupon).

- **Search engine ranking**—Search engines can gather information about the browsing habits of users who conduct searches. A dataset might indicate the search term and other factors involved in the search, and the label itself might describe customer engagement with the results. That engagement could be a simple classification (did or did not click on a result) or it could be a subjective assessment of the quality of the results ("poor", "fair", or "good"). The system can use this information to promote or demote a linked result based on its helpfulness to the user.
- **Sales forecasting**—A business likely has sales numbers from previous quarters or other periods of time that can effectively act as a label for a system to learn from. Most forecasts solve regression problems, where the output from the system is a number that indicates the likely volume or monetary value of sales in the next n time periods.
- **Weather forecasting**—Similar to sales forecasting, weather forecasting relies on known data from the past to make predictions for the future. The forecast can take on numerical data like predicting tomorrow's high temperature, or it can use numerical climate factors to predict storms and other adverse phenomena.
- **Commodity value prediction**—A business might have data concerning how the price or value of products have fluctuated over time. Or, a collectible trading card can increase or decrease in value due to various market factors. In the example of a business product, being able to accurately predict the ideal sale price for a product can increase sales of that product. In the example of the trading card, being able to predict its monetary value at some point in time can ensure the seller has a successful auction.

Unsupervised Learning

In ***unsupervised learning***, a set of labels is not provided. Instead, the model learns by looking for distinct patterns within the data and making observations. The model might figure out how to organize related or similar items into groups. Clustering, therefore, is an unsupervised learning method.

Because there are no labels, an unsupervised model cannot evaluate its performance against any ground truth. This means it is not suitable for problems like classification and regression. In some cases, you may select unsupervised learning because you have no other choice—in other words, a robust set of labeled data is not available to you. However, unsupervised learning is sometimes the preferred approach even when you have a labeled dataset, as it can reveal relationships between data examples that supervised methods might not be able to.

Keep in mind that what makes a dataset labeled vs. unlabeled depends on what problem you're trying to solve. If you want to create a regression model that can estimate a car's fuel economy, then you need a historical dataset that has the fuel economy for each car (i.e., the label). But, you can use the same dataset (with or without fuel economy) to create clusters for each car.

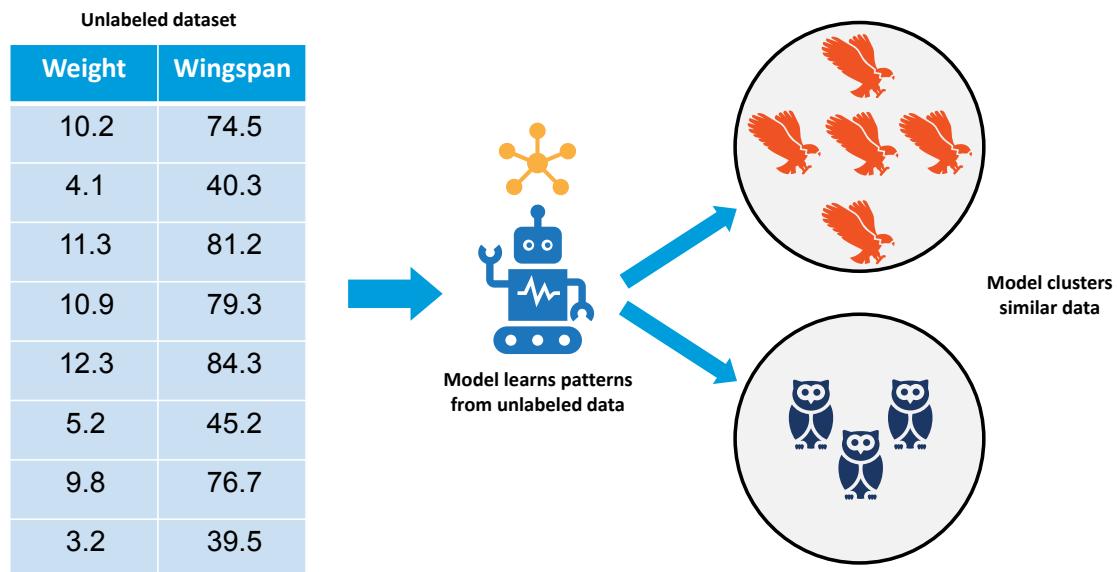


Figure 1–4: The unsupervised learning process applied an unlabeled dataset.



Note: An unsupervised model would not be able to directly label the type of bird, only divide them into clusters. The figure assumes a practitioner or domain expert has identified the type of bird in each cluster.

Examples of Unsupervised Learning Systems

The following are some examples of systems that apply unsupervised learning to solve real-world business problems.

- **Fraud detection**—In the absence of labeled fraud data, fraud detection systems can also learn to divide accounts or transactions into groups that can indicate suspicious activity. For example, most transactions might be placed into one of several well-represented groups, whereas a few outliers are in one or more small groups. These outliers might suggest fraudulent behavior since they deviate from the norm so significantly.
- **Anomaly detection**—Fraud detection is a type of anomaly detection, which can apply to more than just financial concerns. For example, cybersecurity defenses like intrusion detection systems (IDSs) can review network traffic for signs of attack. Advanced attacks can use techniques that are not yet widely known or understood, meaning there is no labeled historical data to draw from. So, an unsupervised system can reveal the presence of outlier data that could indicate a novel attack.
- **Risk management**—Unsupervised systems can also help an organization manage its risk profile. The system can learn from unlabeled datasets that track many different operational activities that contribute to an organization's risk. The system places these activities into different clusters based on their similarities, and a data scientist can use these clusters to derive insights into the likelihood and impact of risks.
- **Bioinformatics**—Biologists might not have labeled datasets to work from, or labeled datasets might not make sense for the given task. For example, protein sequences with similar structures are often involved in similar biological functions. By clustering like sequences together, biologists can more thoroughly interpret the significance of these sequences.
- **Product recommendation**—Recommendation systems that engage in collaborative or content filtering for products can use unlabeled datasets to determine what products a user is most likely to purchase if suggested to them. This can be as simple as clustering the products themselves to push recommendations for a product that is in the same cluster as a product the user purchased, or it can be a more complex analysis of user purchasing behavior.

- **Customer profiling and segmentation**—Businesses place groups of customers into different segments (clusters) so that their marketing efforts are targeted and effective. Some also segment groups of customers by constructing expressive profiles of customers, which help describe those customers in a more qualitative sense. In either case, a machine learning system can learn from a user's behavior and attributes (e.g., their age, gender, nationality, etc.) to determine how they share characteristics with other users.
- **Big data visualization**—Some organizations need to analyze incredibly large volumes of data and cannot easily do so using traditional methods. Unsupervised learning techniques can help those organizations group data into more manageable clusters. This makes it easier to not only produce visualizations from the data, but to produce the most appropriate and effective visualizations for that data cluster. The system therefore saves the data scientist a lot of hard work in wrangling the data.
- **Image analysis**—AI systems can reveal patterns in images that even a human cannot detect. Sometimes spotting a "fake" or doctored image is obvious, but other times it is convincing enough to fool people. Computers, on the other hand, excel at detecting tiny blemishes or deviations in pixel data. They can even detect these issues without any explicit labeling. An unlabeled dataset of images can exhibit certain patterns that the system places into a cluster. Then, that cluster can be analyzed further for signs of image forgery.

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 estimative power of the model.

Semi-supervised learning can be implemented using specialized algorithms and statistical methods. It can also be implemented by adopting an incremental approach to labeling unlabeled data. One of these approaches is called **self-training**. It consists of the following steps:

1. Create a baseline regression or classification model on a sample of the dataset that is actually labeled.
2. Use this baseline model to estimate labels for the portion of the dataset that is unlabeled, as well as confidence values for each estimation. These are not *true* labels, which is why they are sometimes called **pseudo-labels**.
3. Keep only the estimations that are made with high confidence, according to some threshold you define.
4. Use this dataset, which now has more "labeled" data than what you started with, to build another model. Ideally, this model will be an improvement over the baseline.
5. Repeat the process as needed.

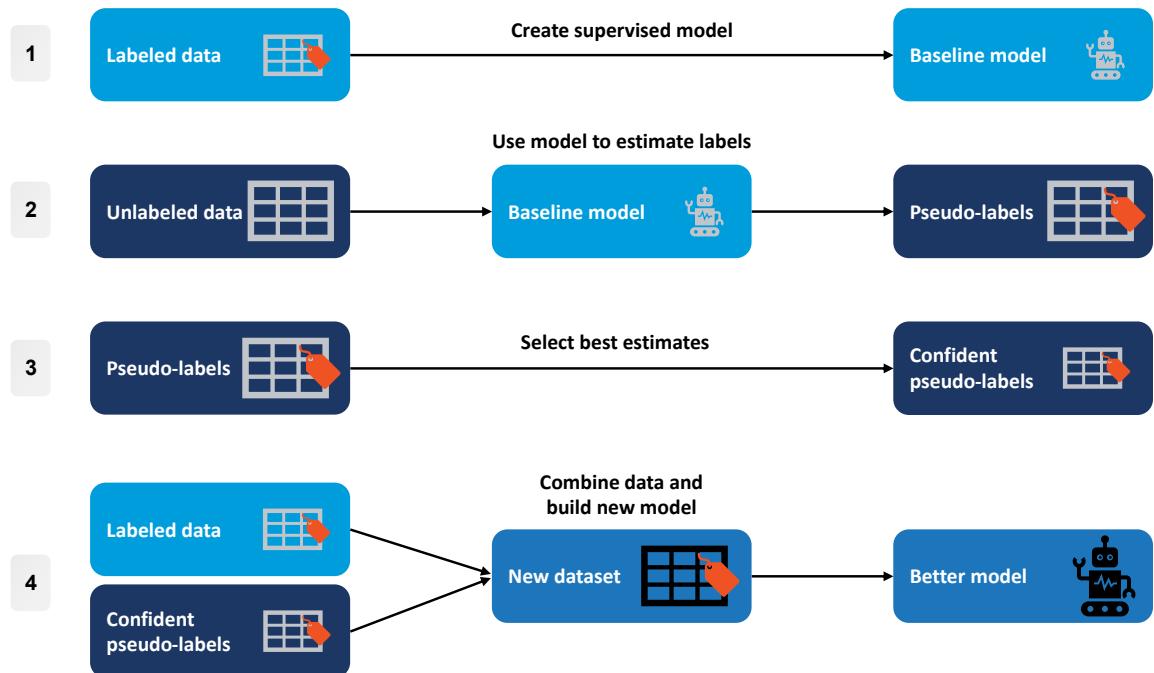


Figure 1–5: A self-training approach for semi-supervised learning.

Co-Training

Another approach to semi-supervised learning is called **co-training**. In co-training, two separate models are created from two separate portions of the labeled dataset (called *views*). Each view describes the same observations, but using different variables (features). When the unlabeled data is introduced, the confident pseudo-labels generated from one model are used to train the other model, and vice versa. Lastly, the estimations from both updated models are combined to generate a single result.

Examples of Semi-Supervised Learning Systems

The following are some examples of systems that apply semi-supervised learning to solve real-world business problems.

- **Dataset labeling**—As explained, semi-supervised techniques like self-training and co-training involve labeling a larger portion of a dataset that starts out as unlabeled. A system might be set up for this express purpose; for example, to prepare data to be handled by a different, supervised system, or to clean the data to facilitate data analysis.
- **Transaction risk assessment**—Some organizations handle thousands or millions of transactions per day, and it may only be feasible for a small proportion to be labeled as fraudulent or subject to some other kind of risk. A semi-supervised system can use this initially labeled data to gradually strengthen its estimations and reveal patterns of risk in a larger volume of transactions.
- **Document categorization**—Textual documents can take a lot of time and effort to label, especially if those documents are lengthy and numerous. The organization might contract personnel to read product reviews and label the reviews' level of satisfaction. If this only accounts for a small percentage of the total review pool, then a semi-supervised system could be used to categorize the rest of the reviews based on the original labeled portion.
- **Content identification**—Large social media companies that host user-generated content, like Twitter and YouTube, have systems in place that automatically scan newly uploaded content (e.g., text, images, and videos) for copyrighted material or material that violates the platform's terms of service. Given the sheer volume of content these sites handle on a daily basis, it's likely that only a very small portion can be labeled by a team of human reviewers. So, a semi-

supervised approach can be useful in augmenting the data to include more labeled examples of acceptable and unacceptable content.

- **Speech analysis**—Applications and devices with a voice interface must be able to derive meaning or sentiment from a user's speech. A virtual assistant, for example, might be able to answer a question asked by a user. As with written text, only a limited amount of voice data the system learns from may be labeled. Applying semi-supervised techniques can make analysis of the larger speech data more fruitful.

Reinforcement Learning

Reinforcement learning is a type of machine learning in which a software agent acts in an environment in order to obtain a reward. This involves the following steps:

1. The agent selects an action from an available set of actions.
2. The agent takes the action in the environment.
3. The environment changes state in some way.
4. The agent is rewarded or not, depending on how its action affected the state of the environment.
5. The process repeats.

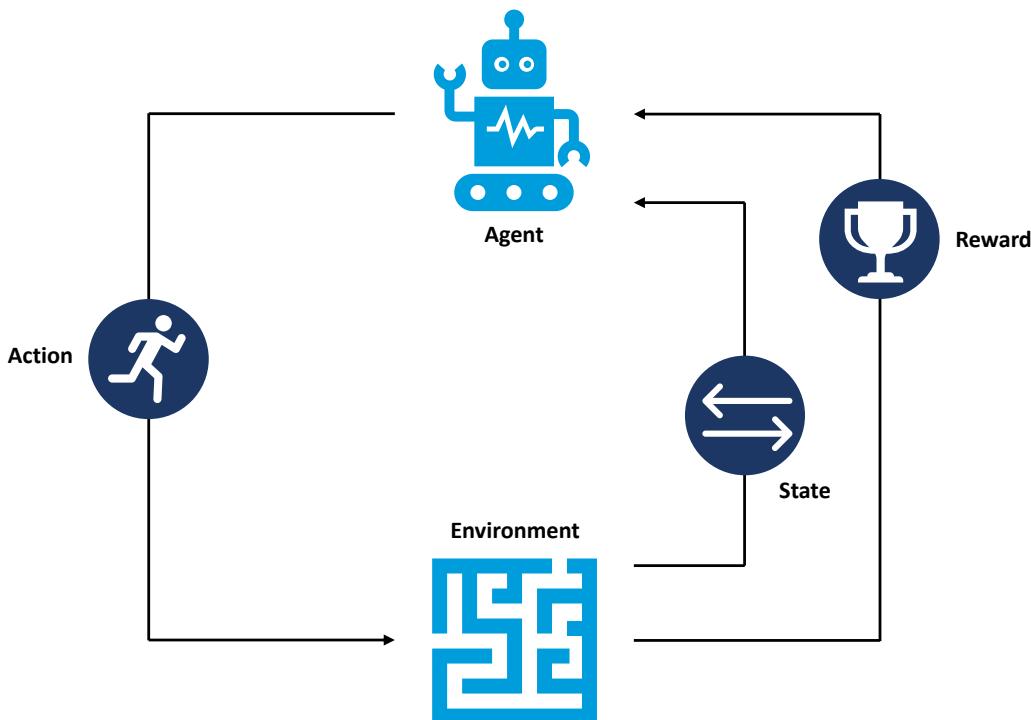


Figure 1–6: The reinforcement learning process.

Aside from the learning process, reinforcement learning differs from supervised and unsupervised learning in that there is no predefined dataset that the agent learns from all at once. Instead, the agent continually learns from the data in the environment as that environment changes state.

Ultimately, the goal of the agent is to maximize the amount of rewards it accumulates. So, it will learn which actions generate the most rewards, and thereby become better at interacting with the given environment. A reward is basically a function that indicates to the agent how it should act—i.e., what actions are desirable and worth repeating in the future. Reward functions can be programmed explicitly into the agent by a machine learning practitioner (extrinsic), or the agent can define rewards itself based on how it models the surrounding environment (intrinsic). An agent can strive for both types of rewards as well.

Examples of Reinforcement Learning Systems

The following are some examples of systems that apply reinforcement learning to solve real-world business problems.

- **Product recommendation**—In a product recommendation system based on reinforcement learning, the system itself is the agent, and the reward it is trying to maximize is user satisfaction. The environment is the collection of products that can be recommended, and the action the agent takes is to select the product from this environment that best maximizes user satisfaction. What constitutes "user satisfaction" can vary—everything from a positive written review analyzed using NLP to a simple yes or no answer to the question, "Was this recommendation helpful?"
- **Industrial automation**—Industrial robots and other machines are a common application of reinforcement learning. The machine can perform a task in a physical environment while learning which tasks are most desirable. A desirable effect might be the successful welding of one component to another, or shaping a piece of metal so that the next machine in the assembly line can work on it. The machine is trying to maximize the desirable effects so that it obtains the most rewards.
- **Vehicle automation**—An automation system based on reinforcement learning can be used to establish a self-driving mechanism in vehicles. The system is trying to maximize the safety of the vehicle and its passengers, as well as maximize its ability to travel efficiently within many different environments (road conditions, weather conditions, construction, etc.). A reward is provided to the self-driving agent so that it can learn what hazards to avoid and what routes to take.
- **Game playing**—Reinforcement learning has proven effective in developing systems that can play card games, board games, and video games. The agent obtains a reward by maximizing its chances of winning the game. It learns which moves or actions in the game strengthen its position and/or weaken its opponent's position. This results in a system that can surpass the skill of any human player. Though, some systems are tuned to be more human like in their play style so that they present a challenge to a human player rather than being impossible to win against.
- **Resource management**—Systems that manage resources can manage them more effectively when based on reinforcement learning. For example, a traffic management system might need to determine when it's best to activate and deactivate traffic lights at each segment of an intersection. A simple system might detect the presence of a car pulling up to the intersection using induction loops embedded in the road; an even simpler system might just change lights on a timer. However, a more intelligent system can analyze various factors like time of day, local traffic patterns, the presence of construction nearby, etc. This ensures that traffic flows as efficiently as possible, minimizing downtime and accidents. In a reinforcement learning context, the system tries to maximize traffic efficiency in order to obtain rewards.

ACTIVITY 1–3

Selecting Approaches to Machine Learning

Scenario

IOT Company has many potential use cases for machine learning, not just determining defects in the manufacturing process. In this activity, you'll discuss how to apply various approaches to solving multiple problems faced by the business.

1. Currently, the HVAC products that IOT Company sells are created on assembly lines, where workers at different points of the line weld the cast metal parts to the product until it gets to a finished state. The line moves relatively slowly, and the work itself is potentially dangerous, so IOT Company wants to investigate ways of automating the process.

Which machine learning approach(es) and mode(s) are best suited to solving this problem, and why?

2. Outside of the manufacturing plant, IOT Company's customer service team must address the questions and issues that customers have regarding the company's products. Right now, the team mostly communicates with customers over the phone or over live chat. The team members struggle to keep up with all of the inquiries they receive, so IOT Company wants to find some way to automate the process of providing customers the help they need.

Which machine learning approach(es) and mode(s) are best suited to solving this problem, and why?

3. On the sales side of IOT Company, the team wants to determine how well each product category is performing. They want to know how many units will be sold and how much revenue each category is likely to generate for the following month. They don't currently have a reliable way of doing this, other than educated guesses based on past sales.

Which machine learning approach(es) and mode(s) are best suited to solving this problem and why?

Summary

In this lesson, you considered AI and machine learning in a business context. You determined how they can impact, challenge, and bring value to the business. You also formulated a machine learning problem to begin addressing business needs through a data-driven approach. And, you explored real-world business applications of machine learning. By undertaking these tasks, you are much better prepared to develop an appropriate AI/ML solution.

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

What machine learning outcomes (classification, regression, clustering, etc.) might be most applicable to your projects?



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

Preparing Data

Lesson Time: 4 hours

Lesson Introduction

Once you've identified the problem you need to solve and how AI can solve it, you can begin working with the data you'll use in the machine learning process. You'll need to prepare your data in various ways before you can build effective models from it.

Lesson Objectives

In this lesson, you will:

- Collect a dataset to use for training and testing a machine learning model.
- Transform data in a dataset to eliminate or minimize data quality issues.
- Apply feature engineering to a dataset to facilitate analysis and modeling tasks.
- Apply specialized techniques to unstructured data like text documents and audiovisual files so that they are in an acceptable state for machine learning.

TOPIC A

Collect Data

There are many factors to consider when you collect 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.

The 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 Structured Query Language (SQL).

Unstructured data, on the other hand, is not as easy to query. Examples include 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, predefined containers like a spreadsheet or database. Nonetheless, unstructured data is often a very important source of information in machine learning, and can represent 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. 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 the context.

There are many alternative names for each row/record. Common ones include data example, data instance, data observation, and data point (especially when graphed). A machine learning model considers these "examples" of some aspect(s) of an environment when it takes them as input.

Those aspects of an environment are the columns, which are called **attributes** or **features**. Features contain the variables the model evaluates to make its estimations. For example, in a dataset you're using to predict the price a house will sell for, features the model uses to make this decision might include the number of bedrooms and bathrooms. These columns are said to be among the features the model uses. The total number of different features it uses are counted to identify the model's **dimensions**.

Sometimes you may refer to the individual data value held in a feature. For example, `bedrooms = 5` describes the `bedrooms` feature as having a particular value (5) for this data example.

Occasionally, the word *feature* is used to refer to this specific combination of variable plus value. In most cases, however, practitioners use the word *feature* to describe the column/variable itself, and just use the term *value* for the specific measurement.

The diagram shows a screenshot of a database table titled "users". A blue callout bubble labeled "Feature" points to the "job" column header. Another blue callout bubble labeled "Data example" points to the first row of data. A third blue callout bubble labeled "Value" points to the value "management" in the second cell of the "job" column.

	user_id	age	job	marital	education	default	housing	loan	con
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	9231c446-cb16-4b2b...	58	management	married	tertiary	no	yes	no	NUL
2	bb92765a-08de-496...	44	technician	single	secondary	no	yes	no	NUL
3	573de577-49ef-42b9...	33	entrepreneur	married	secondary	no	yes	yes	NUL
4	d6b66b9d-7c8f-4257...	47	blue-collar	married	NULL	no	yes	no	NUL
5	fade0b20-7594-4d9a...	33	NULL	single	NULL	no	no	no	NUL
6	c6aae0d4-2a86-4bac...	35	management	married	tertiary	no	yes	no	NUL
7	1fa7d4fb-3e4a-463a...	28	management	single	tertiary	no	yes	yes	NUL
8	d20059f3-84b7-4ec5...	42	entrepreneur	divorced	tertiary	yes	yes	no	NUL
9	0cedabc3-6141-43c6...	58	retired	married	primary	no	yes	no	NUL
10	bc3d8e25-4619-4395...	43	technician	single	secondary	no	yes	no	NUL
11	59b30e4f-753c-47e8...	41	admin.	divorced	secondary	no	yes	no	NUL
12	cd92181a-0e26-4a72...	29	admin.	single	secondary	no	yes	no	NUL
13	f6f04cfb...	53	technician	married	secondary	no	yes	no	NUL
14	8c406417-19a9-4c6c...	58	technician	married	NULL	no	yes	no	NUL
15	139c01b6...	57	services	married	secondary	no	yes	no	NUL
16	cf24cb61...	51	retired	married	primary	no	yes	no	NUL

Figure 2-1: The different portions of data.

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.

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
Irrelevant features	<p>Make sure the dataset includes features 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 style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  Note: Data reduction (eliminating data that doesn't provide a positive contribution to the estimation) is an important step in preparing machine learning data, to avoid unnecessary processing and ensure the best results. </div>
Non-representative data	<p>For each of the features 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> <p>On the other hand, non-representative data can lead to a model making estimations that do not properly reflect the reality of a problem domain. This can result in poor model performance within that domain, as well as bring about ethical risks.</p>
Imbalanced data	<p>Data can be imbalanced, meaning it has a disproportional frequency of certain values, particularly in a target categorical variable. Even representative data can be imbalanced, as the factors involved in real-world problems are not always evenly distributed.</p> <p>You should consider making sure that a dataset can be balanced so that a machine learning model assigns equal weights to each relevant factor. If the data were imbalanced, the model may overestimate or underestimate the importance of certain factors. The balance of data (or the lack thereof) also influences how you evaluate certain models.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  Note: Sometimes, balancing data can negatively impact how representative that data is. This is a tradeoff you'll need to be mindful of as you prepare data for training a model. </div>

Quality Issues	Description
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. This includes failing to learn patterns or just learning the wrong patterns. • Outliers—These are values outside the main distribution, deviating significantly from the rest of the values in the dataset. They may be caused by errors in measurement or execution. They can cause problems with pattern recognition, whether it's a machine learning model or a human analyst. • Noise—Some data values, features, or examples may not be necessary to make good estimations. 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 Quantity Issues

There are several factors that affect how a dataset is both understood and used. One of those factors is the quantity, or volume, of data. "Quantity" in this case can refer to either the number of data examples, the number of features, or both. So, a high-quantity structured dataset can have many rows, many columns, or many rows and columns. A high-quantity unstructured dataset, like a collection of written documents, may include many total words and many different types of words. It's important to make these distinctions because different types of quantity are relevant to different types of problems. For example, some machine learning algorithms perform better with one type over another.

Generally, the more informative data you have (i.e., the more meaningful features), the better the model it will produce. If a dataset has only two features that describe each customer, it might be difficult for the model to understand how those variables relate, and it may fail to find meaningful differences between each of the customers. On the flip side, you could have 80 columns describing your customers in a meaningful way, but if you only had 10 customers on record, your model may not have enough examples to make a proper estimation. Having a large number of examples can help to minimize the influence of a few bad data points.

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, such as the quality factors just mentioned.



Note: One rule of thumb calls for having at least 10 times as many records as the number of features that the model is using. For example, if the model is making a prediction based on 10 columns of data as input, then at least 100 records would be needed to produce a good model. Of course, more than that would be even better.

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 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.

Note that most of these sources incorporate relational databases. Relational databases are not so much a separate source as they are a technique of storing data, typically in an organized, consistent, and normalized format. Such databases usually support applications directly, like applications that process transactions.

Extract, Transform, and Load (ETL)

Extract, transform, and load (ETL) is the process of combining data from multiple sources, preparing the data, and loading the result into a destination. It is the first major phase of the machine learning process that involves hands-on practice with the data. The overall objective of ETL is to essentially take a mess and clean it up. You don't want to start creating models with data from multiple, disparate sources; nor do you want to model data that is inconsistent and has numerous issues. This will significantly impair your model's performance, slow the project down, and lead to unfulfilled business goals. Even if you think your data is clean, or if it just comes from a single source, you still need to put it through ETL to make sure it's in the best state it can be.

ETL is actually one of the most time-consuming tasks in the overall machine learning process when it comes to direct human effort. This is especially true if you're working with large volumes of data, including big data. ETL requires a careful and methodical approach to handling data since every dataset and every application of a dataset to a problem is different.

From a high level, each step of the ETL process can be described as follows:

- **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 analysis and modeling.

Guidelines for Loading Datasets

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



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

Use Python to Load a Dataset

The pandas `DataFrame()` class enables you to define a two-dimensional data structure that may contain columns of different types, similar to a spreadsheet or SQL table. It is commonly used to import structured data files (such as JSON or CSV) for use in machine learning or data analysis projects. The following are some of the functions you can use to load such a data file:

- `df = pandas.read_csv('/path/to/file/dataset.csv')` —This passes in the specified path of a CSV file to load it into the Python environment as a `DataFrame` object.
- `df = pandas.read_excel('/path/to/file/dataset.xlsx')` —Load a Microsoft Excel-compatible workbook as a `DataFrame` object.
- `df = pandas.read_pickle('/path/to/file/dataset.pickle')` —Load a pickle file, Python's binary serialization file format, as a `DataFrame` object.
- `df.info()` —Retrieve various attributes of the data frame, including the number of entries (rows), number of columns, the name of each column, and the data type of each column.
- `df.head()` —Return the first five rows of the data frame.

Ethical Considerations in Data Collection

Many aspects of the machine learning process can bring about ethical challenges. Even before you begin collecting data, you need to be aware of these challenges so that you can identify issues and address them as early in the workflow as possible. The following table discusses some of the more prominent ethical issues concerning data collection, and what impact they may have on the business.

Ethical Issue	Description	Considerations
Personally identifiable information (PII)	AI/ML projects are driven by data, and often that data involves PII, which must be protected to ensure the privacy of the people described by that data. Examples of PII include a person's name, email address, home address, Social Security number, and so forth.	Consider that PII can be used to uniquely identify, contact, or locate an individual without their consent. If the data you collect includes PII, and you don't properly secure it against unauthorized access, you could be harming the subjects of that PII. And, you may run afoul of laws and regulations that govern PII usage, like the European Union's General Data Protection Regulation (GDPR).

Ethical Issue	Description	Considerations
Data usage	Individuals or groups affected by the collection of data will want to be told upfront how you will use that data, but the reality is that you often won't know how the data will be used until you've thoroughly analyzed and modeled it.	Consider that you may not be given consent to use others' data in certain ways until you can explain how exactly that data will be used. Or, you may collect the data under the agreed-upon terms of use, but you may later want to use that data in ways that are not covered by the agreement. This may limit your ability to collect robust sets of data within the given domain.
Data quantity	As mentioned earlier, larger quantities of data can greatly improve the effectiveness of machine learning models. However, this desire for more data is often at odds with principles of privacy and security, which suggest that the less data you collect, the better.	Consider that you may only be able to collect as much data as is feasible within your organization's risk appetite. Your organization may be unwilling to collect volumes and volumes of sensitive data if it cannot guarantee the security of that data. This can hamper the modeling process.
Data bias	Some of the data you collect may exhibit the potential for one or more social biases. For example, it may have been recorded by people with certain prejudices or preconceived notions, whether conscious or unconscious. Or, the process you use to collect the data may itself be biased. For example, you may choose to filter data that indicates outliers and other unwanted aspects, but in doing so, accidentally leave out critical information. Or, the knowledge domain itself may just be lacking in sufficiently robust data.	Consider that a seemingly useful dataset may not be truly representative of the population it is purported to describe. If not examined critically, the dataset may influence decisions that could negatively affect individuals or groups of people.
Intellectual property	In some cases, you'll want to collect data that you or your organization does not own. This is especially true in the case of scientific or other highly specialized fields, where gathering data may not be feasible for the problem you're trying to solve.	Consider that you may not have the rights to view or distribute a set of data that you think would be helpful in your machine learning project. The owner of the data may be disadvantaged by your use of their data without their consent, and you may face legal repercussions.

Guidelines for Addressing Ethical Risks in Data Collection

Use the following guidelines when addressing ethical risks in data collection.

Address Ethical Risks in Data Collection

To address ethical risks in data collection:

- **PII**

- Always consider all PII to be sensitive.
- Refer to privacy guidelines for your country, municipality, or organization for specific lists of PII you may be legally required to protect.
- Use encryption to protect all collected personal data at rest and in transit.
- Ensure that collected personal information is accessible only by authorized users.
- Anonymize personal data before sharing it with a third party.

- **Data usage**

- 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).
- Inform users of all tracking mechanisms used in the project, explaining how and by whom the information is used.
- Promptly delete data that is no longer needed.

- **Data quantity**

- Minimize data collection.
- Consult with data scientists and legal and compliance teams to determine risk of data collection and storage.

- **Data bias**

- Identify the source of the data and how it was collected.
- Identify existing bias concerns with data in the knowledge domain that is relevant to your project.
- Assess your own preconceived notions and biases to see how they may impact the collection process.

- **Intellectual property**

- Research open-source datasets that may be useful for your project to avoid intellectual property issues.
- Ask the owner of proprietary data for a license to use that data in your project.

ACTIVITY 2-1

Loading the Dataset

Data File

/home/student/CAIP/Data Preparation/Data Preparation - KC Housing.ipynb

Scenario

CapitalR Real Estate company wants to use machine learning 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.



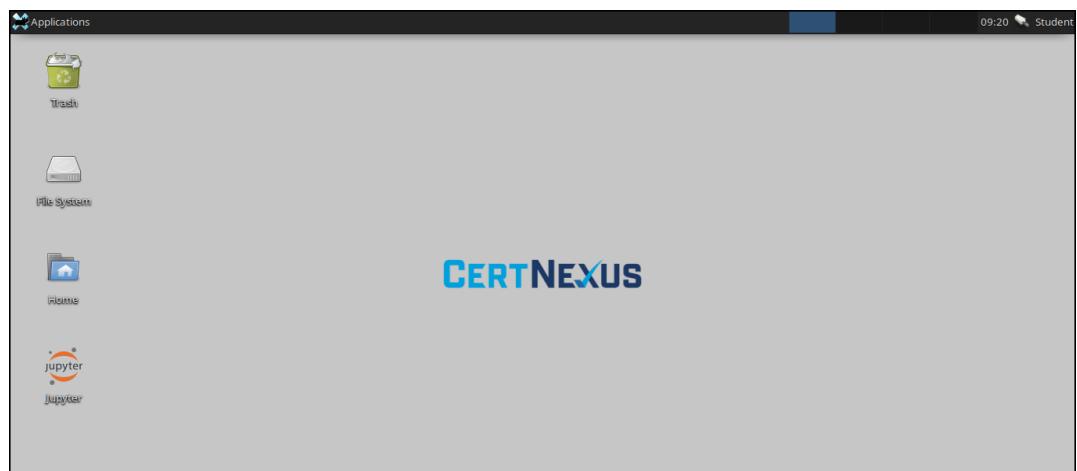
Note: The system on the VM is configured to log the user in automatically. If you are prompted at any time to log in, the account is named **student** and the password is **Pa22w0rd**.



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.

- Start the Oracle VM VirtualBox application.
- In the Oracle VM VirtualBox Manager, from the list on the left, select **CAIP-210** and select **Machine→Start→Normal Start**.
- 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 VM.
- You can use this listing to navigate to folders that contain notebooks you want to open.

- e) Select **CAIP**.



Note: Select the text "CAIP"—not the folder icon next to it.

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

- f) Select **Data Preparation**.

The **Data Preparation** subdirectory contains a subdirectory named **housing_data** and a notebook file named **Data Preparation - KC Housing.ipynb**, among others.

The screenshot shows the Jupyter Notebook interface with a title bar 'jupyter' and tabs 'Files' (selected), 'Running', and 'Clusters'. Below the title bar is a sub-navigation bar showing the current path: '/ CAIP / Data Preparation'. The main area displays a list of files and sub-directories under 'Data Preparation':

- ..
- fashion_data
- housing_data
- imdb_data
- Data Preparation - Fashion.ipynb
- Data Preparation - IMDb.ipynb
- Data Preparation - KC 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**.

	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
1	"7129300520"	"20141013T000000"	221900	3,1	1,180,5650	"1",0,0	"Fair"	7,1180,0,1955,0	"98178"	47.5112,-122.257,1340,5650											
2	"6414100192"	"20141209T000000"	538000	3,2.25	2570,7242	"2",0,0	"Fair"	7,2170,400,1951,1991	"98125"	47.21,-122.319,1690,7639											
3	"5631500400"	"20150225T000000"	180000	2,1	770,10000	"1",0,0	"Fair"	6,770,0,1933,0	"98028"	47.7379,-122.233,2720,8062											
4	"2487200875"	"20141209T000000"	604000	4,3	1960,5000	"1",0,0	"Excellent"	7,1050,910,1965,0	"98136"	47.5208,-122.393,1360,5000											
5	"1954400510"	"20150218T000000"	510000	3,2	1680,8080	"1",0,0	"Fair"	8,1680,0,1987,0	"98074"	47.6168,-122.045,1800,7503											
6	"7237550310"	"20140512T000000"	1.225e+006	4,4.5	5420,101930	"1",0,0	"Fair"	11,3890,1530,2001,0	"98053"	47.6561,-122.005,4760,101930											
7	"1321400060"	"20140627T000000"	257500	3,2.25	1715,6810	"2",0,0	"Fair"	7,1715,0,1905,0	"98003"	47.3007,-122.005,4760,101930											

- 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 the parcel/tract/lot of land that the house occupies. In this context, it is ostensibly an identifier for the house.
- 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**—Categories ranging from "Very poor" to "Excellent" describing the subjective condition of the property.
- grade**—An index from 1 to 13 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. A value of 0 means the house has not been renovated.
- zipcode**—What ZIP Code the house is located within.
- lat**—Latitude of the house's location.
- long**—Longitude of the house's location.
- sqft_living15**—The mean square footage of interior housing living space for the nearest 15 neighbors.
- sqft_lot15**—The mean square footage of the land lots of the nearest 15 neighbors.

- d) Close the **kc_house_data.csv** browser tab.

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

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

The screenshot shows the Jupyter Notebook interface. At the top, there's a header with the Jupyter logo and some navigation tabs: 'Files', 'Running', and 'Clusters'. Below the header, a message says 'Select items to perform actions on them.' Underneath this, the breadcrumb navigation bar is visible, showing the current path: a folder icon followed by '0', then a folder icon followed by 'CAIP', then another folder icon followed by 'Data Preparation' (which is highlighted with a red box), and finally a file icon followed by 'housing_data'. Below the breadcrumb, there's a list of files: a folder icon followed by 'kc_house_data.csv'.



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

- b) Select the **Data Preparation - KC Housing.ipynb** notebook label.

The **Data Preparation - KC 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 block.



Note: If line numbers are not showing along the left side of the code block, 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 frames.
5 import scipy as sp  # Perform scientific computing and advanced mathematics.
6 import sklearn       # Perform feature engineering and machine learning.
7 import category_encoders as ce # Perform data encoding on features.
8 import matplotlib   # Create charts.
9 import matplotlib.pyplot as plt
10 import seaborn as sb # Streamline charting.
11
12 # Summarize software libraries used.
13 print('Libraries used in this project:')
14 print('NumPy {}'.format(np.__version__))
15 print('pandas {}'.format(pd.__version__))
16 print('SciPy {}'.format(sp.__version__))
17 print('scikit-learn {}'.format(sklearn.__version__))
18 print('category_encoders {}'.format(ce.__version__))
19 print('Matplotlib {}'.format(matplotlib.__version__))
20 print('Seaborn {}'.format(sb.__version__))
21 print('Python {}'.format(sys.version))
```

- The `import` statements in lines 1 through 10 enable various software libraries to be used in this program.
- Lines 13 through 21 will print the names of the different libraries, along with the version number of each library installed on the computer.

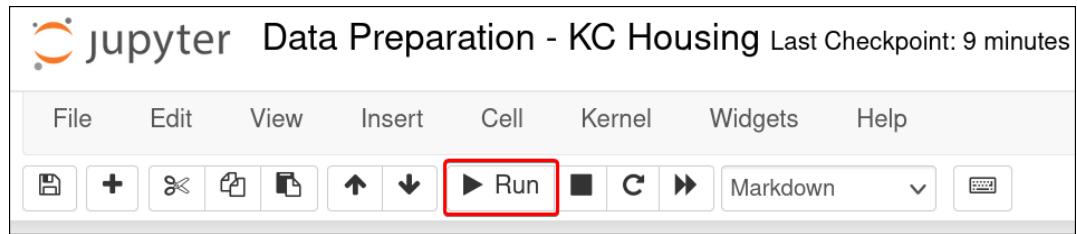
- d) Click within the first code block, as shown.

```

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 frames.
         5 import scipy as sp    # Perform scientific computing and advanced mathematics.
         6 import sklearn        # Perform feature engineering and machine learning.
         7 import category_encoders as ce  # Perform data encoding on features.
         8 import matplotlib      # Create charts.
         9 import matplotlib.pyplot as plt
        10 import seaborn as sb   # Streamline charting.
        11
        12 # Summarize software libraries used.
        13 print('Libraries used in this project:')
        14 print('- NumPy {}'.format(np.__version__))
        15 print('- pandas {}'.format(pd.__version__))
        16 print('- SciPy {}'.format(sp.__version__))
        17 print('- scikit-learn {}'.format(sklearn.__version__))
        18 print('- category_encoders {}'.format(ce.__version__))
        19 print('- Matplotlib {}'.format(matplotlib.__version__))
        20 print('- Seaborn {}'.format(sb.__version__))
        21 print('- Python {}\n'.format(sys.version))
    
```

A border is added around the cell that contains the code block, 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 1.20.3
- pandas 1.3.4
- SciPy 1.7.1
- scikit-learn 0.24.2
- category_encoders 2.5.0
- Matplotlib 3.4.3
- Seaborn 0.11.2
- Python 3.9.7 (default, Sep 16 2021, 13:09:58)
[GCC 7.5.0]
    
```

- 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.

4. Load the dataset.

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

Load the dataset

```
In [ ]: 1 DATA_PATH = os.path.join('.', 'housing_data')
2 print('Data files in this project:', os.listdir(DATA_PATH))
3
4 # Read the raw dataset as a data frame.
5 data_raw_file = os.path.join(DATA_PATH, 'kc_house_data.csv')
6 data_raw = pd.read_csv(data_raw_file)
7 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 5 constructs a text string that includes the full path and file name of the data file.
- Line 6 reads data from the data file into a pandas data frame 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 7 gets the length of (number of rows/records in) `data_raw`, and displays that number in a message.

- b) Click within the code block, and select **Run**.

- c) Observe the output.

```
Data files in this project: ['kc_house_data.csv']
Loaded 21618 records from ./housing_data/kc_house_data.csv.
```

- 21,618 records have been loaded from the `kc_house_data.csv` file.
- These records are now loaded in the `data_raw` data frame object, from which they can be displayed or manipulated through Python code.

5. Keep this notebook open.

ACTIVITY 2–2

Exploring the Dataset

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/Data Preparation/Data Preparation - KC 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 the dataset's features**. 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 what type of data preparation tasks you need to perform before you set up the model.

1. Examine the dataset's features.

- a) Scroll down to view the cell titled **Examine the dataset's features**, and examine the code listing beneath it.
Line 1 will output information about the various data types included in the dataset.
- b) Select the cell that contains the code listing, and select **Run**.

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21618 entries, 0 to 21617
Data columns (total 21 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   id          21618 non-null   int64  
 1   date        21618 non-null   object  
 2   price       21618 non-null   float64 
 3   bedrooms    21618 non-null   object  
 4   bathrooms   21618 non-null   float64 
 5   sqft_living 21618 non-null   int64  
 6   sqft_lot    21618 non-null   int64  
 7   floors      21618 non-null   float64 
 8   waterfront  21618 non-null   int64  
 9   view        21618 non-null   int64  
 10  condition   21618 non-null   object  
 11  grade       21618 non-null   int64  
 12  sqft_above  21612 non-null   float64 
 13  sqft_basement 21618 non-null   int64  
 14  yr_built    21618 non-null   int64  
 15  yr_renovated 21618 non-null   int64  
 16  zipcode     21618 non-null   int64  
 17  lat         21618 non-null   float64 
 18  long        21618 non-null   float64 
 19  sqft_living15 21618 non-null   int64  
 20  sqft_lot15  21618 non-null   int64  
dtypes: float64(6), int64(12), object(3)
memory usage: 3.5+ MB
```

- 21,618 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.
 - 6 columns contain floating point number values: `price`, `bathrooms`, `floors`, `sqft_above`, `lat`, and `long`.
 - 12 columns contain integer number values: `id`, `sqft_living`, `sqft_lot`, `waterfront`, `view`, `grade`, `sqft_basement`, `yr_built`, `yr_renovated`, `zipcode`, `sqft_living15`, and `sqft_lot15`.
 - 3 columns contain string ("object") values: `date`, `bedrooms`, and `condition`.
- The `sqft_above` column is the only one with missing values. It has 21,612 entries instead of the full 21,618.

2. Show example records.

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

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

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

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

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sc
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	
5	7237550310	20140512T000000	1225000.0	4	4.50	5420	101930	1.0	0	0	...	11	
6	1321400060	20140627T000000	257500.0	3	2.25	1715	6819	2.0	0	0	...	7	
7	2008000270	20150115T000000	291850.0	3	1.50	1060	9711	1.0	0	0	...	7	
8	2414600126	20150415T000000	229500.0	3	1.00	1780	7470	1.0	0	0	...	7	
9	3793500160	20150312T000000	323000.0	3	2.50	1890	6560	2.0	0	0	...	7	

10 rows × 21 columns

- You can scroll horizontally to see more columns in the data frame, although not all are shown.
- Even in this small sample, you can see significant variation in prices, number of bedrooms and bathrooms, living space, and so forth.

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

4. Examine descriptive statistics.

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

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

- Line 3 outputs a statistical description of the data contained within the `data_raw` data frame.
- Line 2 sets the context for the statement in line 3, 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	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	grade	sqft_above	sq
count	21618.00	21618.00	21618.00	21618.00	21618.00	21618.00	21618.00	21618.00	21618.00	21612.00	
mean	4579423916.81	540188.73	2.11	2079.94	15107.47	1.49	0.01	0.23	7.66	1788.60	
std	2876816991.64	367305.17	0.77	918.46	41417.10	0.54	0.09	0.77	1.18	828.25	
min	1000102.00	75000.00	0.00	290.00	520.00	1.00	0.00	0.00	1.00	290.00	
25%	2123049093.00	322000.00	1.75	1427.75	5040.00	1.00	0.00	0.00	7.00	1190.00	
50%	3904921185.00	450000.00	2.25	1910.00	7617.50	1.50	0.00	0.00	7.00	1560.00	
75%	7308825087.50	645000.00	2.50	2550.00	10687.75	2.00	0.00	0.00	8.00	2210.00	
max	9900000190.00	7700000.00	8.00	13540.00	1651359.00	3.50	1.00	4.00	13.00	9410.00	

The average (mean) home in this dataset has a price of \$540,188.73, 2.11 bathrooms, 2,079.94 square feet of living space, and 1.49 floors.

5. Keep this notebook open.
-

TOPIC B

Transform Data

Most of the data you collect will not automatically be in a perfect state. You'll likely need to spend time making changes to the data to address any apparent quality issues that could hinder data analysis.

Data Preparation

There are several reasons why you might want to transform data, but the two most common have to do with preparing the data and cleaning it.

Data preparation is the process of altering data so that it more effectively supports other machine learning tasks, particularly analysis and model development. Since these tasks are so vital to meeting business goals, data preparation is a necessary component of achieving success with any machine learning project. There are many individual tasks that can go into preparing data, several of which you'll learn.

Overall, the purpose of preparing data is to correct any issues that you can identify before loading data into its final destination. These issues can be at a macro level (e.g., unstructured data from one source that is poorly fit into a structured format), or they can be at a micro level (e.g., individual data values are incorrect).

Data Cleaning

Data cleaning is really a subset of preparation, and just refers to addressing inaccuracies and other problems with data. This can include duplicated data, data with the wrong data type or formatting, corrupted data, missing data, and so on. To actually "clean" the data could mean to change the offending data, or it could mean to simply remove it. Each has its benefits, and one may be more desirable or practical than the other in certain situations.

For example, if many records have the same mistaken value in the same column, it might be easy to correct that value. Removing too many records could impair the dataset. On the other hand, if only a few records have problematic values, but those values don't follow any identifiable pattern, it may be difficult to correct them. So, you might choose to drop those records instead. The choice comes down to what action you believe will be the most feasible for you to take, while minimizing any negative effects that might appear later on.

Data Wrangling

The entire process of data preparation can be tedious and may take a significant amount of 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. However, several software libraries 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.

Types of Data

When discussing the types of data, you can really be referring to one of two things: the high-level feature types, or the machine-readable data types. The former refers to the overall way that data values represent one or more observations. This has a significant impact on how you can work with that data, including how it's prepared, analyzed, and modeled. There are three major types of features:

- **Quantitative** (or numerical) data holds number values that express magnitude. For example, a Miles Driven feature for a vehicle has a magnitude since it is measuring how much of something there is.
- **Qualitative** (or categorical) data holds a value from a set of values that are typically limited. For example, the Vehicle Type feature might hold values such as sedan, pickup truck, and SUV. Note that there is no inherent ranking in categorical data—no vehicle type is "better" than the others in this example.
- **Ordinal** data is not entirely categorical since it can be ordered. But it's also not entirely quantitative since it doesn't measure magnitude. For example, a Condition feature might describe the condition of a vehicle according to some grading scale, which can be excellent, very good, good, fair, poor, or very poor. An inherent rank/order is represented (for example, a grade of "excellent" is better than a grade of "fair").

On the other hand, data types, also called data formats, refer to the ways that programming languages represent data for execution by computer hardware. These data types have a direct impact on how a program interprets data. Example data types you're probably familiar with include:

- **Integers**, such as 25.
- **FLOATS**, such as 12.78.
- **STRINGS**, such as 'Sedan'.
- **BOOLEANS**, such as True or False.
- **DATETIMES**, such as 2022-12-01.

It's important to understand these distinctions, as features and data types do not always map one to one. For example, not all categorical data is represented by a string. The feature Vehicle Type might contain the integer 1 instead of the string 'Sedan', the integer 2 instead of the string 'Pickup truck', and so on.

Operations You Can Perform on Different Types of Features

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

The following table describes the statistical measures that make sense to be performed on each type of feature.

Feature Type	Description
Quantitative	<p>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: $59 - 18 = 41$

Feature Type	Description
Qualitative	<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 ('Sedan', 'Pickup truck', 'SUV', 'SUV', 'SUV') are shown here.</p> <ul style="list-style-type: none"> • Mode: 'SUV'
Ordinal	<p>Valid measures for the class dataset ('Excellent', 'Very good', 'Good', 'Fair', 'Fair') are shown here.</p> <ul style="list-style-type: none"> • Median: 'Good' • Mode: 'Fair'



Note: Not all numerical data is quantitative. For example, Social Security numbers don't have magnitude, and it would not make sense to perform mathematical operations on such numbers.

Data Irregularities

Lower-level cleaning tasks involve addressing "irregular" data, or any data that doesn't conform to expectations. There are many types of problems that can indicate irregular data, including:

- Corrupted or unusable data.
- Incorrectly formatted data.
- Duplicated data.
- Placeholder data.
- Null or missing data.

You must identify as many instances of these problems as you can and address them as part of your overall data preparation efforts.

Identification of Corrupted or Unusable Data

Corrupted data can take many forms, so there is not necessarily one catch-all method for identifying it. Some corrupted data is more conspicuous and can be identified by the value itself. For example, if you have a feature called `Vehicle Type` and one of the values for a record is '~~XXXX~~', you can be reasonably sure that some sort of encoding error is happening since it includes replacement characters instead of an actual vehicle type. Identifying this would be easy, as you can just filter your dataset to only show non-valid vehicle types, and this will appear. You might then choose to remove the record entirely, or estimate what type is most likely based on several factors.

Another relatively obvious indication of corrupted data is when the data type for a column is something it very clearly shouldn't be. For example, in pandas, each column of a data frame must contain one data type. A column cannot include both integers and floats; only one or the other. When the data is pulled into the data frame, the parsing engine determines the data type based on the way the data appears. If all values in a column are numbers that include decimal points, the column will be cast as a float. However, if just one value deviates, the entire column could be recast. For example, if the string 'Sedan' appears just once in the `Miles Driven` column, the entire column will be strings. So, the regular data will show up as '19871', '33712', '87980', and so on. In this case, you can identify irregular data by the effect it has on data within the same context.

Some data doesn't have such overt signs of corruption, but is still relatively easy to identify. This usually happens when your knowledge of the domain helps you find anomalies. For example, if you had a dataset of major purchases made by a large company, and a feature called `Price`, a purchase price of 3 would almost certainly indicate a mistake. Perhaps the data source you pulled this info from recorded the price in millions, rather than individually. Whatever the reason, you could filter

the dataset to only show values that lie outside an acceptable range; for example, something like 1,000,000 to 10,000,000.

Unfortunately, some corrupted or unusable data can't be easily identified. Maybe one of the purchases has a Price value of 2876300—a perfectly reasonable figure, but one that was just recorded incorrectly. You can try comparing that data to other data values, like checking to see if the purchase price is greater than other, similar purchases, but you may need to do some more external research to be sure. This process of verifying individual data values isn't something you can always automate, and can quickly become tedious. If you detect a pattern of incorrect values in certain portions of the dataset (e.g., all values recorded at a certain time), then you might need to consider that entire portion unusable and drop it.

Correction of Data Formats

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 feature such as Vehicle Type might be stored as a string, an integer, or even a float. When you combine like values from multiple sources, they must use a consistent data type, one that is compatible with the working environment you're using as well as the database you plan to load that data into. These values also need to be in a format that will support both analysis and modeling of data later on.

So, as part of the data cleaning process, you'll need to inspect your dataset to identify any features whose data types need to change. As mentioned before, data objects like a data frame will attempt to determine the optimal data type to use for a column and may be forced to use an unexpected or undesired data type if just one value deviates from the norm. Even if all of the values follow the same pattern, the data type may still not be what you want. For example, it's common practice to use a decimal data type instead of a float when it comes to money. Decimals have the highest level of precision and won't introduce rounding errors. However, many tools will automatically parse any numbers with a decimal point as a float. So, you'll need to convert these values to decimal.

You could also run into issues where numbers with decimal points should have been cast as floats, but were mistakenly made integers. This is a significant loss of precision and will likely have a negative impact on the data's viability. The opposite is a little more common, however—whole numbers being parsed as floats. For example, with the feature Page Count for a dataset of books, you probably want that column to be an integer, since books are typically not printed with half pages. But many tools will automatically make any number a float, so a quantity of 250 will become 250.0. This isn't always a big deal since no precision is lost, but it can make the data harder to interpret.

Thankfully, most data science environments provide functions for easily converting data from one type to another. However, keep in mind that not all conversions are going to work. You can convert the string '1.34' to a float 1.34 because the string contains a number and only a number. But, if you try to convert the string 'two' to an integer, it's probably not going to work. However, you can convert just about any value to a string.

Date Conversion

Most data type conversion is relatively straightforward. However, dealing with dates and times deserves a special mention. In its most basic form, a date or time can be represented as a string, e.g., '2022-12-05 10:25:00' refers to December 5th, 2022 at 10:25 A.M. The problem is that there's a lot of distinct information being conveyed and no easy way to extract individual portions. What if you wanted to retrieve just the month? You could try to parse the string by extracting it from a specific position or between certain characters, but this can get tedious. It's also prone to error. After all, there are many ways to format dates and times. The method you use for parsing the prior string isn't going to work if the value is formatted as 'December 5th, 2022'.

To address these problems, most programming languages implement a special type of data type called a datetime. Each language implements it differently, but they all tend to fulfill the same

purpose: to store a date and/or time in a consistent format that is easy to extract individual components from. So, if you have a datetime variable `dt` whose value is `2022-012-05 10:25:00`, you could call `dt.month()` and it'll retrieve the month. Not only that, but datetimes are flexible, meaning you could retrieve the integer 12 directly, or you could retrieve the string 'December'.

In many cases, if you extract a raw date or time from a text file or database type that your machine learning environment doesn't understand, it will simply pull the value in as a string. However, you can also configure the environment to explicitly parse the values as a datetime. Most environments are smart enough to detect the many different ways to write a date and time, so if some sources use `YYYY-MM-DD` and other sources spell the whole thing out, you don't necessarily have to worry about making them consistent. As long as they're both recognized as dates/times and are parsed as datetimes, you should be good to go. If the environment you're pulling the data into can't automatically convert values to datetime, you may need to use other libraries to do so manually.

Deduplication

Deduplication is the process of identifying and removing duplicate entries from a dataset. Duplicate entries can lead to difficulties with interpreting the data and can impair a model's ability to learn patterns from that data. So, you must find and address them wherever they exist in the dataset.

In the most common cases, duplicated data refers to rows in a table that appear more than once when they should not. This can occur for various reasons, but is usually due to some error in recording the data or consolidating the data from multiple sources. The most obvious indication of this is when two or more rows share the same exact values for every column, like so:

Product Code	Product Name	Product Type	Release Year	MSRP
AN-R730-E	Ansible R73 Elite	Smartphone	2023	\$899.99
AN-R730-E	Ansible R73 Elite	Smartphone	2023	\$899.99
AN-R730-E	Ansible R73 Elite	Smartphone	2023	\$899.99

Most programming libraries provide functions that can automatically recognize fully duplicated rows and drop all of them except for one.

However, some duplicates might exist where only one or two values are the same across multiple rows. If at least one of those repeated values happens to be the primary key, then you know you have a duplicate. The discrepancies with the rest of the columns are likely due to recording error. It may not be clear which row is the "correct" one and which is not, so it might be safer to just drop both rows depending on how frequently they appear and how large the dataset is.



Note: Sometimes, duplication appears in columns instead of rows. You may need to remove one of the columns or consolidate the values in both columns if those values are not identical.

Missing Values

Some machine learning libraries implement algorithms that can handle some missing values on their own, but it is best if you decide exactly how to deal with missing data. If you simply ignore it, the algorithm may cope with the missing data, but the resulting model may not perform as well. When preparing 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. Too many dropped records can limit the model's effectiveness.

The simplest, but often least effective, approach would be to just fill in the missing values with a best guess or some arbitrary default. For example, if a smartphone's MSRP is missing from a product database, you might just fill it in with a value of \$500.

In some cases, the best approach may be to impute missing values. **Imputation** means providing a best estimate to fill in the missing values using statistical techniques.

Imputation Methods

There are numerous imputation strategies. Some examples are described here.

Imputation Method	Description
Mean/mode imputation	Calculate the mean or mode of all items that are not missing in that column, then use the result to fill in missing values. For example, to fill in a missing smartphone MSRP, you would just take the mean of all smartphone MSRPs minus any missing values. This approach is simple, as it preserves the value of the mean/mode and sample size. However, it may not be as good as other methods listed here.
Substitution	Use data from a new record that is not in the sample. For example, you might find another source of smartphone MSRPs that you can substitute the missing one with.
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. For example, you could find another smartphone in the dataset that has similar values for the other features, then reuse that smartphone's MSRP as the MSRP for the smartphone with the missing value.
Cold-deck imputation	Similar to hot-deck imputation, but instead of pulling from the same sample that the missing value is in, you pull from an external sample. For example, you may have access to another electronics vendor's product info, so you'll try to find a similar smartphone in that sample.
Regression imputation	Use a statistical model to identify what the missing value should be, based on data in the record. This method leverages patterns established among other records that are not missing the value to determine what the value should be. This is essentially a machine learning task that treats the record with the missing value as new data to estimate. Once the estimation is made, the record can be returned to the dataset to use on some other machine learning task.

 **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 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.

Time and Processing Power for Data Transformation

Before you start transforming your data, you need to set realistic expectations for the time it will actually take to do those transformations. After all, this step of the process is often the most time consuming; actually building a model may be comparatively quicker. So, you need to be able to manage your time effectively.

One thing you can do to manage your time is, rather than wait for the entire dataset to be transformed, take a smaller slice of the dataset, transform it, then begin preliminary analysis and generate proof-of-concept models. In the meantime, the transformation that applies to the entire dataset can proceed in the background while you get a head start on exploring the possibilities in the data.

Another common tactic is to offload as much of the manual transformation work as possible to an automated pipeline. That way, you can apply repeatable transformation tasks to new datasets as they come in, freeing you up to do other tasks that the project requires.

It's not just your *own* time that you need to manage, either. The time it takes your processing hardware (CPUs and GPUs) to apply transformations to your data is also a concern. It's less of an issue when 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 finish transforming.

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. That's why you need to work carefully with IT personnel to identify the most cost-effective solution for the needs of the business.

Guidelines for Transforming Data

Follow these guidelines when transforming data.

Transform Data

When transforming data:

- **Handle corrupt/unusable data:**
 - Correct data with inconsistent columns between records.
 - Leverage machine learning tools to handle data irregularities.
 - Use your own domain knowledge to identify data that appears to be incorrect.
 - Consider that, often, the only way to handle unusable data is to recreate it from source data.
- **Correct data formats:**
 - Keep in mind that each column in a data frame has one data type.
 - Look out for columns that have been coerced to a specific, undesired data type because of one or more improperly formatted values.
 - Consider that numeric values can take many forms in data files, including integers and floats.
 - Consider that categorical data may use strings or some numeric data type.
- **Convert dates:**
 - Consider that dates come in many formats, and that there are many formal and informal date formatting standards.
 - Keep in mind that dates formatted with the year at the end of the string can be ambiguous.
 - Consider your knowledge of the source data when deciding how to transform dates.
 - For easiest processing, normalize dates in all data sources.
- **Deduplicate data:**
 - Keep in mind that data that has been duplicated between different files can be difficult to discover and correct after it has been transformed.
 - Be systematic when collecting data; keep track of which data has been exported to prevent exporting the same data twice.
 - Deduplicate data if there is a value that is guaranteed to be unique within a data record.
 - Deduplicate data as early as possible in the process to conserve storage space and redundant processing operations.
 - Perform checks to ensure that no duplicate records exist.
- **Handle missing values:**
 - Replace missing data in continuous values with an average, zero, min, or max depending on the characteristics of a given data series.
 - Replace missing category data with the most frequently occurring category.
 - Create a new "missing" category if many records are missing a category.
 - Consider removing a record if there are a number of features missing.

- Drop columns missing values for at least 70% of the records.

Use Python to Transform Data

The pandas library has several functions for performing some of the transformation tasks discussed in this topic, including:

- `pandas.to_numeric('10')` —Convert a non-numeric data type to a numeric data type, if possible.
- `pandas.to_datetime('December 12, 2022 10:25 AM')` —Convert a string with a date and/or time into a proper datetime format. This function can recognize and interpret many different ways of formatting dates and times.
- `pandas.DataFrame.duplicated(df)` —Return a series of Booleans that indicate which row numbers or labels in a data frame (`df`) have duplicate values. By default, for any duplicate rows, the first occurrence is `False` and all other occurrences are `True`.
- `pandas.DataFrame.isna(df)` —Return a data frame of Booleans that indicates which data values are formatted as a missing data type, such as `None`.
- `pandas.DataFrame.notna(df)` —Return a data frame of Booleans that indicates which data values are *not* formatted as a missing data type.
- `pandas.DataFrame.fillna(df, value = 10)` —Fill in any missing values in the data frame with the integer 10.

ACTIVITY 2–3

Transforming Data

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/Data Preparation/Data Preparation - KC 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 **Convert the bedrooms feature to an integer**. Select **Cell→Run All Above**.

Scenario

You've explored the real estate dataset enough to get a good sense of how you'll start cleaning it. You'll apply several transformation techniques to the data, including data type conversion, deduplication, missing data imputation, and dropping outliers.

1. Convert the `bedrooms` feature to an integer.

- Scroll down and view the cell titled **Convert the bedrooms feature to an integer**, and examine the code listing below it.

Recall that the `bedrooms` feature was read as a string object. This feature indicates the number of bedrooms in the house, so it should be numeric. The line of code uses a simple regular expression to look for any data values for `bedrooms` that do not contain digits.

- Run the code cell.
- Examine the output.

	<code>id</code>	<code>date</code>	<code>price</code>	<code>bedrooms</code>	<code>bathrooms</code>	<code>sqft_living</code>	<code>sqft_lot</code>	<code>floors</code>	<code>waterfront</code>	<code>view</code>	<code>...</code>	<code>grade</code>
8103	7461420230	20150325T000000	336500.0	four	1.75	1760	7268	1.0	0	0	...	7
10746	6450302545	20150508T000000	443000.0	three	1.00	1280	5460	1.5	0	0	...	7
12168	1446404015	20140620T000000	200000.0	two	1.00	860	6600	1.0	0	0	...	6
13502	7942600975	20140512T000000	505000.0	four	1.75	1940	4800	1.0	0	0	...	7
14792	4167700210	20140826T000000	240000.0	three	1.75	1520	9600	1.0	0	0	...	8

5 rows × 21 columns

Five records have `bedrooms` values that are causing the entire column to be cast as a string. Instead of numbers, they have the written forms of those numbers. You'll need to deal with them.

- Scroll down and examine the next code cell.

```
1 # Handle irregular number formats.
2 data_raw['bedrooms'].mask(data_raw['bedrooms'] == 'two', '2', inplace = True)
3 data_raw['bedrooms'].mask(data_raw['bedrooms'] == 'three', '3', inplace = True)
4 data_raw['bedrooms'].mask(data_raw['bedrooms'] == 'four', '4', inplace = True)
```

These lines use the pandas `mask()` function to replace the offending string values with their numeric equivalents.

- Run the code cell.

There is no output for this cell.

- f) Scroll down and examine the next code cell.

```
1 data_raw['bedrooms'] = data_raw['bedrooms'].astype(int)
2 data_raw.dtypes
```

This code converts `bedrooms` to an integer type, now that the offending string values have been replaced.

- g) Run the code cell.
h) Examine the output.

id	int64
date	object
price	float64
bedrooms	int64
bathrooms	float64
sqft_living	int64
sqft_lot	int64
floors	float64
waterfront	int64
view	int64
condition	object
grade	int64
sqft_above	float64
sqft_basement	int64
yr_built	int64
yr_renovated	int64
zipcode	int64
lat	float64
long	float64
sqft_living15	int64
sqft_lot15	int64
dtype:	object

The `bedrooms` column is now showing as an int64 type, as expected.

2. Format the `date` column as a datetime type.

- a) Scroll down and view the cell titled **Format the date column as a datetime type**, and examine the code cell below it.

This code will retrieve the `date` values for the first five records.

- b) Run the code cell.
c) Examine the output.

0	20141013T000000
1	20141209T000000
2	20150225T000000
3	20141209T000000
4	20150218T000000
	Name: date, dtype: object

The date appears to be formatted as the year, month, day, and then a timestamp, all without spaces. However, this is a string and not an actual datetime object. Converting it to a datetime will make it easier to work with.

Also, if you were to look at the entire dataset, you'd see that the timestamp for every record is T000000. Therefore, you can just drop that part of the date before you do the conversion.

- d) Scroll down and examine the next code cell.

```

1 # Remove unused timestamp portion.
2 split = data_raw['date'].str.split('T', n = 1, expand = True)
3 data_raw['date'] = split[0]
4 data_raw['date']

```

- Line 2 splits the date on the letter "T" as this begins the timestamp.
 - Line 3 sets `date` equal to the first portion of the split (without the timestamp).
- e) Run the code cell.
f) Examine the output.

```

0      20141013
1      20141209
2      20150225
3      20141209
4      20150218
...
21613   20140521
21614   20150223
21615   20140623
21616   20150116
21617   20141015
Name: date, Length: 21618, dtype: object

```

The result is a set of dates for which the timestamp has been removed.

- g) Scroll down and examine the next code cell.

```

1 data_raw['date'] = pd.to_datetime(data_raw['date'],
                                    format = '%Y%m%d')
2
3 data_raw['date'].head(5)

```

Lines 1 and 2 do the actual conversion to a datetime format.

- h) Run the code cell.
i) Examine the output.

```

0 2014-10-13
1 2014-12-09
2 2015-02-25
3 2014-12-09
4 2015-02-18
Name: date, dtype: datetime64[ns]

```

The data type for `date` is now `datetime64[ns]`, and the components of the date were automatically divided using hyphens.

3. Identify duplicate rows.

- a) Scroll down and view the cell titled **Identify duplicate rows**, and examine the code cell below it.
The `duplicated()` function identifies fully duplicated records.
b) Run the code cell.

- c) Examine the output.

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_ab
6875	567000660	2014-12-04	425000.0	4	2.00	1490	5300	1.0	0	0	...	7	111
6876	567000660	2014-12-04	425000.0	4	2.00	1490	5300	1.0	0	0	...	7	111
12651	809001520	2014-11-05	1850000.0	4	3.25	3480	6000	3.0	0	0	...	8	333
12652	809001520	2014-11-05	1850000.0	4	3.25	3480	6000	3.0	0	0	...	8	333
12653	809001520	2014-11-05	1850000.0	4	3.25	3480	6000	3.0	0	0	...	8	333
15323	321059132	2015-04-27	365000.0	3	1.75	1450	61419	1.0	0	0	...	8	111
15324	321059132	2015-04-27	365000.0	3	1.75	1450	61419	1.0	0	0	...	8	111

7 rows × 21 columns

There are seven rows that include duplicated records. Each row has the same values in every column as at least one other row. This was likely due to some error in the data collection process. In any case, you'll remove the duplicates and leave only one copy.

4. Remove duplicate rows.

- a) Scroll down and view the cell titled **Remove duplicate rows**, and examine the code cell below it.
- Line 1 updates the data frame to remove duplicates.
 - Line 2 selects records with an ID matching one of the duplicates. This will verify whether or not the duplicate was removed.
- b) Run the code cell.
- c) Examine the output.

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_ab
6875	567000660	2014-12-04	425000.0	4	2.00	1490	5300	1.0	0	0	...	7	111

1 rows × 21 columns

As expected, there is only one result for the record, indicating the duplicate was removed.

5. Identify other duplicated data.

- a) Scroll down and view the cell titled **Identify other duplicated data**, and examine the code cell below it.

The `duplicated()` function, by default, only identified *full* duplicates. However, there may still be duplicates that erroneously share the same key value, yet have different values for the other columns.

This code provides `id` as the `subset` argument to the `duplicated()` function. In other words, `id` is acting as a key column.

- b) Run the code cell.

- c) Examine the output.

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft
93	6021501535	2014-07-25	430000.0	3	1.50	1580	5000	1.0	0	0	...	8	
94	6021501535	2014-12-23	700000.0	3	1.50	1580	5000	1.0	0	0	...	8	
313	4139480200	2014-06-18	1384000.0	4	3.25	4290	12103	1.0	0	3	...	11	
314	4139480200	2014-12-09	1400000.0	4	3.25	4290	12103	1.0	0	3	...	11	
324	7520000520	2014-09-05	232000.0	2	1.00	1240	12092	1.0	0	0	...	6	
...
20675	8564860270	2015-03-30	502000.0	4	2.50	2680	5539	2.0	0	0	...	8	
20784	6300000226	2014-06-26	240000.0	4	1.00	1200	2171	1.5	0	0	...	7	
20785	6300000226	2015-05-04	380000.0	4	1.00	1200	2171	1.5	0	0	...	7	
21585	7853420110	2014-10-03	594866.0	3	3.00	2780	6000	2.0	0	0	...	9	
21586	7853420110	2015-05-04	625000.0	3	3.00	2780	6000	2.0	0	0	...	9	

355 rows × 21 columns

There are 355 records that share an `id` value with some other record. This is not necessarily a mistake, however. The same house may have been sold multiple times.

- d) Scroll down and examine the next code cell.

```
1 data_raw[data_raw.duplicated(['id', 'date'], keep = False)]
```

What would be erroneous, however, is if the same house is sold on the same date. That's why this line of code uses both `id` and `date` as duplication subsets.

- e) Run the code cell.
f) Examine the output.

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft
16511	1423400225	2014-07-21	225000.0	2	1.0	1030	9192	1.0	0	0	...	6	1
16512	1423400225	2014-07-21	385000.0	3	2.0	1380	7650	1.0	0	0	...	4	1

2 rows × 21 columns

Two rows share the same `id` and `date`. At least one of these records must have been recorded with the wrong value of either column. It should be safe to just drop one of them.

6. Remove the duplicated data.

- a) Scroll down and view the cell titled **Remove the duplicated data**, and examine the code cell below it.
- Line 2 removes the duplicates with matching `id` and `date`.
 - Line 3 selects the relevant ID to verify that its duplicate was removed.
- b) Run the code cell.
c) Examine the output.

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft
16511	1423400225	2014-07-21	225000.0	2	1.0	1030	9192	1.0	0	0	...	6	1

1 rows × 21 columns

As expected, there is only one instance of this ID and date.

- d) Scroll down and examine the next code cell.

```
1 data_raw.shape
```

This code will verify the shape of the data frame now that duplicates have been removed.

- e) Run the code cell.
f) Examine the output.

```
(21613, 21)
```

There are now 21,613 rows instead of 21,618. There are still 21 columns.

7. Identify missing data.

- a) Scroll down and view the cell titled **Identify missing data**, and examine the code cell below it.
This code uses `isna()` to find the number of missing values in each column.

- b) Run the code cell.
c) Examine the output.

```
id          0
date        0
price       0
bedrooms    0
bathrooms   0
sqft_living 0
sqft_lot    0
floors      0
waterfront  0
view        0
condition   0
grade       0
sqft_above  6
sqft_basement 0
yr_built    0
yr_renovated 0
zipcode     0
lat         0
long        0
sqft_living15 0
sqft_lot15  0
dtype: int64
```

The `sqft_above` column has six missing values.

- d) Scroll down and examine the next code cell.

```
1 missing = data_raw[data_raw['sqft_above'].isna()]
2 missing
```

This code will print the rows with missing data.

- e) Run the code cell.

- f) Examine the output.

floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15
1.0	0	0	...	8	NaN	290	1939	0	98117	47.6870	-122.386	1570
1.0	0	0	...	8	NaN	0	1984	0	98052	47.6601	-122.135	1510
1.5	0	0	...	6	NaN	850	1918	0	98178	47.5094	-122.263	1910
2.0	0	0	...	7	NaN	0	2000	0	98108	47.5440	-122.296	1490
1.0	0	0	...	7	NaN	0	1951	0	98108	47.5442	-122.297	980
2.0	0	0	...	7	NaN	0	1999	0	98065	47.5261	-121.827	1500

By scrolling to the right, you can see the NaN values for sqft_above.

8. Impute missing values for sqft_above.

- a) Scroll down and view the cell titled **Impute missing values for sqft_above**, and examine the code cell below it.

There are many potential ways you could impute the missing values. Here, the code is using mean imputation—specifically, the mean percentage decrease from sqft_living to sqft_above. The above-ground living space will always be less than the entire living space, so the average of this difference can approximate what the former is when given the latter.

- Line 2 calculates the decrease.
 - Line 3 gets the mean of this decrease, minus any missing values.
 - Lines 5 and 6 impute the mean for the missing values.
 - Line 7 converts the column to an integer.
- b) Run the code cell.
c) Scroll down and examine the next code cell.

```
1 data_raw.loc[missing.index]
```

This code will print the rows with missing values again.

- d) Run the code cell.
e) Examine the output.

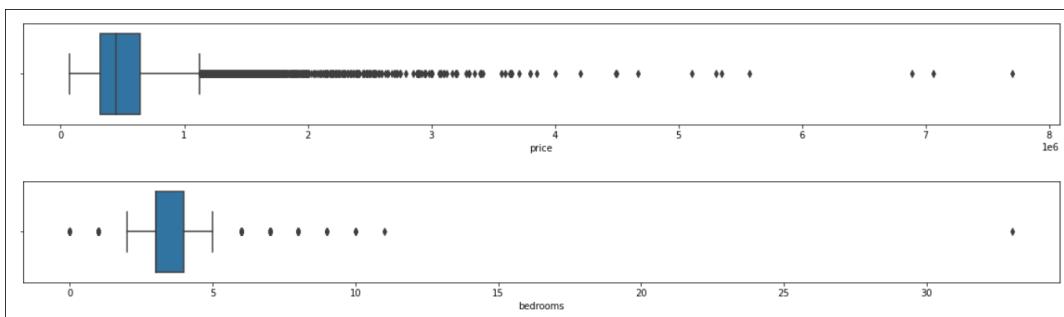
sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat
1580	5000	1.0	0	0	...	8	1383	290	1939	0	98117	47.6870
1600	16510	1.0	0	0	...	8	1400	0	1984	0	98052	47.6601
1780	5336	1.5	0	0	...	6	1558	850	1918	0	98178	47.5094
1460	3044	2.0	0	0	...	7	1278	0	2000	0	98108	47.5440
1450	5456	1.0	0	0	...	7	1269	0	1951	0	98108	47.5442
1470	4675	2.0	0	0	...	7	1286	0	1999	0	98065	47.5261

As you can see, the missing values have been imputed for sqft_above.

9. Identify outliers.

- a) Scroll down and view the cell titled **Identify outliers**, and examine the code cell below it.
- Line 1 iterates through the price and bedrooms features, repeating the steps in lines 2 and 3 for each feature.

- Line 2 specifies the figure size available to the box plot, and line 3 actually generates it, specifying the name of the feature to be plotted, the data frame that holds the data, and formatting options for the box plot.
- b) Run the code cell.
c) Examine the output.



- In both box plots, 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 ($Q_1 - 1.5 \times IQR$) and maximum ($Q_3 + 1.5 \times IQR$) for the distributions.
- The upper box plot 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 box plot 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.

10. Examine data values in the outliers.

- a) Scroll down and view the cell titled **Examine data values in the outliers**, and examine the code cell below it.
- This code will return records where the price of the house is above \$6,000,000.
- b) Run the code cell.
c) Examine the output.

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_living15
3914	9808700762	2014-06-11	7062500.0	5	4.50	10040	37325	2.0		1	2	...	11
7253	6762700020	2014-10-13	7700000.0	6	8.00	12050	27600	2.5		0	3	...	13
9255	9208900037	2014-09-19	6885000.0	6	7.75	9890	31374	2.0		0	4	...	13

3 rows × 21 columns

Three houses have sold for more than \$6,000,000.

- d) Scroll down and examine the next code cell.

```
1 # Houses with more than 11 bedrooms.
2 data_raw[data_raw['bedrooms'] > 11]
```

This code will return records where the number of bedrooms is greater than 11.

- e) Run the code cell.

- f) Examine the output.

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft
15874	2402100895	2014-06-25	640000.0	33	1.75	1620	6000	1.0	0	0	...	7	

1 rows × 21 columns

One house has more than 11 bedrooms.

11. Drop outliers from the dataset.

- a) Scroll down and view the cell titled **Drop outliers from the dataset**, and examine the code cell below it.
- Line 4 filters the training dataset to include only records whose price is less than \$6,000,000.
 - Line 8 is similar to line 4, removing any records for houses with more than 11 bedrooms.
 - The print statements on lines 1, 5, and 9 provide a before-and-after summary so you can see the results of dropping the outliers.



Note: Because the dataset is comparatively large, these outliers may not have much impact on the model, but it won't hurt to remove them from the training set either.

- b) Run the code cell.
c) Examine the output.

```
21613 houses in the dataset before dropping outliers.
21610 houses remain after dropping those priced over $6M.
21609 houses remain after dropping those with more than 11 bedrooms.
```

After dropping the 4 outliers, 21,609 records remain in the dataset.

12. Keep this notebook open.

TOPIC C

Engineer Features

The transformation methods discussed in the previous topic are usually preliminary steps within the larger data preparation process. One of the more advanced and involved steps is to ensure data is ready to be used to train a machine learning model, particularly when it comes to the features. In this topic, you'll round out your data preparation skills by engineering features in a dataset.

Data Preprocessing

Data preprocessing is the task of applying various transformation and encoding techniques to data so that it can be interpreted and analyzed by a machine learning algorithm. The "pre" part of preprocessing implies that the data is being prepared, similar to how ETL prepares data at the early stages. The difference is that, in preprocessing, much of the preliminary transformation is already done, and you're really just focusing on facilitating the machine learning process. Preprocessing is important because machine learning algorithms all have different sets of challenges and requirements, so you need to make sure your data can accommodate them. If you think of your machine learning project as building toward the creation of a model, then preprocessing is the final stepping stone.

There are several techniques that fall under the definition of preprocessing, including:

- Scaling variables using transformation functions.
- Removing unnecessary or unhelpful features.
- Engineering new features.



Note: Some practitioners consider handling missing values to be part of preprocessing, whereas others choose to handle missing values early in the ETL process.

Feature Engineering

Feature engineering is the preprocessing technique of generating and extracting features from data in order to improve the ability for a machine learning model to make estimations. The features you currently have are not necessarily the best possible features for the problem you're trying to solve. By creating new, more useful features, you have a better chance of producing a model that excels at its given task.

Feature engineering is really an umbrella term for multiple types of tasks that can achieve this goal. Several of these tasks will be discussed shortly, including:

- Scaling features.
- Encoding categorical data.
- Binning continuous variables.
- Splitting features.
- Selecting and extracting features to reduce dimensionality.

Before you start engineering new features, you should make sure you've completed some of the earlier transformation tasks that were mentioned, particularly handling missing data and removing duplicate values. You should determine what features can be outright removed from the dataset because they are either redundant or have no estimative power. For example, `Product Code` might make a good primary key in a relational table, but an estimative model is probably not going to learn much from it. Once these preliminary tasks are complete, you can move on to feature engineering.

Feature Scaling

Machine learning algorithms find patterns in data using different approaches. In some cases, the distribution of the data is a primary factor, and the actual magnitude of values is either secondary, unimportant, or actively harmful to the process. So, you may need to apply *scaling* functions to your numeric variables in order to emphasize their distribution, while de-emphasizing the differences in scales. Otherwise, one feature could exert more influence than another simply because it deals with numbers on a larger scale. Scaling is particularly important when using distance-based algorithms like k -nearest neighbor and support-vector machines (SVMs), whereas tree-based algorithms like decision trees and random forests don't require features to be scaled.

Imagine you have a dataset of employees with two features: `Salary` and `Years of Service`. The problem is, the former has numbers that probably extend into the hundreds of thousands, whereas the latter will have a maximum value in the tens. Both of these features could be equally useful to a machine learning task, say if you wanted to determine which employees are most likely to leave the company in the next year. But, since the features are on wildly different scales, distance-based algorithms might treat `Salary` as being much more important. You therefore need to ensure the algorithm sees each feature in terms of distribution of values. The two major approaches for this are normalization and standardization.



Note: You scale data used to train the algorithm, but you also use those same scaling parameters (e.g., the mean of a training feature) to scale the equivalent test data.

Normalization

Normalization involves transforming a feature such that the lowest value is a 0, and the highest value is a 1. So, whatever employee has the lowest salary will have a 0 in place of whatever the true value is. And, the employee with the highest salary will have a 1 instead of the true value. The same exact principle applies to the employees' years of service as well, and any other numeric feature that you believe needs to be scaled. Now the machine learning algorithm will see the distribution of these features rather than the absolute values.

The equation for normalizing a data value is as follows:

$$x' = \frac{x - \min(X)}{\max(X) - \min(X)}$$

Where:

- x is the initial value being normalized.
- x' is the new normalized value.
- $\min(X)$ is the smallest value in the feature that x is taken from.
- $\max(X)$ is the largest value in the feature that x is taken from.

Normalization tends to be useful when the raw data doesn't follow a normal distribution and you want to use the data with an algorithm that doesn't assume any particular type of distribution. It's also good at minimizing the effect of outliers.



Note: The ' $'$ symbol is the prime symbol.

Standardization

Standardization also provides scaling, but does so in a different way. It calculates a 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. You can obtain the z-score for any employee's salary, years of service, etc. Such scores are standardized so that each feature has a mean value of 0 and a standard deviation of 1. Once again, this emphasizes the feature's distribution instead of its absolute scale.

The equation for standardizing a data value is as follows:

$$x' = \frac{x - \mu}{\sigma}$$

Where:

- x is the initial value being standardized.
- x' is the new standardized value.
- μ is the mean of all values in the feature that x is taken from.
- σ is the standard deviation of all values in the feature that x is taken from.



Note: σ is the lowercase Greek letter sigma (also called "little sigma"), and μ is the Greek letter mu.

Since standardization has no upper or lower bounds, it doesn't minimize outliers in the same way as normalization. It's often useful for when the raw data is already normally distributed and you just want to ensure two features are on the same scale.

Additional Transformation Functions

There are other ways to scale features besides normalization and standardization. Some additional transformation functions that can scale data are described in the following table.

Transformation Function	Description
Log	<p>This function calculates the log base 10 of x, log base e of x ($\ln(x)$), or log base 2 of x, where x is a data example.</p> <p>The log transformation function helps to reduce skewness (particularly positive skewness) in non-normally distributed datasets, making them approximate a normal curve. Note that it can only be used on positive numbers.</p>
Cube root	<p>This function raises each data example to a power of $1 / 3$.</p> <p>The cube root function helps reduce positive skewness as well, but not as strongly as a log function. However, unlike log, it can be used on negative numbers and zero, in addition to positive numbers.</p>

Transformation Function	Description
Box–Cox	<p>This function raises each data example to a power of some lambda (λ) value, subtracts 1, then divides that result by λ. This calculation happens for all values of λ (usually between -5 and 5) until the transformed data approximates a normal distribution. Note that whenever λ is 0, the function simply takes the log of the data example.</p> <p>Box–Cox, therefore, also helps to reduce skewness and obtain normally distributed data. It leverages both log transformations and power transformations. There is also a version that can be used to transform negative values.</p>



Note: These are just a few common examples. Depending on the circumstances, various other functions may be appropriate, such as sigmoid, hyperbolic tangent, reciprocals, Yeo–Johnson, O'Brien, and so forth.

Data Encoding

Data encoding is the process of converting data of a certain type into a coded value of a different type. In machine learning, this typically means taking a string of text used in a categorical variable and converting it into a number. The resulting number becomes a new feature within the dataset, and the original feature is either removed or ignored when a model is built.

Data encoding is important because many machine learning algorithms simply can't handle categorical data that is represented in strings. For example, let's say you have a `Color` feature that has the possible values `['Red', 'Green', 'Blue']`. This is a categorical variable and should be treated as such by the algorithm. But rather than feed those strings directly to the algorithm, you'll likely need to convert them to numbers first. However, keep in mind that encoding a categorical variable does not mean that it becomes a numerical feature. The feature is still categorical, even though it is being represented using a number.

Data Encoding Methods

There are various methods for converting categorical features to numbers. Some of the most common encoding methods are listed in the following table.

Encoding Method	Description
Label encoding	<p>Label encoding translates each unique value in a label (the feature you're interested in studying) into separate numbers. So, <code>Color</code> Encoded would have <code>[0, 1, 2]</code> as possible values, where 0 is red, 1 is green, and 2 is blue.</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. That's why it's also called ordinal encoding. In this example, a machine learning algorithm might perceive that blue (with a value of 2) is ranked the highest, which may not be what you intended.</p>

Encoding Method	Description
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.</p> <p>With this method, you create a dummy column for each class of a categorical feature. For example, you might create three columns named <code>IsRed</code>, <code>IsGreen</code>, and <code>IsBlue</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 (red, green, or blue, in this case) more value than the others.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <p>Note: One-hot encoding is sometimes conflated with the very similar technique of dummy encoding. However, dummy encoding creates $n - 1$ columns, whereas one-hot encoding creates n columns, where n is the number of unique values in the categorical variable.</p> </div>
Binary encoding	<p>This encoding scheme converts each categorical value into an ordinal integer, then converts that into binary. Each binary digit becomes a new column. The number of binary digits needed to represent each ordinal integer—and consequently, the number of resulting encoding columns—is $\log_2(n)$ (rounded up), where n is the number of unique categories. Some information is lost in this conversion, but it's more efficient than one-hot encoding because with binary encoding, you don't need one column for each possible value. So, binary encoding is preferred when the number of possible categorical values is high.</p>
Effect encoding	<p>This is similar to binary encoding, except the encoded values are either -1, 0, or 1. Effect encoding is a more advanced technique and is less common.</p>
Frequency 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>With this encoder, each class is encoded as a function of the mean of the target variable (i.e., the label in supervised learning). So, the target variable must be numeric, whereas the variable being encoded is categorical.</p>
Hash encoding	<p>This scheme 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 or thousands of categories.</p>
Base-N encoding	<p>This encoding scheme chooses a number base to convert the categorical values into. Base-1 encoding is the same as one-hot encoding, and base-2 encoding is the same as binary encoding. Otherwise, this particular encoding style confers no additional advantages.</p>

Continuous vs. Discrete Variables

A **continuous variable** is a quantitative variable whose values are uncountable and can extend infinitely within a certain range. Consider a person's age as a feature called `Age`. Based on a target demographic, you might constrain this feature to be between 20 and 29. So, for example, you could

describe someone as being 27 years old. If you were to demonstrate the possible range of values visually on a timeline, it might look like something in the following figure.

Age (Continuous)

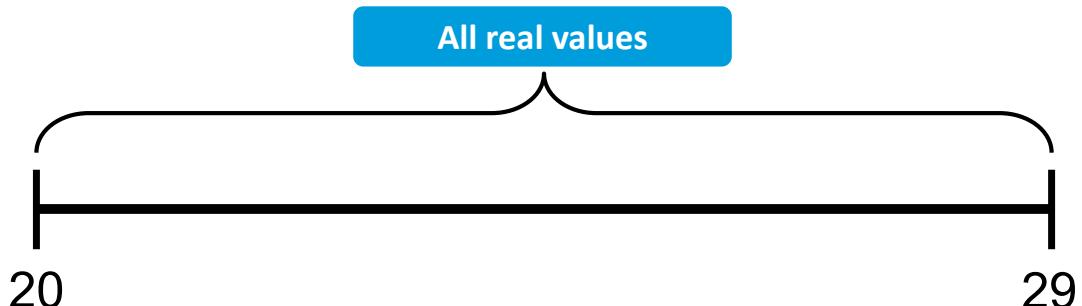


Figure 2–2: A continuous range of values.

This range includes all possible values because there is an infinite number of values between those two points. You could divide the time up into smaller and smaller units—months, weeks, days, hours, minutes, seconds, milliseconds, and so on. You could describe someone's age as 23.4 years, or 23.45 years, or 23.456 years, etc.

Certain analysis methods and machine learning 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 that uses an integer data type is a discrete variable because it is either 20, 21, 22, 23, etc., and cannot be divided into more precise values. As opposed to continuous variables, discrete variables are typically represented as whole numbers.

Observe how this discrete variable creates a fixed number of possible values on the same timeline:

Age (Discrete)

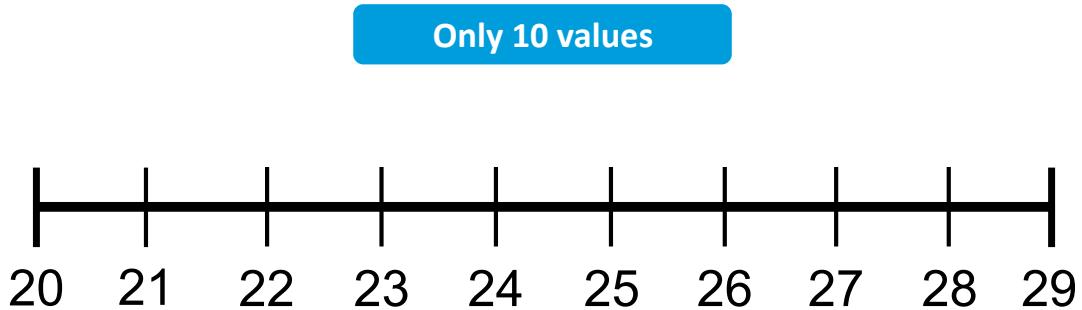


Figure 2–3: A discrete set of values.

The difference between continuous and discrete variables influences how data is treated in multiple phases of the machine learning process, including data transformation and preprocessing. For example, continuous variables are typically formatted as floats, whereas discrete variables are typically formatted as integers.

Continuous Variable Discretization

Some algorithms, like decision trees, have a difficult time working with continuous variables because the tree can just keep splitting over and over again until it becomes too large and inefficient. If you've identified useful features in your dataset that have continuous variables, you may need to engineer discrete features out of them before you input the data to an algorithm.

The process of converting a continuous variable into a discrete variable is called **discretization**. This can greatly simplify and improve the performance of your models. 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**.

Let's say you change the `Age` feature to more accurately reflect the entire adult population. So, this new feature, if kept discrete, would have 112 different possible values (i.e., between 18 years old and 129 years old). So, you could place a person's age into one of those 112 different bins. Now that the feature is discrete, machine learning algorithms like decision trees will be able to handle it.

Bin Determination

Binning requires selecting the number of bins in which to place the values, as well as the range of values between each of the bins—a bin's "width." 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. The 112 bins for `Age` are likely too many regardless. A good rule of thumb is to have no more than 20 bins. So, you might instead transform the feature so that a person's age is either: 18–24, 25–34, 35–44, 45–54, 55–64, 65–74, or 75+ (7 bins).

Or, you might choose a smaller number of bins based on some wider time interval. As the number of bins increases, so too does the complexity of the model, so keep that in mind. Also, no matter how you bin a variable, some amount of information and precision will be lost as compared to the original continuous variable.

As for determining bin width, there are two primary approaches: equal width and equal frequency. In equal-width binning, each bin spans the same range of values. For example, if you specify 8 bins for ages between 18 and 129 years, each bin will have a range of 14 years. The range between minimum (18) and maximum (129) values is simply divided by the number of bins. This approach is sensitive to outliers in the data, and 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, and it is also less sensitive to outliers.

Binning of a feature where the classification label is known (i.e., in supervised learning) 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 feature 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.

Feature Splitting

In some cases, a feature in your dataset might benefit from being partitioned into two or more features. For example, let's say a `Product_Name` feature contains both a family name of the product (such as `Ansible`) and a specific model identifier (such as `R73`). So, one of the values might be '`Ansible R73`'. If variations in product names have an impact on whatever problem your model is trying to solve (e.g., determining name marketability), then this data might be more useful if family names and model identifiers were split into their own separate features. That way the model can

learn from both of these name types independently rather than considering them as just a singular piece of information.

Feature splitting is particularly common with text-based data like names, locations, identifiers, titles—any string that can be compounded with contextually related strings. The choice of whether or not to split such a string will be primarily informed by domain knowledge. Perhaps splitting `Name = 'Mrs. Alma Shepard'` into `Honorific = 'Mrs.'`, `First = 'Alma'`, and `Last = 'Shepard'` will lead to better estimations, or perhaps not. You should consider experimenting with analysis and modeling to see how different splits produce different results.

Feature splitting is also common with date and time values. For example, you might want to take `date = 2022-12-05` and turn it into separate features for `Year`, `Month`, and `Day`. This is much easier if your date feature is already cast as a datetime type—you can simply retrieve each component using a datetime function like `month()`. If it's a string, you'll need to use string splitting functions like `split()`, which can be prone to error if the dates are formatted in different ways.

The Curse of Dimensionality

Adding to a dataset's feature space (also called its dimensionality), like when you split a single string into multiple strings, can make a model more effective. The model is given more potentially useful features to learn from, after all.

Unfortunately, this is not always the case. Some features may be redundant or make the learning process too noisy, which can complicate the analysis and model-building processes and have a negative impact on performance. If the number of data examples stays constant, then at some point, adding to a dataset's dimensionality will actually start reducing the model's ability to learn useful patterns from the data. This is called the *curse of dimensionality*.

Dimensionality Reduction

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 input 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.



Note: The term "dimensionality reduction" can refer to the general process of reducing a dataset's features, but it can also be used to refer to specific algorithms that transform data from high dimensions to lower dimensions. Unless specified, this course uses "dimensionality reduction" in the general sense.

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 learns from 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 which team is going to win a championship, the feature `Win Percentage` probably correlates highly with the features `Points Scored` and `Points Scored Against`, so the three can be merged into one derivative feature that represents the team's overall game-by-game performance. 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.

Dimensionality Reduction Algorithms

You can select and extract features manually, but this becomes tedious in high-dimensional datasets. It's also prone to error, as you might not be making optimal decisions about what to reduce and how. Thankfully, there are many ways to automate the dimensionality reduction process. The following table provides an overview of some of the most common dimensionality reduction algorithms.

Dimensionality Reduction Algorithm	Description
Principal component analysis (PCA)	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 linear variance in the dataset, while dropping the features that contribute very little to the variance.
Singular value decomposition (SVD)	SVD is similar to PCA; both of them decompose matrices of feature values in order to select a subset of features that better represent the data for modeling and analysis. In fact, many implementations of PCA actually use SVD to decompose matrices. SVD tends to work well on sparse matrices (i.e., matrices with mostly zeros).
<i>t</i> -distributed stochastic neighbor embedding (<i>t</i> -SNE)	<i>t</i> -SNE generates probability distributions of data pairs in high dimensions. Similar data examples are given a higher probability, whereas dissimilar examples are given a lower probability. Then, the process is repeated in lower dimensions until the divergence between higher and lower dimensions is minimized. Unlike PCA, <i>t</i> -SNE can retain non-linear variance.
Random forest	Random forests are a type of machine learning algorithm that is commonly used in classification and regression tasks, but they're also effective at feature selection. They can help you determine which features have the largest influence on the model's estimations by returning the weight of each feature as a percentage (out of 100). The higher the percentage, the more important a feature is to the estimator.
Forward feature selection	This algorithm starts a model with no features from the dataset, then iteratively adds features until the performance of the model stops improving.
Backward feature elimination	This algorithm starts a model with all features from the dataset, then iteratively removes features until the performance of the model stops improving.
Factor analysis	Factor analysis measures a latent variable (i.e., a variable that is not directly observed but which is identified through the relationship of other variables). These latent variables are selected as new features because they may exert more influence over a model than the observable variables.

Dimensionality Reduction Algorithm	Description
Missing-value ratio	This algorithm removes any features that have a total number of missing values that exceed some predefined threshold. Missing values don't improve a model's performance, and, in some cases, they can actually hinder its performance. You can adjust the threshold to either increase or decrease the number of features that get removed.
Low-variance filter	This algorithm removes any features that have a total variance lower than some predefined threshold. Low variance means the feature values don't change much between each data example, which doesn't help a model learn. As with missing-value ratio, you can adjust the threshold as needed.
High-correlation filter	This algorithm calculates the correlation coefficient between pairs of features and removes one of those features if the coefficient exceeds a predefined threshold. Correlated features provide similar information to a model, so they may be redundant for the learning process. Once again, you can tune the threshold as desired.

Guidelines for Engineering Features

Follow these guidelines when engineering features.

Engineer Features

When engineering features:

- **Scale features**
 - Identify if scaling is required for the model or algorithm used.
 - Test different scaling transformations with the model or algorithm used.
 - Use normalization when different variables need to be between a given range.
 - Use standardization to preserve distributions when comparing variables with very different scales.
- **Encode features**
 - Convert categories with an inherent, natural ordering to a sequence using ordinals.
 - Maintain a lookup table to generate the category names from the values when reporting or plotting.
 - Use one-hot encoding when an algorithm calls for a vector of categories for each record.
 - Consider hashing labels when there are a large number (100+) of categories.
- **Discretize features**
 - Convert continuous variables to discrete ordinals for models or algorithms that cannot handle discrete data or that train faster with a limited number of ordinals for a variable.
 - Determine how many bins are required based on the use of the data. Most algorithms work better with 5 to 10 bins.
 - Use a uniform distribution when the shape and other characteristics of the distribution should be preserved.
 - Recognize that the loss in precision is often offset by the gain in training speed and/or performance of a model.
- **Split features**
 - Split on punctuation if available in the field.

- Split on whitespace only when data has regular patterns, and each split item does not have embedded spaces.
- Retain only the new fields that are required, such as country code and area code when splitting a phone number.
- Process features after splitting by cleaning up or applying transformation functions again as necessary.
- **Reduce feature dimensionality**
 - Remove a variable that has a high correlation with another variable.
 - Alternatively, join multiple variables that are closely correlated into a single variable with hyphens or spaces.
 - Review the impact to results of removed or reduced variables, as the impact is not always predictable.
 - Try different dimensionality reduction algorithms for a collection of variables.

Use Python to Engineer Features

Several Python libraries, including the default library, pandas, and scikit-learn, have functions for performing some of the feature engineering techniques discussed in this topic, including:

- `scaler = sklearn.preprocessing.MinMaxScaler()` —Create an object that will be used for normalization, also called min–max scaling.
- `scaler = sklearn.preprocessing.StandardScaler()` —Create an object that will be used for standardization
- `scaler.fit_transform(data)` —Return the scaled values of all input data based on an existing scaler object.
- `pandas.get_dummies(data)` —One-hot encode categorical data and automatically create a data frame of those encoded values in dummy columns.
- `binner = sklearn.preprocessing.KBinsDiscretizer(n_bins = 4, encode = 'ordinal', strategy = 'uniform')` —Create an object that will place values into the number of specified bins (4 in this case) using an ordinal integer to identify each bin (i.e., starting at 0 and incrementing by 1). The 'uniform' strategy is equal-width binning, whereas you can use 'quantile' for equal-frequency binning.
- `binner.fit_transform(data)` —Return the binned values of all input data based on an existing binner object.
- `name.split(separator = ',')` —Split a string called name into an array of strings based on a separator. The default separator is whitespace.
- `pca = sklearn.decomposition.PCA(n_components = 2)` —Create an object that will perform principal component analysis (PCA) on a dataset and return two features.
- `pca.fit_transform(data)` —Return the reduced feature values from the input data based on an existing pca object.

Ethical Considerations in Feature Engineering

Just as collecting data can bring about ethical concerns, so too can transforming data through feature engineering. Many of the issues are the same, but need to be considered in a different context. You should be aware of these issues before you commit to making major changes to your data.

Ethical Issue	Considerations
PII	<p>In some cases, techniques like dimensionality reduction can actually help anonymize personal data. But feature engineering can also have the opposite effect, where making changes can expose more about the subject than the raw dataset could.</p>
	<p>For example, splitting features can reveal quasi-identifiers, which are data values that do not directly contain PII, but may be used with other data values to identify an individual. A database query based on a combination of values such as ZIP Code, age, and gender might be used to identify a particular individual.</p>
Data usage	<p>It's difficult to know exactly how a feature will end up until preprocessing is done and you're ready to train a model. You may have changed a feature to the point where it's unrecognizable to most people. There may be questions surrounding the usefulness of the data and whether it might end up producing a bad model.</p>
	<p>For example, you might neglect to scale financial data properly for certain algorithms, which can lead to models making major financial decisions that turn out to be wrong.</p>
Data bias	<p>Much of the tasks involved in feature engineering require you to have at least some preconceived notions about the problem domain. This can lead to an unwarranted sense of "comfort" surrounding the data and how best to transform it, giving way to certain blind spots.</p>
	<p>For example, you may want to bin a feature like people's ages to simplify the model, but the bins you choose may reflect stereotypical attitudes about marketing demographics rather than real-world circumstances.</p>
	<p>Likewise, you may decide to extract a new feature from existing features, but someone could argue that the new feature doesn't adequately convey the significance of the existing features it's derived from. In other words, you may be removing important information.</p>
Proxies for larger social discrimination	<p>There are various attributes of people that are considered protected in some way, such as gender, race, sexual orientation, and religion. It is important to bear in mind that attributes other than these may function as a proxy for those attributes. How you handle them during the feature engineering process can raise ethical concerns.</p>
	<p>For example, certain geographic locations (represented by postal codes, for example) may serve as a proxy for race, religion, national origin, or even age—if members of those categories are predominant in those locations. Modifying these proxy features could unduly influence attitudes surrounding the features they are proxies of.</p>

Guidelines for Addressing Ethical Risks in Feature Engineering

Use the following guidelines when addressing ethical risks in feature engineering.

Address Ethical Risks in Feature Engineering

To address ethical risks in feature engineering:

- **PII**
 - Ensure privacy policies, terms, and conditions are clear.

- Leverage the inherent ability for some feature engineering techniques to anonymize or obscure data.
 - Make sure it is not easy to reverse the transformation of sensitive data.
 - Be on the lookout for quasi-identifiers in engineered data and how they could be used to violate personal data protections.
- **Data usage**
 - Before committing to a change of some feature, envision how the resulting model might be affected differently if you hadn't made the change, or made some other change.
 - If possible, be transparent with stakeholders regarding how the appearance of data may change during the feature engineer process, but its meaning is essentially the same.
 - Always maintain the original, raw data in case you need to revert a change.
 - **Data bias**
 - Leverage your domain knowledge as much as you can, but be careful about relying too much on preconceived notions.
 - Exercise critical thinking regarding the meaning of a feature, its impact on the problem at hand, and the consequences of transforming it.
 - Assume that each machine learning problem requires a fresh approach to feature engineering, even if the dataset is similar to one you've worked on in the past.
 - Aside from maintaining the raw data, make sure you retain the meaning of any features from which you derive new features during the extraction process. Feature names often need to be terse to work well in a programming environment, but this has the side effect of making it difficult to know exactly what the feature is meant to describe.
 - **Proxies for larger social discrimination**
 - Identify how certain features in a dataset may be used as proxies for larger social discrimination.
 - Consider that some proxy features could be removed, since they may not be adding much useful information to the problem at hand.
 - Research other projects that have faced similar issues, and determine how they addressed proxy features.

ACTIVITY 2–4

Engineering Features

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/Data Preparation/Data Preparation - KC 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 **Label encode the condition feature**. Select **Cell→Run All Above**.

Scenario

You've cleaned your data using several different techniques. But the features of the dataset still need work. You need to make sure that any categorical features are properly encoded and that continuous variables are discretized. It's also worth addressing the fact that many of the features are on different scales. And, you may want to simplify the dataset by reducing its dimensionality.

All of these preprocessing techniques are meant to enhance the modeling process, once you're ready to do that.

1. Label encode the condition feature.

- Scroll down and view the cell titled **Label encode the condition feature**, and examine the code cell below it.

There are several categorical features in this dataset, but most of them have already been encoded for you.

- `waterfront` has a binary encoding—either a 0 or a 1 for each. This is because the feature indicates the presence or absence of something.
- `grade` and `view` have been label encoded because they are ordinal features. The higher the grade number and view number, the better the house's construction/design and quality of view, respectively.

However, `condition` uses words to describe the condition of the house. So, this code counts the instances of each descriptive condition.

- Run the code cell.
- Examine the output.

Fair	14029
Good	5678
Excellent	1700
Poor	172
Very poor	30
Name: condition, dtype: int64	

There are five different conditions: `Very poor`, `Poor`, `Fair`, `Good`, and `Excellent`. You can safely assume that the feature is ordinal, and that this is the proper order.

Also, the counts of each condition are printed. There are many more fair houses than houses of any other condition. There are only 30 houses in very poor condition.

- d) Scroll down and examine the next code cell.

```

1 cond_map = [{col: 'condition',
2               'mapping': {
3                 'Very poor': 1,
4                 'Poor': 2,
5                 'Fair': 3,
6                 'Good': 4,
7                 'Excellent': 5
8               }
9             }]
10
11 encoder = ce.OrdinalEncoder(return_df = True, mapping = cond_map)
12 data_raw = encoder.fit_transform(data_raw)

```

- Lines 1 through 9 create a dictionary that maps each word-based condition to an integer. The better the condition, the higher the number.
- Line 11 uses the `category_encoders` library to create an ordinal encoder—another name for a label encoder.
- Line 12 performs this encoding transformation on the dataset.

- e) Run the code cell.

- f) Scroll down and examine the next code cell.

```
1 data_raw['condition'].value_counts()
```

- g) Run the code cell.

- h) Examine the output.

```

3    14029
4    5678
5    1700
2     172
1      30
Name: condition, dtype: int64

```

The same counts appear, but this time showing the encoded values.

2. Examine the distributions of `yr_built` and `yr_renovated`.

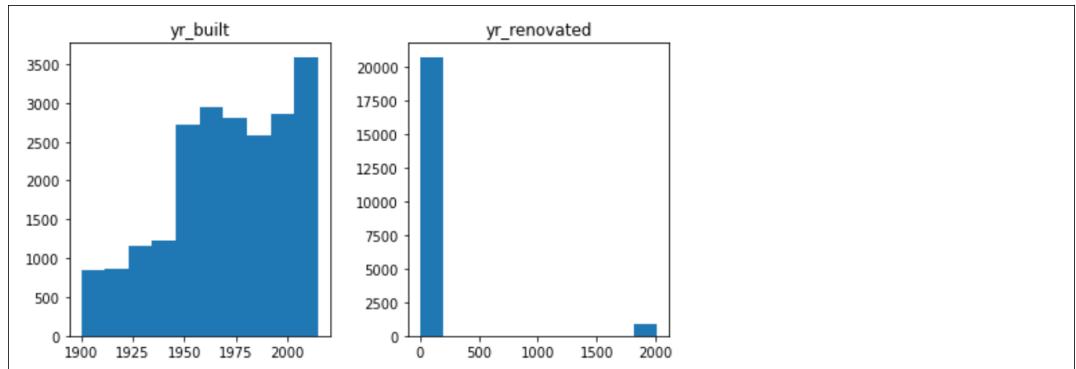
- a) Scroll down and view the cell titled **Examine the distributions of `yr_built` and `yr_renovated`**, and examine the code cell below it.

It's likely that the year values follow a continuous pattern. You might want to bin these values so that you're capturing them as eras, rather than as individual years. This will help simplify the data.

Before you do this, you should look at how these features are distributed, which is what the code in this cell does.

- b) Run the code cell.

- c) Examine the output.



- The `yr_built` histogram shows a distribution that has a left skew. More houses were built in recent years than in the distant past.
 - The `yr_renovated` histograms shows a distribution that is highly right skewed. The overwhelming majority of houses have not been renovated, so they are marked as a 0. This might prompt you to bin `yr_renovated` differently.
- d) Scroll down and examine the next code cell.

```

1 # Exclude zeros.
2 no_zeros = data_raw['yr_renovated'] > 0
3 print('Earliest built: {}'.format(data_raw['yr_built'].min()))
4 print('Latest built:   {}'.format(data_raw['yr_built'].max()))
5 print('-----')
6 print('Earliest renovation: {}'.format(data_raw['yr_renovated'][no_zeros].min()))
7 print('Latest renovation:   {}'.format(data_raw['yr_renovated'][no_zeros].max()))

```

This code will identify the earliest and latest year values for both features. Line 2 removes values with 0 so that you can see the true range of `yr_renovation` values.

- e) Run the code cell.
f) Examine the output.

```

Earliest built: 1900
Latest built:   2015
-----
Earliest renovation: 1934
Latest renovation:   2015

```

- The earliest a house was built was in 1900, and the latest in 2015.
- The earliest a house was renovated was in 1934, and the latest in 2015.

3. Bin the `yr_built` feature.

- a) Scroll down and view the cell titled **Bin the `yr_built` feature**, and examine the code cell below it.
- Lines 2 and 3 create the year bins. This is where having domain knowledge will come in handy, as you can use that to inform how you create the bins. For example, the housing market in King County might have followed specific trends over certain year ranges. In the absence of such knowledge, you will bin in increments of 20 years for a total of 6 bins.
 - Lines 7 and 8 use the `cut()` function to apply the bins to the existing data.
 - Line 9 makes sure the binned column is next to the original one.
 - Line 12 creates an encoded version of the binned column so that it's more conducive to machine learning.
 - Line 13 inserts the encoded column after the binned column.
- b) Run the code cell.

- c) Scroll down and examine the next code cell.

```
1 data_raw[['yr_built', 'yr_builtin_group', 'yr_builtin_encoded']].head(10)
```

- d) Run the code cell.
e) Examine the output.

	yr_built	yr_builtin_group	yr_builtin_encoded
0	1955	1941–1960	2
1	1951	1941–1960	2
2	1933	1921–1940	1
3	1965	1961–1980	3
4	1987	1981–2001	4
5	2001	2001–2021	5
6	1995	1981–2001	4
7	1963	1961–1980	3
8	1960	1941–1960	2
9	2003	2001–2021	5

The original, binned, and encoded versions of the `yr_built` column are shown. Notice that more recent bins receive higher encoded numbers.

4. Bin the `yr_renovated` feature.

- a) Scroll down and view the cell titled **Bin the `yr_renovated` feature**, and examine the code cell below it.

This code is similar to the code that binned `yr_built`, this time applied to `yr_renovated`. The major difference is the bins themselves. There are 5 bins instead of 6, they begin with N/A to indicate no renovation, and they start with 1934 for actual renovations. The gap between bins is still 20 years.

- b) Run the code cell.
c) Scroll down and examine the next code cell.

```
1 data_raw[['yr_renovated', 'yr_ren_group', 'yr_ren_encoded']].head(10)
```

- d) Run the code cell.
e) Examine the output.

	yr_renovated	yr_ren_group	yr_ren_encoded
0	0	N/A	0
1	1991	1975–1994	3
2	0	N/A	0
3	0	N/A	0
4	0	N/A	0
5	0	N/A	0
6	0	N/A	0
7	0	N/A	0
8	0	N/A	0
9	0	N/A	0

As before, the binned and encoded columns are shown with the original.

5. Compare the scales of the quantitative features.

- a) Scroll down and view the cell titled **Compare the scales of the quantitative features**, and examine the code cell below it.

You might have noticed earlier, but many of the quantitative features in the dataset are on different scales. This code will demonstrate that by showing their descriptive statistics.

- b) Run the code cell.
c) Examine the output.

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	sqft_basement	sqft_living15	sqft_lot15
count	21609.00	21609.00	21609.00	21609.00	21609.00	21609.00	21609.00	21609.00	21609.00
mean	3.37	2.11	2078.73	15105.03	1.49	1787.51	291.22	1986.28	12767.04
std	0.91	0.77	912.87	41423.79	0.54	824.49	441.83	684.96	27305.74
min	0.00	0.00	290.00	520.00	1.00	290.00	0.00	399.00	651.00
25%	3.00	1.75	1425.00	5040.00	1.00	1190.00	0.00	1490.00	5100.00
50%	3.00	2.25	1910.00	7617.00	1.50	1560.00	0.00	1840.00	7620.00
75%	4.00	2.50	2550.00	10685.00	2.00	2210.00	560.00	2360.00	10083.00
max	11.00	8.00	13540.00	1651359.00	3.50	9410.00	4820.00	6210.00	871200.00

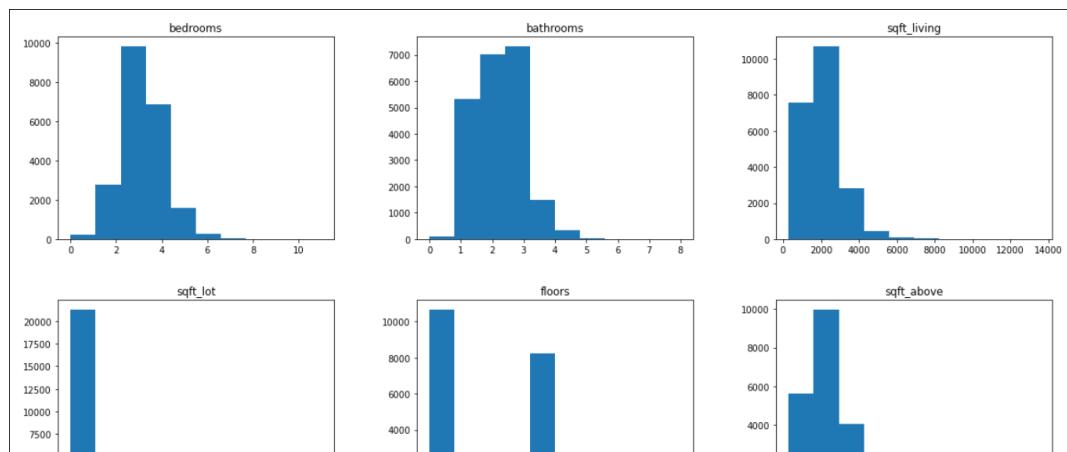
A feature like `bedrooms` has a minimum of 0 and max of 11, whereas a feature like `sqft_living` has a minimum of 290 and a maximum of 13,540. Because some algorithms will be affected by differences in scales, you may want to ensure these features are on the same scale so as to emphasize the distribution of their values, and not the absolute magnitude.

6. Compare the distributions of the quantitative features.

- a) Scroll down and view the cell titled **Compare the distributions of the quantitative features**, and examine the code cell below it.

This code will show histograms of the quantitative features.

- b) Run the code cell.
c) Examine the output.



Most of these features are right skewed, with fewer values at the higher ends. Since you want to make sure the scaling technique you apply will maintain the features' distributions, you'll use this as a point of comparison later.

7. Normalize the quantitative features.

- a) Scroll down and view the cell titled **Normalize the quantitative features**, and examine the code cell below it.

- Line 3 creates a copy of the dataset. You'll store the normalized values in a separate data frame so that you can more easily work with those values separately when you start training machine learning models.
- Line 4 creates a scaler object using `MinMaxScaler()`, also called normalization.
- Line 5 applies the scaler to the dataset's quantitative features.



Note: In most cases, you don't need to scale the dependent variable. That's why `price` is not included in this scaling operation.

- Run the code cell.
- Scroll down and examine the next code cell.

```
1 with pd.option_context('float_format', '{:.2f}'.format):
2     display(data_norm[quant_vars].describe())
```

- Run the code cell.
- Examine the output.

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	sqft_basement	sqft_living15	sqft_lot15
count	21609.00	21609.00	21609.00	21609.00	21609.00	21609.00	21609.00	21609.00	21609.00
mean	0.31	0.26	0.13	0.01	0.20	0.16	0.06	0.27	0.01
std	0.08	0.10	0.07	0.03	0.22	0.09	0.09	0.12	0.03
min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	0.27	0.22	0.09	0.00	0.00	0.10	0.00	0.19	0.01
50%	0.27	0.28	0.12	0.00	0.20	0.14	0.00	0.25	0.01
75%	0.36	0.31	0.17	0.01	0.40	0.21	0.12	0.34	0.01
max	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

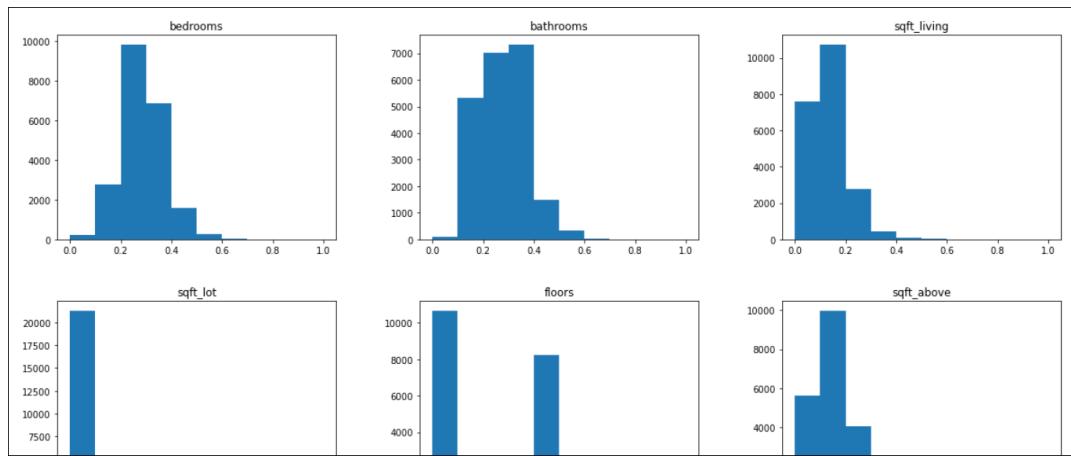
Each quantitative feature now has a minimum of 0 and a maximum of 1. Therefore, they are all on the same scale.

- Scroll down and examine the next code cell.

```
1 # Verify that distributions haven't changed.
2 data_norm[quant_vars].hist(figsize = (20, 15), grid = False);
```

- Run the code cell.

- h) Examine the output.



Despite the change in scale after normalization, the distributions of each feature have not changed.



Note: In some cases, you may actually want to alter the distributions of each feature. For example, you could apply a transformation so that each feature approximates a bell-shaped normal distribution curve.

8. Perform PCA to reduce the dimensionality of the quantitative features.

- Scroll down and view the cell titled **Perform PCA to reduce the dimensionality of the quantitative features**, and examine the code cell below it.
 - Line 4 creates a PCA object that will calculate two new features derived from some set of features. This effectively reduces the dimensionality of the dataset.
 - Line 6 fits the PCA transform to the quantitative data.
 - Line 8 creates a new data frame to hold these PCA features.
- Run the code cell.
- Scroll down and examine the next code cell.

```
1 data_reduced = pd.DataFrame(reduced, columns = ['PCA1', 'PCA2'])
2 data_reduced
```

Each of the derived features won't necessarily have a qualitative meaning, so this code assigns them the generic labels `PCA1` and `PCA2`.

- Run the code cell.

- e) Examine the output.

	PCA1	PCA2
0	-0.268815	-0.086649
1	0.176238	-0.091541
2	-0.235679	-0.009263
3	-0.188635	0.105507
4	-0.190766	0.021193
...
21604	0.487795	-0.353714
21605	0.207574	-0.075495
21606	0.029207	-0.342203
21607	0.145388	-0.180666
21608	0.029201	-0.342235
21609 rows × 2 columns		

All of the rows still represent the houses, but the columns have changed. The nine quantitative features have been reduced to only two. The numeric values in these columns were mathematically derived from the values that were in the nine quantitative columns. The idea is that these two columns now represent an approximation of the initial feature space. The relationships between the records and the features are mostly preserved, but are now in a more compact form. The values may not have much meaning to a human observer, but they can still be meaningful to a machine learning algorithm.

9. Save the datasets.

- a) Scroll down and view the cell titled **Save the datasets**, and examine the code cell below it.

This code adds the PCA features to the normalized set of features. If you planned on creating a model from the PCA features, you'd use them by themselves as input. For convenience, they're all being rolled up into one data frame.

- b) Run the code cell.
c) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21609 entries, 0 to 21608
Data columns (total 27 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   id               21609 non-null   int64  
 1   date              21609 non-null   datetime64[ns]
 2   price             21609 non-null   float64 
 3   bedrooms          21609 non-null   float64 
 4   bathrooms         21609 non-null   float64 
 5   sqft_living       21609 non-null   float64 
 6   sqft_lot          21609 non-null   float64 
 7   floors             21609 non-null   float64 
 8   waterfront        21609 non-null   int64  
 9   view              21609 non-null   int64
```

PCA1 and PCA2 have been added to the end of the data frame.

- d) Scroll down and examine the next code cell.

```
1 # Serialize data frames to preserve integrity.  
2 data_raw.to_pickle('kc_house_data_prep.pickle')  
3 data_norm.to_pickle('kc_house_data_prep_norm.pickle')
```

- Line 2 saves the non-normalized dataset as its own pickle file. A pickle file is Python's serialization format for saving objects as binary files so that they can be loaded back into a Python environment.
 - Line 3 saves the normalized dataset, along with the PCA features, as its own pickle file. You'll use these prepared datasets to create machine learning models in the next lesson.
- e) Run the code cell.

There is no output, but the files should have been saved to **/home/student/CAIP/Data Preparation**.

10. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.



Note: When you shut down a notebook kernel, you may be prompted to confirm that you want to leave the page. If so, select **Leave page**.

- Close the **Data Preparation - KC Housing** tab in Firefox, but keep a tab open to **CAIP/Data Preparation** in the file hierarchy.

TOPIC D

Work with Unstructured Data

So far, you've worked with data that follows a standard tabular structure. But not all data useful for machine learning will be in this form. In this topic, you'll process unstructured data so that it's ready for machine learning.

Unstructured Data Examples

There are several sources of data can be considered unstructured. The following are some of the most common that you'll encounter:

- **Textual data**, which refers to written instances of natural language. This can cover a great deal of different forms, including novels, poetry, news articles, scientific articles, technical manuals, blog posts, product reviews, social media posts—pretty much any digital artifact that features the written word.
- **Image data**, which can refer to either bitmap (raster) images (binary files consisting of specific pixel values) or vector images (files consisting of mathematical formulas that draw geometric shapes). The former can either be drawn digitally, photographed digitally, or scanned from an analogue source and converted into a digital format. The latter is typically drawn digitally.
- **Video data**, which typically refers to frames of encoded bitmap images sequentially displayed over some predefined rate of time. Like individual images, videos can be created digitally, recorded from a digital camera, or digitally scanned from an analogue source like photographic film. Videos may also contain audio data.
- **Audio data**, which represents sound waves by sampling an audio signal sequentially over some predefined rate of time. Audio can be created digitally, recorded from a digital microphone, or digitally sampled from an analogue source like phonograph (vinyl) records.

Word Embedding

Consider a dataset where one of the features is `Response`. This is a user's written response to a chatbot. This feature is not quite categorical, and even though the order of words is important, it's not quite ordinal in the sense that one word is necessarily more important than another. You need some way to transform this data so that it's easier to analyze and more conducive to modeling.

Embedding is the process of condensing a language vocabulary into vectors of relatively small dimensions. For example, the average adult American knows the meaning of about 40,000 words in the English language. Rather than have one feature for every one of these 40,000 words, which would present a huge problem for performance, 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. When trained on certain models, particularly neural networks, words with similarities are placed close to each other in the embedded space so that the meaning of a clause or sentence is more easily identified.

The following figure shows the embedded space for words representing different forms of written media and words for how those media are divided. The embedding is able to correctly place a form near its corresponding division.

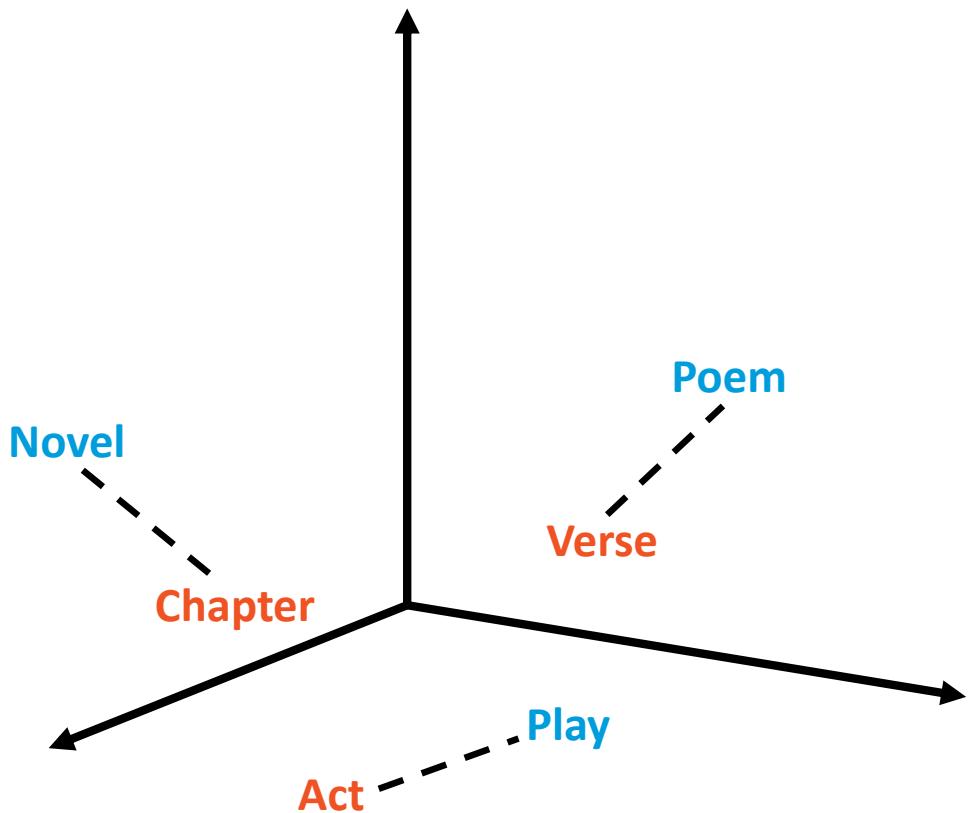


Figure 2–4: An embedded space of written forms and their divisions.

Many practitioners before you have already trained useful word embeddings, so you should consider leveraging these pre-trained embeddings in your own projects.

Word Embedding Tools

If you 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.
- **fastText**, an extension of Word2vec that partitions words into n -grams, a form of tokenization. For example, the n -grams for the word "drink" might be "dri," "rin," and "ink." The vector for "drink" 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.
- **TF-IDF**, or term frequency-inverse document frequency, represents the proportion of words that appear in each document, while also considering how many documents in the entire corpus contain those words.
- **GloVe**, or global vectors, a technique that incorporates matrix factorization to calculate the frequency of words within a certain context, like a document or corpus.

Text Data Transformation Techniques

Embedded vectors are typically not created directly from textual content. Instead, some degree of preprocessing is involved. There are several of these techniques for handling text data that you

should be aware of. The following table uses a line from Walt Whitman's poetry collection *Leaves of Grass* as an example:

'The shadows, gleams, up under the leaves of the old sycamore-trees'

Technique	Description
Bag of words	<p>A bag of words is a list of each individual word in a document without respect to grammar, punctuation, or any other language component that may be extraneous to the word itself. It is usually represented in a collection object like an array or dictionary. In the case of the latter, the key is the word and the value is the number of times it appears in the document.</p> <p>A bag of words using the example might produce a dictionary like:</p> <pre>{'the': 3, 'shadows': 1, 'gleams': 1, 'up': 1, 'under': 1, 'leaves': 1, 'of': 1, 'old': 1, 'sycamore': 1, 'trees': 1}</pre>
Tokenization	<p>Tokenization partitions a document into smaller units. In fact, <i>n</i>-grams are a form of tokenization. There is also word tokenization, where a document is simply broken up into words, and character tokenization, where each character is partitioned.</p> <p>An example of character tokenization applied to the word "gleams" and put into a list:</p> <pre>['g', 'l', 'e', 'a', 'm', 's']</pre>
Removing stop words	<p>A stop word is any word that is very common in text and whose inclusion typically provides little to no value. Words like "the," "a," "an," and so on, are usually excised from a document and put into a list of stop words. When an algorithm operates on text, it can refer to this list and ignore any such words. Other than universally common words, a stop word can also be any word that appears very frequently in the document in question. So, a list of stop words might be created by taking the frequency of all words in a document and extracting the top results.</p> <p>Considering the entire poetry collection rather than just a single line, the stop words from this line might be:</p> <pre>['the', 'up', 'under', 'of']</pre> <p>The word "leaves" is also very common, so it might be a candidate for a stop word. Though, because of its thematic importance, it may be best to leave it alone.</p>
Stemming	<p>Stemming removes the affix of a word in order to reduce the inflection of that word. In most cases, this means a suffix, but stemming can also be applied to prefixes. The result of the operation is the word stem, the base form of the word itself. Suffixes include plurals like "s" and "es"; gerunds ("ing"); past tense ("ed"); adverbs ("ly"); and so on.</p> <p>If run through a stemming program, the word "shadows" becomes:</p> <pre>'shadow'</pre> <p>However, stemming is somewhat limited. For the most part, it simply chops off the suffix without regard to the true dictionary root word. For example, stemming the irregular plural "leaves" produces:</p> <pre>'leav'</pre>

Technique	Description
Lemmatization	<p>Lemmatization addresses the shortcomings of stemming through morphology; that is, the analysis of word parts and structures and how they change.</p> <p>Lemmatization attempts to derive the <i>lemma</i>, or the canonical dictionary form of a word. The lemma can be derived through cutting off a suffix, like in stemming, but also through more intelligent methods, like analyzing word tense and grammatical mood.</p> <p>When "leaves" is lemmatized, the outcome will be:</p> <p>'leaf'</p> <p>This is the base dictionary form of the word. However, lemmatization isn't perfect, as "leaves" in a different context might be a verb whose lemma is actually "leave"—to depart.</p>

Image Data Representation

Just like textual data, image data must be transformed before it can be used effectively. Typically, raw images are represented as an array of numeric values, where each cell represents a pixel. In a grayscale image, the number in a cell represents that pixel's intensity, where 0 is black and 255 is white. Simple color images are often represented with RGB (red, green, blue) values for each pixel. Most deep learning algorithms like convolutional neural networks (CNNs) compress this kind of data to eliminate "noise" in the image and only retain the features of the image that are most useful for determining patterns. This is called a *latent representation* of an image because the patterns being learned are "hidden" within the network. When building a CNN, you can extract these latent representations from the neural network as one-dimensional vectors. The vectors include all of the relevant features and can therefore be used in place of the original image in your dataset.

You might want to extract these vectors if you plan to input the image data into another model, as it will speed up the process and potentially improve results if the new model doesn't create its own latent representations. You can also measure the similarity between feature vectors using a distance metric like cosine similarity, which calculates the cosine of the angles between each vector. As the angle decreases, cosine increases, as does the level of similarity. So, to streamline your image data, you might remove any feature vector from the dataset that is highly similar to another feature vector.

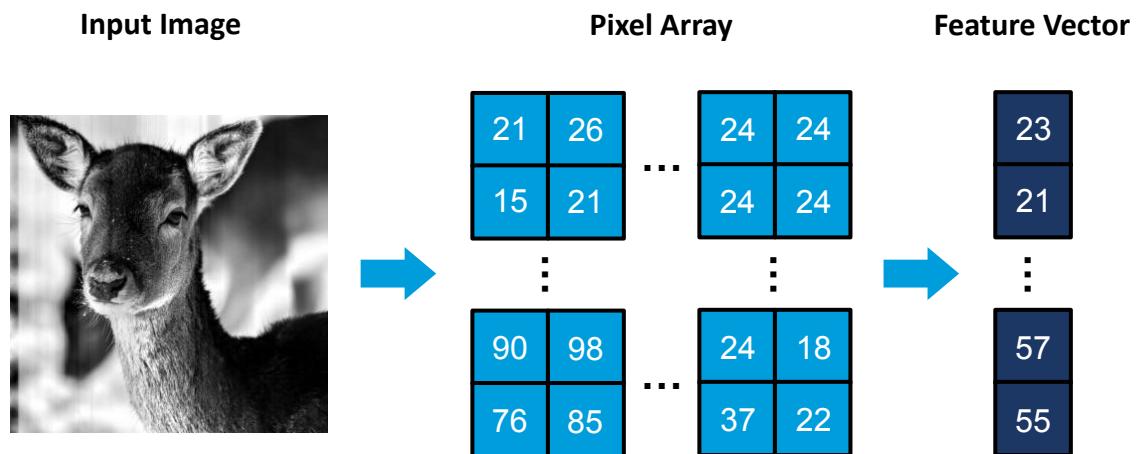


Figure 2–5: Extracting a feature vector from a grayscale pixel array.

Image Grayscale Conversion

Although you can rely on a model's latent representations, you'll probably want to do at least some preprocessing yourself before you use image data as training. One of the most common techniques is to reduce the dimensionality of an image. This is not to be confused with reducing an image's *pixel dimensions*, which will be discussed shortly. Instead, recall that features are what constitute a data example's dimensionality. In the case of images, color information makes up a lot of an image's features.

Consider the image of the deer that was just shown. Here's what the original image looks like.



Figure 2–6: Original full-color deer image.

In a pixel array, there would be three separate values for each pixel—intensity of red, intensity of green, and intensity of blue. That means the image has three dimensions: the x-axis location of the pixel, the y-axis location of the pixel, and the z-axis color values of the pixel. It would look something like the following:



$$\begin{bmatrix} [37 \ 19 \ 7] & [37 \ 19 \ 7] & \cdots & [142 \ 163 \ 192] & [142 \ 163 \ 192] \\ [37 \ 19 \ 7] & [37 \ 19 \ 7] & & [142 \ 163 \ 192] & [142 \ 163 \ 192] \\ \vdots & & \ddots & & \vdots \\ [35 \ 17 \ 13] & [35 \ 17 \ 13] & & [192 \ 182 \ 183] & [192 \ 182 \ 183] \\ [38 \ 20 \ 16] & [38 \ 20 \ 16] & \cdots & [194 \ 184 \ 185] & [194 \ 184 \ 185] \end{bmatrix}$$

Figure 2-7: A pixel array of the original full-color deer image.

Although color information can be useful, most algorithms perform just fine with grayscale images. The shading of each pixel relative to its surrounding pixels is often enough for a neural network to retain important information about the image that would otherwise be latent in a color space.



By converting the deer image to grayscale, it now only has two dimensions: the x-axis and y-axis, with a single value at their intersection.



$$\begin{bmatrix} 22 & 22 & \cdots & 161 & 161 \\ 22 & 22 & & 161 & 161 \\ \vdots & & \ddots & & \vdots \\ 21 & 21 & & 184 & 184 \\ 24 & 24 & \cdots & 186 & 186 \end{bmatrix}$$

Figure 2–8: The reduced pixel array of the grayscale image.

This is a significant reduction in unnecessary information, which will speed up training considerably—especially important when your dataset has thousands or millions of full-color images.

Image Scaling

You can also transform an image's *pixel dimensions*. In other words, you can resize it—usually by making it smaller. Just like removing color, reducing the total number of pixels will simplify the data you use as input, making it easier for the algorithm to learn the latent representations. There are two approaches to resizing: scaling and shaping.

To scale an image is to change the overall number of pixels while preserving the image's original **aspect ratio**. Aspect ratio in two-dimensional images is the measurement of one dimension relative to the measurement of the other dimension, usually expressed as width to height in a reduced form. The original deer image is 1,920 pixels wide and 1,280 pixels tall, or 1920×1280 . This total pixel count is the image's **resolution**. The aspect ratio of this image is commonly expressed as 3:2, but can also be depicted as the fraction $\frac{3}{2}$, which is 1.5. For every 3 pixels in width, there are 2 pixels in height. The image is therefore wider than it is tall.

The following figure shows, proportionally, what would happen if you scaled the deer image down to a quarter of its original size.



Figure 2–9: Scaling the image down.

The image goes from a resolution of 1920×1280 to 480×320 . The aspect ratio is still the same 3:2, which you can tell by looking at the image—it has the same rectangular shape. The image is still visually recognizable, and likewise, a neural network will be able to deduce the latent representations without spending as much time as it would on the full-sized image.

Image Reshaping

Many algorithms only take square images as input. This simplifies the input even further, particularly if your dataset of images has a mix of different aspect ratios. A square image has an aspect ratio of 1:1. If you reshape a non-square image to a square one, one of two things must happen:

- The image is either expanded (if it's tall) or contracted (if it's wide) to fill the square. This "skews" the image.
- Part of the image must be removed. This is effectively the same as cropping the image.

The decision between these two approaches comes down to what appears in the image itself, where the important detail appears, how much extraneous information there is in the image, and what the network needs to learn from the image.

For example, in the deer image, the right side of the image has plenty of out-of-focus background space that probably won't be all that useful to a task like classifying the type of animal in the picture. So, it makes sense in this case to take the cropping approach. In the figure, the right side of the image has been cropped, and the resulting square image puts the deer in the center of the frame.

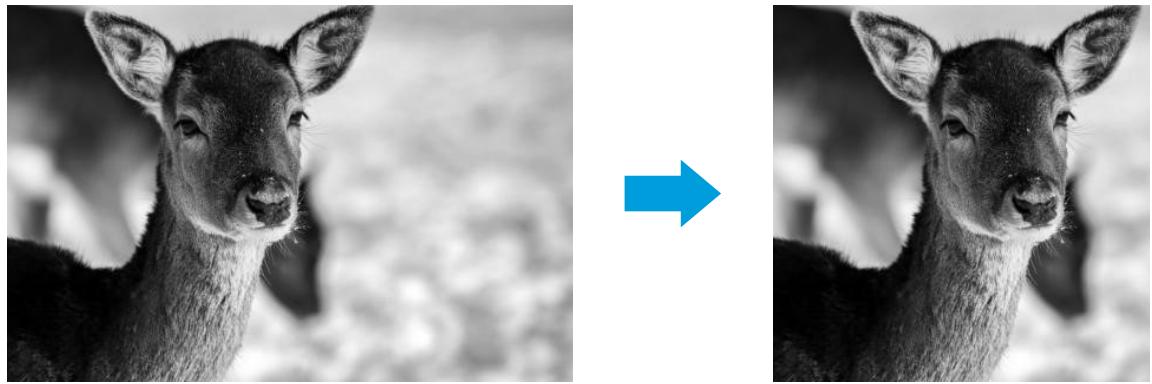


Figure 2–10: Cropping the image so that it's square.

The resolution of the image is now 320×320 , a 1:1 ratio. If you were to instead contract the image instead of cropping it, it would look like the following figure.

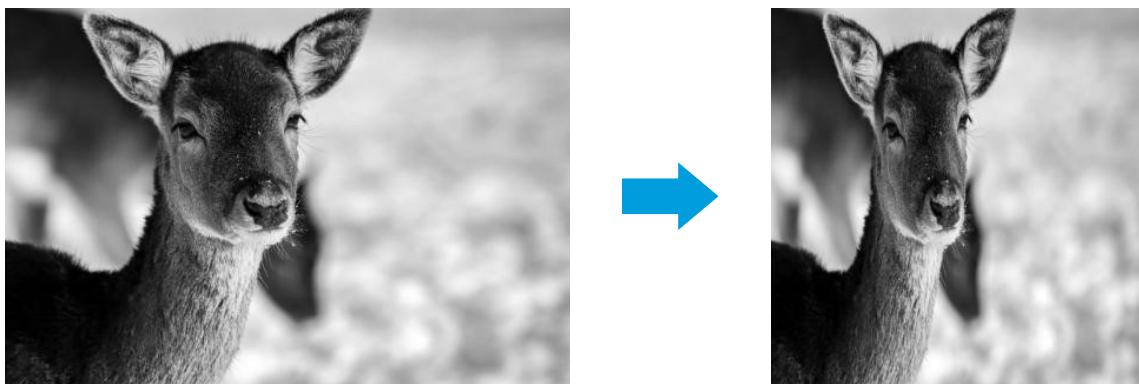


Figure 2-11: Contracting the image so that it's square.

The contracted image is also 320×320 , and therefore 1:1. The frame of the image is mostly preserved, but appears skewed in order to fit within a different aspect ratio.

You could even scale this image down more, if desired. Neural networks are very good at learning patterns from images that are so small, even a human has trouble discerning them. Some networks can learn from images as small as 25×25 pixels, or even smaller. However, the more detail that the network needs to learn from the image, and the more complex the network's assigned task, the larger the image will need to be.

Image Perturbation

To **perturb** an image means to distort the pixels of an image without compromising the overall information contained within that image that the neural network is meant to learn. There are several ways to perturb an image, including:

- Mirror/flip the image horizontally or vertically.
- Rotate the image clockwise or counterclockwise.
- Offset/translate some portion of the image along either axis.
- Adding random noise or other filters and effects to the image.

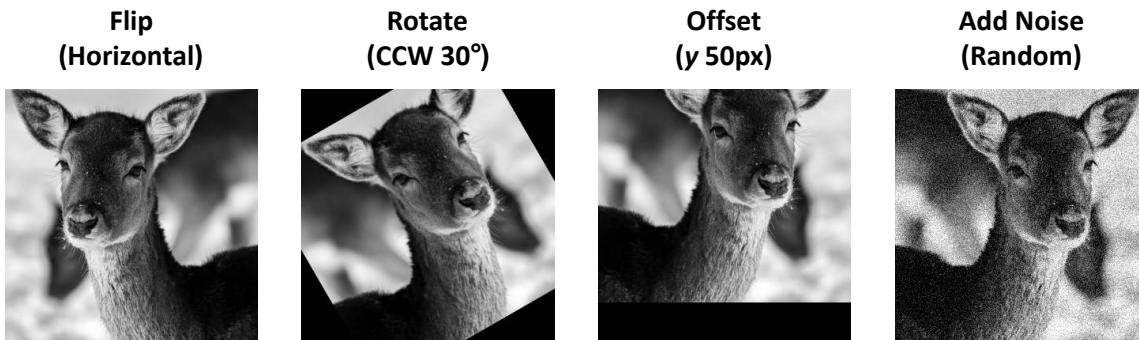


Figure 2-12: Perturbing an image.

Perturbation is a useful technique for avoiding the problem of overfitting, in which a neural network (or other machine learning model) learns the wrong patterns and is only effective at making decisions on the input data, rather than new data.

For example, let's say you want the neural network to classify the subject of an image as either a deer or a moose. If all the labeled deer images you use as input show the deer looking to the left of the frame, but the new image you want it to classify shows the deer looking to the right, the network may be unable to recognize the deer. Or, the network may focus on the center image—but what if the deer is off to one side? Some images, especially photographs, may be blurry or noisy—if the network only learns from "clean" images, will it be able to correctly classify an "unclean" one?

Perturbation ensures the input data is sufficiently varied for the neural network to learn the correct patterns.

Image Augmentation

Perturbation is also commonly called **augmentation** when it is used to generate multiple modified copies of an image. For example, you might take each source image and produce a flipped copy, a rotated copy, and an offset copy—all of which are fed into the network as separate data samples. So, you'd have three times the amount of input data (assuming you're not using the source image as input).

Randomized Perturbation

Although you could apply the same perturbations to all images in your dataset, it may be more beneficial to randomize them. That way, the network isn't stuck with the same problem: learning repeated superficial details instead of the true patterns. You can randomize images in terms of the types of perturbations, but also in how those perturbations are applied. For example, you might allow a random number generator to determine how many degrees to rotate each image by.

Image Normalization and Standardization

Since an image can be represented as an array of numbers, it is effectively numeric data—and numeric data can be normalized and standardized.

The formula for normalizing an image, as well as its output, is no different than if you had normalized any other dataset: the pixel values are scaled from 0 to 1, where 0 is the lowest value (i.e., the value closest to black) and 1 is the highest value (i.e., the value closest to white). So, the output will contain decimal values instead of integers between 0 and 255 (or whatever other grayscale or color scheme the image was encoded with).

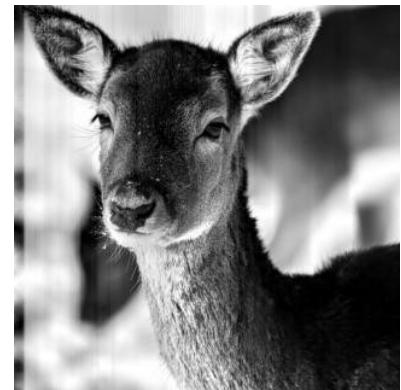
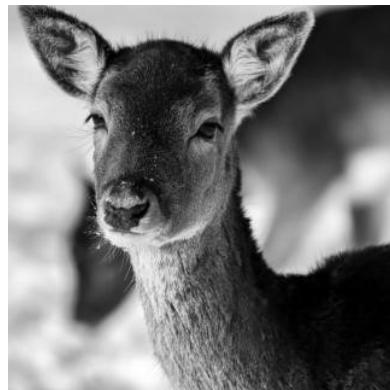


Figure 2–13: Normalizing an image.

In this example, the outcome of normalization is not dramatically different, at least from a human point of view.

Likewise, standardizing an image will center the pixel values around 0 as the mean, with their pixel values being the number of standard deviations above or below this mean.

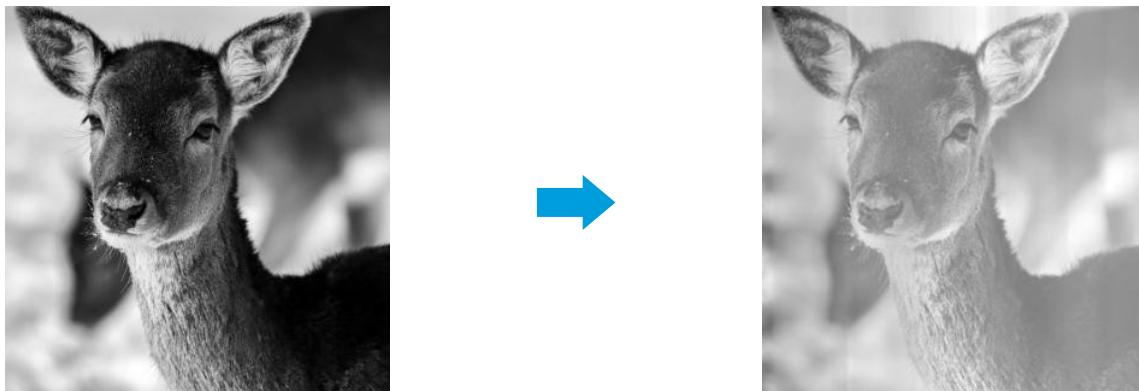


Figure 2-14: Standardizing an image.

In this example, the scaled image is more visually different than the baseline image.

As you might expect, performing either of these feature scaling techniques (not to be confused with resolution scaling) will ensure the neural network sees the distribution of pixel values rather than the absolute pixel values. So, the network may be able to learn the true patterns in the image more efficiently.

Video Data Preprocessing

Recall that videos are essentially just individual images (frames) of the same resolution that are arranged in order, displayed over a period of time. This means that you can apply pretty much any of the image preprocessing techniques you just saw to each frame of a video.

Of course, you'll need to extract those frames from the video file first. There are some important things to keep in mind when doing this:

- **Frame rate**—Each video file has a *frame rate*. The frame rate is the number of frames that are displayed every second. Film is typically shot at 24 frames per second (fps), whereas most commercial video cameras shoot at 30 fps or 60 fps. If your video is 10 minutes long and shot at 60 fps, it will contain 36,000 frames. So, the data examples you input into the network may be so numerous that they become difficult to manage, as well as slow the network down. You can alleviate this somewhat by reducing the frame rate of the video. For example, you could drop every other frame and set the frame rate at 30 fps on playback. Visually the video won't be as smooth, but the neural network should still have plenty of information to learn from.
- **Data storage**—A related problem is one of data storage. The amount of storage space it takes to hold all of the frames from a video as individual image files will almost always be greater than the storage space taken up by the video file itself. This is because video files use compression technology to encode the differences between one frame and the next, rather than encode every single frame independently. Make sure you have enough storage space to handle thousands or millions of frames.
- **Resolution scaling**—You should almost always scale down video frames to a much smaller resolution. Even low-resolution video can slow a network down due to the sheer number of frames it must look at.
- **Consistent transformations**—If you perturb or otherwise apply some transformation to one frame of a video, you should apply those same transformations to all other frames of the same video. That way, the neural network will better learn useful information from each video. If the frames in a video are constantly rotating, changing size, changing orientation, and so on, it may be difficult for the network to understand how they all fit together. You can apply different transformations to different videos—as long as they're internally consistent.
- **Frame sequence**—Neural networks will need to consider video frames within an overall sequence. After all, that sequence is important to the context of the video, whereas no sequence is implied by individual images. Make sure you are naming and categorizing frames in a sequence,

such as 000001.jpg, 000002.jpg, and so on. An iterative naming scheme like this will keep the frames in the correct order.

Video Preprocessing Example

The slide shows two videos: the one on the left is the original video, and the one on the right is a preprocessed version. The images are to scale. The following transformations were applied:

- Halved frame rate from 30 fps to 15 fps.
- Reduced resolution from 1920×1080 to 180×180.
- Cropped to a 1:1 square shape.
- Converted to grayscale.
- Rotated 10 degrees clockwise.
- Flipped vertically.
- Applied blur effect.
- Standardized pixel data.

The preprocessed video on the right was converted to an animated GIF, whereas the original video is an MP4 file.

Video files often contain audio as well. For many tasks, the visual aspect is what's important, so you can safely discard the audio portion. If the audio is important to the network's assigned task, you'll need to process that audio using separate methods.

Audio Data Primer

Like image data, audio data must be transformed before it can be processed by a neural network or other machine learning model. First, you need to understand the three aspects of sound waves that are most relevant to audio processing:

- **Amplitude:** The magnitude of the signal, usually measured in decibels (dB).
- **Frequency:** The number of times a sound wave is repeated per second, usually measured in hertz (Hz).
- **Time:** The duration of the signal, usually measured in seconds.

Amplitude and frequency are both functions of time. A neural network should be able to learn from how both of these factors change over time.

Another important aspect of audio data you should be aware of is the sample rate. In the real world, audio signals are continuous. In order to be represented digitally, audio signals must be discrete. The conversion process is called **sampling**, which records the value (i.e., the amplitude) of a continuous audio signal at a certain point in time, at a certain rate. The sampling rate is measured in Hz, like the frequency. With a sampling rate of 1,000 Hz (1 kHz), the audio file has 1,000 different measurements of amplitude per second. In a 5-second audio clip, you would have 5,000 samples, or 5,000 total amplitude values.

The sampling rate therefore determines the maximum frequency that a digital (discrete) audio signal can reproduce from an analogue (continuous) one. The frequency range of human hearing is approximately between 20 Hz and 20 kHz. For various reasons, sampling rate should double the frequency that it intends to capture. This is why audio CDs have a sampling rate of 44,100 Hz (44.1 kHz), and why more modern audio formats typically have a sampling rate of 48,000 Hz (48 kHz). However, a sampling rate of 22,050 Hz (22.05 kHz)—half the rate of audio CDs—is still used in scenarios where lower quality audio is necessary. There are other sampling rates as well, but these are the most common.

Audio Waveforms

When you load an audio file into a programming environment for use in machine learning, you typically get an array of amplitude values for each sample. Depending on the library, these can be represented in different ways. In many cases, the amplitude may be represented as the change in pressure around the microphone as a float value. It's also common to see the values measured (or converted to) decibels.

You can use a [waveform](#) to visually represent the amplitude of an audio signal as change in pressure over time.

$[-0.0074 \quad -0.0064 \quad \dots \quad -0.0005 \quad -0.0001]$

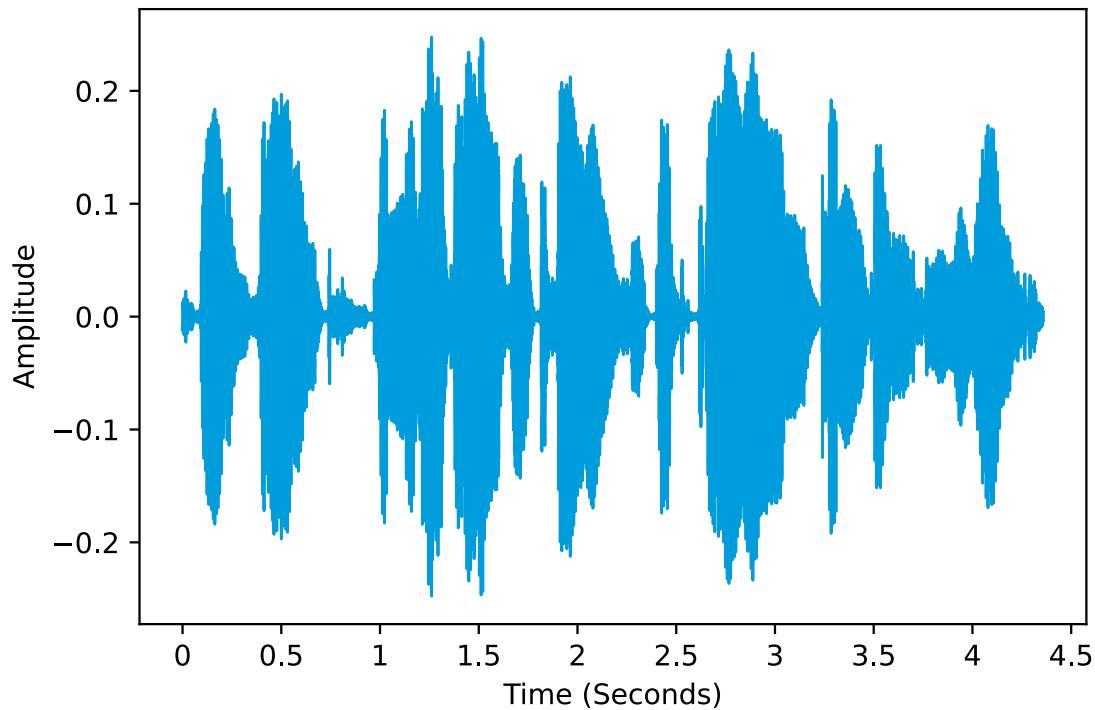


Figure 2-15: The amplitude array and corresponding waveform of a four-second audio file.

There are various crests and valleys in the waveform, indicating changes in amplitude as the speaker recites the lines of poetry. Raw amplitude data and its corresponding waveform are useful for human interpretation, but are not always enough for a neural network to learn from since they only demonstrate the loudness of a recording.

Fourier Transformation

To extract more useful features out of an audio file that a neural network can learn from, you can apply a [Fourier transformation](#) to your data. Put simply, a Fourier transformation decomposes an audio signal into its constituent frequencies. After all, most sound recordings exhibit multiple frequencies—someone's voice changing as they speak, or different instruments being played in the case of music. Instead of just indicating amplitude over time like a waveform, a Fourier transformation indicates amplitude *at each frequency* over time. By applying this transformation, you are able to extract all three dimensions of audio data mentioned earlier: amplitude, frequency, and time.

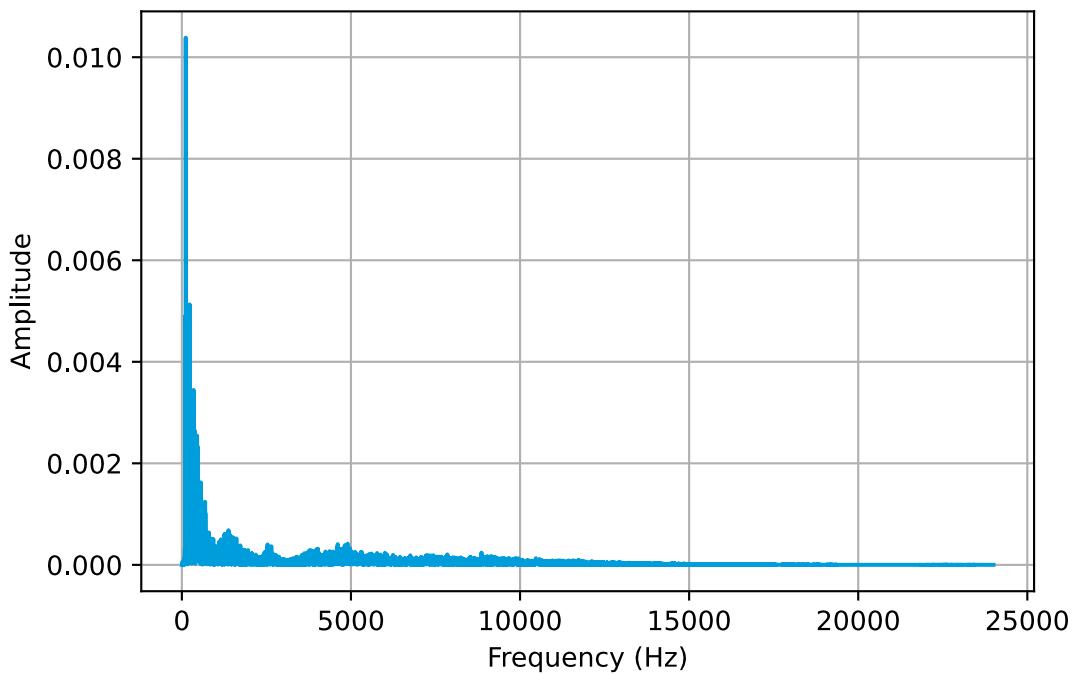


Figure 2–16: Amplitude over frequency.

In the figure, frequency and amplitude (on an absolute scale) are plotted. This shows that the loudest portions of the audio clip are at the lower frequencies. This makes sense, as most of the audio comes from a man speaking, and the typical frequency range for an adult male voice is between 85 Hz and 155 Hz.

There are several different Fourier techniques, the details of which are beyond the scope of this course. For now, it's sufficient to know that Fourier transformations help reveal more useful information about your audio that you can feed into a neural network.

Audio Spectrograms

To depict all three dimensions of an audio signal, you can create a [**spectrogram**](#). A spectrogram shows time along the x-axis, frequency along the y-axis, and amplitude as gradations of color as in a heatmap.

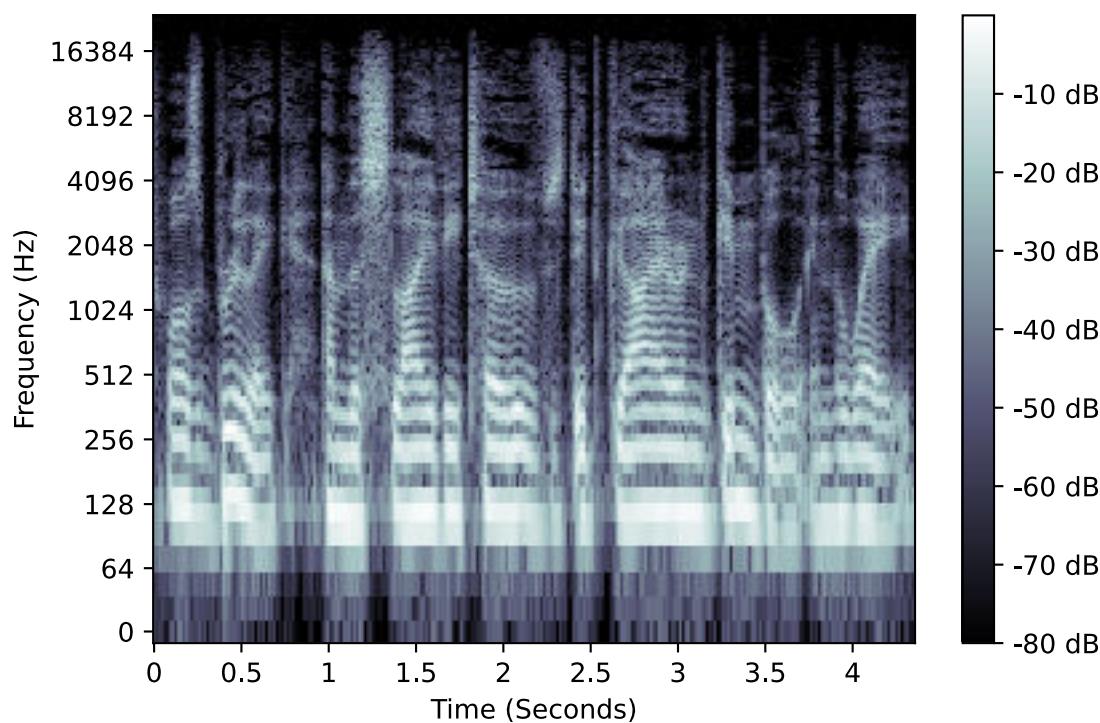


Figure 2-17: A spectrogram of the speech file.

It's important to note that dB measurements can be confusing. In casual conversation, when people refer to dB, they are usually referring to sound pressure level (SPL). Essentially, this measures dB according to human hearing on an absolute scale. So, a 0 dB signal is the "floor" of what is perceptible to human hearing, whereas anything larger than 120 dB is likely to inflict pain. When it comes to audio processing, dB is used to compare the amplitude of two signals on a relative scale. So, 0 dB means both signals have the same amplitude. In many audio processing libraries (as well as professional audio equipment), 0 dB is set as the threshold by which any sample above 0 is loud enough that it distorts the signal. This is all to say that most dB measurements you'll see in an audio processing library will be negative.

So, in the figure, lighter portions indicate louder parts, whereas darker portions indicate quieter parts. The loudest portions are, as expected, at the low frequencies that are typical of male speech. You can also tell that frequency as a function of amplitude tends to follow a pattern over the duration of the clip. There are intermittent gaps of quiet at different frequencies as the man speaks, which correspond to the poetic meter (the rhythmic structure of each line as a series of stressed and unstressed syllables).



Note: The y-axis in this figure is logarithmic.

Mel-Frequency Cepstral Coefficients (MFCCs)

Fourier transforms are perhaps the most basic methods for transforming audio signals. Another method you might want to research more on your own time is mel-frequency cepstral coefficients (MFCCs). These coefficients represent the power of an audio signal in a frequency range after the signal's power spectrum has been transformed using a non-linear scale. They usually incorporate Fourier transformation as the first of several steps.

MFCCs help indicate the pitch of an audio signal and can be plotted using spectrograms, where the y-axis is the pitch as measured by the number of "mels" (from "melody"). MFCCs can also be plotted in chromagrams, which map time to traditional musical scales (i.e., C, C#, D, D#, and so on).

Audio Data Preprocessing

Besides techniques like Fourier transformations that extract audio features, you can apply other preprocessing techniques to your audio data.

- **Reduce sampling rate**—Although 44.1 kHz is a good sampling rate for audio playback, neural networks can work with lower sampling sizes. 22.05 kHz is good enough for most neural networks, and many audio libraries default to this sampling rate. Reducing sampling rate can save on storage space and neural network processing time, as the smaller the sampling rate, the less information there is.
- **Reduce bit depth/bit rate**—Bit depth is another dimension by which the amount of information in an audio file is measured. It is essentially the number of binary digits that represent one sample. The bit rate is the number of bits processed per second of audio. Both have an effect on the quality of the audio, with higher quality leading to larger file sizes. So, you may wish to reduce either, or both. Audio CDs have a 16-bit depth, whereas modern high-quality sources have a 24-bit depth. Note that compressed audio formats (like MP3) reduce the bit rate of files compared to uncompressed audio formats (like WAV).
- **Convert stereo to mono**—The source audio may have sound recorded in two separate channels—in other words, stereo sound. It's usually beneficial to convert stereo tracks into mono (one channel) to reduce complexity. A neural network may not need separate channels to perform its assigned task, assuming no important spatial information will be lost.
- **Normalize/standardize values**—You can apply these familiar feature scaling techniques to audio just like you can with other types of data. The purpose remains the same: to emphasize the distribution of values rather than absolute values.
- **Add or remove silence**—Adding and removing silence serve a similar purpose as reshaping or perturbing an image; you're either trying to get all of your files into a consistent format, or apply subtle variations so that a neural network is not overfitting to the input. In either case, most neural networks expect audio input to be the same length, so you might want to add or remove silence to short or long files as necessary.

Audio Preprocessing Example

The slide shows two sound clips: the one on the left is the original clip, and the one on the right is a preprocessed version. The following transformations were applied:

- Reduced the sampling rate from 48 kHz to 22.05 kHz.
- Normalized the sampling data.
- Added half a second of silence to both ends.

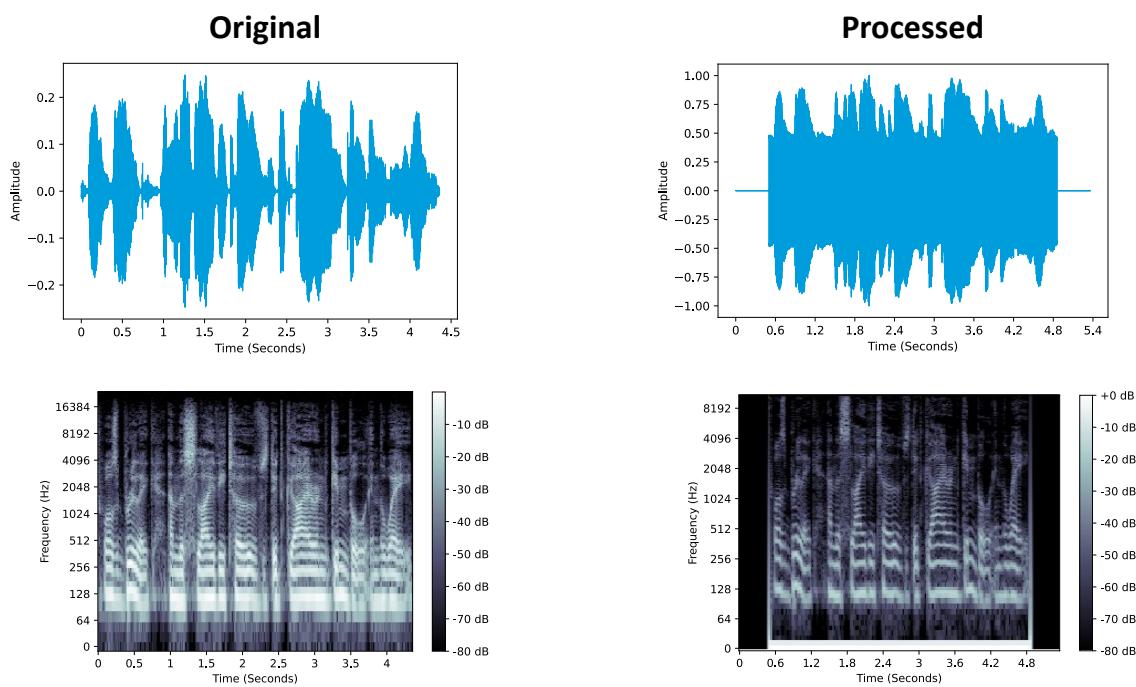


Figure 2-18: The waveforms and spectrograms of each file.

From the waveform you can tell that the processed version has less dynamic range, as there is less of a difference between the quiet parts and the loud parts. The spectrogram reveals a similar overall pattern in both versions, though the processed one has subtle reductions in the loudness of certain frequency ranges. Both visuals reveal the addition of silence to either end of the clip.

Depending on your own hearing ability, you may be able to discern the difference between both clips. In particular, the processed version is of lower quality. Even so, a neural network should be able to learn useful patterns for carrying out its assigned task—for instance, determining the gender, nationality, and age of the speaker.

Guidelines for Working with Unstructured Data

Follow these guidelines when working with unstructured data like text, images, videos, and audio.

Work with Unstructured Data

When working with unstructured data:

- **Text**
 - Perform embedding on text data to reduce its dimensionality.
 - Consider leveraging existing embeddings rather than creating your own.
 - Use techniques like bag of words, stop words, and tokenization to reduce the complexity of a textual dataset.
 - Use lemmatization to find the canonical dictionary forms of each word.
- **Images**
 - Convert color images to grayscale.
 - Scale your images down to smaller dimensions, preferably no larger than 256×256 pixels.
 - Reshape your images so that they have a square aspect ratio.
 - Apply perturbation techniques like rotation and translation to add variation to images.
 - Consider feature scaling pixel data using normalization or standardization.
 - Recognize that some neural network libraries perform these tasks for you.
- **Video**

- Extract a reduced number of frames from a video to alleviate storage and processing issues.
- Recognize that videos contain many frames, and extracting even a sample of them will take up a significant amount of storage space.
- Ensure you are downscaling and reshaping video frames like you would with individual images.
- Apply consistent transformations to all frames in one video.
- Ensure frame files are named and categorized sequentially so as to maintain their context within the video.
- **Audio**
 - Generate waveforms to analyze the amplitude of an audio file over time.
 - Apply a Fourier transformation to audio files to derive frequency information as a function of time and amplitude.
 - Generate spectrograms to analyze all three dimensions of a Fourier-transformed audio signal.
 - Reduce the sampling rate of an audio file to 22.05 kHz for greater processing efficiency.
 - Consider reducing bit depth/bit rate in an audio file to save on storage and processing efficiency.
 - Convert stereo tracks to mono to simplify the input.
 - Consider applying feature scaling techniques to audio data.
 - Ensure all audio clips used as input are the same length, and add or remove silence as necessary.

Use Python to Process Text

The third-party spaCy library offers several ways for processing text, including:

- `nlp = spacy.load('package')` —Load a spaCy model with the specified name to use with your text.
- `doc = nlp(text)` —Create a spaCy Language object out of an object containing text.
- `doc.sents` —Create an iterable object for extracting the sentences in the `doc` object.
- `sentence.text` —Create an iterable object that tokenizes a sentence object into words.
- `token.is_stop` —Determine whether a tokenized word is considered a stop word.
- `token.lemma_` —Retrieve the lemma of each tokenized word.

Use Python to Process Images

The skimage library, an offshoot of scikit-learn, has various functions for processing images. You can also use Matplotlib to display images in your programming environment.

- `image = skimage.io.imread('image.jpg')` —Load an image as a numeric pixel array.
- `image = matplotlib.image.imread('image.jpg')` —An alternative method for reading an image.
- `image_gray = skimage.color.rgb2gray(image)` —Convert a color image to grayscale.
- `matplotlib.pyplot.imshow(image_gray, cmap = 'gray')` —Plot the image using a grayscale color map. By default, Matplotlib uses a different color map, so you must override it if you want the image to be truly grayscale.
- `matplotlib.image.imsave('image_gray.jpg', image_gray, cmap = 'gray')` —Save an image to local storage using a grayscale color map.
- `skimage.transform.rescale(image_gray, 0.25)` —Scale an image down to 25% of its original size, preserving its aspect ratio.
- `skimage.util.crop(image_gray, ((0, 0), (0, 160)))` —Crop an image using values that correspond to the image's top, bottom, left, and right, respectively. In this instance, 160 pixels are being cropped from the right.
- `skimage.transform.resize(image_gray, (320, 320))` —Reshape an image to fit into the provided resolution. The image will either expand or contract to fill the 320×320 space, depending on its original shape.

- `skimage.transform.rotate(image_gray, 30)` —Rotate the image counterclockwise by 30 degrees. Use negative values for clockwise.
- `numpy.fliplr(image_gray)` —Flip the image horizontally.
- `numpy.flipud(image_gray)` —Flip the image vertically.
- `tform = skimage.transform.SimilarityTransform(translation = (0, 50))` —Create a transformation object that will translate (offset) the image 50 pixels from the bottom.
- `skimage.transform.warp(image_gray, tform)` —Perform the transformation using the `tform` object.
- `skimage.util.random_noise(image_gray)` —Add random noise to the image.
- `skimage.filters.gaussian(image_gray)` —Add a blur effect to the image.
- Use the scikit-learn `MinMaxScaler()` and `StandardScaler()` classes on the image object to perform normalization and standardization, respectively.

Use Python to Process Video

The open-source, cross-platform FFmpeg is one of the most common video-processing utilities. The FFmpy library offers a Python wrapper for issuing FFmpeg commands.

- `ff = ffmpy.FFmpeg(inputs = {'video.mp4': None}, outputs = {'out_frames/%06d.jpg': '-vf fps=fpss=29.97/2,scale=320:-1 -q:v 5'})` —This creates an object that, when issued, will do the following:
 - Take `video.mp4` as input.
 - Convert the video to JPEG outputs, with `%06d.jpg` indicating the naming scheme to use—each image will be incremented by one and padded with zeros for a total of six digits.
 - Apply an FPS video filter to take the original video frame rate (29.97) and halve it.
 - Apply a scale video filter to make the images 320 pixels wide. The `-1` indicates that the height will be the number of pixels it takes to maintain the video's aspect ratio.
 - Use a quality of 5 for the JPEG outputs. The lower the number, the higher the quality.
- `ff.cmd` —Display the FFmpeg command that will run to check it for errors.
- `ff.run()` —Run the command.

You can then return to `skimage` and `Matplotlib` to process the frames:

- `images = skimage.io.imread_collection('out_frames/*.jpg')` —Read multiple images at once as an iterable array of image arrays.
- Use the same image transformation functions listed in the previous procedure on the images in the collection.
- Use `imsave()` to save the transformed frames back to images.

Use Python to Process Audio

The librosa library provides functionality for processing audio.

- `samples, sampling_rate = librosa.load('audio.wav', sr = None, mono = True)` —Load an audio file in mono with its native sampling rate. The sample data itself will be in `samples`, and the sampling rate in `sampling_rate`. If you want to reduce the sampling rate, provide it for the `sr` argument.
- `librosa.display.waveform(y = samples, sr = sampling_rate)` —Generate a waveform of the audio file.
- `stft_mat = librosa.stft(samples)` —Return a matrix of sample data that has had a short-term Fourier transform applied to it.
- `db = librosa.amplitude_to_db(np.abs(stft_mat), ref = np.max)` —Convert the default amplitude values to a relative decibel scale.
- `librosa.display.specshow(db, y_axis = 'log', x_axis = 'time', sr = sampling_rate)` —Generate a spectrogram from the dB-converted sample values, using a logarithmic scale and the native sampling rate.

- `numpy.append(samples, np.zeros(24000))` —Add silence (zero amplitude) to the end of the audio track. In this case, because the track's native sampling rate is 48,000, half a second of silence will be added.
- Use the scikit-learn `MinMaxScaler()` and `StandardScaler()` classes on the `audio` object to perform normalization and standardization, respectively.
- `soundfile.write('processed_audio.wav', samples, samplerate = sampling_rate)` —Save the sample data as an audio file.

ACTIVITY 2–5

Working with Text Data

Data Files

/home/student/CAIP/Data Preparation/Data Preparation - IMDb.ipynb

/home/student/CAIP/Data Preparation/imdb_data/imdb_data_sample.pickle

Before You Begin

Jupyter Notebook is open.

Scenario

You do consulting work for an online retailer. Users can leave written reviews on the online storefront for any product they purchase. 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.

An automated system 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. Before you can even begin building such a system, you need to get the review data itself into a form that is better suited for language analysis and modeling. You'll start by applying some text transformation techniques to a dataset of movie reviews.



Note: The full dataset was retrieved from <https://ai.stanford.edu/~amaas/data/sentiment/>.

1. From Jupyter Notebook, select **CAIP/Data Preparation/Data Preparation - IMDb.ipynb** to open it.

2. Import the relevant software libraries.

- View the cell titled **Import software libraries**, and examine the code cell below it.
- Run the code cell.
- Verify that the version of Python is displayed, as are the versions of the other libraries that were imported.



Note: As the warning states, you can ignore it because the VM doesn't have a GPU.

3. Load the dataset.

- Scroll down and view the cell titled **Load the dataset**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
Data files in this project: ['imdb_data_sample.pickle']
Loaded 500 records from ./imdb_data/imdb_data_sample.pickle.
```

500 records were loaded from **imdb_data_sample.pickle**. This file contains a small sample of an overall dataset of 50,000 movie reviews posted on IMDb, an online database that catalogues various information about movies, TV shows, and other video content.

4. Get acquainted with the dataset.

- Scroll down and view the cell titled **Get acquainted with the dataset**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Review      500 non-null    object  
 1   Positive?   500 non-null    int64  
dtypes: int64(1), object(1)
memory usage: 7.9+ KB
None
```

	Review	Positive?
0	This film is terrible. Every line is stolen fr...	0
1	"Journey of Hope" tells of a poor Turkish fami...	1
2	I live in Ottawa where this film was made and ...	0
3	Standard procedure for Swedish movies today se...	0
4	Having seen both "Fear of a Black Hat" and "Th...	1
5	It is very rare for a film to appeal to viewer...	1
6	"Birth of the Beatles", for being a US televisi...	1
7	A female vampire kills young women and paints ...	0
8	As the number of Video Nasties I've yet to see...	0
9	an very good storyline, good thrill to it	0

- There are 500 records and 2 columns.
- Each record is a separate review.
- `Review` is a string object column that contains the text of the review.
- `Positive?` is an integer column that indicates whether the review was labeled positive (1) or not (0). This dataset was manually labeled and is often used for classification and/or sentiment analysis purposes. For now, you'll just work with the text in the `Review` column.

5. Extract a sample review.

- Scroll down and view the cell titled **Extract a sample review**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
'In the opinion of several of my friends and family members, including myself, this is the f
inest of the entire gamut of Tarzan movies. Johnny Weissmuller never played the part as well
in the following issues in the series. It definitely rates a "10" in my collection of film
s.'
```

The review is full of characters like punctuation, numbers, and common words. You'll need to streamline this text to make it more amenable to analysis and modeling.

6. Tokenize the sample text into sentences.

- Scroll down and view the cell titled **Tokenize the sample text into sentences**, and examine the code cell below it.

This code uses a third-party text processing library called spaCy to create a document from the sample text.

- Line 2 specifies the local path to the spaCy model that you'll be using.
- Line 4 loads the model.
- Line 7 creates a spaCy document—an object type specific to spaCy—from the sample text.
- Lines 9 and 10 print each sentence from the document.

- Run the code cell.

- c) Examine the output.

```
In the opinion of several of my friends and family members, including myself, this is the finest of the entire gamut of Tarzan movies.
Johnny Weissmuller never played the part as well in the following issues in the series.
It definitely rates a "10" in my collection of films.
```

Each sentence identified by spaCy is printed. Note that the line breaks are being treated as different sentences.

7. Tokenize a sentence into words.

- a) Scroll down and view the cell titled **Tokenize a sentence into words**, and examine the code cell below it.

To demonstrate the tokenization of individual words, this code is taking a single sentence from the overall document.

- Line 2 creates a list where each list element is the string representation of a sentence (the `orth_` attribute). The purpose is to extract the text of a single sentence.
- Line 4 tokenizes the first sentence.
- Lines 6 and 7 print the tokenized words.

- b) Run the code cell.
c) Examine the output.

```
In
the
opinion
of
several
of
my
friends
and
family
members
,
including
myself
,
this
is
the
finest
of
the
entire
gamut
of
Tarzan
movies
.
```

Each individual "word" (as defined by spaCy) is printed. Note that numbers and punctuation are counted as words.

8. Identify the parts of speech for each token.

- a) Scroll down and view the cell titled **Identify the parts of speech for each token**, and examine the code cell below it.

This code creates a data frame that will assign the parts of speech (the `pos_` attribute) to each word token.

- b) Run the code cell.

- c) Examine the output.

Word	Part of Speech
0 In	ADP
1 the	DET
2 opinion	NOUN
3 of	ADP
4 several	ADJ
5 of	ADP
6 my	PRON
7 friends	NOUN
8 and	CCONJ
9 family	NOUN
10 members	NOUN
11 ,	PUNCT

The table lists what part of speech each token is. For example, `In` is an adposition (an alternative term for a preposition), `the` is a determiner (a word that clarifies what a noun refers to), `,` is punctuation, and so on.

9. Identify stop words.

- a) Scroll down and view the cell titled **Identify stop words**, and examine the code cell below it.

This code creates a data frame that will indicate whether or not a word token in the sentence qualifies as a stop word according to spaCy. A stop word is a word so common that it should be excluded from the text so as not to exert undue influence.

- b) Run the code cell.
c) Examine the output.

Word	Stop Word
0 In	True
1 the	True
2 opinion	False
3 of	True
4 several	True
5 of	True
6 my	True
7 friends	False
8 and	True
9 family	False
10 members	False
11 ,	False
12 including	False
13 myself	True
14 .	False

Words like `In` and `the` are considered stop words, whereas `opinion` and `friends` are not.

10. Stem the text.

- a) Scroll down and view the cell titled **Stem the text**, and examine the code cell below it.

Since spaCy does not have a stemmer, this code uses the Natural Language Toolkit (NLTK) to perform stemming on the tokenized sample. Stemming obtains the affix of a word to reduce that word to a base form, making it easier to work with.

There are multiple stemming algorithms in NLTK, and each one can produce different results. In this case, on line 4, you're using `SnowballStemmer()` to see how it does.

- b) Run the code cell.

- c) Examine the output.

	Word	Stem
0	In	in
1	the	the
2	opinion	opinion
3	of	of
4	several	sever
5	of	of
6	my	my
7	friends	friend
8	and	and
9	family	famili
10	members	member
11		

You can see that some tokens were properly stemmed, but that the resulting stem is not quite a real word. For example, family became famili, and including became includ. That's why a more advanced technique like lemmatization is usually preferred.

11. Lemmatize the text.

- a) Scroll down and view the cell titled **Lemmatize the text**, and examine the code cell below it.

This code uses the same sentence sample, but this time uses spaCy to retrieve each word's lemma. A lemma is the canonical dictionary form of a word.

- b) Run the code cell.
c) Examine the output.

	Word	Lemma
0	In	in
1	the	the
2	opinion	opinion
3	of	of
4	several	several
5	of	of
6	my	my
7	friends	friend
8	and	and
9	family	family
10	members	member
11	,	,
12	including	include
13	myself	myself
14		

The lemmatization did a better job of obtaining root words. For example, family didn't change because it's already in the canonical form, and including became include, which is the canonical form. Also, a verb like is became be.

12. Transform the sample text.

- a) Scroll down and view the cell titled **Transform the sample text**, and examine the code cell below it.

Now it's time to actually transform the text data. This function, when called, will:

- On lines 6 through 9, use spaCy to tokenize the text.
- On lines 12 and 13, remove punctuation, spaces, numbers, and stop words.
- On line 16, use a regular expression to replace the full words with their lemmas.
- On lines 18 and 19, compile all of the lemmas into a single string.
- On line 21, separate each lemma in the string by a space.

- b) Run the code cell.

- c) Examine the output.

The function to transform the text has been defined.

- d) Scroll down and examine the next code cell.

```
1 # Apply transformation to sample.
2 transform_text(sample_text)
```

First, you'll try the transformation out on the one sample review.

- e) Run the code cell.
f) Examine the output.

'opinion friend family member include fine entire gamut Tarzan movie Johnny Weissmuller play follow issue series definitely rate collection film'

As expected, the text of the review has been streamlined.

- g) Scroll down and examine the next code cell.

```
1 # Compare to sample before transformation.
2 sample_text
```

- h) Run the code cell.
i) Examine the output.

'In the opinion of several of my friends and family members, including myself, this is the finest of the entire gamut of Tarzan movies. Johnny Weissmuller never played the part as well in the following issues in the series. It definitely rates a "10" in my collection of films.'

You can see significant differences between the original form of the review and the review after it underwent text processing.

13. Transform all reviews in the dataset.

- a) Scroll down and view the cell titled **Transform all reviews in the dataset**, and examine the code cell below it.

This code uses a lambda function to apply the transformation to every review in the dataset.

- b) Run the code cell.



Note: This will take a few minutes to execute.

- c) Examine the output.

	Review	Review_cleaned	Positive?
0	This film is terrible. Every line is stolen fr...	film terrible line steal mm italian dub versio...	0
1	"Journey of Hope" tells of a poor Turkish fami...	Journey Hope tell poor turkish family odyssey ...	1
2	I live in Ottawa where this film was made and ...	live Ottawa film wish God awful flick try supp...	0
3	Standard procedure for Swedish movies today se...	standard procedure swedish movie today start t...	0
4	Having seen both "Fear of a Black Hat" and "Th...	having see fear Black Hat Spinal Tap honestly ...	1

The first five reviews are printed, as are the transformed versions of those reviews.

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 **Data Preparation - IMDb** tab in Firefox, but keep a tab open to **CAIP** in the file hierarchy.
-

ACTIVITY 2–6

Working with Image Data

Data Files

/home/student/CAIP/Data Preparation/Data Preparation - Fashion.ipynb

All files in /home/student/CAIP/Data Preparation/fashion_data/

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 determines 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 sample database of product images that include photographs of clothing articles taken by both the clothing brands and your own customers who post their purchases online. You'll need to process these images before they can be used to create the automated system.



Note: The dataset was retrieved from <https://github.com/alexeygrigorev/clothing-dataset-small>, which is itself a subset of the full dataset found at <https://www.kaggle.com/datasets/agrigorev/clothing-dataset-full>.

1. From Jupyter Notebook, select **CAIP/Data Preparation/Data Preparation - Fashion.ipynb** to open it.
2. Import the relevant software libraries.
 - a) View the cell titled **Import software libraries**, and examine the code cell below it.
 - b) Run the code cell.
 - c) Verify that the version of Python is displayed, as are the versions of the other libraries that were imported.
3. Load the dataset.
 - a) Scroll down and view the cell titled **Load the dataset**, and examine the code cell below it.
 - This code walks through the **./fashion_data** directory and all its subdirectories, looking for all files.
 - Line 6 uses Matplotlib's `imread()` function to read each image as an array of pixel data.
 - b) Run the code cell.

- c) Examine the output.

```
Loaded images from ./fashion_data.
Loaded images from ./fashion_data/outwear.
Loaded images from ./fashion_data/shorts.
Loaded images from ./fashion_data/skirt.
Loaded images from ./fashion_data/shirt.
Loaded images from ./fashion_data/t-shirt.
Loaded images from ./fashion_data/longsleeve.
Loaded images from ./fashion_data/shoes.
Loaded images from ./fashion_data/hat.
Loaded images from ./fashion_data/pants.
Loaded images from ./fashion_data/dress.

Loaded 200 total images.
```

- 200 total images were loaded from several subfolders in the `./fashion_data` folder.
- This is a small sample of an overall dataset of 5,000+ clothing images.
- Each subfolder is a category of clothing. In other words, the data is already labeled. This will be important later, when you actually build a neural network to classify articles of clothing. For now, you don't need to retain the image labels.

4. Preview the image data in raw pixel form.

- a) Scroll down and view the cell titled **Preview the image data in raw pixel form**, and examine the code cell below it.
- b) Run the code cell.
- c) Examine the output.

```
[array([[[ 86,  85,  80],
       [ 65,  64,  59],
       [ 59,  58,  53],
       ...,
       [206, 205, 201],
       [206, 205, 201],
       [205, 204, 200]],

      [[ 70,  67,  62],
       [ 88,  85,  80],
       [ 85,  82,  77],
       ...,
       [206, 205, 201],
       [206, 205, 201],
       [206, 205, 201]],

      [[ 82,  79,  72],
       [ 81,  81,  71]]]
```

The result is a multi-dimensional array.

- Each element of the outer-most array (technically a Python list) is a separate image.
- The second outer-most array is a NumPy array read by `imread()` that contains the red, green, and blue (RGB) values for each pixel in each image.
- Each NumPy image array is divided into what can be thought of as rows and columns, where each intersection of row and column describes a pixel in that spot of the image. For example, `[86, 85, 80]` describes the pixel in the upper left-most corner of the first image.
- By default, `imread()` converts images into an integer-based RGB format like you see here. These values indicate the intensity of each of the three colors from 0 to 255. The RGB value `[86, 85, 80]` is a gray color, which you'll see shortly.
- The only exception to this is if `imread()` reads a PNG image. It automatically converts those to float values from 0 to 1 indicating the relative intensity of each color. Since all of the images in this dataset are JPEGs, you'll only see integer values.
- It can be difficult to interpret the nested arrays like this, so you'll view the data as a truncated data frame for a more tabular perspective.

- d) Scroll down and examine the next code cell.

```

1 # First image only.
2 img_samp = images[0]
3 print('Shape of first image: ' + str(img_samp.shape))
4
5 temp_list = []
6
7 # Prepare values to put into a data frame.
8 for i in img_samp:
9     temp_list.append(list(i))
10
11 df_img_samp = pd.DataFrame(temp_list)
12
13 # Retrieve first and last 3 rows and columns.
14 df_img_samp.iloc[np.r_[0:3, -3:0], np.r_[0:3, -3:0]]

```

This code will convert the first image's pixel array into a data frame.

- e) Run the code cell.
f) Examine the output.

Shape of first image: (534, 400, 3)						
	0	1	2	397	398	399
0	[86, 85, 80]	[65, 64, 59]	[59, 58, 53]	[206, 205, 201]	[206, 205, 201]	[205, 204, 200]
1	[70, 67, 62]	[88, 85, 80]	[85, 82, 77]	[206, 205, 201]	[206, 205, 201]	[206, 205, 201]
2	[82, 79, 72]	[84, 81, 74]	[81, 78, 71]	[206, 205, 201]	[206, 205, 201]	[206, 205, 201]
531	[135, 134, 130]	[140, 139, 135]	[145, 144, 140]	[179, 178, 176]	[178, 177, 175]	[178, 177, 175]
532	[134, 133, 129]	[138, 137, 133]	[144, 143, 139]	[178, 177, 175]	[178, 177, 175]	[178, 177, 175]
533	[133, 132, 128]	[137, 136, 132]	[143, 142, 138]	[178, 177, 175]	[177, 176, 174]	[177, 176, 174]

- The shape of the first image is 534 rows by 400 columns, with 3 RGB values for each cell.
- The data frame shows a truncated view of the image's pixel values. It displays the values for the 3 pixels in each corner. For example, rows 0 through 2 and columns 0 through 2 are the upper left-most corner; rows 531 through 533 and columns 0 through 2 are the lower left-most corner; and so on.
- Remember, this is a truncated view of the pixel data, so there are many more rows and columns in between the ones you see here.

- g) Scroll down and examine the next code cell.

```

1 # Check for different image sizes.
2 print('Shape of fourth image: ' + str(images[3].shape))

```

It's possible that all of the images are the same shape and size, but you'll check to make sure.

- h) Run the code cell.
i) Examine the output.

Shape of fourth image: (888, 400, 3)

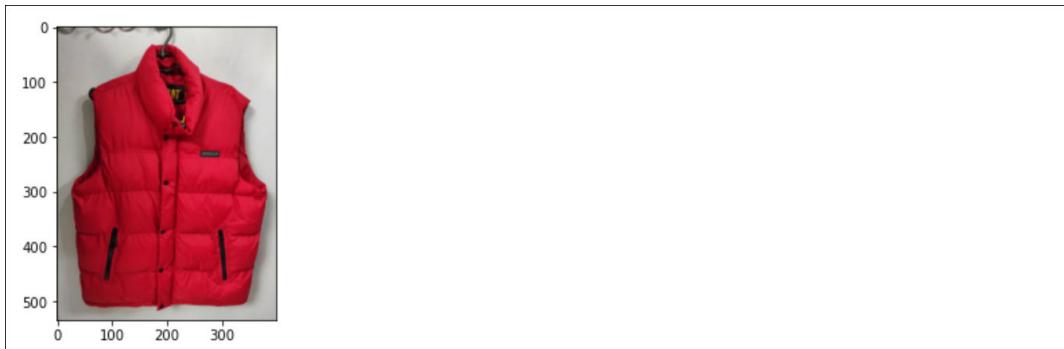
As it turns out, not all images in this dataset are the same size. The fourth image has 888 rows and 400 columns. You'll need to make the images a uniform shape and size before you use the data as input.

5. Display the images in visual form.

- a) Scroll down and view the cell titled **Display the images in visual form**, and examine the code cell below it.

This code uses Matplotlib's `imshow()` function. Since the image is just an array of pixel data, it can be plotted like any other data.

- Run the code cell.
- Examine the output.



The first image in the dataset, which you'll use as a sample going forward, is a red jacket (categorized as "outwear" in the dataset). The background of the photograph confirms the upper left-most gray pixel value mentioned earlier.

- Scroll down and examine the next code cell.

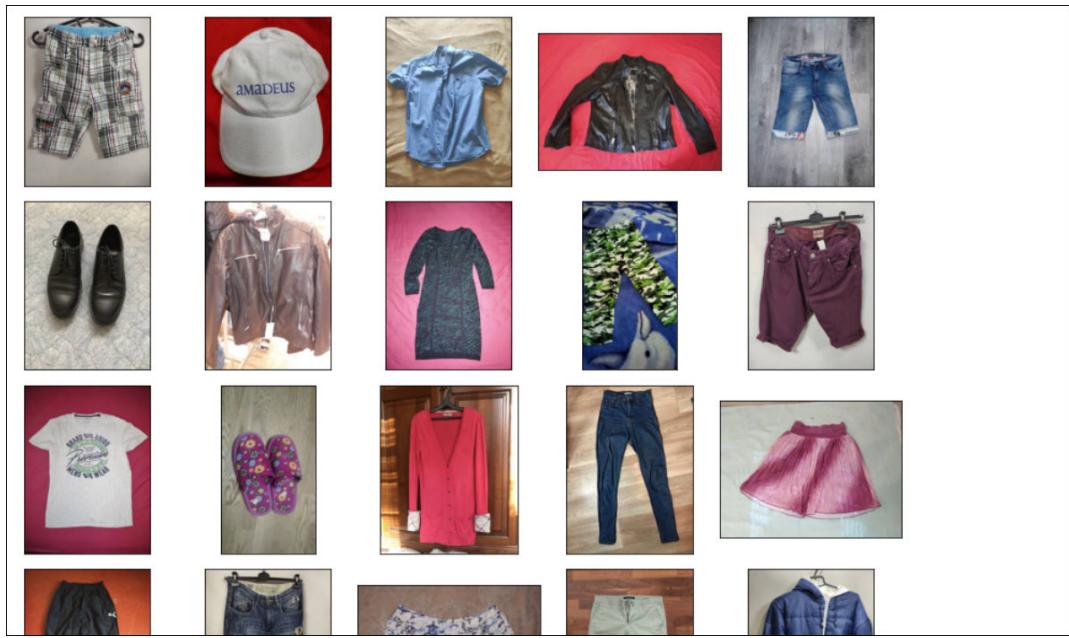
```

1 import random
2
3 # Get a better distribution of different types of clothing.
4 random.seed(1)
5 random.shuffle(images)
6
7 def plot_img_grid(images):
8     '''Plots a grid of the first 25 images from an array of image data.'''
9     fig, axes = plt.subplots(nrows = 5, ncols = 5, figsize = (12, 12))
10
11    for i, ax in zip(range(25), axes.flatten()):
12        ax.imshow(images[i], cmap = 'gray')
13
14    # Turn off axis ticks for readability.
15    for ax in axes.flatten():
16        ax.set_xticks([])
17        ax.set_yticks([])
18
19    fig.tight_layout()
20
21 plot_img_grid(images)
  
```

- Line 5 randomly shuffles the images so there's a better mix of each category of clothing.
- The function that starts on line 7, when called, will plot a grid of the first 25 images.

- Run the code cell.

- f) Examine the output.



You can see a mix of photographs of different types of clothing.

6. Convert the sample image to grayscale.

- Scroll down and view the cell titled **Convert the sample image to grayscale**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
array([[ 0.33275294,  0.2504      ,  0.22687059,  ...,  0.80362392,  0.80362392,
       0.79970235],
       [ 0.26383137,  0.33441961,  0.3226549 ,  ...,  0.80362392,  0.80362392,
       0.80362392],
       [ 0.31032471,  0.31816784,  0.30640314,  ...,  0.80362392,  0.80362392,
       0.80362392],
       ...,
       [ 0.52519255,  0.54480039,  0.56440824,  ...,  0.69830706,  0.69438549,
       0.69438549],
       [ 0.52127098,  0.53695725,  0.56048667,  ...,  0.69438549,  0.69438549,
       0.69438549],
       [ 0.51734941,  0.53303569,  0.5565651 ,  ...,  0.69438549,  0.69046392,
       0.69046392]])
```

The pixel array values for the grayscale conversion are printed. Notice that the values are floats instead of the integers you saw earlier. By default, scikit-image converts pixel data to float values. You'll convert these back to integer values since they're likely more familiar (and they match the results of the `imread()` function).

- Scroll down and examine the next code cell.

```
1 from skimage import img_as_ubyte
2
3 # Convert to integer format that matches imread() result.
4 img_samp_t = img_as_ubyte(rgb2gray(img_samp))
5 print('Shape of image: ' + str(img_samp_t.shape))
6 img_samp_t
```

The `img_as_ubyte()` function converts the data to an unsigned 8-bit integer format.

- Run the code cell.

- f) Examine the output.

```
Shape of image: (534, 400)
array([[ 85,  64,  58, ..., 205, 205, 204],
       [ 67,  85,  82, ..., 205, 205, 205],
       [ 79,  81,  78, ..., 205, 205, 205],
       ...,
       [134, 139, 144, ..., 178, 177, 177],
       [133, 137, 143, ..., 177, 177, 177],
       [132, 136, 142, ..., 177, 176, 176]], dtype=uint8)
```

- The shape of the image confirms the grayscale transformation; the third dimension is gone.
- The familiar 0–255 integer values for the image are printed.
- Now that the image is grayscale, the integer values are not referring to RGB, but simply to shade intensity. For example, the upper left-most pixel is now only represented by one value—85.

- g) Scroll down and examine the next code cell.

```
1 # Show the image.
2 plt.imshow(img_samp_t, cmap = 'gray');
```

This will show the actual image like before. However, images with intensity values are not inherently "gray" or any other color. They simply indicate the relevant intensity of a pixel's "color" at the given location. So, you need to apply a color map to the image if you actually want it to look a certain way. That's what the `cmap = 'gray'` argument does.



Note: Neural networks don't care what color map is used; this is just for display purposes.

- h) Run the code cell.

- i) Examine the output.

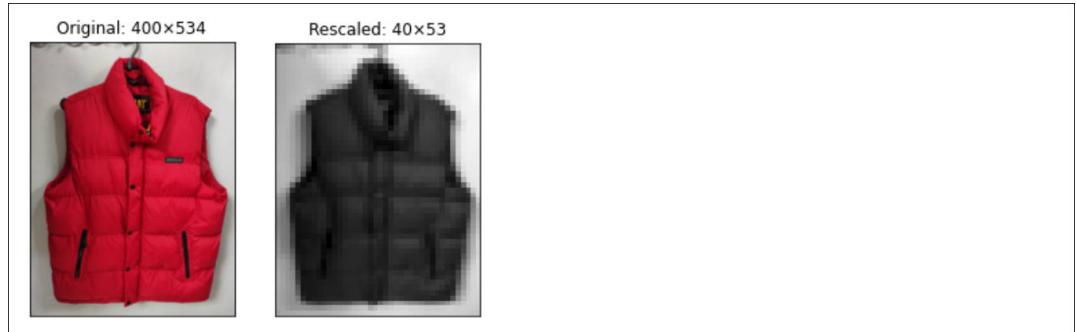


The RGB color information has been removed in favor of a single shade that varies in intensity. You can still identify the object, as should a neural network.

7. Scale the sample image down.

- Scroll down and view the cell titled **Scale the sample image down**, and examine the code cell below it.
 - Line 3 will reduce the grayscale image down to one tenth of its original size while preserving its aspect ratio.
 - The function that begins on line 5, when called, will plot the original image side by side with the image that has had transformations applied to it.
- Run the code cell.

- c) Examine the output.



- The image on the right has one tenth the pixels as the original image on the left.
- Pixel resolution is typically written as width \times height, which corresponds to columns by rows—the opposite of how tabular data is usually written.
- The scaled-down version is 40 pixels wide and 53 pixels tall.
- Because this is a scaling operation, the aspect ratio of the image is preserved.
- Even though the output has scaled the smaller image up so that it matches the original image, you can tell it is much lower in quality. This is because there are fewer pixels to display.
- Despite this loss of visual fidelity, a neural network should still be able to learn from this scaled-down image. And, because the image is smaller, it has less of a memory footprint—which can be a boon to performance.

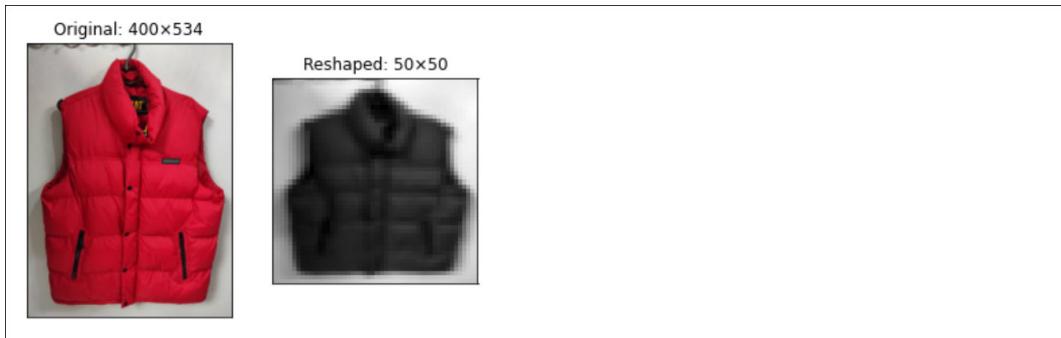
8. Expand the sample image to make it a square shape.

- a) Scroll down and view the cell titled **Expand the sample image to make it a square shape**, and examine the code cell below it.

This code will resize the image by fitting it into a square aspect ratio that has a 50×50 pixel resolution.

- b) Run the code cell.

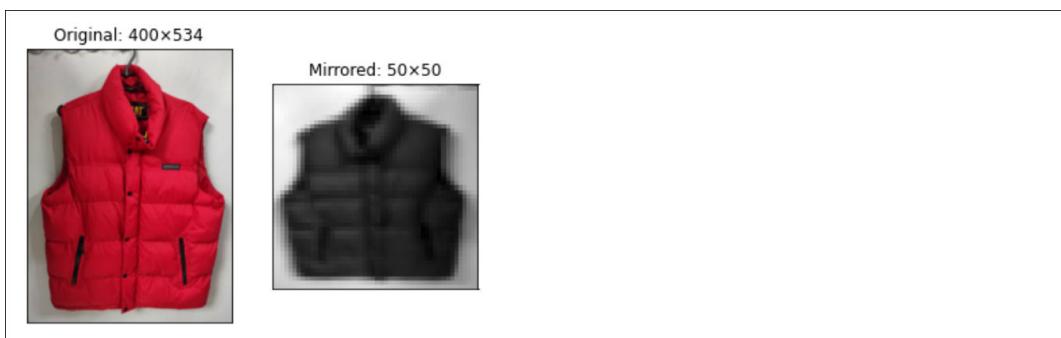
- c) Examine the output.



- Both the resolution and the aspect ratio have changed.
- The transformed image is now square, which neural networks tend to work better with.
- As a result of the resizing operation, the image has been squeezed, and the jacket appears wider than it normally does.
- Alternatively, you could have cropped the image to make it square, but this might have removed some useful information from the image. Part of the jacket would have been cut off from the image.
- Another alternative would be to pad the image with white or black pixels until it conforms to a square shape. The neural network might learn the wrong patterns from this irrelevant data, however.
- There's no one "correct" approach to reshaping all images. It often depends on the images themselves. For example, some of the other photographs have the actual clothing item in the center of the frame, with more background surrounding the item (e.g., the shoes in row 2, column 1 of the prior grid of images). It'd probably be easier to crop these images for a square shape. At the same time, you may not want to remove too much background, since your neural network may need to make decisions on new data that includes backgrounds.

9. Flip the sample image horizontally.

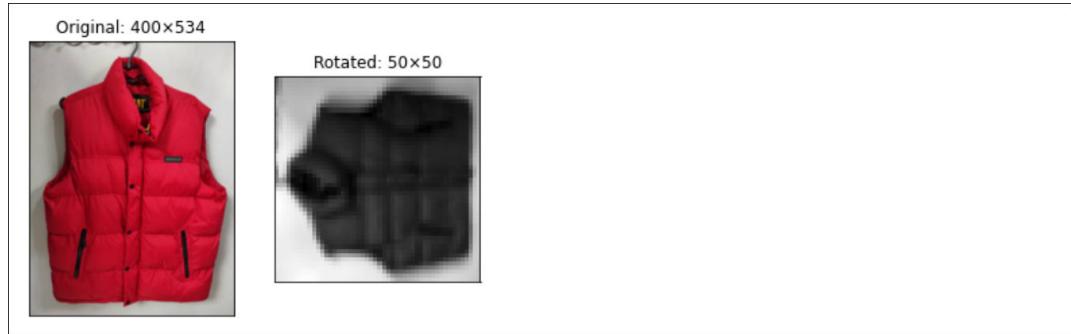
- Scroll down and view the cell titled **Flip the sample image horizontally**, and examine the code cell below it.
- Run the code cell.
- Examine the output.



- The jacket has been flipped horizontally, or "mirrored."
- This kind of perturbation might not have much of an impact with this particular data, since most clothing items are symmetrical. Still, there are subtle variations that could be affected by flipping the image. In this example, the collar in the original image is skewed to the right, so it might be a good idea to show the neural network an alternative image where the collar is skewed to the left.

10. Rotate the sample image by 90 degrees.

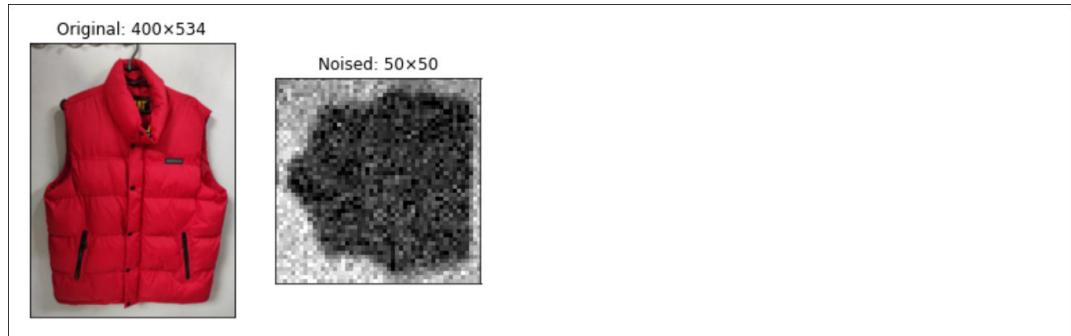
- Scroll down and view the cell titled **Rotate the sample image by 90 degrees**, and examine the code cell below it.
- Run the code cell.
- Examine the output.



- The image is rotated counterclockwise by 90 degrees.
- The neural network might need to make decisions based on images that are not in a typical orientation. This is common with photos taken using mobile phones. Even if it doesn't, seeing different orientations will likely help the neural network learn the true patterns in the image.

11. Add noise to the sample image.

- Scroll down and view the cell titled **Add noise to the sample image**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

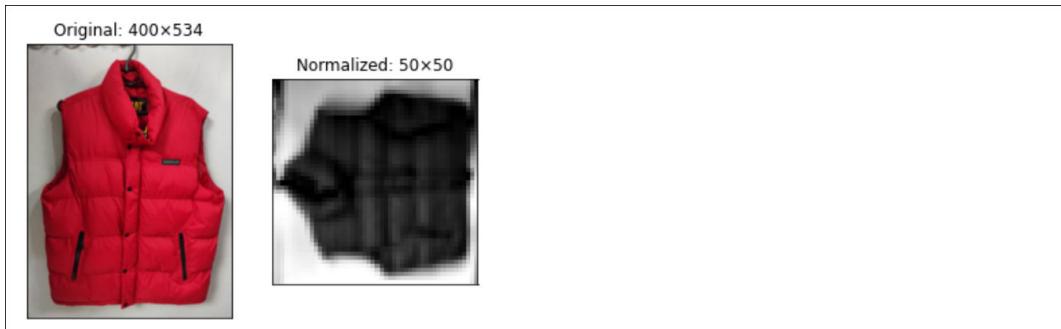


- Random "noise" was added, resulting in a fuzzy, grainy image.
- Not all photographs are taken cleanly, so it can be helpful to add noise to your input data. That way the neural network will be better at determining what detail is important in an image and what is not.
- Visually, the image is almost unrecognizable. A neural network might be able to do better than the human brain at identifying the jacket, but it could be negatively affected by low-quality data.
- Not all transformations will be worth doing. Rather than risk it, you'll discard this transformation.

12. Normalize the sample image.

- Scroll down and view the cell titled **Normalize the sample image**, and examine the code cell below it. The feature scaling code should look familiar.
- Run the code cell.

c) Examine the output.



- Compared to the image after it was first rotated, the effect of normalization is subtle from a visual perspective, but the image has changed slightly.
- Feature scaling the image can help emphasize the distribution of pixel values rather than the absolute values.
- You could continue to transform the image in different ways, but this is probably sufficient.

13. Transform all images in the dataset.

- Scroll down and view the cell titled **Transform all images in the dataset**, and examine the code cell below it.
 - Line 3 begins a `for` loop that will apply all of the prior transformations (except for adding noise) to each image in the dataset.
 - Lines 4, 5, and 6 convert the image to grayscale, scale the image down, and make the image square, respectively.
 - Lines 8 through 11 randomly flip the image horizontally, or do not flip it at all.
 - Lines 13 and 14 randomly rotate the image 90 degrees counterclockwise, 90 degrees clockwise, 180 degrees, or not at all.
 - Applying perturbations randomly can help reduce bias in the neural network. Otherwise, it might be too focused on specific transformations and be unable to generalize to new data.
 - Lines 16 and 17 normalize the image.
- Run the code cell.

- c) Examine the output.



The first 25 clothing images are shown again, this time with the transformations applied.

14. How else might you "augment" the data before you use as it as input to a neural network?

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 **Data Preparation - Fashion** tab in Firefox, but keep a tab open to **CAIP** in the file hierarchy.
-

Summary

In this lesson, you collected data, then began to prepare that data for machine learning by performing preliminary transformation tasks. You then applied more advanced feature engineering techniques to the data—a crucial step toward creating effective machine learning models.

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?

What kind of transformation and preprocessing tasks would be most applicable to the type of data you work with?



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

Training, Evaluating, and Tuning a Machine Learning Model

Lesson Time: 2 hours

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 an appropriate machine learning algorithm and model data using that algorithm.
- Evaluate a trained machine learning model to identify shortcomings or areas for improvement, then make these improvements by tuning various aspects of the model.

TOPIC A

Train a Machine Learning Model

To create a model that can make intelligent decisions, you need to input data to one of several machine learning algorithms.

Machine Learning Models

A **machine learning model**, usually just called a model, is a mathematical representation of data applied to an algorithm. It is similar to the idea of a traditional statistical model in that it puts forth assumptions about a population based on available sample data. Machine learning models differ in that they are specifically built upon machine learning algorithms (i.e., algorithms that can induce learning), and that they are always designed to output an estimation. The machine learning model is often the ultimate "product" of the machine learning process—it is the thing that you create to actually perform the necessary task, whatever that may be.

The general process for creating and using a model is as follows:

1. The practitioner selects an algorithm to use.
2. The practitioner (or an automated process) feeds data into the algorithm. This is called the *training* data, or training sample.
3. The algorithm outputs a model based on the training data.
4. The practitioner (or an automated process) feeds the model new data that it hasn't seen before.
5. The model makes an estimation (i.e., a prediction or some other decision) about this new data.

So, a model is an implementation of an algorithm. It is specific to whatever problem you are trying to solve, whereas an algorithm can generate many kinds of models if given different data or configured in different ways.

The **training** process in step 3 is of particular importance, as it actually creates the model. This is where the algorithm does all the hard work, taking your input data and performing many calculations on it. This is also called **fitting** a model.

You, the practitioner, don't do much during training, but everything you do leading up to training (and even things you do after) will have an impact on the success of that training. This is also the point in the process where computational power becomes a factor, as even when run on high-end hardware, certain models can take hours, days, or even weeks to finish training. Preparing to train a machine learning model is therefore crucial to get the results you want out of the process.

A Note on Terminology

It is common to think of machine learning models as making predictions, but not all conclusions a model can draw are predictive in the strictest sense. It's more precise to refer to the output of a model as an *estimation*. For example, the model might predict what height someone will be when they are older. The model could also make a non-predictive estimation, like what height someone is right now if only given their age and nationality. The model could also make proactive decisions, like suggesting a diet and exercise regimen to someone based on their height and age.

Machine Learning Algorithms

Before you train your initial machine learning model, you'll have to select an algorithm or algorithms that you'll use to produce the outcome you require. For example, if you needed to perform a classification task such as determining whether industrial equipment is at risk of failing based on various inputs—environmental conditions, average operation time, age, size, and so forth—you could use logistic regression, random forest, support-vector machines (SVMs), or one of several other algorithms.

The following figure places some example algorithms into the four main machine learning modes (supervised, unsupervised, semi-supervised, and reinforcement) and three outcomes/tasks (regression, classification, and clustering). This is not an exhaustive list of algorithms, just some of the most common ones used in the field.

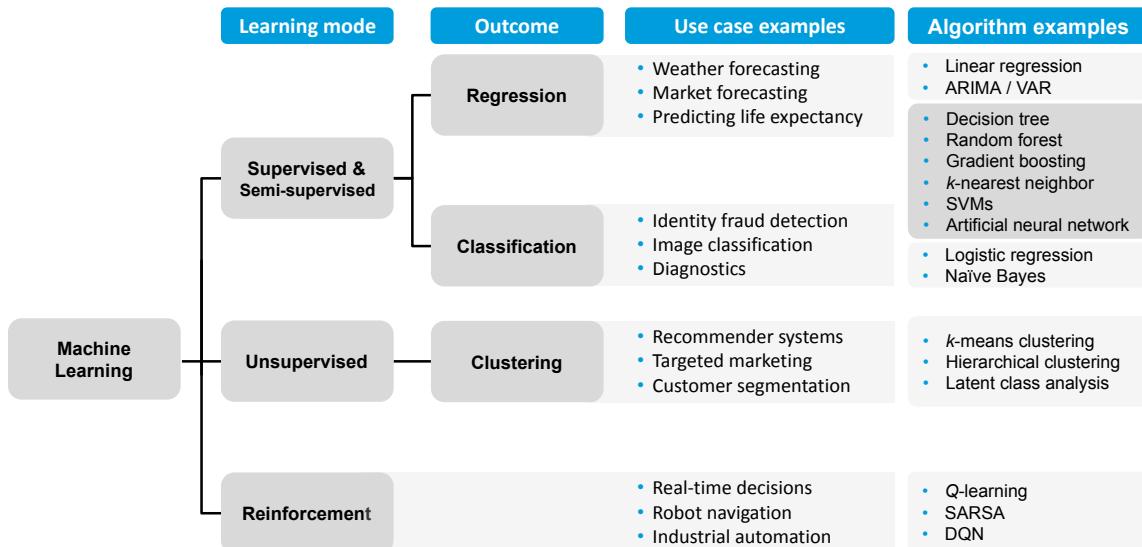


Figure 3-1: Example machine learning algorithms.

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. Also note that some of these terms refer to groups of related algorithms rather than specific algorithms. For example, there are multiple types of decision tree algorithms, but they tend to operate in a similar way and produce the same type of output.

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 machine learning practitioners may not be able to immediately select the best algorithm for a particular situation without some experimentation.

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 earlier in the data science process. Mapping algorithms to the outcomes/tasks that you expect is a good place to start, as you may be able to rule out some algorithms that don't apply to your situation. For example, logistic regression is primarily used for classification tasks, so you'll probably decide not to train a logistic regression model if you just need to forecast sales figures. However, even if the desired task is clear, there may still be many suitable algorithms. Why choose logistic regression instead of a decision tree, for instance? What about using a support-vector machines (SVMs) model instead?

There are some factors you can look for that differ among algorithms that perform the same basic type of task, even when you use the same datasets. These factors include:

- **Training speed.** Some algorithms take much longer to train a model than others even when you use the same dataset.
- **Model effectiveness.** Some algorithms produce models that perform better than others at their given task.
- **Data preparation requirements.** Some algorithms require more data preparation than others in order to be effective, or they require different kinds of preparation.

- **Complexity.** Some algorithms, especially those in deep learning, can be much more complex and difficult to work with than others.
- **Transparency/explainability/interpretability.** Some algorithms are more closed off or **black box** than others, meaning it is harder to determine why the model made a certain decision.
- **Availability.** Each programming library supports a selection of algorithms, and the library you use may not support the algorithm you're looking for.

Of course, if you have the capacity and the ability, you should consider training multiple models using different algorithms. That way you'll be able to compare each model and choose the one that suits your needs the best.

Bias and Variance

Machine learning models can (and will) encounter errors in training. The two major types of error that can arise in machine learning models are bias and variance.

Bias measures the difference between the model's estimations and the actual ground truth (labeled) values. A model with high bias is simplified to make the **target function** less complex, easier to understand, and easier for a machine to learn. In other words, it makes many assumptions about the target function. Some algorithms, such as linear regression, tend to err on the side of high bias. High-bias models tend to have lower estimative performance on complex problems that fail to meet the simplifying assumptions they have made.

Variance, which is standard deviation squared, measures the variability of the model's estimations. It 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 like decision trees are higher in variance than algorithms like linear regression.

Model Generalization

When you train a machine learning model, you'll typically want it to make a reasonably good estimation on any new datasets that it might encounter—beyond the dataset that was originally used to train it. This characteristic is called **generalization**. A model that generalizes well is effective beyond the original conditions it was trained under, and therefore can be applied across a spectrum of problems and still be relatively useful. Conversely, a model that does not generalize well is only useful in very limited circumstances.

Both bias and variance relate directly to the concept of generalization. A high amount of either will directly impact the model's ability to generalize.

- **High bias** leads to **underfitting**. An underfit model is too simple to be useful to new data, as it cannot derive pertinent information from the dataset. It simply cannot learn the appropriate patterns from which to make an effective estimation.
- **High variance** leads to **overfitting**. An overfit model is too complex and maps to the training data too tightly. It therefore will perform poorly on new, unseen datasets since it has failed to separate the signal from the noise and learn the true estimation pattern.

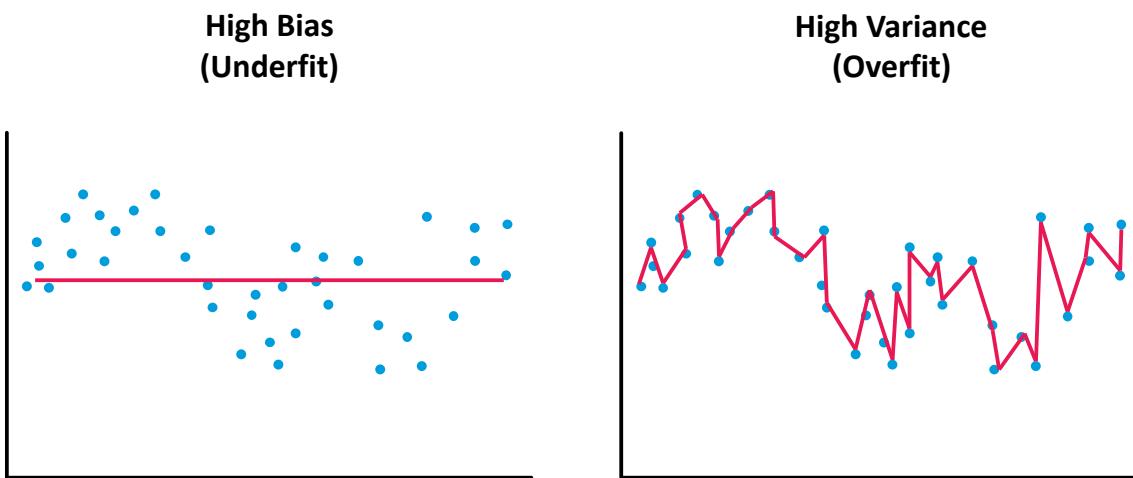


Figure 3-2: A model that underfits training data vs. a model that overfits that same data. Neither model is able to find the curve that is the true pattern.

Underfitting can usually be avoided by choosing more complex algorithms, increasing the number of features, and reducing the effect of techniques like regularization. Overfitting is usually more common, and there are several techniques, including cross-validation and regularization, that can reduce its effects. Adding more data examples and reducing the number of features (among other data preparation methods) can also help your model avoid overfitting.

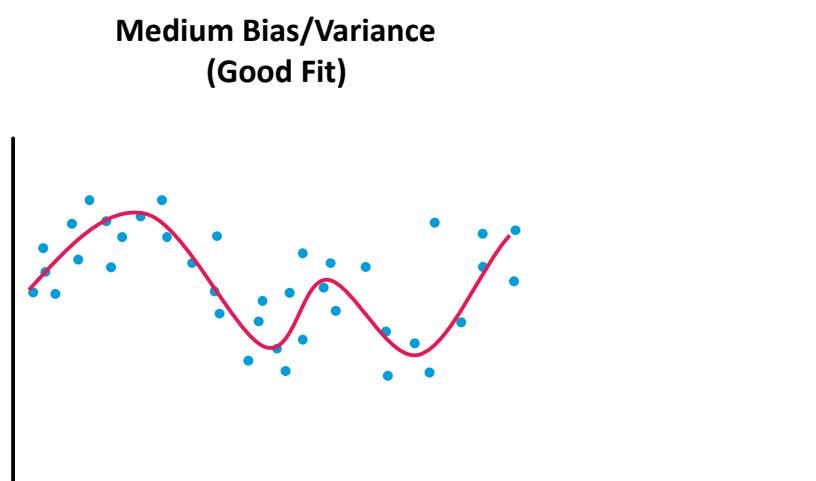


Figure 3-3: A model that achieves a good fit, neither underfitting nor overfitting to the training data.

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 bias and 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 tends to decrease. The sweet spot is the point where the smallest number of total errors are produced.

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 configure a machine learning model with controls that you can adjust. 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	Optimal complexity	More complex
Less likely to be influenced by relationships between features and target outputs	Effective in finding 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")

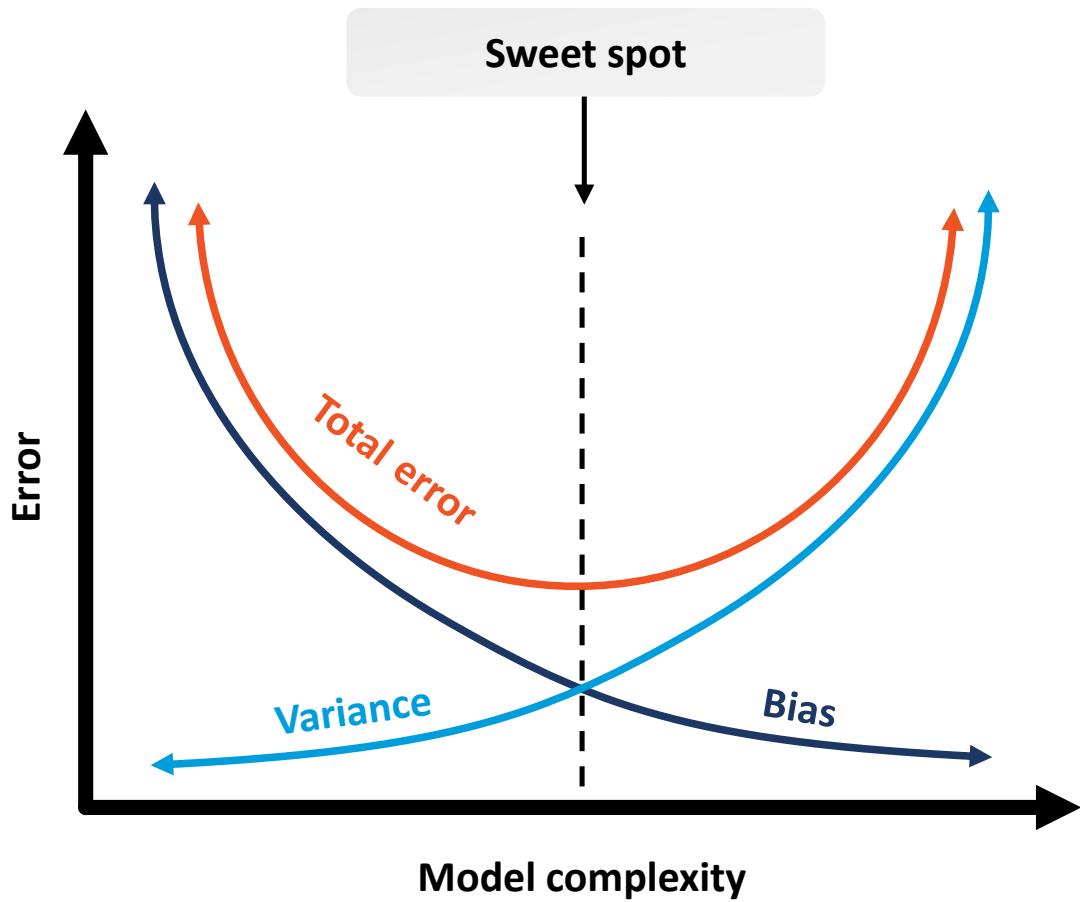


Figure 3-4: Finding a bias-variance balance that doesn't overfit or underfit the model.

The Holdout Method

The **holdout** method involves splitting up the original dataset so that you have multiple sets. It is essentially a process of sampling the data and one of the most basic techniques in addressing the problem of overfitting. You create two or three of these subsets, where each subset is used for a different purpose.

- **Training set:** Data you use to train the model. In supervised machine learning, the learning algorithm operates on the training set that includes target labels for the ground truth.
- **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 some settings of the algorithm. You will be familiar with the data in the validation set, and you will have access to the labels for it. This particular set is optional.

- **Test set:** This is another testing set with data that the model hasn't seen during training. If you're incorporating a validation set, you should not use the test set to tune performance, but rather only for testing the final fit of the model. That way, you can more effectively determine how well the model generalizes to new data it hasn't been tuned on.

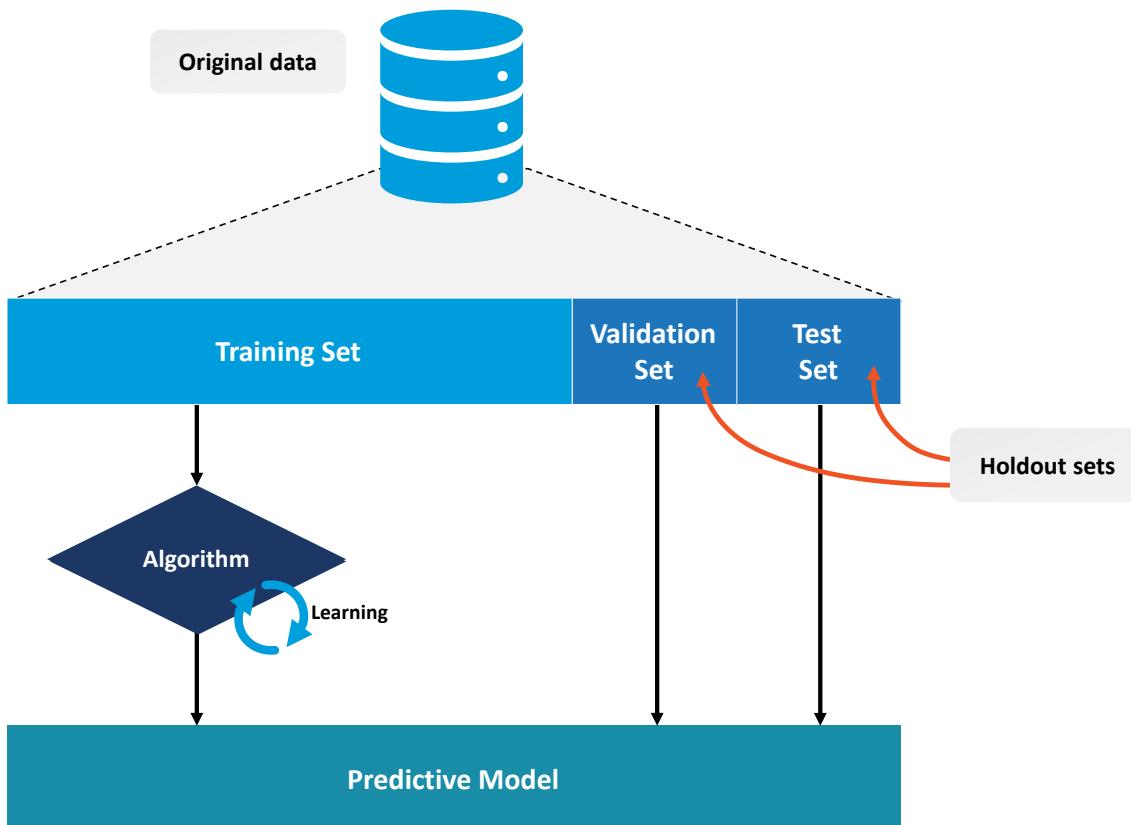


Figure 3–5: Using the holdout method to create a validation and testing set.

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 that you can adjust, since performance tuning is the primary reason for using a validation set. Also, fewer samples will be used in training if the data is split three ways, so you may choose to incorporate a validation set only if your dataset has a large number of samples.

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 between 70–80% training and 20–30% 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.

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 learned by the 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 a random forest, the maximum number of decision trees in the forest 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.

Parametric vs. Non-Parametric Algorithms

The algorithms that generate machine learning models can be defined as parametric or non-parametric.

- **Parametric** algorithms generate a fixed number of model parameters. No matter how much data you input to the algorithm for training, it will always produce a model with the same number of model parameters. Algorithms like linear regression and logistic regression are parametric.
- **Non-parametric** algorithms can generate a potentially infinite number of model parameters. The more data you input to the algorithm, the more model parameters it can generate. Algorithms like decision trees are non-parametric.

Guidelines for Training Machine Learning Models

Follow these guidelines when training machine learning models.



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

Train a Machine Learning Model

When training a machine learning model:

- **Select the right algorithm for the job.** One machine learning algorithm may be more effective at accomplishing a specific task than another. Or, consider selecting multiple algorithms and comparing the results to choose the right one.
- **Avoid unnecessary complexity.** In general, you should use the simplest or most computationally efficient method that meets your requirements.
- **Prioritize model generalization.** It's important for a model to be able to generalize well to new data. This usually means paying attention to factors like bias and variance and how they can restrict a model's ability to generalize.
- **Sample the input data.** Divide your dataset into training, testing, and (optionally) validation subsets. This can help mitigate problems in machine learning models such as overfitting.
- **Identify algorithm hyperparameters.** Different hyperparameters used with the same algorithm can drastically alter the effectiveness of your model. Be mindful of any hyperparameters that you are configuring before training, or that are configured for you by default.

Use Python to Split the Dataset

The scikit-learn library provides the `sklearn.model_selection.train_test_split()` function, which you can call to split a dataset into randomized training and testing subsets. Consider the following code statement:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7,  
random_state = 1)
```

- The first argument provides `x` as the dataset *without* the label data.
- The second argument provides `y` as *only* the label data.
- The `train_size` argument indicates that 70% of the data examples will appear in the training subset, and consequently, 30% will appear in the test subset. You can also use `test_size` to specify the inverse.
- The `random_state` argument provides a pseudorandom seed value to ensure the sampling output is the same every time you call the function with that seed. In this case, `1` is just an arbitrary value.
- The result of this call is that `X_train` will be your training features, `X_test` will be your test features, `y_train` will be your training labels, and `y_test` will be your testing labels.

ACTIVITY 3-1

Training a Machine Learning Model

Data Files

/home/student/CAIP/Training/Training - KC Housing.ipynb
 /home/student/CAIP/Training/housing_data/kc_house_data_prep.pickle
 /home/student/CAIP/Training/housing_data/kc_house_data_prep_norm.pickle

Before You Begin

Jupyter Notebook is open.

Scenario

After your data transformation and feature engineering efforts, the housing dataset is in pretty good shape. You're ready to start building a machine learning model to predict the price of a house given a set of features. In this activity, you'll start off simple by creating a simple regression model.

1. From Jupyter Notebook, select **CAIP/Training/Training - KC Housing.ipynb** to open it.
2. Import the relevant software libraries.
 - a) View the cell titled **Import software libraries**, and examine the code cell below it.
 - b) Run the code cell.
 - c) Verify that the version of Python is displayed, as are the versions of the other libraries that were imported.
3. Load the datasets.
 - a) Scroll down and view the cell titled **Load the datasets**, and examine the code cell below it.
 This code loads the pickle files you saved in the previous lesson into their own data frames.
 - `df` is the non-normalized data.
 - `df_norm` is the normalized data with PCA.
 - b) Run the code cell.
 - c) Examine the output.

```
Data files in this project: ['.ipynb_checkpoints', 'kc_house_data_prep.pickle', 'kc_house_data_prep_norm.pickle']
Loaded 21609 records from ./housing_data/kc_house_data_prep.pickle.
Loaded 21609 records from ./housing_data/kc_house_data_prep_norm.pickle.
```

21,609 records were loaded for both datasets.

4. Show correlations with `price`.
 - a) Scroll down and view the cell titled **Show correlations with price**, and examine the code cell below it.
 Before you begin training a model, it's worthwhile to determine how features correlate with one another, particularly how they correlate with the dependent variable. You may be able to drop some features that have high or low correlations.
 This code creates a matrix of cross correlations and then prints correlations with `price` only.



Note: You can also do this as part of the feature engineering process.

- b) Run the code cell.
- c) Examine the output.

Pearson correlations with price:

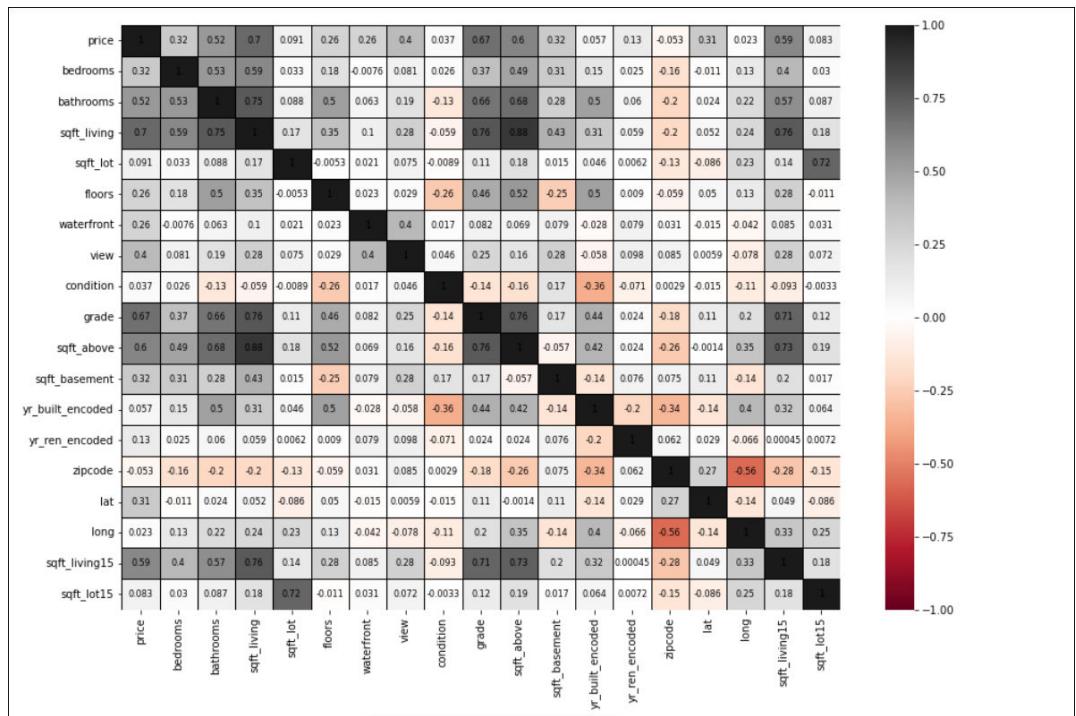
price	1.000000
sqft_living	0.698797
grade	0.673815
sqft_above	0.602144
sqft_living15	0.591702
bathrooms	0.523482
view	0.397864
sqft_basement	0.320141
bedrooms	0.316551
lat	0.312897
waterfront	0.264023
floors	0.259689
yr_ren_encoded	0.129226

- 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_lot` and `sqft_lot15`) or `yr_built`.
- You can disregard the perfect 1.000000 standard correlation value for `price`. (Of course, it correlates perfectly with itself.)

5. Analyze cross correlations.

- a) Scroll down and view the cell titled **Analyze cross correlations**, and examine the code cell below it.
 - Line 2 creates a list of rows and columns to be dropped from the correlation matrix. The `id` feature is not relevant to predicting `price`, and `yr_built` and `yr_renovated` have been encoded as different features, so the original ones are not necessary.
 - Lines 3 and 4 drop the rows and columns.
 - Lines 8 through 10 call the Seaborn library's `heatmap()` function, passing in the correlation values to be displayed in the heatmap.
- b) Run the code cell.

- c) Examine the output.



- 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 gray tones highlight values with high positive correlation, whereas lighter gray tones highlight values with low positive correlations.
- Darker orange tones highlight values with high negative correlation, whereas lighter orange tones highlight values with low negative correlations.
- The darkest values appear diagonally where features intersect with themselves.
- You will use these correlations to determine what features to remove before training the model.

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

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

This code performs the holdout method for splitting the dataset into training and testing sets.

- Line 6 assigns the `price` as the label, the dependent variable you're trying to predict.
- Lines 8 through 10 keep a subset of the features as training input. The following is a rationale for why certain features have been removed:
 - `id` and `date`—These value are primarily used to identify the house and have little to 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 0.60 correlation with `price`. So of these two features, `sqft_living` will be the more useful one to predict the `price`.
 - `sqft_lot` and `sqft_lot15`—Both of these features showed little correlation with any other features but themselves.
- Lines 13 through 15 call the `train_test_split()` method to split various columns from the original dataset into four separate datasets:

- `X_train` contains the independent variables that will be used to train the model.
- `X_test` contains the independent variables that will be used to test the model after it has been trained.
- `y_train` contains the dependent variable (`price`) for each row used to train the model.
- `y_test` contains the dependent variable (`price`) for each row used to test the model.



Note: `x_train/test` and `y_train/test` is a conventional naming scheme for dataset splits in machine learning. You could use more descriptive variables instead.

- 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.
- Lines 18 through 23 compare the number of rows and columns in the original data to those in the training and testing sets.

- a) Run the code cell.
- b) Examine the output.

```
Original set:      (21609, 25)
-----
Training features: (16206, 15)
Testing features:  (5403, 15)
Training labels:   (16206, 1)
Testing labels:    (5403, 1)
```

- The training features and labels include 16,206 of the original 21,609 records, whereas the testing features and labels include only 5,403 records.
- The features datasets include 15 of the original 25 columns, whereas the labels include only 1 column (for `price`).

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

8. Build a linear regression model (Round 1).

- a) Scroll down and view the cell titled **Build a linear regression model (Round 1)**, and examine the code cell below it.
 - Line 5 creates an object from the scikit-learn `LinearRegression()` class, which provides the algorithm that will create the model.
 - Line 8 starts the timer.
 - Line 9 trains (fits) 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) Run the code cell.

- c) Examine the output.

```
Model took 16.43 milliseconds to fit.
```



Note: Your time results will likely differ than the results in the screenshot.

- 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.

9. Compare the first 10 predictions to actual values (Round 1).

- a) Scroll down and view the cell titled **Compare the first 10 predictions to actual values (Round 1)**, and examine the code cell below it.

This code creates a function that will print a data frame of prediction results, compared to the actual price values for the test set.

- Line 2 creates a data frame for the results, using `n` (default of 10) as the number of results to store.
- Line 3 inserts the predictions on the test set after the actual price values.
- Line 4 formats the results to make them more readable.

- b) Run the code cell.
c) Scroll down and examine the next code cell.

```
1 # Make predictions on test set.
2 pred_prices = regressor.predict(X_test)
3
4 show_n_results(y_test, pred_prices)
```

- Line 2 calls the `predict()` function to generate predicted prices for the houses in the test dataset.
 - Line 4 calls the custom `show_n_results()` function to print the prediction results.
- d) Run the code cell.
e) Examine the output.

	price	price_pred
4983	\$450,000	\$550,843
10258	\$313,950	\$408,749
4045	\$704,300	\$495,485
21276	\$320,000	\$410,125
10498	\$410,000	\$357,474
2199	\$347,000	\$512,557
9216	\$1,100,000	\$1,084,409
9980	\$342,000	\$399,520
2323	\$552,500	\$440,790
6098	\$1,150,000	\$1,170,945

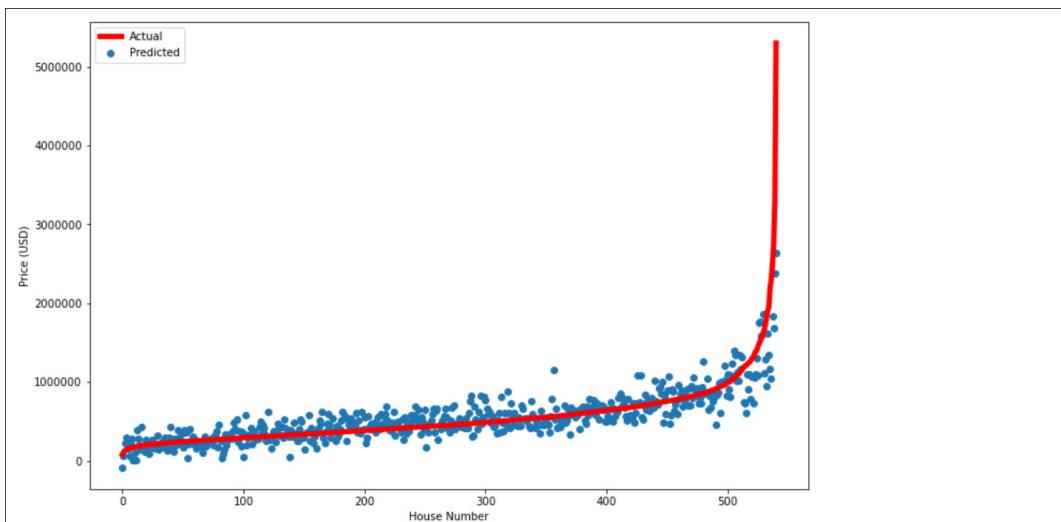
- The first ten actual prices can be compared with the predicted prices.
- While the predicted prices and actual prices correlate roughly, the current level of effectiveness may be inadequate. The model could use some improvement later on.

10. Plot the prediction residuals (Round 1).

- a) Scroll down and view the cell titled **Plot the prediction residuals (Round 1)**, and examine the code cell below it.

The `plot_residuals()` function is defined on lines 1 through 21. This function generates a combination line/scatter chart that will compare predicted prices (shown as dot markers) to actual prices (plotted in a line). This is also called a residual plot, since it plots the residuals (differences) between the actual and predicted values.

- b) Run the code cell.
c) Examine the output.



- The chart shows a red line representing actual prices and blue dots showing predicted prices.
- The closer a dot is to the line, the closer the prediction comes to the actual price.
- 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 cluster more closely to the red line.

11. Evaluate the regression model's score (Round 1).

- a) Scroll down and view the cell titled **Evaluate the regression model's score (Round 1)**, and examine the code cell below it.
- 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. It compares those predictions to the actual prices of those houses.
 - The `score()` function uses an appropriate algorithm to rate the performance of the regression model. The best possible score that can be returned is 1. The score can also be negative.
 - Line 3 outputs the score.
- b) Run the code cell.
c) Examine the output.

```
Score: 0.68991
```

You may be able to make some adjustments to improve on this score. For now, you've successfully trained a preliminary model from this data.

12. Keep this notebook open.

TOPIC B

Evaluate and Tune a Machine Learning Model

After training a machine learning model, you'll need to assess how well it performs. That way you can make continual improvements to the model until it reaches a level of effectiveness that meets your business needs.

Iterative Tuning

As you work with machine learning models, you must constantly attend to various challenges that will lead to ineffective and faulty results. Or, results that don't quite measure up to your expectations. After all, machine learning is probabilistic. 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 or decisions 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.

There are many ways to tune a machine learning model, several of which are discussed throughout the rest of this course. Most of them involve training and retraining the model so that its skill gradually increases until meeting the threshold that satisfies your requirements. Likewise, there are many ways to measure model skill that you'll learn about. The key takeaway at this point is understanding the overall tuning process, which tends to follow this pattern:

1. Train the model.
2. Test the model's basic purpose (e.g., its ability to predict something).
3. Adjust some aspects of the model.
4. Evaluate the model using various metrics.
5. Adjust the model again.
6. Evaluate the model again to see if improvements have been made.
7. Repeat as many times as necessary.

Evaluation Metrics

In machine learning, *evaluation metrics* are used to assess the skill, performance, and characteristics of a model. They do this by measuring some aspect of a model and/or its results. There are many such evaluation metrics, each one based on a different measurement. You're not limited to using just one metric, but each type of machine learning outcome (regression, classification, clustering, etc.) has its own metrics. So if you're training a classification model, it won't do much good to use a clustering metric like silhouette analysis.

Evaluation metrics are crucial to developing any type of machine learning model, regardless of whether the model is supervised or unsupervised. Supervised metrics tend to evaluate the estimations a model makes on the validation and/or test sets. Since these sets are labeled and were not used to train the model, you can find out how closely the model gets to the ground truth. Metrics usually report this level of performance in the aggregate—in other words, you won't be evaluating how the model performs on a single data example, but all data examples in the set

combined. Unsupervised metrics tend to look at the model's characteristics, as there are no labels with which to measure performance.

Evaluating model performance is important for a couple of major reasons. First, you can determine whether or not you are "satisfied" with the model according to whatever expectations you've established or business needs you've been tasked with fulfilling. If the model doesn't meet the desired level of performance, you may abandon it. Alternatively, the other reason to use evaluation metrics is to inform the tuning process. Rather than tune hyperparameters at random and see what happens, you'll be able to tune those hyperparameters in order to maximize one or more metrics.

Goodhart's Law

Goodhart's Law, named after economist Charles Goodhart, can essentially be paraphrased as: "When a measure becomes a target, it ceases to be a good measure." In other words, if you focus too much on achieving good results for a specific measurement, then the measurement itself becomes the goal. If you attempt to exert too much control over a system through one metric, this could lead to a false sense of accomplishment. In the field of machine learning, you might optimize a model so that it performs extremely well on one metric, but it might also perform poorly on other, equally important metrics. The model's overall skill could suffer as a result. So, as you learn about the various metrics in machine learning, keep in mind that there is no one objectively "perfect" metric for every circumstance, and that you may need to consider multiple metrics in the evaluation process.

Learning Curves

A **learning curve** is a method of visually comparing the number of data examples to the changing score of some metric. Along the x-axis is the number of training examples, and along the y-axis is whatever metric you're evaluating the model with. Two points are plotted: one for the training score (i.e., the model performs estimations on the training data after being trained on that data), and one for the validation score (typically using cross-validation). These scores are computed for several different sizes of the input data, then all of the points are connected via a line.

The purpose of a learning curve is twofold: it can help you determine whether adding more examples to the training dataset is likely to increase the model's score, and it can help you determine whether the model is more prone to errors of bias or errors of variance. For determining the effect of more data examples, you're looking for both score lines to converge. If they do converge at some sample size, then that means the score is not really changing between training and validation, implying that adding more data examples won't improve the model. If the lines don't yet converge, it implies that there's still room to improve the model by adding more data.

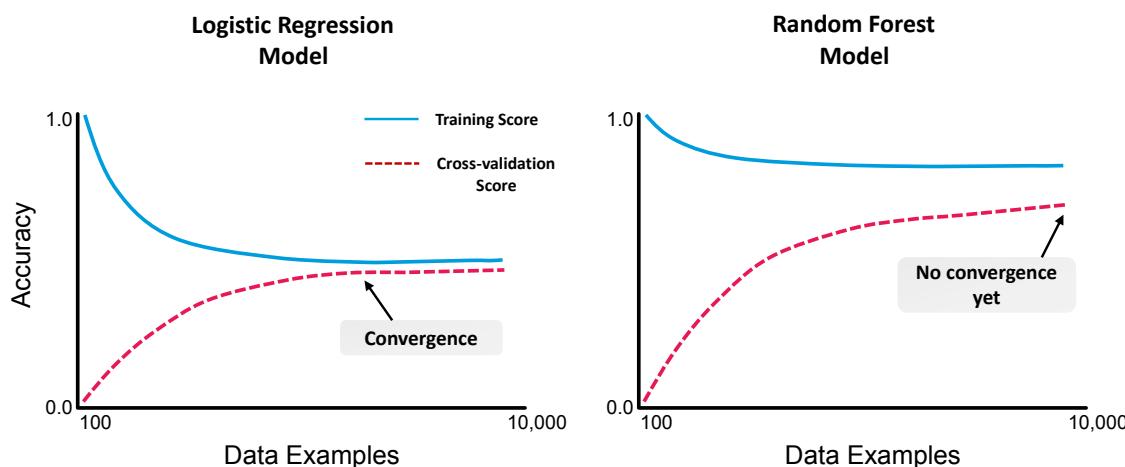


Figure 3-6: Comparing how the number of data examples impacts the skill of two different models.

In the figure, the logistic regression classifier on the left converges at around 6,000 data examples and its accuracy stays constant after that. It therefore won't benefit from more data. On the other hand, the random forest model on the right has a validation score that is still well below the training score after 10,000 examples, so it could benefit from more data.

Note that the training score is almost always going to be higher than the validation score, assuming that the "score" is any metric that evaluates how *well* the model performs. That's because the model is making estimations about the same data it learned from, so naturally it's going to be better at estimating that data. Higher overall training scores indicate low bias, whereas low training scores indicate high bias. Larger gaps between training and validation scores indicate high variance, whereas small gaps indicate low variance. Recall that high bias can lead to underfitting, and high variance can lead to overfitting. So, you can use learning curves to help you identify these issues in your models.

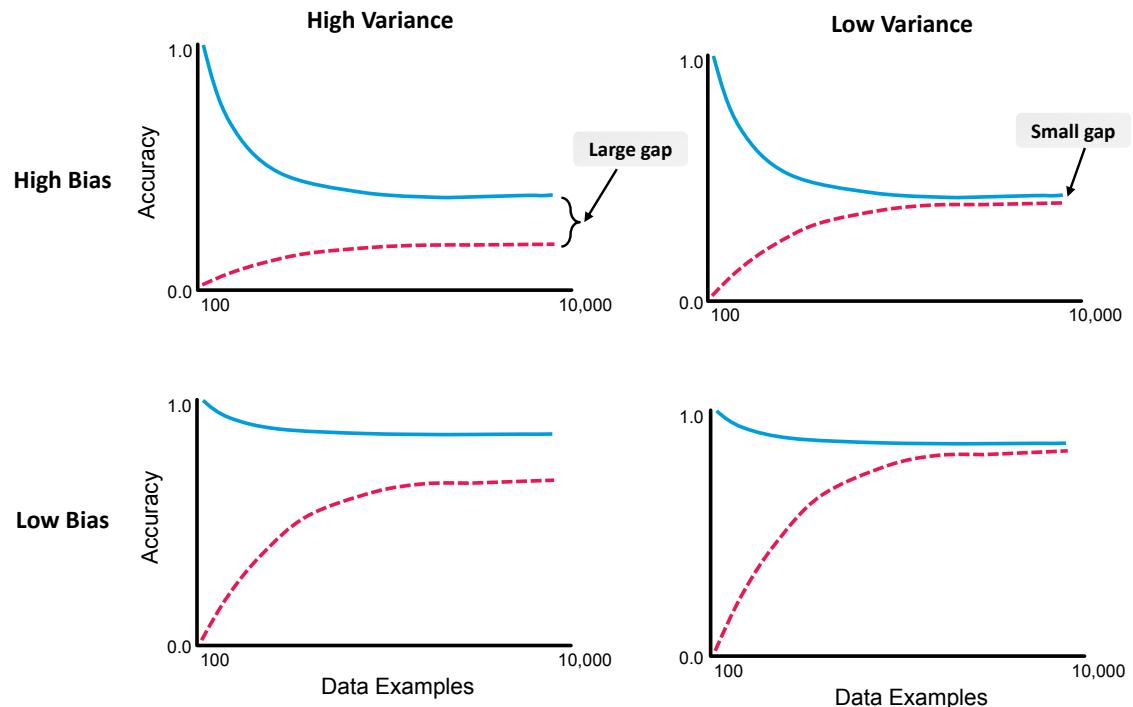


Figure 3-7: How bias and variance issues appear on a learning curve.



Note: These figures depict learning curves for classification models. Learning curves can evaluate any supervised model. In the case of regression models, they might be measuring error, so the training error in that case will always be lower than the validation error. High errors indicate high bias, and vice versa.

Model Optimization

There are various approaches to optimizing a machine learning model. Some are focused on generating the best estimation results for a given metric, whereas others are more concerned with the inner workings of the model. The following list summarizes some of the most important factors in optimizing models:

- Cross-validating training data.
- Regularizing model parameters.
- Tuning hyperparameters.
- Adjusting the complexity of model structure.
- Reducing training run time.

Most of these techniques are discussed in more depth in this topic.

Cross-Validation

Cross-validation (also called rotation estimation or out-of-sample testing) is a technique for partitioning data in order to improve a model's ability to generalize to new data. There are actually several cross-validation techniques, the most basic of which is the holdout method.

Cross-Validation Method	Description
<i>k-fold cross-validation</i>	<p>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.</p> <p>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.</p>
<i>stratified k-fold cross-validation</i>	<p>Similar to k-fold cross-validation, this alternative 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 0 and 70% is in class 1, then 30% of the data in each fold will be in class 0, and 70% will be in class 1.</p> <p>The stratification method is therefore most suitable in cases of class imbalance.</p>
<i>Leave-p-out cross-validation (LPOCV)</i>	<p>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.</p>
<i>Leave-one-out cross-validation (LOOCV)</i>	<p>The same as LPOCV when p is set to 1. It's common to do this because as p increases, the amount of trial combinations increases dramatically, leading to significant performance issues.</p>

Regularization

One of the methods you can use to tune a model is regularization. **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 estimations 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.

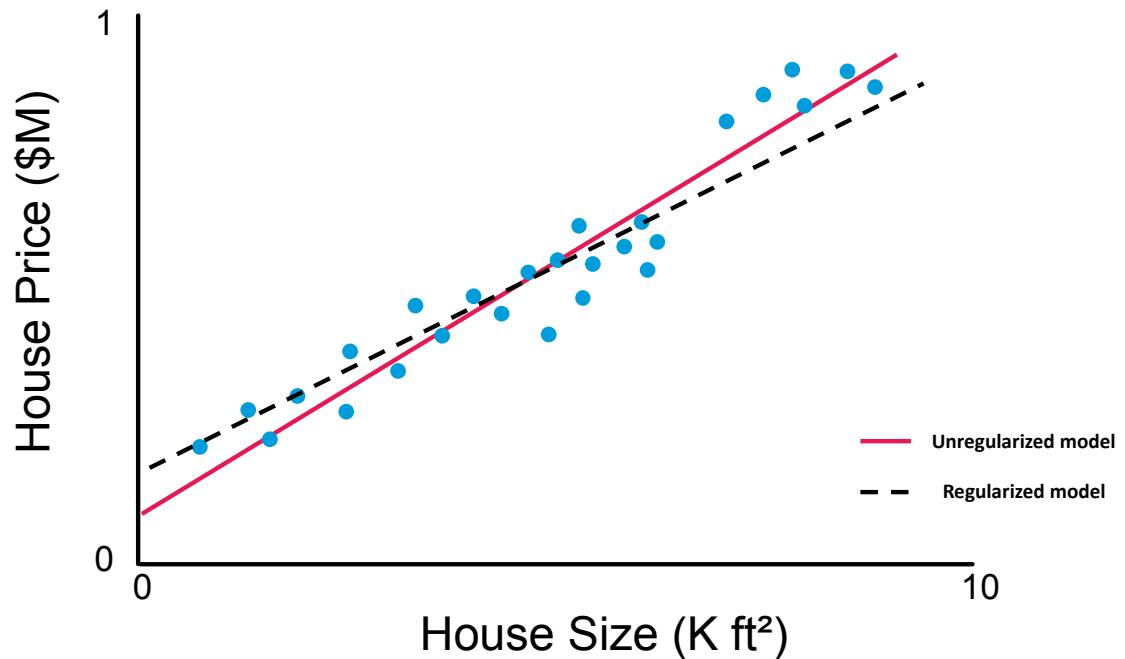


Figure 3–8: A linear model without regularization (solid red line) vs. with regularization (dashed black line). The former fits the training data better, but the latter generalizes to new data better.

Model Structure

The structure of the model is an optimization factor that is concerned more with a model's inner workings than its raw estimative skill. Recall that one of the criteria for selecting an algorithm is its complexity. Simple algorithms can produce simple models. In this case, "simple" means a model that has relatively few model parameters. You can achieve simplicity by selecting a parametric algorithm so that the number of parameters are knowable and set beforehand; and you can also reduce the number of parameters by likewise reducing the dimensionality of your training data. However, keep in mind that simple does not always imply better.

Simple models may be optimal in that they are easy to understand and explain, but complex models are optimal in that they may be more skillful at a given task. Neural networks are the foremost examples of complex models that can achieve great results. If feasible, adding more dimensionality to data fed into non-parametric algorithms can also increase model complexity. Bear in mind that complex models can be difficult, if not impossible, to adequately explain; they can also be harder to implement consistently if there is not enough robust data to draw from.

Model Training Time

The run time of model training is also an important concern. This is another criterion of algorithm selection that can be vital in optimizing the model. It's also closely tied together with the model's structure, as complex models usually take more time to train than simple ones, and vice versa. So, you need to consider both factors together, making sure not to optimize one without also seeing how it affects the other.

You can reduce training time in a few ways.

- **Simplify the model.** The techniques mentioned previously apply here.
- **Parallelize training.** *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 configurations 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 train a model.
- **Train using GPU hardware.** For deep learning tasks, *graphics processing units (GPUs)* are often better than *central processing units (CPUs)*. 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. This is a requirement of both graphics processing and deep learning, thus the reason why GPUs are popular in the field of AI. GPUs can also be parallelized for even greater training efficiency.
- **Offload training to the cloud.** Rather than purchasing GPU cards and setting up your own hardware, you can provision similar capabilities using a cloud hosting service such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform. All of these major services enable you to create virtual instances of GPUs and related hardware, which you can easily scale up and scale down as needed. However, the costs of using these services can add up quickly, so you may want to experiment with local hardware first, then move your projects to the cloud when you need to scale massively.



Note: Depending on how you deploy your model to a production environment, it may only need to be trained once or every so often. In these cases, training time may be less of a concern.

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 the 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.

Guidelines for Evaluating and Tuning Machine Learning Models

Follow these guidelines when evaluating and tuning machine learning models.

Evaluate and Tune a Machine Learning Model

When evaluating and tuning machine learning models:

- **Tune models iteratively.** Do not simply train a model and consider it finished. You need to refine it using a process of evaluation, tuning, and retraining. You may need to go through this process multiple times before you get optimal results.

- **Use generalization techniques to reduce overfitting.** Techniques like cross-validation and regularization are both common ways of ensuring a model avoids overfitting to the training data.
- **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.
- **Consider the optimal complexity of your models.** Make sure the model is not too complex if training time and explainability are of paramount importance; but not too simple if the problem you need solving requires complex decision-making logic.
- **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.
- **Select the right hardware.** Make sure the systems you use are appropriate for the tasks you need to accomplish, and they are appropriately configured.
- **Use models in combination.** Whenever feasible, try to combine the output of multiple models when making intelligent business decisions.

Use Python to Perform Cross-Validation

The `sklearn.model_selection` module has several classes for performing cross-validation. The following classes only perform data splitting:

- `kf = sklearn.model_selection.KFold(n_splits = 5, shuffle = True)` —Creates a cross-validation object that will split the training data into five folds. The `shuffle = True` argument indicates that the data will be shuffled randomly before being split.
- `kf.split(x)` —Uses the `kf` object to perform cross-validation on the `x` dataset, returning the indices with which to split the data into train and test subsets.
- `skf = sklearn.model_selection.StratifiedKFold(n_splits = 5, shuffle = True)` —Creates an object that will return stratified folds based on the percentages of each class provided as labels. The data examples from each class will be shuffled before splitting.
- `skf.split(x, y)` —Performs cross-validation on the `x` data, while using the data in `y` as the class labels for stratification.
- `lpo cv = sklearn.model_selection.LeavePOut(p = 3)` —Creates a cross-validation object using LPOCV. In this case, the number of examples in the test set is 3.
- `loocv = sklearn.model_selection.LeaveOneOut()` —Creates an LOOCV object. You can also set `p = 1` in the `LeavePOut()` class to achieve the same effect.
- You can use the same `split()` method to retrieve the split indices for LPOCV/LOOCV.

You can also perform splitting, fitting, and evaluation all in one function:

- `cv = sklearn.model_selection.cross_validate(estimator, X, y, cv = 5)` —Performs cross-validation by fitting a model object (`estimator`) with training data (`X`) and labels (`y`). The `cv` argument takes a cross-validation generator as input. By default, specifying an integer will use `KFold()` or `StratifiedKFold()` with the integer as the number of folds.
- The `cv` object is a dictionary that contains results from the cross-validation process, including the time it took to fit each split, the time it took to score each split, and the actual score values for each split. If you specify `return_estimator = True`, you can also retrieve the models that were fit for each data split.

Ethical Considerations in the Training Process

The training, tuning, and evaluation process is subject to some of the same ethical concerns as prior steps of the machine learning workflow. There are also some new concerns to be aware of.

Ethical Issue	Considerations
Conclusion bias	<p>In this case, bias doesn't just refer to social biases (although that is also a potential effect); it refers to a model drawing any type conclusion and making any type of decision that is inaccurate or otherwise of poor quality.</p>
	<p>For example, failing to tune the proper hyperparameters, or failing to evaluate the model based on several important metrics (recall Goodhart's Law) can lead you to release a model that appears satisfactory, but is actually flawed. If the model makes critical decisions, such as those in the medical field, it can cause more harm than good.</p>
Conflicting interests	<p>According to privacy legislation, the user reigns supreme. According to machine learning, the model reigns supreme. If you try to serve both interests, you may find that they are at odds with one another. This is because privacy is best upheld by minimizing data usage, whereas models are usually optimized by including as much useful data as possible.</p>
	<p>For example, if you want to build a model that can anticipate which products your customers are likely to buy next so you can make targeted recommendations, your model might be exceptional if it trained on every bit of personal information about each customer. But if this data were compromised, it would cause significant harm to the users, and major legal repercussions to your organization.</p>
Black box models	<p>The black box problem is a significant obstacle in machine learning. Even machine learning experts may face a situation where they cannot adequately explain why a model makes the decisions it does. This negatively affects the transparency of the product, which poses a problem for both the user and the developer.</p>
	<p>For example, a model may be used to profile people, such as analyzing their behavior, preferences, identity, appearance, and so on. Since there are no well-defined rules in place to prevent a black box model from making potentially life-altering decisions, such a model can be a risk that is difficult to justify.</p>

Guidelines for Addressing Ethical Risks in the Training Process

Use the following guidelines when addressing ethical risks in the model training process.

Guidelines for Addressing Ethical Risks in the Training Process

To address ethical risks in the training process:

- **Conclusion bias**
 - Identify as many factors that go into training your model as possible—opportunities for regularization, cross-validation, hyperparameter optimization, etc.
 - Time and resources permitting, perform many training and tuning sessions across multiple different configurations to see how your results change.
 - Identify the evaluation metrics that are most relevant to your model and the problem it is trying to solve; but don't become hyperfocused on solving just one or two that seem like the best.
 - Brainstorm ways in which your model may draw the wrong conclusions, and what effect those wrong conclusions will have.

- Consider how attempts to optimize training time and model simplicity/complexity may lead to problems when the model is applied to real-world scenarios.
- Evaluate your model's real-world, in-production performance on a regular basis so you can identify major issues as early as possible.
- **Conflicting interests**
 - Be aware of any privacy legislation your organization is subject to, as well as the guidelines and requirements set out by that legislation concerning how private data should and should not be used.
 - When training on sensitive user data, err on the side of the user being more important than the model.
 - Open a dialog with your users so you can solicit their concerns about how a model might use their data.
 - Recognize that your model does not need to be perfect, just good enough to do the job.
- **Black box models**
 - Whenever possible, train simple models as well as complex ones, in case the simple models perform just as well.
 - If explainability is more important to your business needs than model skill, consider selecting an inherently explainable algorithm, like decision trees.
 - Use tools like Shapley Additive Explanations (SHAP) and ELI5 (both Python libraries) that can reveal feature importances and other interpretability factors that may not be obvious through normal training approaches.
 - Look into explainable AI solutions, also called XAI, that may be able to help you avoid the black box problem.

ACTIVITY 3–2

Evaluating and Tuning Machine Learning Models

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/Data Preparation/Data Preparation - KC 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 PCA data into training and testing sets and labels**. Select **Cell→Run All Above**.

Scenario

You've trained an initial model on the real estate data, but your job isn't done. Machine learning is all about making adjustments and fine tuning your models to get the best possible results. So, in this activity, you'll try some other approaches to training models to see if you can improve their predictive capabilities.

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

- Scroll down and view the cell titled **Split the PCA data into training and testing sets and labels**, and examine the code cell below it.

This code cell performs the same basic train/test split operation as before, but this time on just the PCA columns. The idea is to see whether or not the reduced dataset can produce better results than the full dataset, or at least comparable results.

The `price` variable will still be used as the label.

- Run the code cell.
- Examine the output.

```
Original set:      (21609, 27)
-----
Training features: (16206, 2)
Testing features:  (5403, 2)
Training labels:   (16206, 1)
Testing labels:    (5403, 1)
```

The same split of data was produced—75% training, 25% testing.

2. Build a linear regression model (Round 2).

- Scroll down and view the cell titled **Build a linear regression model (Round 2)**, and examine the code cell below it.

This code is essentially the same as before. The same linear regression algorithm is used to create the model.

- Run the code cell.

- c) Examine the output.

```
Model took 9.85 milliseconds to fit.
```

The model was a little bit faster to train, likely due to the reduced number of features. The effect would be more noticeable on much larger datasets.

3. Compare the first 10 predictions to actual values (Round 2).

- a) Scroll down and view the cell titled **Compare the first 10 predictions to actual values (Round 2)**, and examine the code cell below it.

This code makes predictions on the PCA test set and obtains prediction results.

- b) Run the code cell.
c) Examine the output.

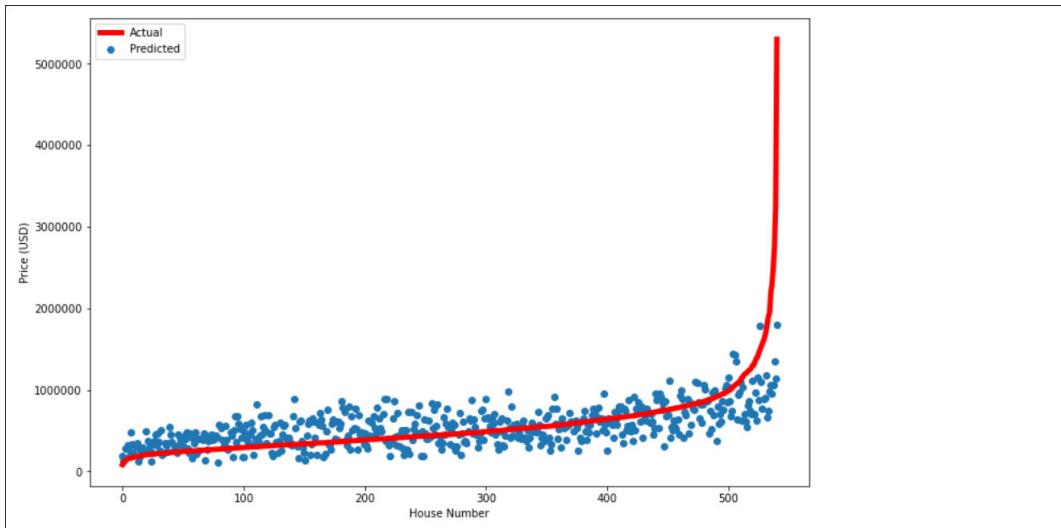
	price	price_pred
4982	\$450,000	\$697,333
10254	\$313,950	\$336,451
4044	\$704,300	\$447,219
21267	\$320,000	\$470,153
10494	\$410,000	\$492,128
2199	\$347,000	\$701,287
9213	\$1,100,000	\$927,310
9976	\$342,000	\$722,146
2323	\$552,500	\$267,302
6097	\$1,150,000	\$841,164

The first ten predictions are shown next to their actual values. The predictions are clearly different than the original model, though reviewing the residuals and score will perhaps be more illuminating.

4. Plot the prediction results (Round 2).

- a) Scroll down and view the cell titled **Plot the prediction results (Round 2)**, and examine the code cell below it.
b) Run the code cell.

- c) Examine the output.



The dots appear farther apart from the line than in the original linear regression model. It's possible that this model performed worse.

5. Evaluate the regression model's score (Round 2).

- Scroll down and view the cell titled **Evaluate the regression model's score (Round 2)**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

Score: 0.4516

This confirms that the PCA-based model performed worse than the one based on the full dataset. It's important to understand that not every change you make to the data or the model will improve its performance, even if the change seems worthwhile. This is a reality of experimentation. In this case, the data was likely simplified too much as a result of decomposing to two features, losing some vital information in the process.

Thankfully, there are other approaches to the tuning process you can try.



Note: In some cases, you may be fine with a model performing slightly worse if it means that training time is significantly reduced, or if the model itself is simplified and easier to work with.

6. Try a different algorithm.

- Scroll down and view the cell titled **Try a different algorithm**, and examine the code cell below it.

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 algorithm that employs a combination of regression algorithms to come up with the best fit.

You'll be training this algorithm on the full dataset, rather than the PCA data.

- Run the code cell.



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

- c) Examine the output.

```
Model took 6.70 seconds to fit.
Score: 0.88183
```

- 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 certain metrics outweigh the need for producing quick results.
- d) Scroll down and examine the next code cell.

```
1 # Make predictions on test set.
2 pred_prices_forest = forest.predict(X_test)
3
4 show_n_results(y_test, pred_prices_forest)
```

- e) Run the code cell.
f) Examine the output.

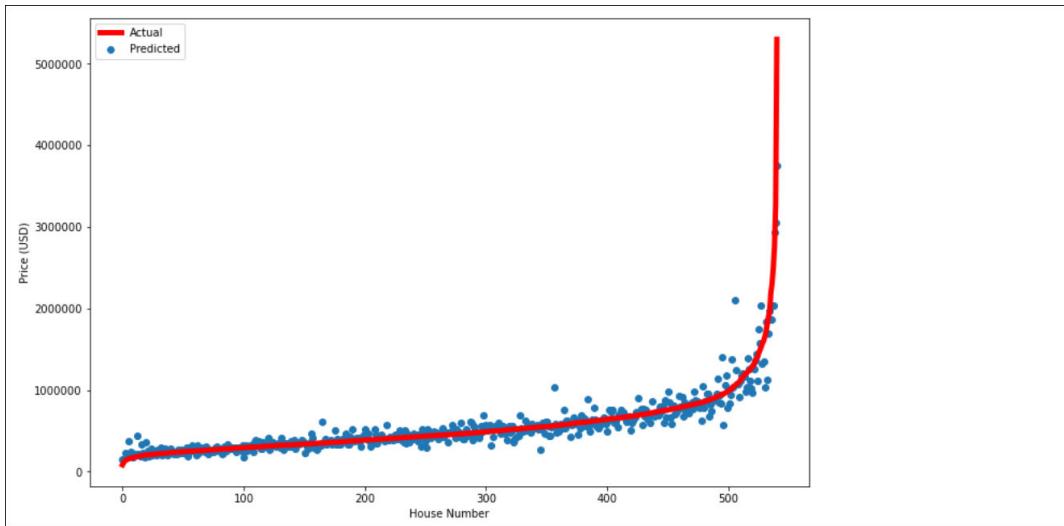
	price	price_pred
4983	\$450,000	\$505,972
10258	\$313,950	\$427,777
4045	\$704,300	\$529,922
21276	\$320,000	\$338,912
10498	\$410,000	\$438,218
2199	\$347,000	\$383,068
9216	\$1,100,000	\$1,190,575
9980	\$342,000	\$326,728
2323	\$552,500	\$485,330
6098	\$1,150,000	\$1,065,124

- Most of these predictions are fairly close to the actual prices.
g) Scroll down and examine the next code cell.

```
1 plot_residuals(y_test, pred_prices_forest)
```

- h) Run the code cell.

- i) Examine the output.



As expected, the dots are closer to the actual prices line, indicating lower residuals.

7. Plot learning curves.

- Scroll down and view the cell titled **Plot learning curves**, and examine the code cell below it. This function generates learning curves for the random forest model. Learning curves generate learning performance over time and can be used to identify models that might benefit from more training examples, as well as models that may be overfitting or underfitting.
- Run the code cell.
- Scroll down and examine the next code cell.

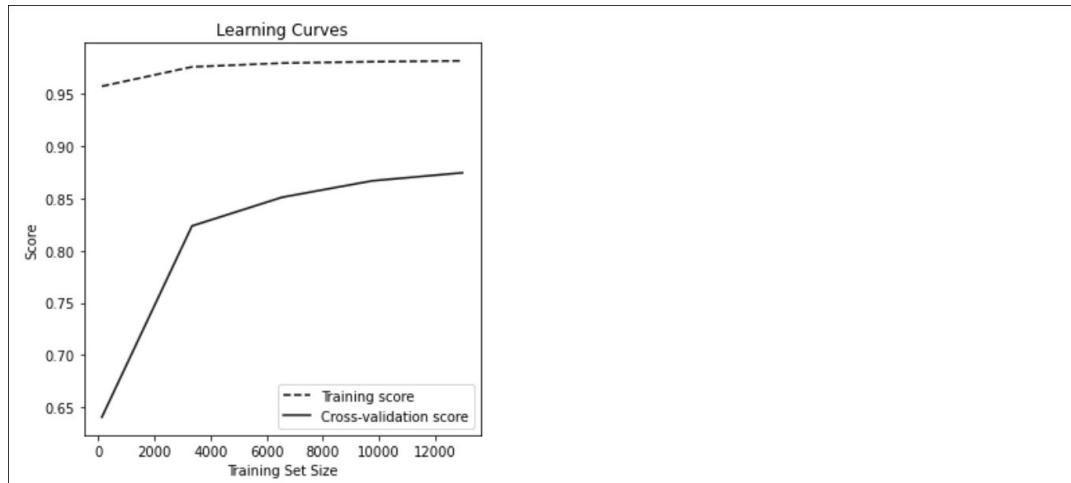
```
1 | plot_learning_curves(forest, X_train, y_train.values.ravel())
```

- d) Run the code cell.



Note: This may take a few minutes to complete.

- e) Examine the output.



Learning curves are plotted for both training and cross-validation datasets. The gap between the two curves indicates that adding more training instances is likely to help. The gap also indicates that the model suffers from some variance. Although the gap may look large, keep in mind that the scale of the y-axis starts at around 0.60 rather than 0. So, the gap is narrower than it might seem at first, which means that the variance is relatively low. Still, the variance isn't insignificant, so there are some signs of overfitting. Also, because both lines are close to 1 on the y-axis (i.e., they have a high score), the bias is also relatively low.

Increasing the number of training instances and applying regularization to the current learning algorithm would likely decrease the variance and increase the bias. You could also consider running the PCA data on the random forest algorithm to simplify the model, or find other ways to reduce the number of features.

8. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel**→**Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **Training - KC Housing** tab in Firefox, but keep a tab open to **CAIP** in the file hierarchy.

Summary

In this lesson, you trained, evaluated, and tuned a machine learning model. You now have an overview of the general process and can begin building models from the many algorithms that are featured in the bulk of this course.

What factors are most important for you when selecting a machine learning algorithm?

How do you plan on addressing training time issues in your machine learning projects?



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.

4 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 estimate 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 the most basic type of regression analysis. It demonstrates 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.

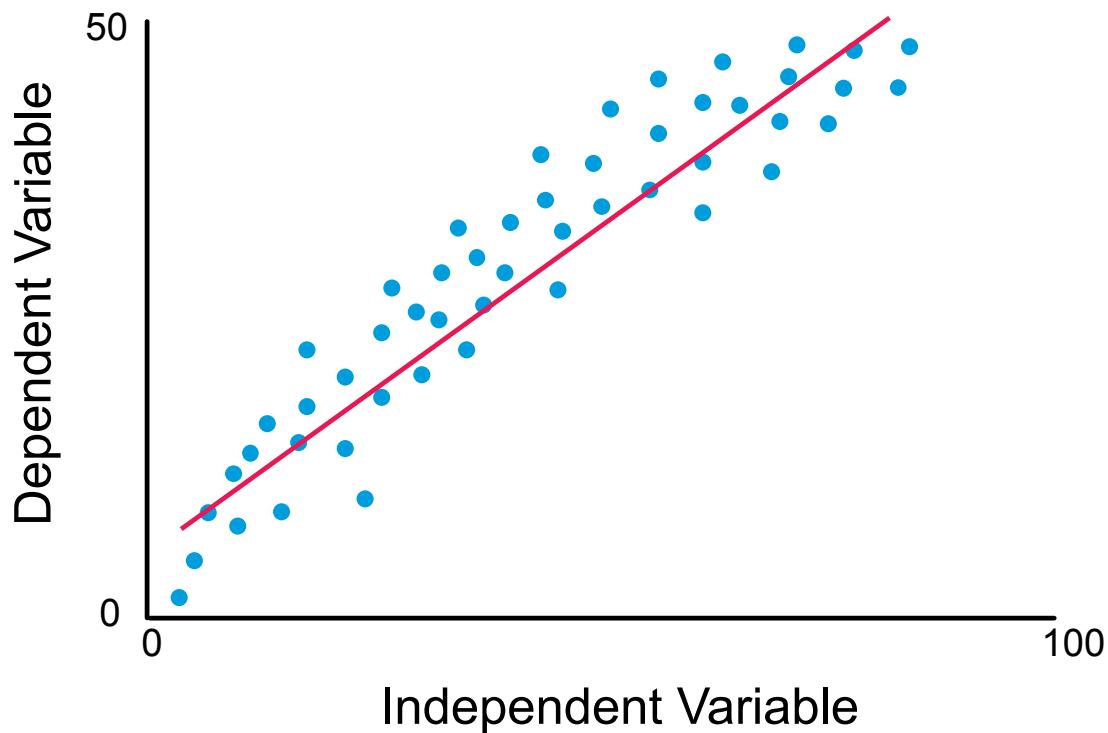


Figure 4–1: A linear regression model in which the data fits to a straight line.



Note: This particular example shows a positive correlation between the two variables. Linear regression can also work with variables that have a negative correlation (i.e., the data points would go from the top left to the bottom right in this graph, and so would the line).

The 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 Example Data

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 (each a different vehicle), 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.

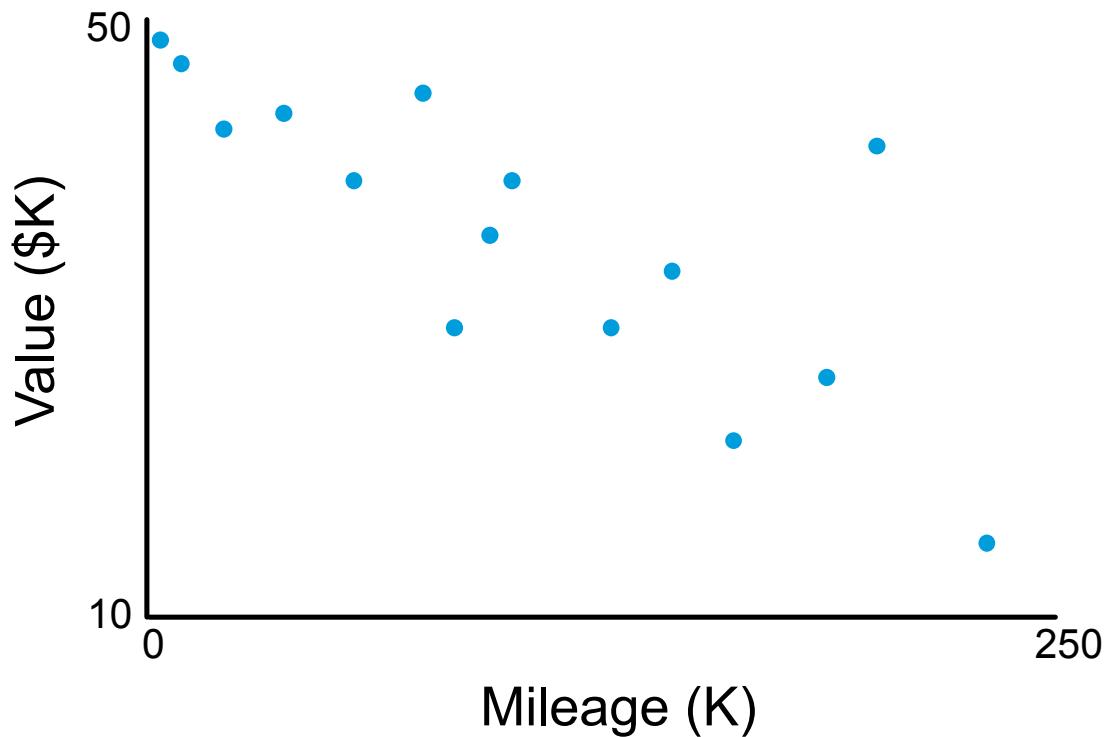


Figure 4–2: Mapping a vehicle's worth as it correlates to miles driven.

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.

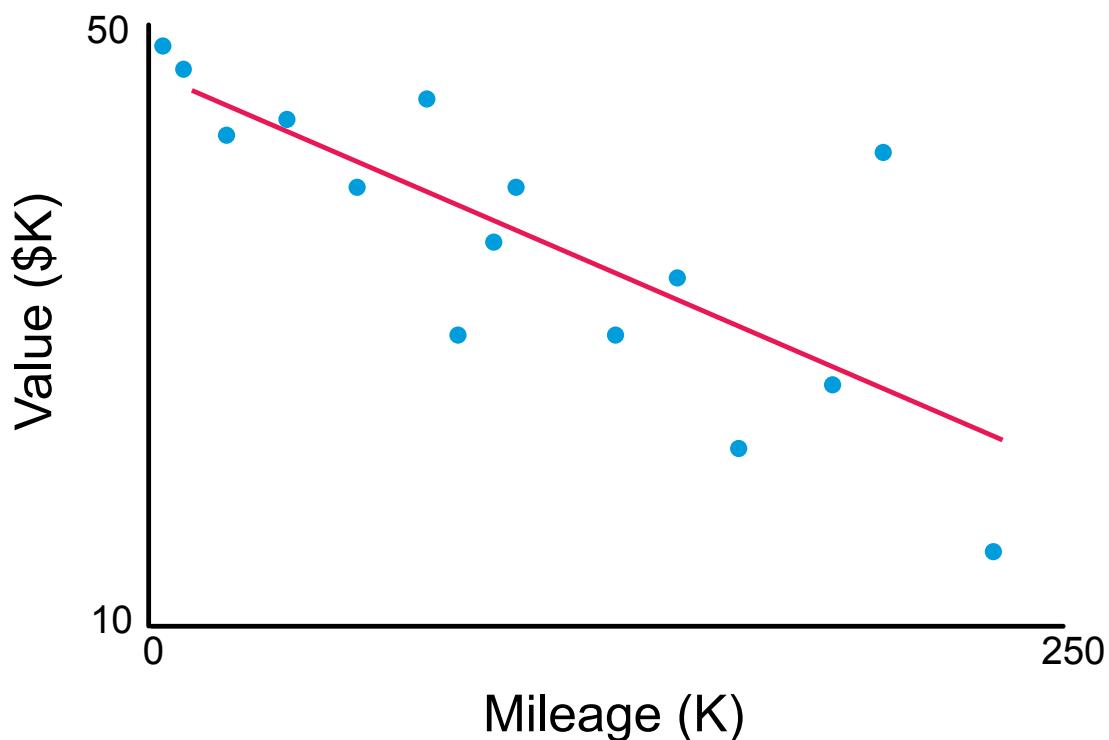


Figure 4–3: Fitting a straight line to the vehicle dataset.

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 approximately \$34,740. When graphed, this might look like the following.

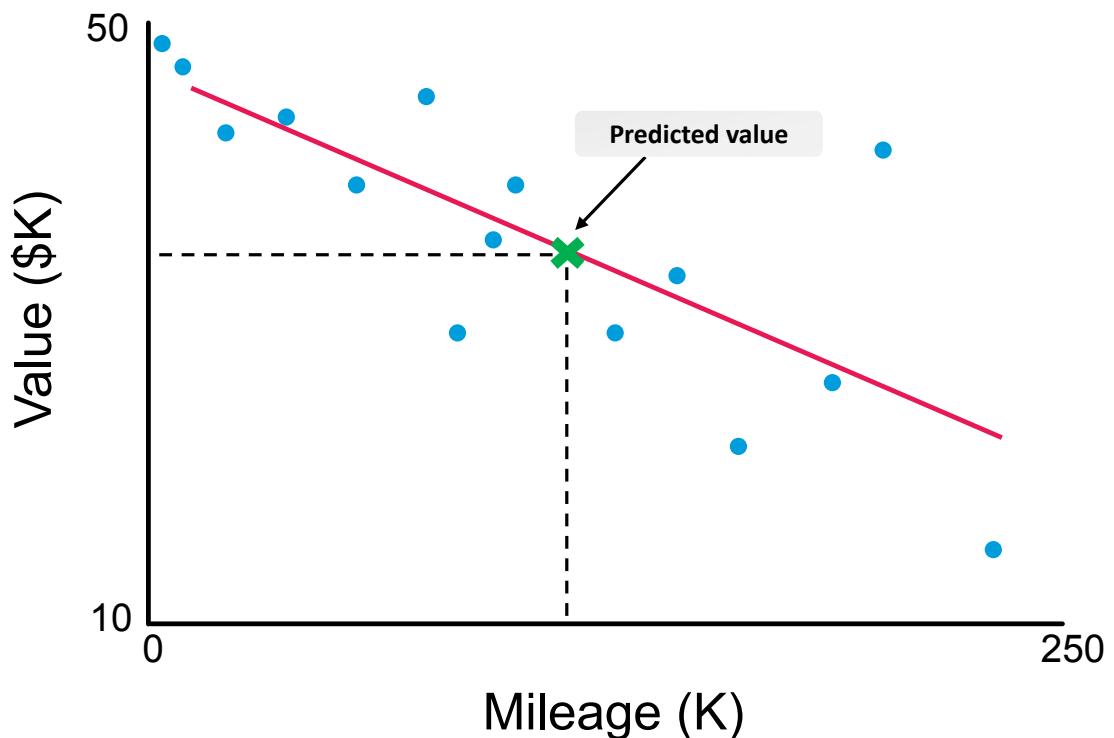


Figure 4–4: Making a prediction based on the line of best fit.

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, its fuel economy, 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 estimated line of best 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 estimate (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).
- 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 a prediction or other estimation 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 estimate 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:

$$\begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

When the values are plugged in:

$$\begin{bmatrix} 1 & 194.98 \\ 1 & 224.63 \end{bmatrix}^{-1} \begin{bmatrix} 43.00 \\ 13.25 \end{bmatrix} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} 238.76 \\ -0.89 \end{bmatrix}$$

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.

Inversion

If you're curious, the inverse of the example \mathbf{x} matrix looks like the following:

$$\begin{bmatrix} 1 & 194.98 \\ 1 & 224.63 \end{bmatrix}^{-1} = \begin{bmatrix} 7.58 & -6.58 \\ -0.03 & 0.03 \end{bmatrix}$$

The 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 \mathbf{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:

$$\widehat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \cdot \mathbf{X}^T \mathbf{y}$$

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.

Transposition and Inversion

If you're curious, $\mathbf{x}^T \mathbf{x}$ (using just the last two values in the dataset) looks like the following:

$$\mathbf{X}^T \mathbf{X} = \begin{bmatrix} 1 & 1 \\ 194.98 & 224.63 \end{bmatrix} \begin{bmatrix} 1 & 194.98 \\ 1 & 224.63 \end{bmatrix} = \begin{bmatrix} 2 & 419.61 \\ 419.61 & 88475.8373 \end{bmatrix}$$

The inversion of that is:

$$(\mathbf{X}^T \mathbf{X})^{-1} = \begin{bmatrix} 2 & 419.61 \\ 419.61 & 88475.8373 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{88475.8373}{879.1225} & -\frac{419.61}{879.1225} \\ -\frac{419.61}{879.1225} & \frac{2}{879.1225} \end{bmatrix}$$



Note: The technique of using the normal equation to derive optimal model parameters is also called *ordinary least squares* regression.

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.

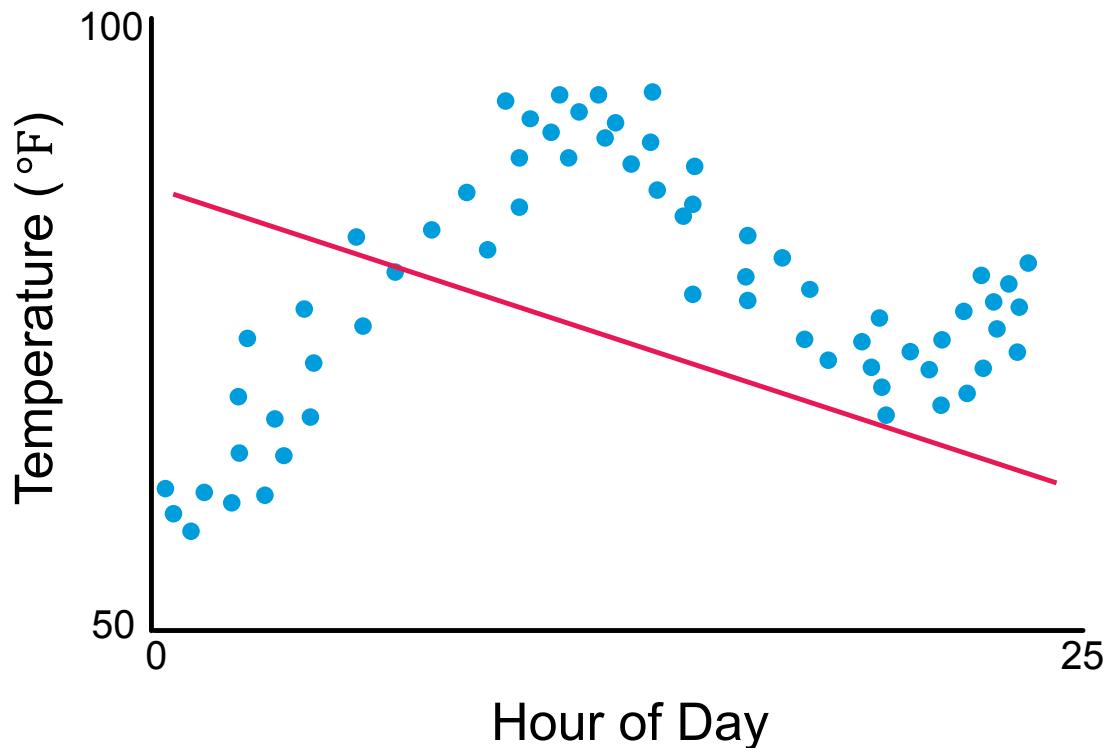


Figure 4–5: Attempting to fit a straight line to complex, non-linear data.

To obtain a higher-order fit, you add more columns of data to the \mathbf{x} matrix. This can be represented as:

$$\begin{bmatrix} 1 & x_1^1 & x_1^2 & \dots & x_1^k \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_n^1 & x_n^2 & \dots & x_n^k \end{bmatrix}$$

In this case, k represents the total number of columns you're adding to the matrix.



Note: In these examples, the superscripts for x are referring to the indices of the columns. For example, x^2 means column 2, *not* x squared.

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.

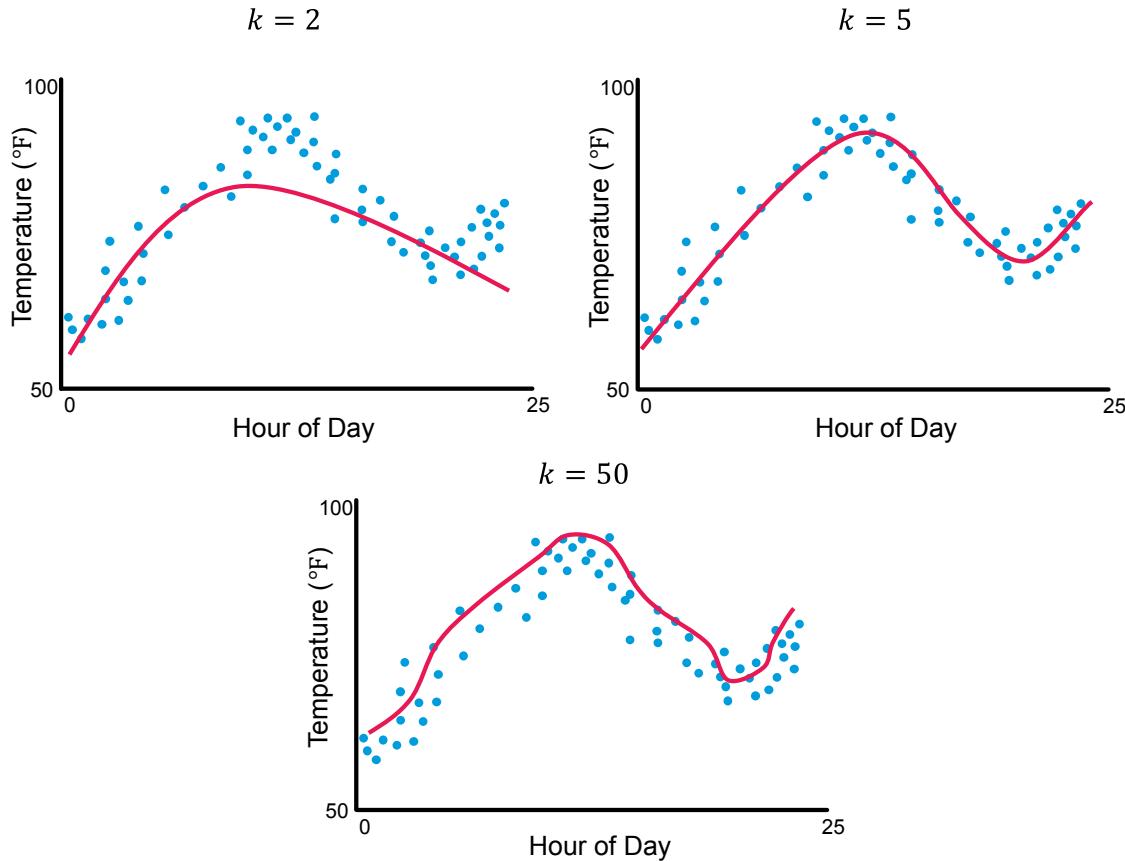


Figure 4–6: New line fits based on different k values.

Linear Model with Multiple Parameters

In the initial 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:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d$$

Where:

- d is the total number of model parameters. This is also the total number of features in the dataset, as each model parameter in linear regression maps to a feature.
- θ_i is the i^{th} model parameter (and the intercept).
- x_i is the i^{th} value of the feature.

This is called ***multivariate linear regression***, and it can also be represented in sigma notation:

$$\hat{y} = \theta_0 + \sum_{i=1}^d \theta_i x_i$$

In matrix form, inputting multiple estimative 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:

$$\begin{bmatrix} 1 & \text{mi}_1 & \text{mpg}_1 & \text{tank}_1 & \text{age}_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \text{mi}_n & \text{mpg}_n & \text{tank}_n & \text{age}_n \end{bmatrix}$$



Note: In this example, mi is mileage (miles driven), mpg is miles per gallon (the fuel economy), tank is the capacity of the fuel tank (in gallons), and age is the age of the vehicle.

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 Functions

To find the best possible fit for a regression 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 estimations, in regression, it's more common to evaluate how *poorly* it estimates—in other words, its cost.

A **cost function** attempts to quantify the error between the estimated 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 regression, the simplest and often most effective cost function is the **mean squared error (MSE)**. Once the error (difference) between the estimated 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 estimated value of the variable.
- $J(\boldsymbol{\theta})$ is the cost function itself.

You can also take the square root of MSE to get the **root mean squared error (RMSE)**. RMSE is to MSE as standard deviation is to variance—it makes the results more interpretable by putting them in the same scale as the label value the model is estimating.

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:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Relative Error

Both of the mean error functions mentioned are most useful when comparing models whose estimated values are in the same scale. If you're tuning a model based on the same dataset, then this is likely to be the case. However, there may be circumstances where you want to compare the performance of regression models that estimate values on different scales. This is where the relative equivalents of each cost function come in handy. The *relative* squared error (RSE) and *relative* absolute error (RAE) both output scores that you can use for this type of comparison.

The 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 the independent variables in 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 regression model. R^2 is usually 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 fails to pass through 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 estimate) is explainable by the model. The other 24% cannot be explained by the model.

Although R^2 is typically positive, it can also be negative. This happens when the model actually performs *worse* than the baseline (i.e., random chance), which can be the result of the model learning the wrong patterns in the data. Despite using squaring operations in its calculations, the most common definition of R^2 subtracts those operations from 1, which is what can lead to a negative output.

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 estimative 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

# Load CSV file.
dataset = np.loadtxt('/path/vehicle_data.csv', delimiter = ',')
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. One common method of reducing overfitting is to simplify the model through regularization.

Guidelines for Building Regression Models 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 estimate 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 data frame or array with the training set (not including labels), whereas the second argument is a data frame 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 data frame or array with the validation/test set (not including labels), whereas the second argument is a data frame 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 4-1

Building a Regression Model Using Linear Algebra

Data Files

/home/student/CAIP/Linear Regression/Linear Regression - Power Plant.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/Linear Regression - Power Plant.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 cell below it.
 - b) Run the code cell.
 - c) Verify that **cc_power_plant_data.csv** was loaded with 9,568 records.
3. Get acquainted with the dataset.
 - a) Scroll down and view the cell titled **Get acquainted with the dataset**, and examine the code cell below it.
 - b) Run the code cell.

- c) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9568 entries, 0 to 9567
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Temperature    9568 non-null   float64
 1   ExhaustVacuum  9568 non-null   float64
 2   AmbientPressure 9568 non-null   float64
 3   RelativeHumidity 9568 non-null   float64
 4   EnergyOutput    9568 non-null   float64
dtypes: float64(5)
memory usage: 373.9 KB
None

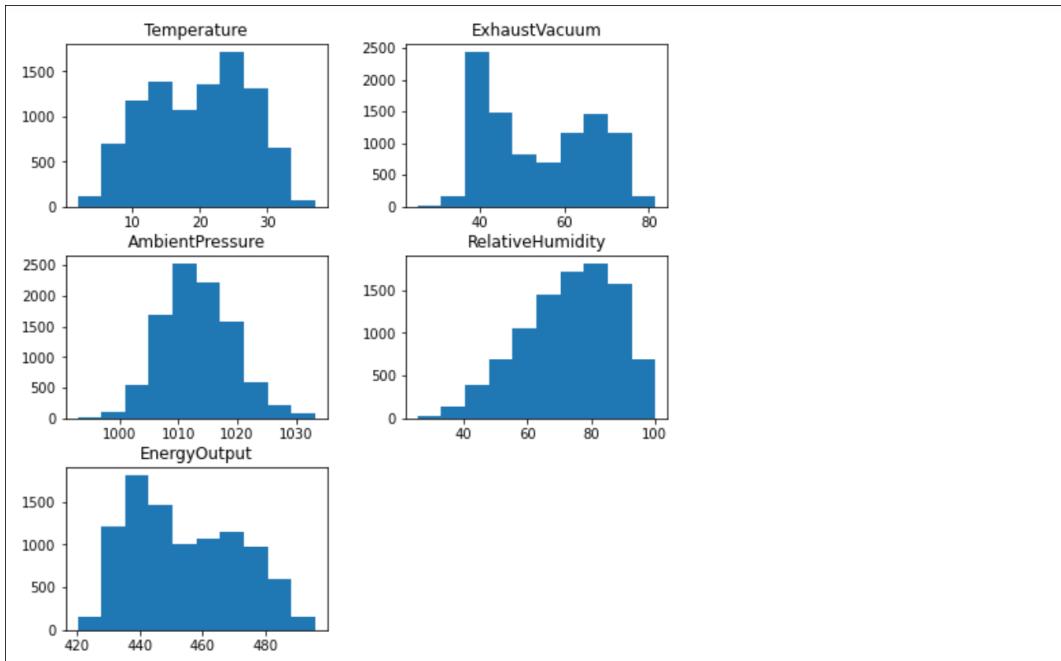
   Temperature  ExhaustVacuum  AmbientPressure  RelativeHumidity  EnergyOutput
0       8.34        40.77       1010.84        90.01        480.48
1      23.64        58.49       1011.40        74.20        445.75
2      29.74        56.90       1007.15        41.91        438.76
3      19.07        49.69       1007.22        76.79        453.09
4      11.80        40.66       1017.13        97.20        464.43
5      13.97        39.16       1016.05        84.60        470.96
6      22.10        71.29       1008.20        75.38        442.35
7      14.47        41.76       1021.98        78.41        464.00
8      31.25        69.51       1010.25        36.83        428.77
9      6.77        38.18       1017.80        81.13        484.31
```

- 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 distributions of the features.

- Scroll down and view the cell titled **Examine the distributions of the features**, and examine the code cell below it.
- Run the code cell.

- c) 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 descriptive statistics.

- Scroll down and view the cell titled **Examine descriptive statistics**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

	Temperature	ExhaustVacuum	AmbientPressure	RelativeHumidity	EnergyOutput
count	9568.00	9568.00	9568.00	9568.00	9568.00
mean	19.65	54.31	1013.26	73.31	454.37
std	7.45	12.71	5.94	14.60	17.07
min	1.81	25.36	992.89	25.56	420.26
25%	13.51	41.74	1009.10	63.33	439.75
50%	20.34	52.08	1012.94	74.97	451.55
75%	25.72	66.54	1017.26	84.83	468.43
max	37.11	81.56	1033.30	100.16	495.76

- Compared to the other features, `AmbientPressure` seems to exhibit a low amount of variance (and standard deviation). Also, its minimum and maximum values are relatively close together.
- The scales of `AmbientPressure` and `EnergyOutput` seem 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. Show correlations with EnergyOutput.

- Scroll down and view the cell titled **Show correlations with EnergyOutput**, and examine the code cell below it.

- b) Run the code cell.
- c) Examine the output.

```
Pearson 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 `EnergyOutput`, 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 dataset.

- a) Scroll down and view the cell titled **Split the dataset**, and examine the code cell below it.
- b) Run the code cell.
- c) Examine the output.

```
Original set: (9568, 5)
-----
Training features: (7176, 4)
Testing features: (2392, 4)
Training labels: (7176, 1)
Testing labels: (2392, 1)
```

The original training dataset has now been split into two: one subset to continue using as training, the other to use for testing. Note that the `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 cell below it.

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) Run the code cell.
- c) Examine the output.

```
Linear regression model took 7.78 milliseconds to fit.
Variance score on test set: 0.91
```

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 10 predictions to actual values.

- a) Scroll down and view the cell titled **Compare the first 10 predictions to actual values**, and examine the code cell below it.
- b) Run the code cell.

- c) Examine the output.

	EnergyOutput	EnergyOutput_pred
7466	431.32	436.20
8561	453.55	457.51
7777	478.47	474.29
176	472.47	464.26
6342	473.41	477.28
74	479.28	475.27
2155	441.78	442.35
9057	466.06	457.90
1675	448.43	445.15
771	437.72	443.16

10. Calculate 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 cell below it.
- b) Run the code cell.
- c) Examine the output.

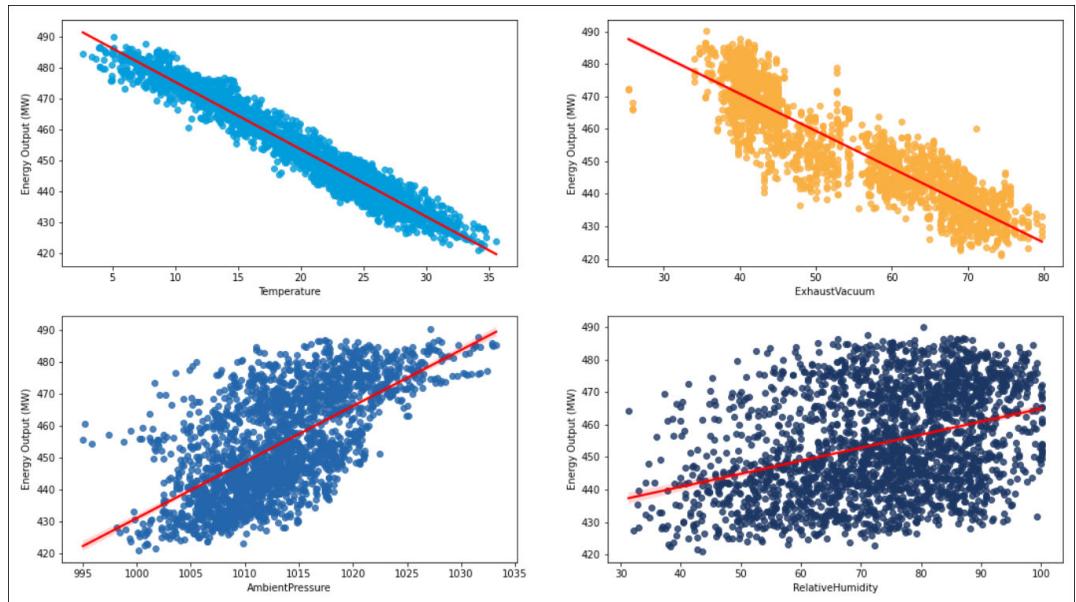
```
Cost (mean squared error): 25.87
```

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 cell 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) Run the code cell.

c) Examine the output.

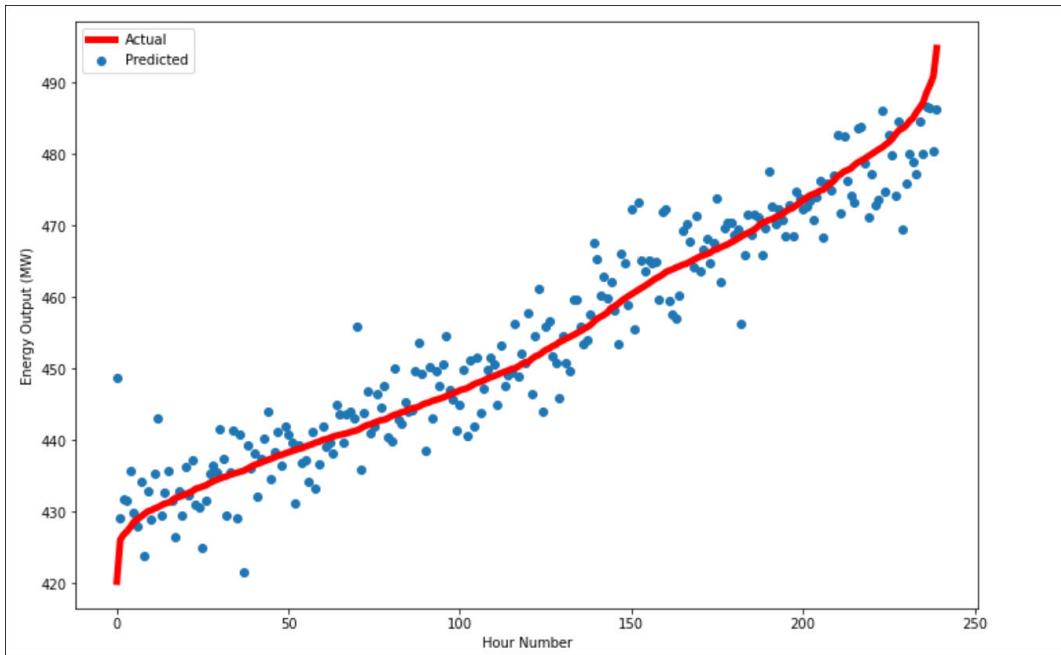


- Each feature is plotted against the `EnergyOutput` label, with a line of best fit generated from the linear regression.
- Note that the appearance of each scatter plot seems to align with the correlations identified earlier. For example, `Temperature` and `ExhaustVacuum` both exhibited high negative 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 well.

12. Plot the prediction residuals.

- Scroll down and view the cell titled **Plot the prediction residuals**, and examine the code cell below it.
- Run the code cell.

- c) Examine the output.



- The dataset has been sorted by the actual energy output per hourly measurement, which is shown as the red line.
- The predicted energy output at each hourly measurement is shown as a blue dot in a scatter plot. The farther the dot is from the line, 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.

- a) Scroll down and view the cell titled **Retrieve the model parameters**, and examine the code cell below it.

This code will print out the model parameters that the linear regression model uses to make its predictions.

- b) Run the code cell.
c) Examine the output.

```
Temperature coefficient: -1.66684747
ExhaustVacuum coefficient: -0.27545747
AmbientPressure coefficient: 0.50257581
RelativeHumidity coefficient: -0.09689120
```

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.

- a) Scroll down and view the cell titled **Manually calculate the model parameters using the normal equation**, and examine the code cell below it.

This code manually forms the normal equation that is used in simple linear regression.

- b) Run the code cell.

- 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 an estimative 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 **Linear Regression - Power Plant** tab in Firefox, but keep a tab open to **CAIP/Linear Regression/** in the file hierarchy.

TOPIC B

Build Regularized Linear Regression Models

The simple models you created earlier work well in many cases, but that doesn't mean they'll be 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 estimative 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. **Multicollinearity** refers to the same concept, but can extend to more than two features.

Normal Equation with Ridge Regression

When using ridge regression, the normal equation becomes:

$$\widehat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \cdot \mathbf{X}^T \mathbf{y}$$

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 estimative 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. The α value specifies which regression technique exerts more influence over the result.

- **0** favors the result produced by *ridge* regression (ℓ_2).
- **1** favors the result produced by *lasso* regression (ℓ_1).
- A value between **0** and **1** specifies a result influenced by both ridge and lasso, with values closer to 0 producing a result more like ridge, and values closer to 1 more like lasso.

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, α (alpha) is used instead of λ (lambda) to refer to the regularization strength, and `l1_ratio` instead of alpha to refer to the weight of lasso vs. ridge in elastic net regularization.

Guidelines for Building Regularized Linear Regression Models

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 estimative power.
- Consider using lasso regression with datasets that have a small to moderate number of features that have moderate estimative 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 effect 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.

ACTIVITY 4-2

Building a Regularized Linear Regression Model

Data Files

/home/student/CAIP/Linear Regression/Linear Regression - California Housing.ipynb

/home/student/CAIP/Linear Regression/housing_data/cali_house_data.pickle

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 California. Once again, you want to be able to train a model to predict the value of a home in a certain 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.



Note: This dataset was retrieved from the following source: https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

1. From Jupyter Notebook, select **CAIP/Linear Regression/Linear Regression - California Housing.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 cell below it.
 - b) Run the code cell.
 - c) Verify that **cali_house_data.pickle** was loaded with 20,640 records.
3. Get acquainted with the dataset.
 - a) Scroll down and view the cell titled **Get acquainted with the dataset**, and examine the code cell below it.
 - b) Run the code cell.

- c) Examine the output.

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count Dtype  
 --- 
 0   MedInc       20640 non-null   float64 
 1   HouseAge     20640 non-null   float64 
 2   AveRooms     20640 non-null   float64 
 3   AveBedrms    20640 non-null   float64 
 4   Population    20640 non-null   float64 
 5   AveOccup     20640 non-null   float64 
 6   Latitude      20640 non-null   float64 
 7   Longitude     20640 non-null   float64 
 8   target        20640 non-null   float64 
dtypes: float64(9)
memory usage: 1.4 MB
None

MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  Longitude  target
0      8.3252    41.0     6.984127  1.023810   322.0     2.555556  37.88    -122.23  4.526
1      8.3014    21.0     6.238137  0.971880  2401.0     2.109842  37.86    -122.22  3.585
2      7.2574    52.0     8.288136  1.073446   496.0     2.802260  37.85    -122.24  3.521
3      5.6431    52.0     5.817352  1.073059   558.0     2.547945  37.85    -122.25  3.413
4      3.8462    52.0     6.281853  1.081081   565.0     2.181467  37.85    -122.25  3.422
5      4.0368    52.0     4.761658  1.103627   413.0     2.139896  37.85    -122.25  2.697
6      3.6591    52.0     4.931907  0.951362  1094.0     2.128405  37.84    -122.25  2.992
7      3.1200    52.0     4.797527  1.061824   1157.0     1.788253  37.84    -122.25  2.414
8      2.0804    42.0     4.294118  1.117647  1206.0     2.026891  37.84    -122.26  2.267
9      3.6912    52.0     4.970588  0.990196   1551.0     2.172269  37.84    -122.25  2.611

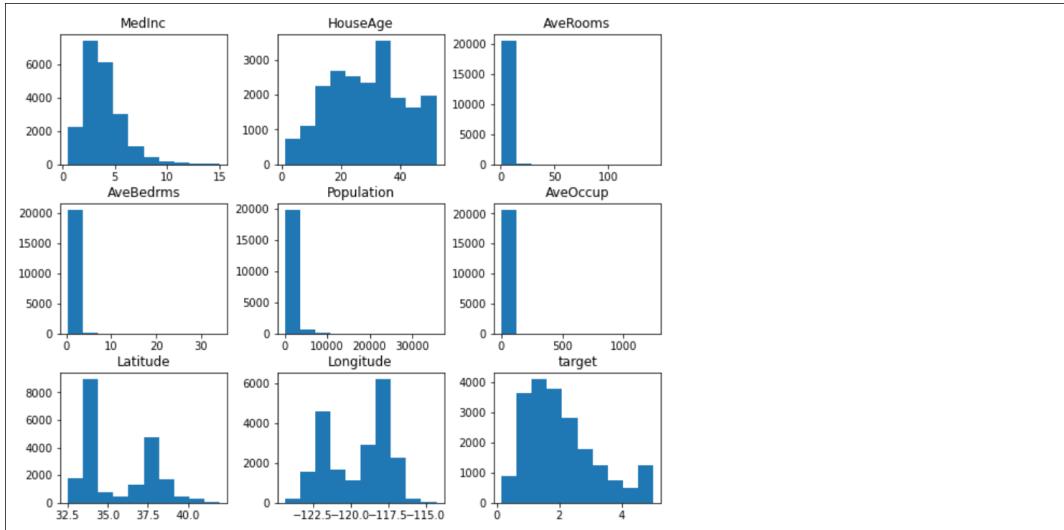
```

- The training set includes 20,640 rows and 9 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 *block group* of houses in California. A block group, defined by the U.S. Census Bureau, is a type of geographical unit that comprises multiple census blocks (the smallest unit that represents multiple houses in the same area).
- The columns/features are:
 - `MedInc` is the median income for residents in the block group.
 - `HouseAge` is the median age (in years) of the houses in the block group.
 - `AveRooms` is the mean number of rooms in houses in the block group.
 - `AveBedrms` is the mean number of bedrooms in houses in the block group.
 - `Population` is the total population of the block group.
 - `AveOccup` is the mean number of occupants in houses in the block group.
 - `Latitude` and `Longitude` are the latitude and longitude of the block group, respectively.
 - `target` is the median value of houses in the block group in hundreds of thousands of dollars.

4. Examine the distributions of the features.

- Scroll down and view the cell titled **Examine the distributions of the features**, and examine the code cell below it.
- Run the code cell.

- c) Examine the output.



Some highlights include:

- The median income is right skewed, indicating that most residents' incomes are on the lower end.
- The house ages seem to fluctuate, with a spike somewhere around 35 years.
- The average number of rooms, average number of bedrooms, total population, and average number of occupants are all heavily right skewed. This suggests the presence of very high outliers.
- The median house value (i.e., target) seems to be somewhat right skewed, with most houses having a value just under \$200,000.

5. Examine descriptive statistics.

- Scroll down and view the cell titled **Examine descriptive statistics**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	target
count	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00
mean	3.87	28.64	5.43	1.10	1425.48	3.07	35.63	-119.57	2.07
std	1.90	12.59	2.47	0.47	1132.46	10.39	2.14	2.00	1.15
min	0.50	1.00	0.85	0.33	3.00	0.69	32.54	-124.35	0.15
25%	2.56	18.00	4.44	1.01	787.00	2.43	33.93	-121.80	1.20
50%	3.53	29.00	5.23	1.05	1166.00	2.82	34.26	-118.49	1.80
75%	4.74	37.00	6.05	1.10	1725.00	3.28	37.71	-118.01	2.65
max	15.00	52.00	141.91	34.07	35682.00	1243.33	41.95	-114.31	5.00

The feature scales seem to be disproportionate, especially when comparing lower-scale features like `MedInc` (whose mean is 3.87) and higher-scale features like `Population` (whose mean is 1,425.48).

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. Show correlations with target (median house value).

- Scroll down and view the cell titled **Show correlations with target (median house value)**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
Pearson correlations with target:
```

```
target      1.000000
MedInc     0.688075
AveRooms   0.151948
HouseAge   0.105623
AveOccup   -0.023737
Population -0.024650
Longitude  -0.045967
AveBedrms -0.046701
Latitude   -0.144160
Name: target, dtype: float64
```

- It looks like the only feature that has a strong correlation with median house value is `MedInc`.
- `AveRooms` and `Latitude` have slight correlation (positive and negative, respectively).
- The rest of the features have essentially negligible correlation with `target`. You'll drop these from training, since they're unlikely to be of much use to the model.

7. Split the dataset and drop columns that won't be used for training.

- Scroll down and view the cell titled **Split the dataset and drop columns that won't be used for training**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
Original set: (20640, 9)
-----
Training features: (20640, 3)
Training labels: (20640, 1)
```

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`).

Also note that `AveRooms` and `Latitude` are being used in training despite having low correlation with the target variable. This is to ensure that the model isn't overly simplistic, and that it can learn from multiple features. However, it's possible that dropping these features won't negatively impact the model's skill.

8. Standardize the features.

- Scroll down and view the cell titled **Standardize the features**, and examine the code cell below it.
 - This function takes input data and scales it uses `StandardScaler()`, which applies the z -score formula.
 - Line 10 calls the function on the `x` data.
- Run the code cell.
- Examine the output.

```
The features have been standardized.
```

- d) Scroll down and examine the next code cell.

```
1 with pd.option_context('float_format', '{:.2f}'.format):
2     display(X[training_cols].describe())
```

- e) Run the code cell.
f) Examine the output.

	MedInc	AveRooms	Latitude
count	20640.00	20640.00	20640.00
mean	0.00	0.00	0.00
std	1.00	1.00	1.00
min	-1.77	-1.85	-1.45
25%	-0.69	-0.40	-0.80
50%	-0.18	-0.08	-0.64
75%	0.46	0.25	0.97
max	5.86	55.16	2.96

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.

9. Train a model and calculate its scores.

- a) Scroll down and view the cell titled **Train a model and calculate its scores**, and examine the code cell below it.

This function both performs cross-validation on the dataset and trains the model:

- On line 6, the function takes `in_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.

- b) Run the code cell.
c) Examine the output.

The function to train the model has been defined.

The function will be called in the next code block.

10. Evaluate several regularized linear regression models.

- a) Scroll down and view the cell titled **Evaluate several regularized linear regression models**, and examine the code cell below it.

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).
- On line 10, 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).

- Run the code cell.
- Examine the output.

The function to evaluate the linear regression models has been defined.

You'll call this function in the next few code blocks.

- Scroll down and examine the next code cell.

```
1 | model_eval(500, 0.5)
```

- Run the code cell.
- Examine the output.

```
Regularization: None
-----
Mean variance score on test set: 0.4178
Cost (mean squared error): 0.7263

Regularization: Ridge
-----
Mean variance score on test set: 0.4195
Cost (mean squared error): 0.7247

Regularization: Lasso
-----
Mean variance score on test set: -0.0892
Cost (mean squared error): 1.3703

Regularization: Elastic net
-----
Mean variance score on test set: -0.0892
Cost (mean squared error): 1.3703
```

- In this function call, you used an arbitrary starting point of 500 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 the highest variance score and the lowest MSE (though only barely compared to the non-regularized model). The latter score is of particular interest, as this is what you're trying to minimize.
- Lasso and elastic net regression performed considerably worse than the others. This is likely due to the strength of regularization being so high. There are only a few features the model learns from, and lasso is probably removing one or more of them, causing the model to fail to learn patterns in the data. For elastic net, it seems like even being halfway between lasso and ridge is enough to remove useful features.
- You'll reconfigure these hyperparameters to hopefully get better results.

- g) Scroll down and examine the next code cell.

```
1 model_eval(0.1, 0.3)
```

- h) Run the code cell.
i) Examine the output.

```
Regularization: None
-----
Mean variance score on test set: 0.4178
Cost (mean squared error): 0.7263

Regularization: Ridge
-----
Mean variance score on test set: 0.4178
Cost (mean squared error): 0.7263

Regularization: Lasso
-----
Mean variance score on test set: 0.4073
Cost (mean squared error): 0.7410

Regularization: Elastic net
-----
Mean variance score on test set: 0.4146
Cost (mean squared error): 0.7318
```

- In this function call, you changed the regularization hyperparameter to 0.1, a significant decrease in strength compared to your first run.
- You changed the elastic net penalty to 0.3, which favors the ℓ_2 norm (ridge).
- For these hyperparameters, lasso and elastic net regression are much improved. However, the simple model and ridge regression both have the lowest MSE—though not by much.
- Whereas the highly regularized ridge regression model performed slightly better than the simple model, the ridge model with low regularization strength is essentially equal to the simple model. This might be due to a lack of collinearity in the data, as there is not much for ridge regression to improve upon.
- You won't always see massive gains through regularization.

11. Are you satisfied with this MSE value? In other words, would you stop there and finalize the model? Why or why not?

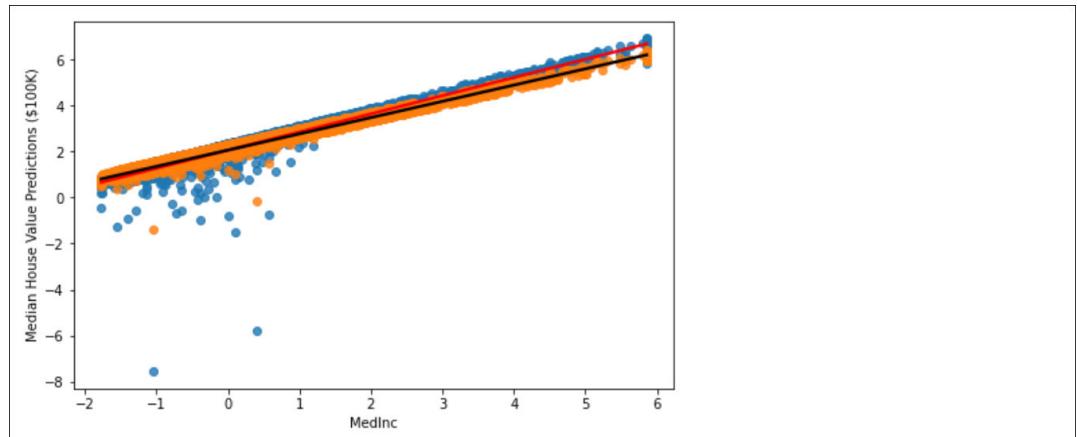
12. Plot lines of best fit for the MedInc (median income) feature.

- a) Scroll down and view the cell titled **Plot lines of best fit for the MedInc (median income) feature**, and examine the code cell below it.

This code will generate a chart showing the line of best fit for two regression models. Elastic net is being chosen as the regularized model for demonstration purposes.

- b) Run the code cell.

- c) Examine the output.

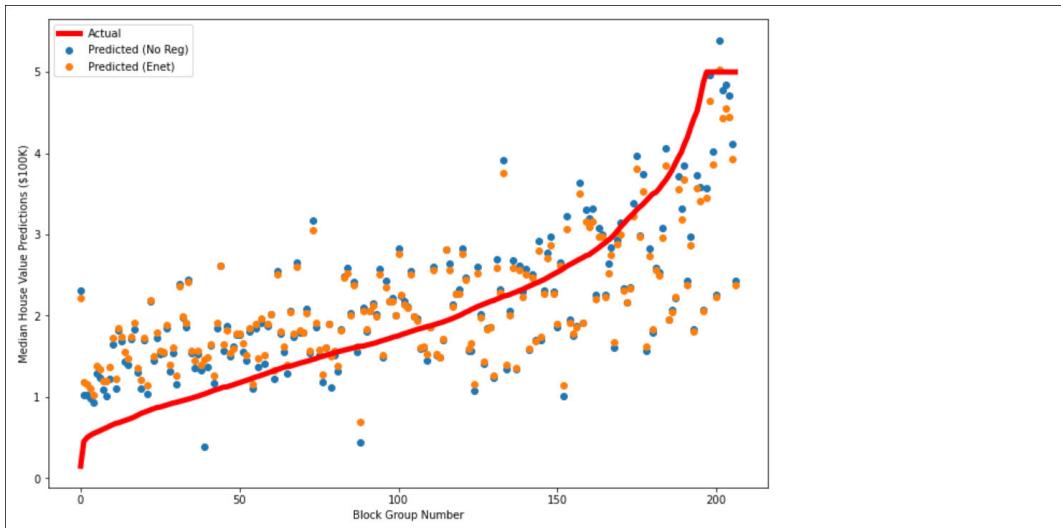


- `MedInc` is being demonstrated here because it had the strongest correlation with the median house value.
- The red line is the unregularized line of best fit, whereas the black line is the line of best fit using elastic net with the hyperparameters that were deemed "good enough."
- The blue dots are 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. For example, the elastic net model has much fewer negative house value predictions (orange dots) than the non-regularized model (blue dots).

13. Plot the prediction residuals.

- Scroll down and view the cell titled **Plot the prediction residuals**, and examine the code cell below it.
This code will generate a residuals chart for both models.
- Run the code cell.

c) Examine the output.



- The dataset has been sorted by the actual median house value, which is shown as the red line.
- The predicted value of a house using unregularized linear regression is shown as a blue dot in a scatter plot. The farther the dot is from the line, the higher the amount of error between predicted and actual values.
- The predicted value of a house using elastic net regression is shown as an orange dot in a scatter plot.
- Each model gives slightly different predictions, and some predictions are closer to the target than others.

14. Keep this notebook open.

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

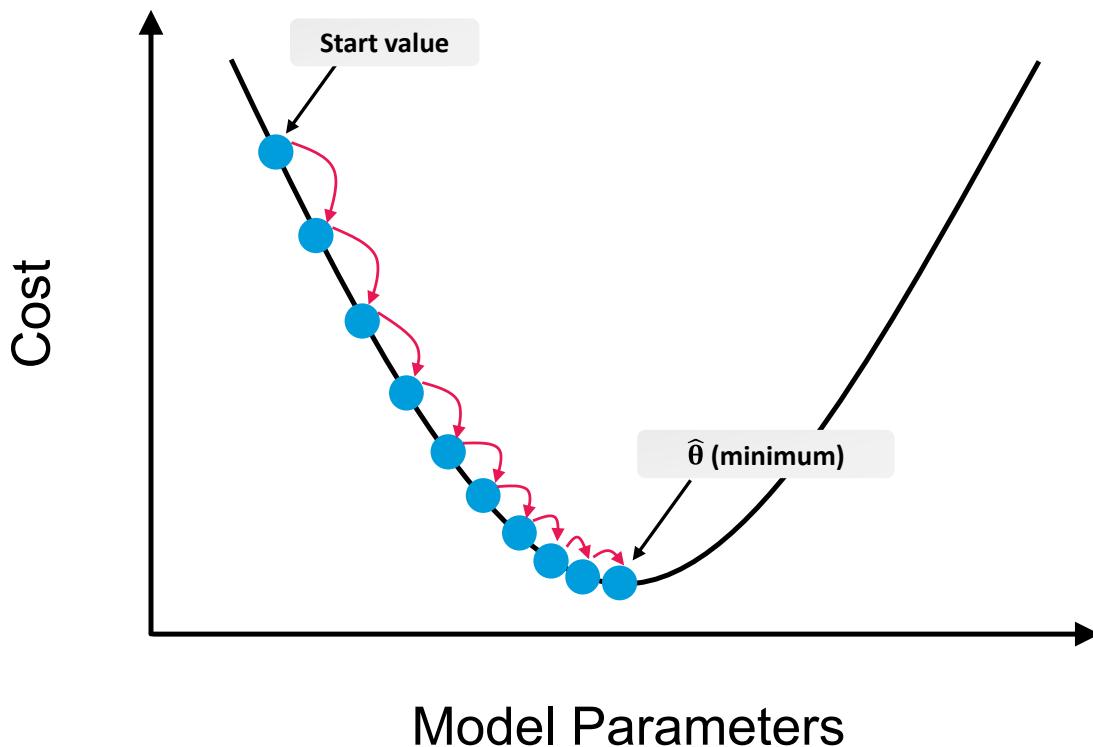
The closed-form nature of the normal equation means it will directly provide you with the model parameters that best minimize the cost function. The expense of computing the matrix inverse as applied to large datasets necessitates a different approach. In an iterative approach, the model attempts to 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 $\hat{\theta}$).



Model Parameters

Figure 4-7: 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. According to noted AI researcher Andrew Ng, a good rule of thumb is that a dataset with 10,000 or more examples and/or features will usually train too slowly for the normal equation and will likely be a better candidate for 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 kinds of cost functions, gradient descent is not guaranteed to converge at the global minimum. Whenever possible, use a convex cost function.

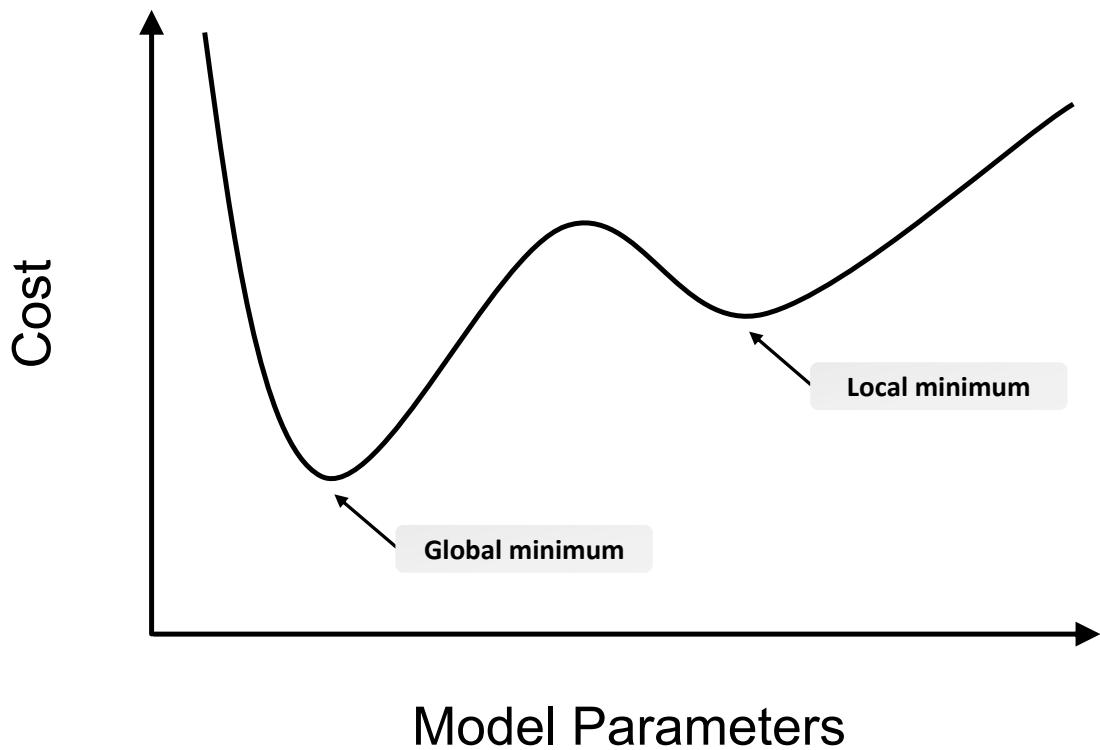
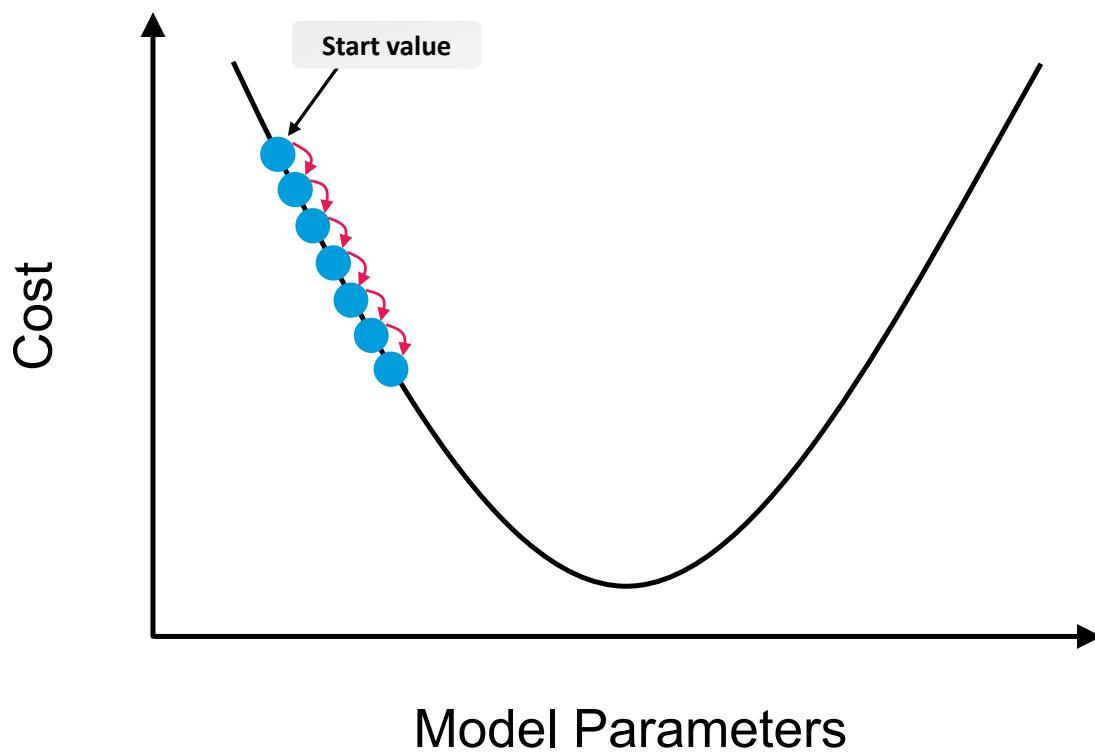


Figure 4-8: Comparing the global minimum to one local minimum.

Learning Rate

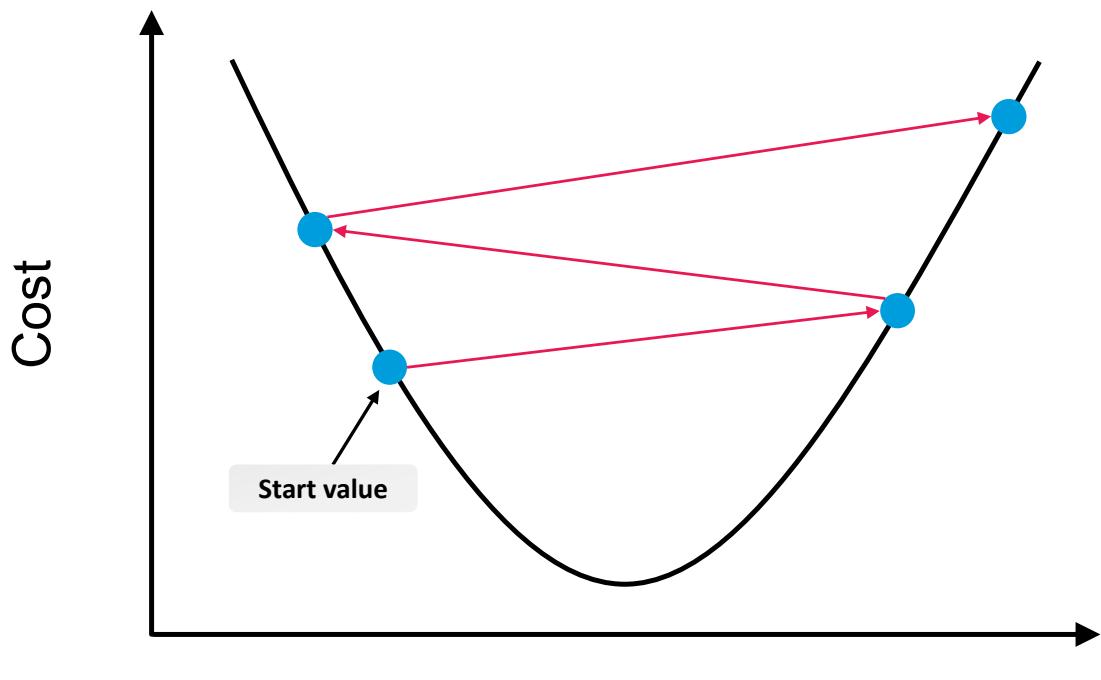
In gradient descent, the size of each "step" down the valley is called the **learning rate**. The learning rate is a hyperparameter (α) of the model that you must tune to get the best results. If the learning rate is small, more "steps" will be required to converge at the minimum. Too small, and the descent could end up taking a very long time.



Model Parameters

Figure 4-9: 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.



Model Parameters

Figure 4-10: 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. This is one advantage the normal equation has over gradient descent—there's no need to tune a learning rate.



Note: In addition to tuning the learning rate, you also can speed up convergence by scaling your features.

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 estimative 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.
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 Iterative Linear Regression Models

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 dataset 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 4–3

Building an Iterative Linear Regression Model

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/Linear Regression/Linear Regression - California 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 original dataset using the holdout method**. Select **Cell→Run All Above**.

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 California housing model.

1. Split the original dataset using the holdout method.

- Scroll down and view the cell titled **Split the original dataset using the holdout method**, and examine the code cell below it.
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.
- Run the code cell.
- Examine the output.

```
Original set:      (20640, 9)
-----
Training features: (15480, 3)
Testing features:  (5160, 3)
Training labels:   (15480, 1)
Testing labels:    (5160, 1)
```

2. Standardize the features.

- Scroll down and view the cell titled **Standardize the features**, and examine the code cell below it.
The data is being prepared in the same way as before—scaling values using the *z*-score.
- Run the code cell.
- Examine the output.

```
The features have been standardized.
```

3. Why is it important to scale down features, such as through standardization, when using an iterative cost minimization technique like gradient descent?

4. Train a model and calculate its cost and training time.

- Scroll down and view the cell titled **Train a model and calculate its cost and training time**, and examine the code cell below it.

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.

- Run the code cell.
- Examine the output.

The function to train the model and calculate its cost has been defined.

5. Evaluate linear regression models using both closed-form and iterative solutions.

- Scroll down and view the cell titled **Evaluate linear regression models using both closed-form and iterative solutions**, and examine the code cell below it.

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` (0.001), 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.

- Run the code cell.
- Examine the output.

The function to evaluate the linear regression models has been defined.

6. 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.

7. Continue evaluating the linear regression models using different initial learning rates.

- a) Scroll down and examine the next code cell.

```
1 model_eval(0.09)
```

- b) Run the code cell.
c) Examine the output.

```
Model: Ridge regression (closed form)
-----
Linear regression model took 9.24 milliseconds to fit.
Cost (mean squared error): 0.7222

Model: Ridge regression (gradient descent)
-----
Linear regression model took 8.16 milliseconds to fit.
Cost (mean squared error): 0.8746
```

- In this function call, you used an arbitrary starting point of 0.09 as the initial learning rate (`eta0`).
 - The mean squared error (MSE) for the SGD model is worse.
 - Because the dataset is relatively 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 examine the next code cell.

```
1 model_eval(0.08)
```

- e) Run the code cell.
f) Examine the output.

```
Model: Ridge regression (closed form)
-----
Linear regression model took 1.46 milliseconds to fit.
Cost (mean squared error): 0.7222

Model: Ridge regression (gradient descent)
-----
Linear regression model took 38.32 milliseconds to fit.
Cost (mean squared error): 0.8388
```

- In this function call, you decreased the learning rate slightly.
 - The MSE for the SGD model has improved, but is still worse than the closed-form model.
- g) Scroll down and examine the next code cell.

```
1 model_eval(0.01)
```

- h) Run the code cell.

- i) Examine the output.

```
Model: Ridge regression (closed form)
-----
Linear regression model took 2.03 milliseconds to fit.
Cost (mean squared error): 0.7222

Model: Ridge regression (gradient descent)
-----
Linear regression model took 10.60 milliseconds to fit.
Cost (mean squared error): 0.7402
```

- The SGD model seems to be gradually improving as you progressively decrease the learning rate.
- j) Scroll down and examine the next code cell.

```
1 model_eval(0.001)
```

- k) Run the code cell.
l) Examine the output.

```
Model: Ridge regression (closed form)
-----
Linear regression model took 9.00 milliseconds to fit.
Cost (mean squared error): 0.7222

Model: Ridge regression (gradient descent)
-----
Linear regression model took 5.29 milliseconds to fit.
Cost (mean squared error): 0.7246
```

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.

8. 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?

9. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **Linear Regression - California Housing** tab in Firefox, but keep a tab open to **CAIP** in the file hierarchy.

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 or otherwise estimate 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.

5

Building Forecasting Models

Lesson Time: 2 hours, 30 minutes

Lesson Introduction

There are many other algorithms that can perform regression analysis beyond just linear regression. Forecasting is one such alternative that excels at making numeric predictions from data that follows time-based patterns—everything from the weather to stock prices, and much more.

Lesson Objectives

In this lesson, you will:

- Build forecasting models based on univariate time series data.
- Build forecasting models based on multivariate time series data.

TOPIC A

Build Univariate Time Series Models

The simplest application of forecasting involves a single variable that changes over consistent periods of time. This is where you'll start building forecasting models.

Forecasting

Forecasting is a task that involves making predictions about future events based on the analysis of relevant past events. In machine learning, the term "forecasting" usually refers to a type of forecasting called **time series** forecasting. Time series forecasting is actually a subset of regression analysis because it attempts to identify the relationships between continuous variables. What makes time series forecasting distinct from typical regression tasks is that each successive value of an independent variable relates to the previous value of the independent variable as the next step in a sequence.

Forecasting is very popular in fields like sales, business planning, risk assessment, and any other task that requires future predictions based on past data. There are many algorithms that perform time series forecasting, including complex neural networks used in deep learning.

Univariate Time Series

A **univariate** time series includes a single variable whose change is recorded in consistent increments of time. Consider a task in which a ride-sharing service wants you to develop a model that can predict the number of passengers that will use the service for the first few months in the next year. That way, the service can determine the optimal number of drivers to hire for that month. If the data were in a univariate time series format, it might look something like this:

Month	Number of Passengers (Thousands)
2021-01	129
2021-02	137
2021-03	135
2021-04	156
2021-05	159
2021-06	179
2021-07	183
2021-08	173
2021-09	170
2021-10	173
2021-11	155
2021-12	132
2022-01	130
2022-02	135
2022-03	146

Month	Number of Passengers (Thousands)
2022-04	148
2022-05	145
2022-06	169
2022-07	178
2022-08	181
2022-09	167
2022-10	161
2022-11	145
2022-12	137

As you can see, each row is a month (in this case, over a period of two years). The data is therefore sequential, and values for each feature are directly dependent on those values in the past. So any predictions about future passenger numbers must consider each historical row of data as a member of a sequence, rather than as an independent observation. This also means that row order in a time series dataset is important, and new records must follow this order if the dataset is to grow.

Time series can follow any time interval, not just months like in the example. They can be ordered by weeks, days, hours, seconds, and so on. The important thing is that the interval is consistent across each observation.



Note: Despite having what appears to be two variables, this is still a univariate problem. The details of the time interval are not part of the algorithm's analysis.

Autoregressive Integrated Moving Average (ARIMA)

Autoregressive integrated moving average (ARIMA) is one of the most common algorithms used in univariate time series forecasting.

Each term in the name ARIMA actually helps to describe what the algorithm does:

- **Autoregressive** describes the dependent relationship between one observation and one or more other observations. The gap between each observation in a time series is referred to as the *lag*, and observations are said to be *lagged* if they are some relative degree apart from each other. In the example table, the lag between the record at 2021-03 and 2021-05 is two—not because they happen to be two months apart, but because they are two records apart.
- **Integrated** means that an observation is subtracted from the previous observation ("differenced") to make the time series *stationary*. A stationary time series is one whose statistical properties (e.g., mean and variance) are constant over time. This makes prediction easy, since these values don't change. Later, the process of making the time series stationary is reversed in order to reveal the predictions on the original, non-stationary series.
- **Moving average** describes the dependent relationship between one observation and the residual errors from a moving average applied to lagged observations.

These three concepts are actually implemented as three different hyperparameters when training an ARIMA model. Those hyperparameters are:

- **p**—The number of lags to include in the model. So, a value of 5 would cause autoregression to consider an observation in context with the five observations before it.
- **d**—The number of times that the observations are differenced. When there is a clear trend in the data, it's common to start with a value of 1 so that the minimum number of differences is performed in order to make the time series stationary.

- q —The number of lagged forecast errors to include in the model. This value is used to remove any remaining autocorrelation—the correlation between lagged observations as a function of that lag—from the stationary series. A good starting value is 1.

You can set 0 for any of these hyperparameters in order to remove that functionality from the model. In other words, it's possible to have an AR model, an I model, an MA model, and so on.

There are various analysis methods you can use to determine the best hyperparameter values. However, libraries like `auto.arima` for R and `pmdarima` for Python perform these analyses for you to determine the optimal p , d , and q to use in your forecast.

Seasonality

ARIMA can incorporate the concept of *seasonality*, in which a pattern repeats over some number of time periods s . A seasonal time series exhibits this repeating pattern. For example, since the ride-sharing data is recorded monthly, your s value might be 12, since passenger counts tend to follow a yearly pattern. The busiest months tend to be in the summer, whereas the slowest months are in the winter. If you wanted to predict passenger count in February, seasonal ARIMA models would consider past years' February passenger counts when making the prediction.

"Seasonal," however, should not be taken literally—any time series data can be seasonal if it exhibits a repeating pattern over some period of time. If you were forecasting the ambient temperature of a campus on an hour-by-hour basis, you might set s to 24 since temperatures tend to follow a daily pattern.

ARIMA Example: Ride-Sharing Passengers

Consider what the prior table of ride-sharing passenger data looks like when graphed.

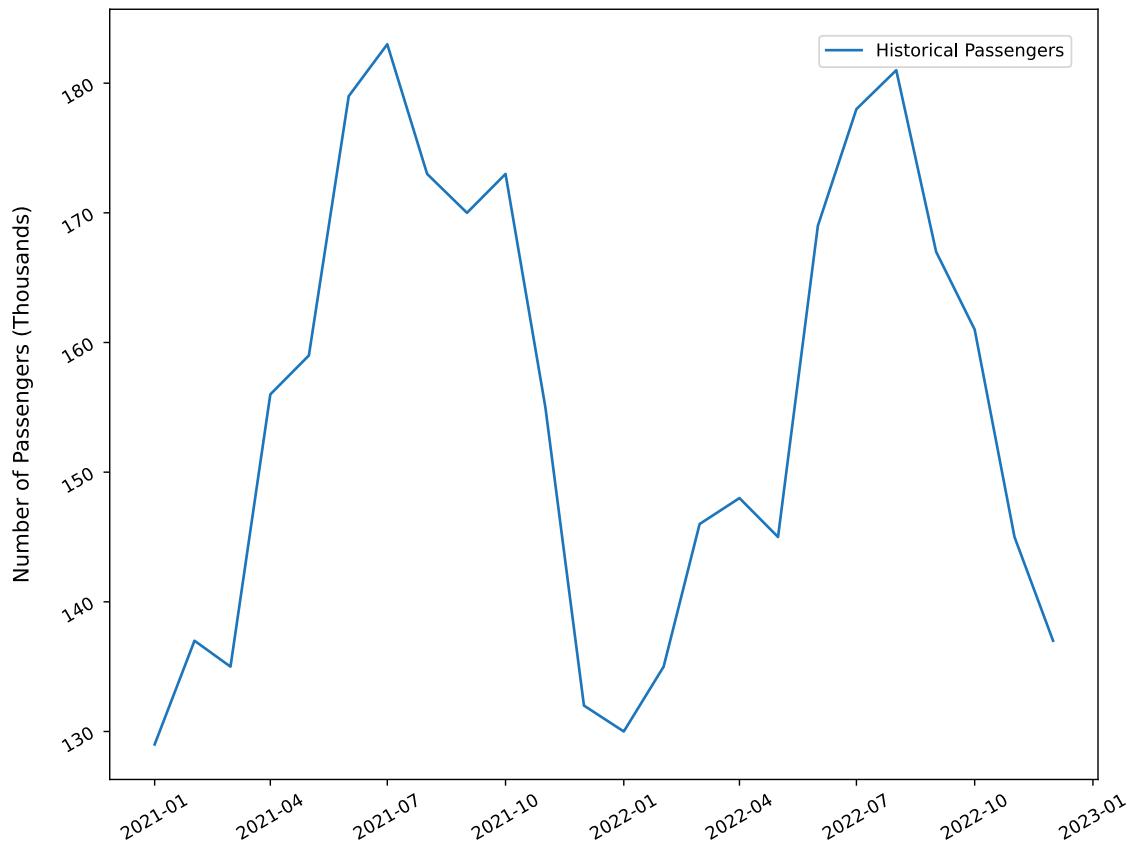


Figure 5-1: A line graph of the historical ride-sharing data.

As you can see, in 2021, there was an upward trend in the summer months (in the Northern Hemisphere), whereas the number of passengers started to dip as winter approached. This pattern seems to have repeated for 2022.

Now, to use ARIMA for forecasting, the algorithm will be configured with the following AR, I, and MA:

- $p = 3$
- $d = 1$
- $q = 1$

So, the autoregressive lag is 3, whereas both the differencing count and the number of lagged forecast errors are both 1. After fitting the data to the model, you can specify a start and end time to forecast. In this case, the model will forecast the next six months (i.e., starting on January 2023).

In addition, with most libraries, you can configure a separate seasonality parameter for *each* aspect of ARIMA. You can just make these the same values as the non-seasonal parameters, though you can also change them as desired. However, you do need to set the seasonal ***periodicity***. In the case of the ride-sharing example, a periodicity of 12 should be sufficient for the reason mentioned earlier.

The results are as follows.

Month	Forecasted Number of Passengers (Thousands)
2023-01	129.58
2023-02	133.34
2023-03	128.62
2023-04	141.51
2023-05	145.58
2023-06	175.81

These forecasted values can be plotted alongside the historical values, as in the following figure.

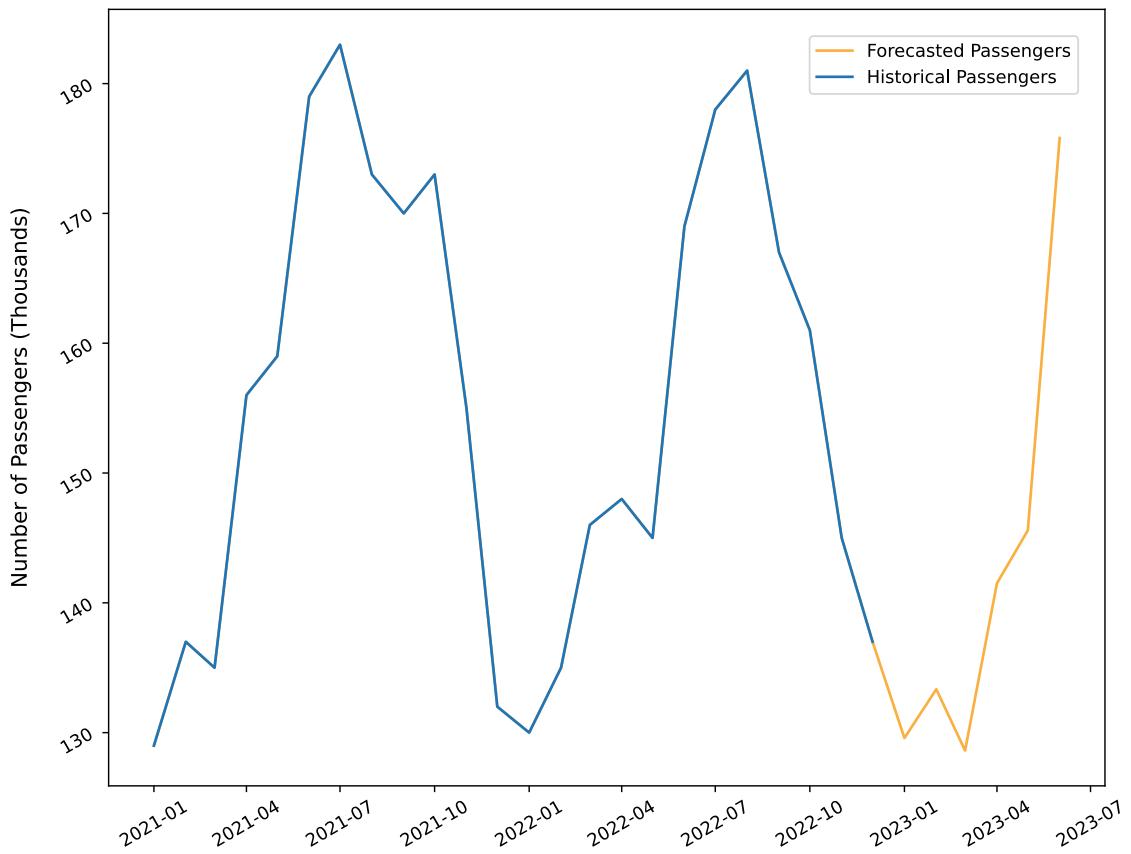


Figure 5-2: A graph of the passenger count values forecasted by the ARIMA model.

Obviously, more training data will lead to a more effective model, as 24 observations are not a particularly robust sample for the model to learn from. Also, in practice, you'd want to split the data into separate training and test sets to evaluate the model's performance, as with any other supervised algorithm.

Guidelines for Building Univariate Time Series Models



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

Follow these guidelines when you are building univariate time series models.

Build a Univariate Time Series Model

When building a univariate time series model:

- Ensure your time series dataset follows a consistent pattern for the time of each observation.
- Consider splitting your data into 60% training and 40% testing data when you have many observations.
- Split time series data into segments instead of selecting values at random, with training data always earlier than test data.
- Consider using differences rather than raw values when forecasting time series data.
- If you're unsure what values to select for p , d , and q in an ARIMA model, consider starting with 1 for the latter two, and a small integer for the former, then tune from there.
- Consider using a library that can automatically analyze a model to determine the optimal ARIMA hyperparameters.

- Be sure to incorporate seasonality into your model if it follows a repeatable pattern.
- Whenever feasible, plot your model to better illustrate trends in the model's forecasts.

Use Python for Univariate Time Series Models

The `statsmodels` package in Python includes a `tsa.arima.model` module that provides the `ARIMA()` class for univariate time series forecasting. The following are some of the objects and functions you can use to build such a model.

- `model = statsmodels.tsa.arima.model.ARIMA(y_train, order = (3, 1, 1), seasonal_order = (3, 1, 1, 12))` —This constructs an ARIMA model object with a p of 3, and a d and q of 1. These same values are incorporated into the model's seasonality, which also uses 12 as the periodicity.
- You can use this class object to call the same `fit()` and `score()` methods as you would with other regression models. You can also call `summary()` to get a detailed output of the model's attributes, including its coefficients, error scores, and distribution properties.
- `model.predict(start = '2023-01-01 00:00:00', end = '2023-06-01 00:00:00')` —Use the model to make predictions for the next time intervals that fall between `start` and `end`. In this example, the interval is one month, so the model will make six total predictions.

ACTIVITY 5–1

Building a Univariate Time Series Model

Data Files

/home/student/CAIP/Forecasting/Forecasting - Solar Power.ipynb
/home/student/CAIP/Forecasting/electricity_data/elect_net_gen_solar.csv

Before You Begin

Jupyter Notebook is open.

Scenario

You work for a research firm in the energy industry. You've been tasked with identifying trends in electricity generation across the country. Specifically, you want to be able to forecast how much electrical energy will be generated in the next year per each source of energy. This can help the industry execute plans that adequately meet the forecasted demand.

You'll start by looking at electrical energy consumption by solar power, which has become increasingly popular in recent years. You've collected data from the U.S. Energy Information Administration (EIA) regarding the net electricity generation from solar power for each month for the past 50 years. You'll build a univariate time series model using ARIMA to try to predict how much electrical energy will be generated using solar power in the next year. Later, you hope to extend your forecasting efforts to other sources of power so you can compare your findings.

1. From Jupyter Notebook, select **CAIP/Forecasting/Forecasting - Solar Power.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 cell below it.
 - b) Run the code cell.
 - c) Verify that **elect_net_gen_solar.csv** was loaded with 591 records.

This dataset includes recordings of the net electricity generated from solar power in the entire United States for each month. It was obtained from the EIA.
3. Get acquainted with the dataset.
 - a) Scroll down and view the cell titled **Get acquainted with the dataset**, and examine the code cell below it.
 - b) Run the code cell.

- c) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
Index: 591 entries, 1973 01 to 2022 03
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   ElectNetGen  459 non-null    float64
dtypes: float64(1)
memory usage: 9.2+ KB
None

ElectNetGen
Date
1973 01      NaN
1973 02      NaN
1973 03      NaN
1973 04      NaN
1973 05      NaN
...
2021 11    7873.778
2021 12    6354.878
2022 01    8003.896
2022 02    9202.560
2022 03    11890.566
591 rows × 1 columns
```

- There is one column: `ElectNetGen`.
- `ElectNetGen` contains floats.
- The data frame has a `Date` index that records the month and year of each observation. To make things easier, you'll convert this to a datetime format.
- The net electricity generated is measured in gigawatt-hours (GWh), which is one million kilowatt-hours (kWh).
- Several of the early years have missing values. Solar power is a recent phenomenon, so this makes sense. You'll need to handle these values, however.

4. Convert the `Date` index to datetime format for processing.

- a) Scroll down and view the cell titled **Convert the Date index to datetime format for processing**, and examine the code cell below it.

The `to_period('M')` portion on line 1 indicates that the date should be specified to the months.

- b) Run the code cell.
c) Examine the output.

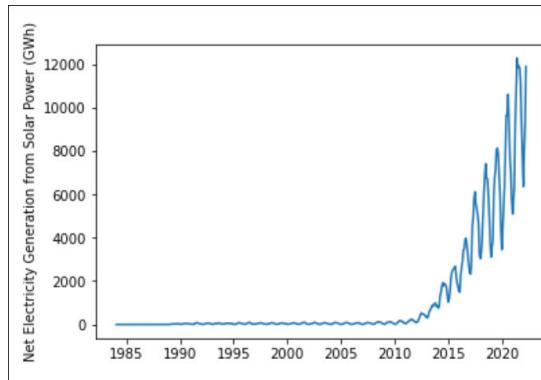
ElectNetGen	
Date	
1973-01	NaN
1973-02	NaN
1973-03	NaN
1973-04	NaN
1973-05	NaN

The `Date` index is now correctly formatted.

5. Plot the net electricity generation.

- a) Scroll down and view the cell titled **Plot the net electricity generation**, and examine the code cell below it.
b) Run the code cell.

- c) Examine the output.



- There has clearly been an increase in solar power energy usage over the years, with the most dramatic increases beginning around the mid-2010s.
- Because prior years saw little to no electricity generated from solar power, it may not be particularly useful to include that data in training. So, you'll treat every measurement before 2015 as an outlier.

6. Remove past months where solar-powered electricity generation was minimal.

- a) Scroll down and view the cell titled **Remove past months where solar-powered electricity generation was minimal**, and examine the code cell below it.
- b) Run the code cell.
- c) Examine the output.

ElectNetGen	
Date	
2015-01	1155.351
2015-02	1483.554
2015-03	2072.257
2015-04	2379.118
2015-05	2504.149
...	...
2021-11	7873.778
2021-12	6354.878
2022-01	8003.896
2022-02	9202.560
2022-03	11890.566

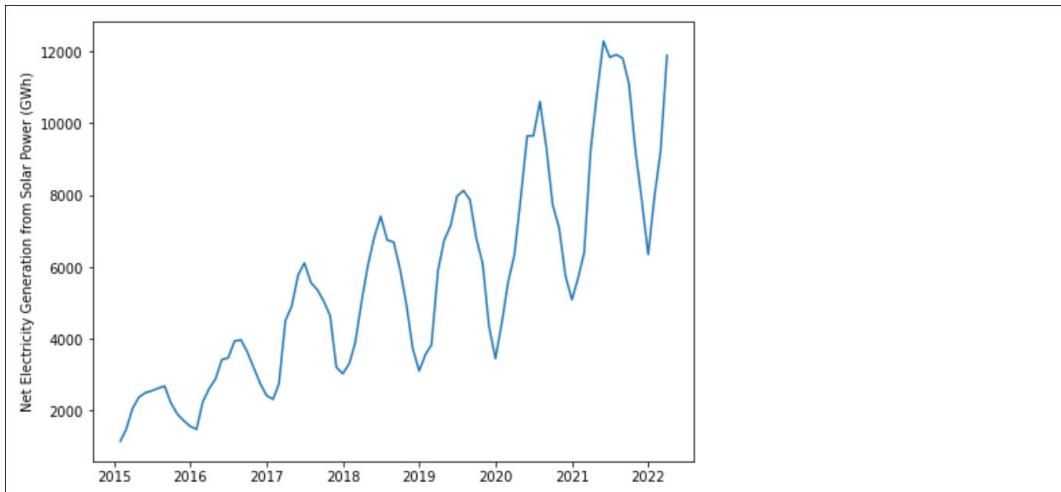
87 rows × 1 columns

The dataset now includes measurements from January 2015 onward. This has also removed the missing data.

7. Determine the seasonality of the dataset.

- a) Scroll down and view the cell titled **Determine the seasonality of the dataset**, and examine the code cell below it.
- If there is seasonality in the data, accounting for it in the model will likely improve its forecasting skill.
- b) Run the code cell.

- c) Examine the output.



From this graph, it seems like the electrical energy readings fluctuate in a yearly pattern. The beginning of the year (winter in the Northern Hemisphere) starts off low, then climbs to higher usage around the mid-year point (summer in the Northern Hemisphere). From there, it begins to fall in the later months of the year. The pattern then repeats for the next year.

To be sure, though, you'll confirm your supposition using a more statistical approach.

- d) Scroll down and examine the next code cell.

```

1 from statsmodels.tsa.seasonal import seasonal_decompose
2
3 # Obtain seasonality decomposition using moving averages.
4 seasonal_decomp = seasonal_decompose(df['ElectNetGen'],
5                                     model = 'multiplicative', # Changing magnitude.
6                                     period = 12)
7
8 fig = plt.figure(figsize = (8, 6))
9 seasonal_decomp.seasonal.plot()
10 plt.show();

```

- Lines 4 through 6 use the `statsmodels seasonal_decompose()` function to obtain seasonal information about the dataset. This process is called decomposition.
- Line 5 uses a multiplicative model as opposed to an additive one. The latter is used when the magnitude of the seasonality stays relatively constant over time. The former is used when the seasonal fluctuations increase or decrease over time. Since the seasonality of the electricity generation data is increasing over time, a multiplicative approach is used.

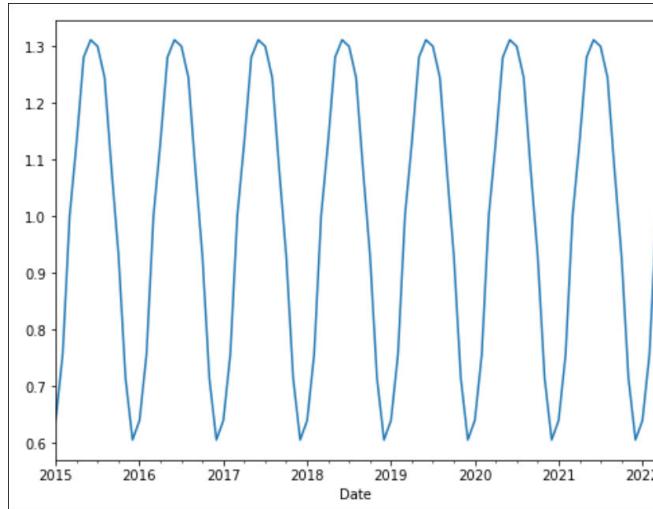


Note: This is *not* referring to the overall trend of values increasing, but the fact that the "peaks" and "valleys" of each season are increasing proportionally over time.

- Line 6 specifies the seasonal period. Since you hypothesized that the seasonal pattern is yearly, you are defining that period as 12 time increments (i.e., 12 months).
- Line 9 retrieves only the `seasonal` attribute from the decomposition data. There are more statistical measures associated with decomposition, but this one should suffice.

- e) Run the code cell.

- f) Examine the output.



Each year exhibits a smooth fluctuation, starting at the beginning of the year with a continuous ascent to its peak in the middle of the year, then a continuous descent toward the valley at the end of the year. This generally follows the peak and valley in each year of the raw data plot, confirming a 12-month seasonal pattern.

8. Split the dataset.

- Scroll down and view the cell titled **Split the dataset**, and examine the code cell below it.
 - There is no separate label to predict—just the one variable—so you'll create one set of training and testing data.
 - Unlike a problem solved by linear regression or some other supervised algorithm, your split must not be randomized. The time series must be maintained to get useful results. That's why `shuffle = False` will result in a test set that is the last part of the dataset, in order. The first part will be in the training set.
 - The `train_size = 80` argument is telling the function to assign a specific number of rows to the training set, with the rest going to the test set. Although a split of 60% training and 40% testing is more typical when you have a larger number of observations, this particular dataset has trouble remaining stationary with so few training observations.
- Run the code cell.
- Examine the output.

```
Original set: (87, 1)
-----
Training features: (80, 1)
Testing features: (7, 1)
```

9. Train a seasonal ARIMA model.

- Scroll down and view the cell titled **Train a seasonal ARIMA model**, and examine the code cell below it.
 - Lines 4 through 6 use the `ARIMA()` class from `statsmodels` to create the model.
 - For the non-seasonal `order` argument:
 - The number of lags is 1 (`p`, or autoregression).
 - The number of times the observations are differenced is 1 (`d`, or integration).
 - The number of lagged errors is 1 (`q`, or moving average).

The values being supplied are somewhat arbitrary, and they're not necessarily optimal, but they're a good starting point.

In the `seasonal_order` argument:

- The p , d , and q are the same.
- The period is 12. This ensures that the electrical energy measurements are modeled in the yearly context you confirmed earlier.

Line 8 fits the model. The training data was supplied as the first argument to `ARIMA()`, so it doesn't need to be specified here.

	Note: The call to <code>fit()</code> is suppressing a warning about the model being unable to optimize one of the statistical results. Despite this warning, the model is still useful.
---	--

- Run the code cell.
- Examine the output.

SARIMAX Results						
Dep. Variable:	ElectNetGen	No. Observations:	80			
Model:	ARIMA(1, 1, 1)x(1, 1, 1, 12)	Log Likelihood	-513.687			
Date:	Thu, 15 Sep 2022	AIC	1037.375			
Time:	16:08:06	BIC	1048.398			
Sample:	01-31-2015	HQIC	1041.737			
	- 08-31-2021					
Covariance Type:	opg					
	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.6343	0.215	2.952	0.003	0.213	1.055
ma.L1	-0.8838	0.131	-6.732	0.000	-1.141	-0.627
ar.S.I.12	-0.9406	0.160	-5.879	0.000	-1.254	-0.627

The output summarizes quite a bit about the model you just trained, including:

- The AR, I, and MA hyperparameters for both non-seasonal and seasonal aspects.
- The span of the time series in the training set ("sample").
- Model parameters (coefficients) for the autoregressive lag and moving average for both non-seasonal and seasonal aspect. Since you set 1 for each hyperparameter, there is only one lag and one moving average per aspect.
- Standard error and z-score values for the lags and moving averages.
- p-values for the lags and moving averages. Assuming an alpha (statistical significance) of 0.05, anything below this is desired, as it implies that you can reject the null hypothesis that the lags and moving averages have no significant affect on the model. Since they are indeed below 0.05, you can be confident that they do affect the model.
- Skew and kurtosis values for the dataset.
- And many more.

10. Forecast electricity generation using the testing set.

- Scroll down and view the cell titled **Forecast electricity generation using the testing set**, and examine the code cell below it.
 - This code uses the model to make predictions on the test set.
 - The `start` argument sets the start of the prediction to the same time value that begins the testing set.
 - The `end` argument does likewise, but for the last value.
- Run the code cell.

- c) Examine the output.

2021-09	10117.964875
2021-10	9452.071297
2021-11	7785.945458
2021-12	7056.843174
2022-01	7802.635454
2022-02	8772.876773
2022-03	10759.823003
Freq:	M
Name:	predicted_mean
Dtype:	float64

These are the predictions that the model made for the same time period that the testing set covers—September 2021 to March 2022, the last 7 months in the full dataset.

11. Calculate 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 cell below it.
- As with any other regression algorithm, you can evaluate an ARIMA model using the mean squared error (MSE).
 - On line 3, the `squared = False` argument is actually obtaining the *root* mean squared error (RMSE) to put the results on a scale that is closer to the actual data.
- b) Run the code cell.
- c) Examine the output.

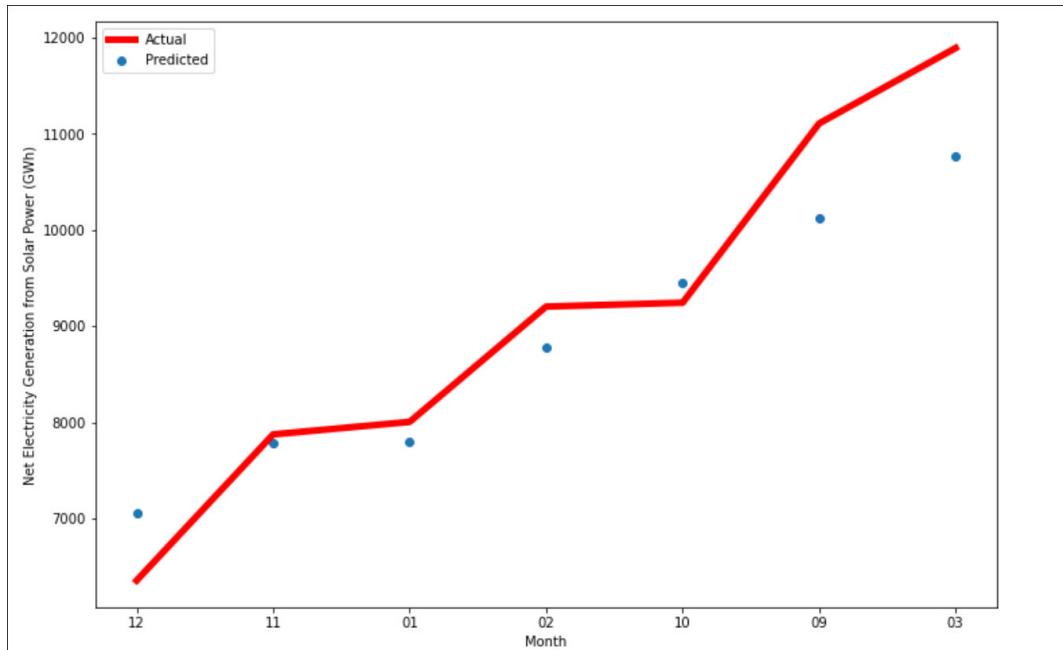
Cost (root mean squared error): 657.36
--

The RMSE is 657.36. Recall that, in context, each observation was in the thousands of GWh.

12. Plot the prediction residuals.

- a) Scroll down and view the cell titled **Plot the prediction residuals**, and examine the code cell below it.
This is essentially the same residual plot function you've seen before.
- b) Run the code cell.

- c) Examine the output.



- The dataset has been sorted by the actual electricity generated per month in the test set, which is shown as the red line.
- The predicted electrical energy generation at each monthly measurement is shown as a blue dot in a scatter plot. The farther the dot is from the line, the higher the amount of error between predicted and actual values.
- You can see that a few predictions are pretty close to the line, like month 11 (November 2021) and month 01 (January 2022).
- Other predictions are farther away, like month 09 (September 2021) and month 03 (March 2022).

13. Forecast electricity generation values into the future.

- a) Scroll down and view the cell titled **Forecast electricity generation values into the future**, and examine the code cell below it.

Now that you have your model, you can use it to forecast into the future. This code fits the model on new start and end dates—a span of one year after the last recorded observation.

- b) Run the code cell.
c) Scroll down and examine the next code cell.

```

1 # Set up new data frame for forecasting results.
2 forecasts = forecasts.rename('ElectNetGen')
3 forecast_df = pd.DataFrame(forecasts, index = forecasts.index, columns = ['ElectNetGen'])
4 forecast_df = df.append(forecast_df, sort = False)
5 forecast_df.tail(12)

```

This code assembles a new data frame that will hold the forecasted values, as well as date values for the historical dates. This will make plotting easier.

- d) Run the code cell.

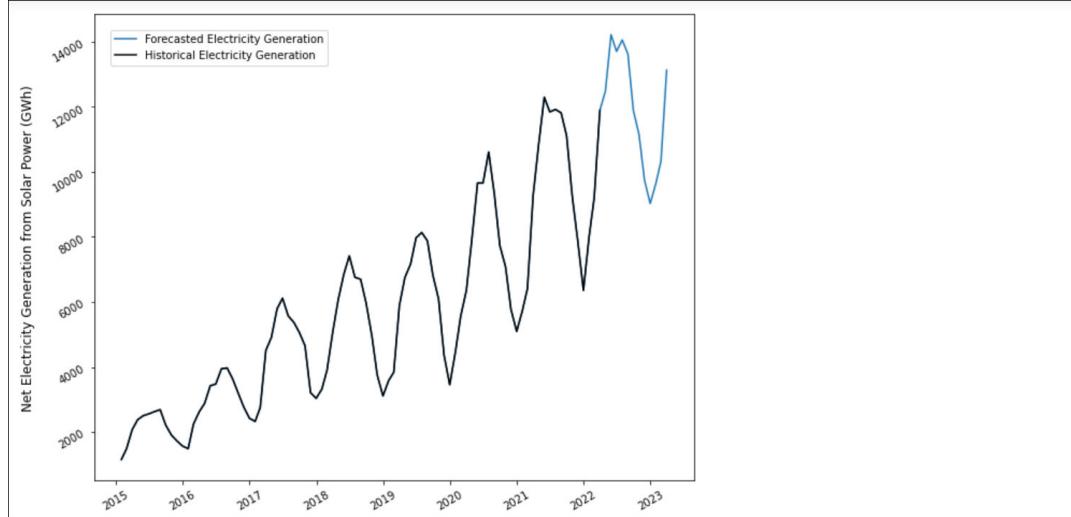
- e) Examine the output.

ElectNetGen	
2022-04	12474.754527
2022-05	14213.993267
2022-06	13699.264502
2022-07	14052.294391
2022-08	13623.327553
2022-09	11883.335309
2022-10	11139.778076
2022-11	9740.027194
2022-12	9023.377771
2023-01	9639.798996
2023-02	10328.210460
2023-03	13125.841188

The forecasted electrical energy generation values (in GWh) for the next 12 months in the U.S. are printed.

14. Visualize the forecasting trend.

- a) Scroll down and view the cell titled **Visualize the forecasting trend**, and examine the code cell below it.
- Lines 5 through 9 set up the line plot for the forecasted data values.
 - Lines 12 through 16 set up the line plot for the historical data values.
- b) Run the code cell.
- c) Examine the output.



- The historical electricity generation values are the black line.
- After March 2022, the blue line continues with the forecasted electricity generation values through to the following year.
- The model predicted that electrical energy generation from solar power will rise in the middle of 2022 to around 14,000 GWh, then fall to around 9,000 GWh at the end of the year, then start rising again in 2023. As in previous years, this demonstrates a seasonal, yet overall upward, trend in solar power usage.

15. How might you try to improve the skill of this forecasting model?

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 **Forecasting - Solar Power** tab in Firefox, but keep a tab open to **CAIP/Forecasting/** in the file hierarchy.
-

TOPIC B

Build Multivariate Time Series Models

You've built a simple ARIMA model using a univariate model, but if your time series data is more complex, you'll need to take a different approach.

Multivariate Time Series

A **multivariate** time series includes multiple variables whose changes are recorded in consistent increments of time. Just like in other applications of machine learning, most problems have several factors that influence the outcome of a prediction or other model estimation. In a multivariate time series, you're interested in analyzing how multiple variables relate to one another within the context of time. In other words, the variables not only depend on their own values in the past, but on the values of other variables as well.

Consider a task in which you need to predict the relative humidity in a data center. That way, you can trigger an alert before the humidity gets to be too low or too high, either of which can damage electronics. You have various Internet of Things (IoT) sensors positioned in and around the data center so that you can efficiently record and access weather data. The following is a truncated version of a multivariate dataset generated from these sensors:

Time	Outside Temperature (°F)	Wind Speed (mph)	Relative Humidity in Data Center (%)
00:00:00	54	9	49
01:00:00	54	8	46
02:00:00	56	7	45
03:00:00	58	11	43
04:00:00	60	13	43
05:00:00	61	17	42

Each row is an hour, so this dataset follows a consistent and sequential time interval, just like a univariate time series would. The three variables—outside temperature, wind speed, and relative humidity indoors—have various interdependent relationships that a time series analysis will attempt to analyze and reveal. Standard ARIMA cannot address multivariate time series problems like this one, so you'll need to use a different algorithm.

Vector Autoregression (VAR)

Vector autoregression (VAR) is an alternative to ARIMA that can forecast multivariate time series data. It is able to map the relationship between multiple variables by making each variable in time a function of its own past values and the past values of all other variables. Just like in ARIMA, the "autoregressive" part of VAR refers to the gap between each observation in time as a *lag*.

Consider that you have two variables, y_1 and y_2 . For both variables, you can calculate the autoregression for a single lag, also called a one-order or single-order model, represented as AR(1):

$$y_{1,t} = c_1 + a_{1,1}y_{1,t-1} + a_{1,2}y_{2,t-1} + e_{1,t}$$

$$y_{2,t} = c_2 + a_{2,1}y_{1,t-1} + a_{2,2}y_{2,t-1} + e_{2,t}$$

Where:

- t is the time interval.
- c_1 and c_2 are the constants (intercepts) for the first and second variables, respectively.
- $a_{1,1}$ is the first model coefficient for the first variable.
- $a_{1,2}$ is the second model coefficient for the first variable.
- $a_{2,1}$ is the first model coefficient for the second variable.
- $a_{2,2}$ is the second model coefficient for the second variable.
- e_1 and e_2 are the error terms for each variable.

Each variable in the time series has its own equation, so y_3 , y_4 , and so on, would follow this pattern. However, if you incorporated these two new variables, they would need to be added to each equation as well.

That was just one lag, though. Like an ARIMA model, a VAR model may benefit from multiple lags. To create a two-order model, you would need to effectively double the amount of addition operations in each equation above to account for the fact that the lags have doubled. This is where vectors and matrices come into play. Here's a VAR(2) model:

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} y_{1,t-1} \\ y_{2,t-1} \end{bmatrix} + \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} y_{1,t-2} \\ y_{2,t-2} \end{bmatrix} + \begin{bmatrix} e_{1,t} \\ e_{2,t} \end{bmatrix}$$

Notice how the matrix of a values is multiplied against two different lags (y_{t-1} and y_{t-2}).

Again, the more variables you add, the more complex the calculations become. $\text{VAR}(p)$, where p is the total number of lags (i.e., the order), and k is the total number of variables, can be represented like so:

$$\begin{bmatrix} y_{1,t} \\ \vdots \\ y_{k,t} \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_k \end{bmatrix} + \begin{bmatrix} a_{1,1} & \cdots & a_{1,k} \\ \vdots & \ddots & \vdots \\ a_{k,1} & \cdots & a_{k,k} \end{bmatrix} \begin{bmatrix} y_{1,t-1} \\ \vdots \\ y_{k,t-1} \end{bmatrix} + \cdots + \begin{bmatrix} a_{1,1} & \cdots & a_{1,k} \\ \vdots & \ddots & \vdots \\ a_{k,1} & \cdots & a_{k,k} \end{bmatrix} \begin{bmatrix} y_{1,t-p} \\ \vdots \\ y_{k,t-p} \end{bmatrix} + \begin{bmatrix} e_{1,t} \\ \vdots \\ e_{k,t} \end{bmatrix}$$

Or, in simplified notation:

$$\mathbf{y}_t = \mathbf{c} + \mathbf{A}\mathbf{y}_{t-1} + \cdots + \mathbf{A}\mathbf{y}_{t-p} + \mathbf{e}_t$$

Optimal Order Determination

There are several statistical methods for evaluating lag order. One of the most commonly used in VAR is the Akaike information criterion (AIC). AIC estimates prediction error in a set of data given a lag order. So, you could run multiple models with different lag order values, and whichever one produced the lowest AIC is the optimal choice (from that selection).

Endogenous and Exogenous Variables

A VAR model can also handle variables differently based on whether or not they are endogenous or exogenous to the model. An ***endogenous*** variable is explained by other variables in the model; an ***exogenous*** variable is not explained by other variables in the model. Any variables you wish to predict are by definition endogenous, whereas variables that have an affect on the model but are not predicted are exogenous. This is roughly equivalent to dependent and independent variables in standard regression analysis.

Consider a scenario where a farmer is trying to predict crop yield for an upcoming season. The crop yield and the quality of the soil are both endogenous, whereas the amount of pesticide used and the rainfall are exogenous. This is because:

- Crop yield is the variable you're trying to predict, so it is automatically *endogenous* to the model.
- Soil quality is affected by the rainfall (and possibly pesticide usage), so it is *endogenous* to the model.
- Pesticide usage is not affected by any of the other variables, so it is *exogenous* to the model.
- Rainfall is also not affected by any of the other variables, so it is likewise *exogenous* to the model.

VAR modeling libraries may require you to specify which variables are endogenous and which are exogenous. Of course, it's up to you to determine this. Domain knowledge regarding the problem you're trying to solve is crucial. If you have any gaps with such knowledge, you should consider doing more research in the relevant field.

Stationarity

If you recall, the "integrated" portion of ARIMA makes a time series stationary by differencing each consecutive observation in time. ***Stationarity*** is the property of a forecasting model by which the statistical attributes of a variable, like mean, variance, and covariance, are kept constant rather than varying over time.

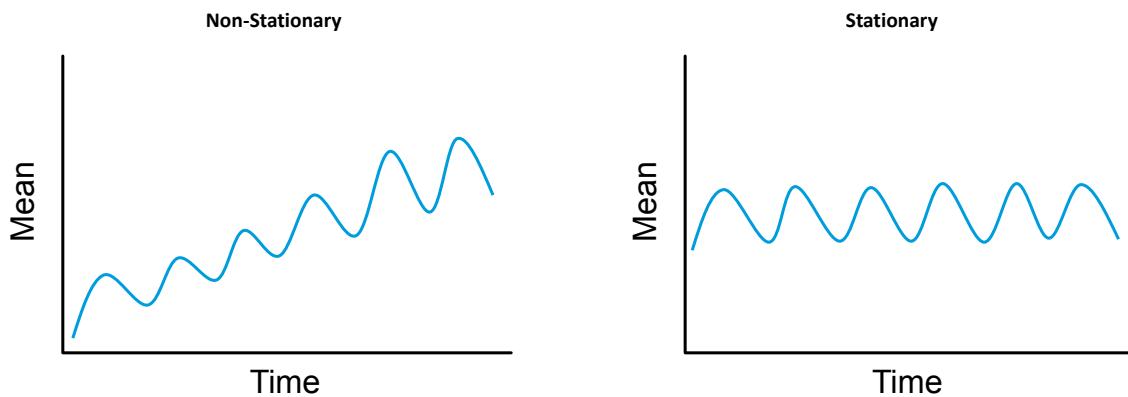


Figure 5–3: A non-stationary time series vs. a stationary one.

A non-stationary time series will lead to a model with fluctuating levels of effectiveness, as there may be large differences in statistics at different points within the series. A stationary time series, on the other hand, enables the model to make predictions based on the whole view of data at all relevant points in time.

VAR models therefore require stationary time series data. Depending on the library you use, you may need to perform this as part of a preprocessing effort before you begin fitting the model. The simplest method of making a time series stationary, as described previously, is by differencing the current observation from the observation in the previous lag. If you plan to make your model seasonal, you can also use seasonal differencing, in which the current observation is differenced from the previous observation in the same season (e.g., an observation in June of one year will be

differenced from the observation in June of the previous year). There are other transformation techniques to make a model stationary, including taking the log or the square root.

In order to know whether or not your attempt at stationarity was effective, you can run various tests that evaluate the stationarity of a time series. Two of the most common tests are the augmented Dickey–Fuller (ADF) test and the Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test. In the former, the null hypothesis is that the time series has a *unit root*, which essentially means that its statistical properties are *not* constant over time—in other words, not stationary. The latter test has the opposite null hypothesis—that the time series *is* stationary.

Either test is useful, and can help you determine which of the methods used to achieve stationarity is most effective.

VARMA

You can incorporate a moving average (MA) in VAR models to produce a VARMA model. VARMA models are more complex than standard VAR models and are less common.

VAR Example: Data Center Humidity

The historical data of relativity humidity readings each hour inside a data center for one day is shown in the following graph.

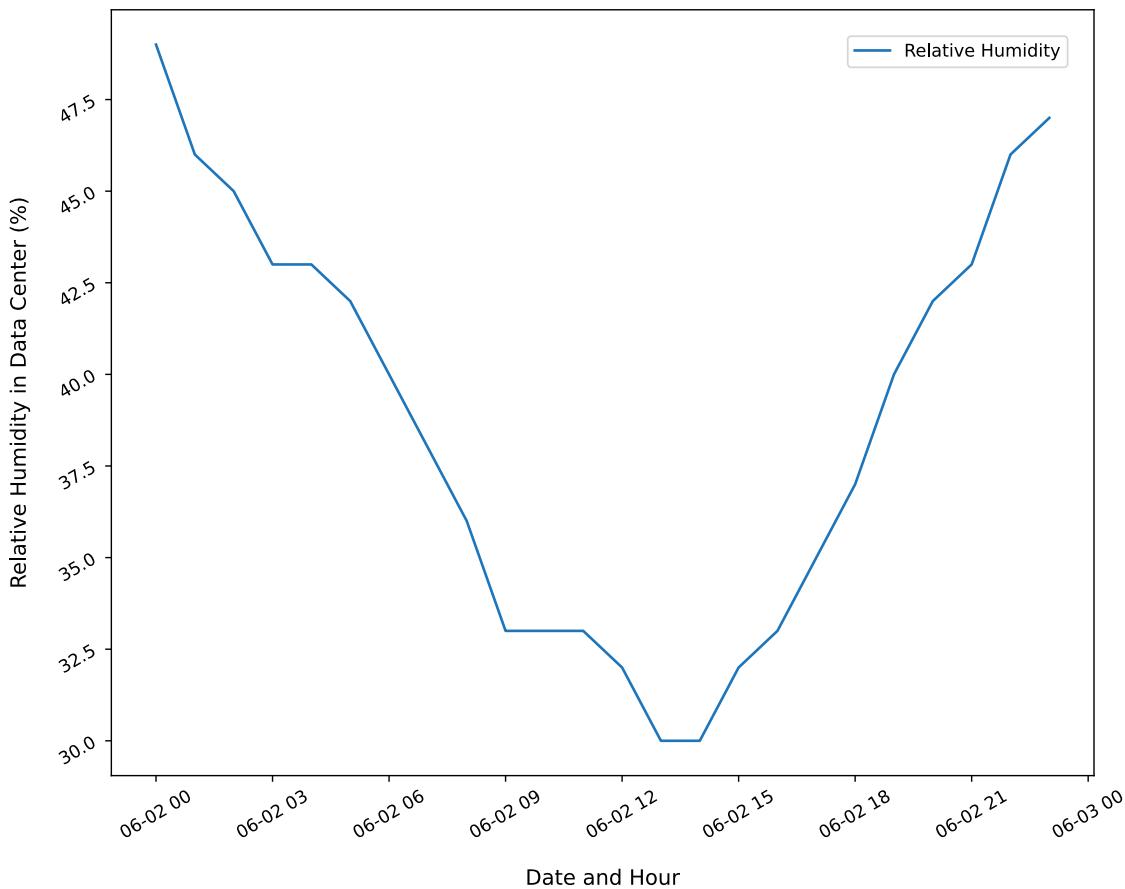


Figure 5–4: A line graph of the historical data center data.

As you can see, toward midday, the relative humidity in the data center dropped, whereas it began to climb during the night.

Before creating a VAR model, the stationarity of each variable is evaluated using the ADF test. This test returns the following *p*-values for each variable:

- Temperature: **0.00**
- Wind Speed: **0.99**
- Relative Humidity: **0.03**

Assuming a standard level of significance (alpha) of 0.05, you have sufficient evidence to reject the null hypothesis for Temperature and Relative Humidity because both *p*-values are lower. In other words, you can be confident that these variables are stationary. Wind Speed, on the other hand, has a *p*-value much higher than 0.05, so you cannot reject the null hypothesis. In other words, it is non-stationary.

Simple differencing can make Wind Speed stationary. After doing so, its *p*-value drops to 0.04, below the level of significance.

Now you can use this time series to train a VAR model. In particular, you want to forecast the first six hours of the next day. The results of a VAR(3) model are as follows.

Time	Forecasted Temperature (°F)	Forecasted Wind Speed Diff (mph)	Forecasted Relative Humidity in Data Center (%)
00:00:00	61.02	0.90	42.48
01:00:00	62.56	1.62	41.41
02:00:00	62.56	0.27	40.58
03:00:00	65.88	0.17	38.18
04:00:00	70.62	-0.48	35.29
05:00:00	68.73	1.57	34.64

In this case, all variables were considered endogenous to the model, and all were forecasted. Also note this table is showing the differenced wind speed values that were generated to make the variable stationary, rather than the true wind speed values.

Since you're most interested in forecasting the relative humidity of the data center, just that is shown in the following figure.

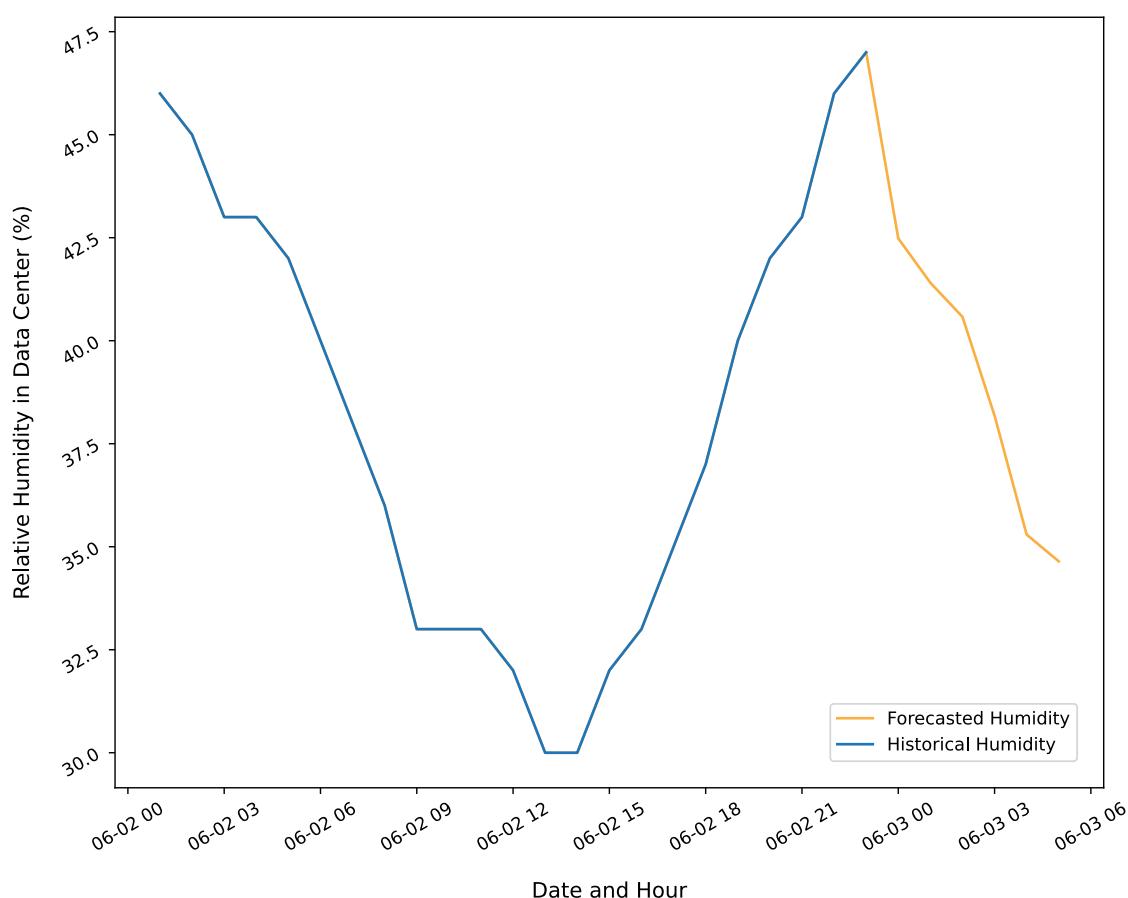


Figure 5–5: A graph of the relative humidity in the data center forecasted by the VAR model.

The forecasted humidity shows a downward trend in the early morning, much like the previous day.

Guidelines for Building Multivariate Time Series Models

Follow these guidelines when you are building multivariate time series models.

Build a Multivariate Time Series Model

When building a multivariate time series model:

- Ensure your time series follows a consistent pattern over time, just like a univariate dataset.
- Consider splitting your data into 60% training and 40% testing data when you have many observations.
- Split time series data into segments instead of selecting values at random, with training data always earlier than test data.
- Using your own domain knowledge, consider how each variable is or is not influenced by other variables to help you determine which are endogenous and which are exogenous.
- If you have gaps in your domain knowledge, research other relevant sources to help you determine which variables are endogenous and which are exogenous.
- Use tests like ADF or KPSS to evaluate the stationarity of your variables.
- Use differencing or other transformation techniques to turn a non-stationary variable into a stationary one.
- Keep in mind that all variables should be stationary before being used to train a VAR model.
- Consider using a library that can automatically determine the optimal number of lags to include in the model.

- Whenever feasible, plot your model to better illustrate trends in the model's forecasts.

Use Python for Multivariate Time Series Models

The `statsmodels` package in Python includes a `tsa.vector_ar.var_model` module that provides the `VAR()` class for multivariate time series forecasting. The following are some of the objects and functions you can use to build such a model.

- `model = statsmodels.tsa.vector_ar.var_model.VAR(train_endo, train_exo)` —This constructs a VAR model object with a set of endogenous variables (`train_endo`) and a set of exogenous variables (`train_exo`).
- `model_fit = model.fit(3)` —This fits the VAR model with 3 as the maximum number of lags.
- You can use this class object to call the `score()` method as you would with other regression models. You can also call `summary()` to get a detailed output of the model's attributes, including its coefficients, error scores, and distribution properties.
- `lags = model_fit.k_ar` —Use this attribute to retrieve the number of lags in the fit model.
- `model.predict(model_fit.params, start = lags, end = lags + 5, lags = lags)` —Use the model parameters (`model_fit.params`) to make predictions for the next time intervals that fall between `start` and `end`. In this example, `start` is equal to the number of lags, and `end` is the next five lags ahead of that (for a total of six predictions). The `lags` argument also takes the number of lags.
- `model_fit.forecast(y = train_endo[-lags:], steps = 6)` —An alternative method for forecasting/predicting the next number of specified steps. The `y` argument slices the last number of rows from the training set, where that number is equal to the number of lags.

You can also use the `tsa.stattools` module to test stationarity. The following are some of the functions you can use to perform such tests:

- `statsmodels.tsa.stattools.adfuller(train_var)` —This returns a tuple of various results of the ADF test on the provided variable. The second value in the tuple is the *p*-value.
- `statsmodels.tsa.stattools.kpss(train_var)` —This returns a tuple of various results of the KPSS test on the provided variable. The second value in the tuple is the *p*-value.

ACTIVITY 5–2

Building a Multivariate Time Series Model

Data Files

/home/student/CAIP/Forecasting/Forecasting - Wage Growth and Inflation.ipynb

/home/student/CAIP/Forecasting/economics_data/Raotbl6.csv

Before You Begin

Jupyter Notebook is open.

Scenario

In addition to research on U.S. energy consumption, you also do research on U.S. economics. You've been tasked with identifying how economic factors like gross national product (GNP) and inflation/deflation are changing over time. That way, your results can be used to anticipate economic challenges in the near future.

You have a time series dataset, but this time it has several variables associated with it. A univariate algorithm like ARIMA won't do. You'll need to use a multivariate algorithm like VAR to be able to forecast all of these economic factors.

1. From Jupyter Notebook, select CAIP/Forecasting/Forecasting - Wage Growth and Inflation.ipynb to open it.

2. Import the relevant libraries and load the dataset.

- View the cell titled **Import software libraries and load the dataset**, and examine the code cell below it.
- Run the code cell.
- Verify that **Raotbl6.csv** was loaded with 123 records.

This dataset includes various economic factors in the U.S. from 1959 to 1989. Although the data is fairly old, it can still be used to demonstrate forecasting using a multivariate time series model.

3. Get acquainted with the dataset.

- Scroll down and view the cell titled **Get acquainted with the dataset**, and examine the code cell below it.
- Run the code cell.

- c) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 123 entries, 0 to 122
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   date    123 non-null    object  
 1   rgnp    123 non-null    float64 
 2   pgnp    123 non-null    float64 
 3   ulc     123 non-null    float64 
 4   gdfco   123 non-null    float64 
 5   gdf    123 non-null    float64 
 6   gdfim   123 non-null    float64 
 7   gdfcf   123 non-null    float64 
 8   gdfce   123 non-null    float64 
dtypes: float64(8), object(1)
memory usage: 8.8+ KB
None

   date  rgnp  pgnp  ulc  gdfco  gdf  gdfim  gdfcf  gdfce
0  1959-01-01  1606.4  1608.3  47.5   36.9  37.4   26.9   32.3  23.1
1  1959-04-01  1637.0  1622.2  47.5   37.4  37.5   27.0   32.2  23.4
2  1959-07-01  1629.5  1636.2  48.7   37.6  37.6   27.1   32.4  23.4
3  1959-10-01  1642.4  1650.9  48.8   37.7  37.8   27.1   32.5  23.8
```

This dataset has 123 rows and 9 columns. Except for the `date` column, all of them are floats, and there does not appear to be any missing data. The columns are:

- `date`, which records each observation every 3 months. In other words, these are quarterly observations for each year.
- `rgnp`, the real gross national product (GNP), the value of all products and services produced by a country's residents. In recent times, GNP has largely been replaced by the gross national income (GNI), which determines the same thing but using a different calculation.
- `pgnp`, the potential GNP. Potential GNP uses a constant rate of inflation, whereas real GNP uses a variable rate of inflation. Potential GNP is therefore an estimate, and is often used to measure the GNP of the following fiscal quarter.
- `ulc`, the unit labor cost (ULC). ULC measures labor productivity by calculating the average cost of labor per unit of output.
- `gdfco`, the fixed weight deflator for personal consumption expenditure (excluding food and energy). In GNP/GDI, a deflator is a measure of both price inflation and deflation.
- `gdf`, the fixed weight deflator for the GNP. This deflator covers the entire GNP, rather than just one aspect.
- `gdfim`, the fixed weight deflator for imports.
- `gdfcf`, the fixed weight deflator for food in personal consumption expenditures.
- `gdfce`, the fixed weight deflator for energy in personal consumption expenditures.

The purpose of the article this dataset is taken from is to determine the causal relationship between wages and prices. In this machine learning project, you're interested in forecasting these factors for several quarters after the final observation.

First, however, you need to prepare the time series.

4. Convert the `date` column to a datetime index.

- Scroll down and view the cell titled **Convert the date column to a datetime index**, and examine the code cell below it.
- Run the code cell.

- c) Examine the output.

	rgnp	pgnp	ulc	gdfo	gdf	gdfim	gdfcf	gdfce
date								
1959-01	1606.4	1608.3	47.5	36.9	37.4	26.9	32.3	23.1
1959-04	1637.0	1622.2	47.5	37.4	37.5	27.0	32.2	23.4
1959-07	1629.5	1636.2	48.7	37.6	37.6	27.1	32.4	23.4
1959-10	1643.4	1650.3	48.8	37.7	37.8	27.1	32.5	23.8
1960-01	1671.6	1664.6	49.1	37.8	37.8	27.2	32.4	23.8
...
1988-07	4042.7	3971.9	179.6	131.5	124.9	106.2	123.5	92.8
1988-10	4069.4	3995.8	181.3	133.3	126.2	107.3	124.9	92.9
1989-01	4106.8	4019.9	184.1	134.8	127.7	109.5	126.6	94.0
1989-04	4132.5	4044.1	186.1	134.8	129.3	111.1	129.0	100.6
1989-07	4162.9	4068.4	187.4	137.2	130.2	109.8	129.9	98.2

123 rows x 8 columns

- The date column has been turned into the data frame's index, and is now in the proper format.
- The earliest observation is from January 1959, and the last observation was in July 1989.

5. Plot all features.

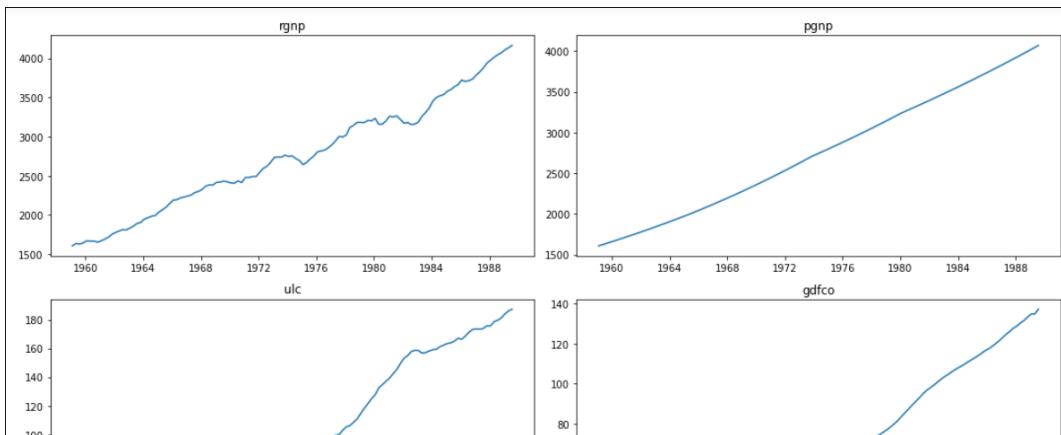
- a) Scroll down and view the cell titled **Plot all features**, and examine the code cell below it.

This function, when called, will plot every feature in the dataset as a separate line graph.

- Lines 3 through 5 set up the figure subplots for each feature.
- Line 7 begins a `for` loop that will iterate through each column/feature so that each can be added to a subplot.
- Lines 9 through 29 check to see if the function call includes prediction data. If it does, lines 11 through 26 plot both the historical data and the predicted data. Otherwise, lines 28 and 29 just plot the historical data.

- b) Run the code cell.

- c) Examine the output.



Each feature is plotted.

- All of the features are generally trending upward.
- rgnp, ulc, gdfim, gdfcf, and gdfce have occasional dips, with gdfim and gdfce being the most pronounced around the middle of the 1980s.
- pgnp, gdfo, and gdf are fairly consistently rising, particularly pgnp.

6. Split the dataset.

- a) Scroll down and view the cell titled **Split the dataset**, and examine the code cell below it.

- Since there is no single label to predict, you'll create one set of training and testing data.
 - Like with ARIMA or any other time series model, the split for a VAR model must not be randomized. This is why `shuffle = False`.
 - 115 observations will be assigned to the training set, and the rest to the test set. With a larger dataset, a more proportional split might be beneficial. In this case, a smaller test set is optimal.
- b) Run the code cell.
c) Examine the output.

```
Original set: (123, 8)
-----
Training features: (115, 8)
Testing features: (8, 8)
```

7. Test the time series for stationarity.

- a) Scroll down and view the cell titled **Test the time series for stationarity**, and examine the code cell below it.

This function, when called, will perform both ADS and KPSS tests on all features to determine their stationarity (or lack thereof).

- Lines 5 and 6 ignore warnings regarding the KPSS scores that will be explained shortly.
 - Lines 10 through 17 retrieve the ADS and KPSS scores and compile them. Note that there are several results of each test, but since you're most interested in the *p*-value, that's the only one you'll retrieve. The *p*-value is at index 1 in the results.
 - Lines 19 and 20 construct a data frame of the test results.
- b) Run the code cell.
c) Examine the output.

	ADF p-value	KPSS p-value
<code>rgnp</code>	0.976808	0.01
<code>pgnp</code>	0.994343	0.01
<code>ulc</code>	0.994495	0.01
<code>gdfco</code>	0.997674	0.01
<code>gdf</code>	0.997371	0.01
<code>gdfim</code>	0.960921	0.01
<code>gdfcf</code>	0.996961	0.01
<code>gdfce</code>	0.929620	0.01

Assuming an alpha (statistical significance) of 0.05, anything below this means you can reject the null hypothesis. So:

- For ADF, the null hypothesis is that the time series has a unit root—i.e., it's not stationary. Since all of the *p*-values are above 0.05, you cannot reject this null hypothesis. In other words, ADF suggests that none of the features are stationary.
- For KPSS, the null hypothesis is the opposite—that the time series *is* stationary. Since all of the *p*-values are below 0.05, you can reject this null hypothesis. In other words, KPSS suggests that none of the features are stationary.
- Both tests agree. You'll need to make the features stationary before you proceed.



Note: The warnings that were suppressed indicate that *p*-values for KPSS may actually be lower than what is being reported. In any case, the conclusion is the same.

8. Difference the features to make them stationary.

- a) Scroll down and view the cell titled **Difference the features to make them stationary**, and examine the code cell below it.

Line 2 uses pandas' `diff()` function to take the difference between one data element and another. The default is to use the element in the prior row. It also drops the first row since it'll have a missing value.

- Run the code cell.
- Examine the output.

	ADF p-value	KPSS p-value
<code>rgnp</code>	0.00	0.10
<code>pgnp</code>	0.33	0.01
<code>ulc</code>	0.01	0.01
<code>gdfco</code>	0.67	0.01
<code>gdf</code>	0.61	0.01
<code>gdfim</code>	0.00	0.10
<code>gdxfc</code>	0.03	0.01
<code>gdfce</code>	0.04	0.10

The results show some improvement.

- `rgnp`, `ulc`, `gdfim`, `gdxfc`, and `gdfce` all have ADF *p*-values below alpha, as desired.
- Of those, only `rgnp` and `gdfim` have KPSS *p*-values above alpha, as desired.
- `pgnp`, `gdfco`, and `gdf` still have large ADF *p*-values and low KPSS *p*-values, which you want to avoid.

If one round of differencing doesn't produce the desired results, another round might. So, you'll difference the time series a second time.

- Scroll down and examine the next code cell.

```

1 # Second differencing.
2 X_diff = X_diff.diff().dropna()
3
4 with pd.option_context('float_format', '{:.2f}'.format):
5     display(test_stationarity(X_diff))

```

Line 2 applies the same differencing operation, but this time to the already-differenced data. For consistency, you'll apply this second round of differencing to all features, even the ones that were made stationary in the first round.

- Run the code cell.
- Examine the output.

	ADF p-value	KPSS p-value
<code>rgnp</code>	0.00	0.10
<code>pgnp</code>	0.00	0.10
<code>ulc</code>	0.00	0.10
<code>gdfco</code>	0.00	0.10
<code>gdf</code>	0.00	0.10
<code>gdfim</code>	0.00	0.10
<code>gdxfc</code>	0.01	0.10
<code>gdfce</code>	0.00	0.10

All of the features are now stationary.

9. What is the difference between an exogenous variable and an endogenous variable, and why is this important?

10. Determine the optimal lag order for a VAR model.

- Scroll down and view the cell titled **Determine the optimal lag order for a VAR model**, and examine the code cell below it.

To start with, you want to determine a good lag order for the VAR model, rather than supply an arbitrary number.

- Line 3 creates the VAR model on the differenced training data. Note that the first argument in `VAR()` is for supplying endogenous variables, whereas the second (optional) argument is for supplying exogenous variables. In this case, the assumption is that all of the variables are endogenous, so only the first argument is used.
- Without adequate domain knowledge, it can be difficult to tell which variables are endogenous and which are exogenous. Here, since you want to predict all of the variables, it's appropriate to treat them all as endogenous.
- Lines 8 through 9 fit the model on a range of different lag orders—from 1 to 9. This is the number of lags the model will use in its calculations. So, the code is creating a VAR(1) model, a VAR(2) model, and so on, all the way to VAR(9).
- Line 10 retrieves the AIC value for each fit, which you'll use to determine which lag order is optimal.

- Run the code cell.
- Examine the output.

AIC
Lag order
1 -1.575275
2 -1.879888
3 -2.083729
4 -2.238175
5 -2.143585
6 -2.711646
7 -3.184860
8 -3.777092
9 -5.362435

- The lower the AIC, the better.
- With the exception of going from VAR(4) to VAR(5), increasing the lag order decreases the AIC.
- The lowest AIC, and therefore the optimal lag order, from this set is 9, so you'll create a VAR(9) model.
- It's possible that an even higher-order model will be better, but VAR(9) should be good enough for a first attempt.

11. Train a VAR model using the optimal lag order.

- Scroll down and view the cell titled **Train a VAR model using the optimal lag order**, and examine the code cell below it.

- Line 1 creates the VAR model again.
- Line 2 fits the model using a lag order of 9, the optimal number you just identified.
- Line 3 prints the model parameters.

- Run the code cell.

- c) Examine the output.

	rgnp	pgnp	ulc	gdfco	gdf	gdfim	gdfcf	gdfce
const	3.418504	0.042473	-0.112856	0.062907	-0.010802	-0.149534	-0.038464	-0.178736
L1.rgnp	-0.873544	-0.006685	0.014354	-0.002444	-0.000116	0.005180	-0.001551	-0.015609
L1.pgnp	-1.086610	0.084172	-0.296108	-0.154218	0.038962	0.057240	0.120775	0.678776
L1.ulc	-6.459951	-0.160868	-0.538248	0.023282	-0.002949	0.618174	0.011763	0.064490
L1.gdfco	-0.755460	-0.017761	-0.047289	-0.918690	-0.033519	0.417048	-0.173029	-0.200624
...
L9.gdfco	9.279105	0.127019	-0.071998	-0.221985	-0.111364	-0.148293	-0.038265	-0.174798
L9.gdf	17.315635	-1.197180	-2.049459	0.091355	0.346067	2.590370	0.640226	5.411070
L9.gdfim	6.367623	-0.078356	-0.287804	-0.168161	-0.097690	-0.145009	0.048420	-0.312453
L9.gdfcf	-6.280837	0.397566	0.352019	-0.063645	-0.006649	-0.935462	-0.354426	-1.263333
L9.gdfce	-6.404416	-0.107878	0.243754	0.069833	0.011173	-0.264462	-0.119433	-0.244649

73 rows × 8 columns

These are the parameters the model will use to make its predictions. Note that every feature has its own set of parameters that includes a constant term, the first lag for each feature, the second lag for each feature, and so on, until the ninth and final lag for each feature.

12. Forecast the features using the testing set.

- a) Scroll down and view the cell titled **Forecast the features using the testing set**, and examine the code cell below it.
- Line 1 retrieves the lag order from the model.
 - Line 2 specifies how many steps out the forecast will take. In this case, it's the length of the testing set plus 6. This will leave enough room for the model to forecast both the testing set and 6 quarters after it (i.e., 6 new quarters after the last observation in the entire dataset).
 - Line 5 conducts the forecast using the model.
 - Lines 7 through 10 set up the datetime indices for the forecasts.
 - Line 12 constructs a data frame of the forecasted values.
- b) Run the code cell.

- c) Examine the output.

	rgnp	pgnp	uic	gdfo	gdf	gdflm	gdfcf	gdfce
1987-10	-14.233775	-1.870785	-1.131133	-0.830680	-0.200126	2.642161	-0.319408	0.651013
1988-01	24.325838	-1.712522	-0.158721	-0.181857	0.218529	-2.187089	2.523647	1.218391
1988-04	56.501626	1.318235	1.960552	1.774484	0.750960	0.266142	-1.142962	0.256895
1988-07	-42.980135	-0.188752	2.295153	-0.211500	0.164765	-0.134131	0.346343	-0.402016
1988-10	23.163390	0.657848	-4.059509	-1.133189	-0.903459	-1.778257	-1.102373	-5.411998
1989-01	-107.369891	-1.436585	1.941890	0.773813	-0.276434	-3.672836	-0.373006	-4.987087
1989-04	-34.478345	-0.294396	1.833708	-0.398364	0.300643	-1.523516	0.551775	3.706254
1989-07	14.711352	1.623452	2.276685	0.731362	-0.013038	2.743329	-0.767901	-2.404085
1989-10	46.035076	1.527938	-3.329985	-0.802790	0.169056	1.679495	1.594767	2.859980
1990-01	53.961704	1.965332	-3.322797	0.187198	-0.860853	-0.041353	-2.014250	2.204786
1990-04	-143.493999	-0.350026	2.971762	-0.077652	-0.159922	-0.729605	-1.662739	-2.164832
1990-07	118.295678	-1.136324	-2.301520	-0.364267	-0.294658	1.733233	2.180093	-2.517001
1990-10	1.872659	-1.145262	-0.838195	-0.078137	0.599189	0.472629	1.388989	2.261060
1991-01	62.794031	0.017265	0.826342	0.917842	0.658071	1.799255	0.315402	4.452431

The forecasts for each feature are printed.

- The forecasts begin on October 1987, which is the first quarter included in the test set.
- The forecasts end on January 1991, which is 6 quarters after the last quarter in the test set (July 1989).
- The problem with these forecasts is that they're on the scale of the stationary time series. You'll need to reverse the differencing to get neater results.

13. Reverse the stationarity to get the data back to its original scale.

- Scroll down and view the cell titled **Reverse the stationarity to get the data back to its original scale**, and examine the code cell below it.
 - Line 3 begins a `for` loop that will reverse the differencing done to each feature.
 - Lines 5 and 6 start with the second differencing operation and reverse that. This reversal operation subtracts a data element from the element before it, then adds that result to the cumulative sum (`cumsum()`) of the data frame.
 - Line 9 reverses the first differencing operation. A data element is added to the result of the cumulative sum of the previous operation.
 - Line 10 drops the result of the initial reversal from the data frame.
- Run the code cell.
- Examine the output.

	rgnp	pgnp	uic	gdfo	gdf	gdflm	gdfcf	gdfce
1987-10	3907.866225	3899.029215	173.268867	126.669320	120.499874	106.042161	117.980592	95.551013
1988-01	3967.258289	3918.645909	172.479013	127.156783	121.618276	107.997232	120.784832	99.220417
1988-04	4083.151978	3939.580837	173.649710	129.418731	123.487638	110.218446	122.446110	103.146715
1988-07	4156.065532	3960.327013	177.115561	131.469179	125.521765	112.305529	124.453730	106.670999
1988-10	4252.142477	3981.731038	176.521903	132.386438	126.652433	112.614354	125.358978	104.783284
1989-01	4240.849530	4001.698476	177.870135	134.077510	127.506667	109.250343	125.891220	97.908482
1989-04	4195.078238	4021.371519	181.052075	135.370218	128.661545	104.362817	126.975237	94.739934
1989-07	4164.018299	4042.668014	186.510699	137.394288	129.803385	102.218619	127.291352	89.167301
1989-10	4178.993435	4065.492446	188.639339	138.615568	131.114281	101.753916	129.202235	86.454649
1990-01	4247.930276	4090.282210	187.445182	140.024047	131.564324	101.247859	129.098868	85.946782
1990-04	4173.373117	4114.721948	189.222787	141.354873	131.854445	100.012197	127.332762	83.274083
1990-07	4217.111637	4138.025362	188.698872	142.321432	131.849909	100.509768	127.746749	78.084384
1990-10	4262.722816	4160.183513	187.336762	143.209853	132.444561	101.479968	129.549725	75.155745
1991-01	4371.128025	4182.358930	186.800993	145.016117	133.697285	104.249423	131.668103	76.679537

Now, the prediction values are in the proper scale.

14. Calculate 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 cell below it.

This code generates RMSE values for all features in the test set compared to the predictions. Line 7 ensures that only the predictions matching the test set, and not the 6 new predictions, will be included.

- b) Run the code cell.
c) Examine the output.

Cost (RMSE)	
rgnp	96.312284
pgnp	15.622959
ulc	4.128757
gdfco	0.847404
gdf	0.418068
gdfim	5.064099
gdfcf	1.460046
gdfce	9.007686

The error values for each feature are printed.

- Note that `rgnp` and `pgnp` are on larger scales than the others, so it makes sense that their error values are higher.
- Interestingly, despite being on similar scales, the error in the former is much higher than the latter.
- You could use these results to tune your model until it begins showing improvement. For the purposes of this project, this should be sufficient.

15. Visualize the forecasting trend on new data.

- a) Scroll down and view the cell titled **Visualize the forecasting trend on new data**, and examine the code cell below it.

This code will only retrieve the last 6 predictions from the model, which are the predictions for fiscal quarters that go beyond what was in the original dataset. It also appends this data to the full dataset so that the forecast is more easily plotted.

- b) Run the code cell.
c) Examine the output.

	rgnp	pgnp	ulc	gdfco	gdf	gdfim	gdfcf	gdfce
1989-10	4178.993435	4065.492446	188.639339	138.615568	131.114281	101.753916	129.202235	86.454649
1990-01	4247.930276	4090.282210	187.445182	140.024047	131.564324	101.247859	129.098868	85.946782
1990-04	4173.373117	4114.721948	189.222787	141.354873	131.854445	100.012197	127.332762	83.274083
1990-07	4217.111637	4138.025362	188.698872	142.321432	131.849909	100.509768	127.746749	78.084384
1990-10	4262.722816	4160.183513	187.336762	143.209853	132.444561	101.479968	129.549725	75.155745
1991-01	4371.128025	4182.358930	186.800993	145.016117	133.697285	104.249423	131.668103	76.679537

The 6 new forecasted quarters are shown.

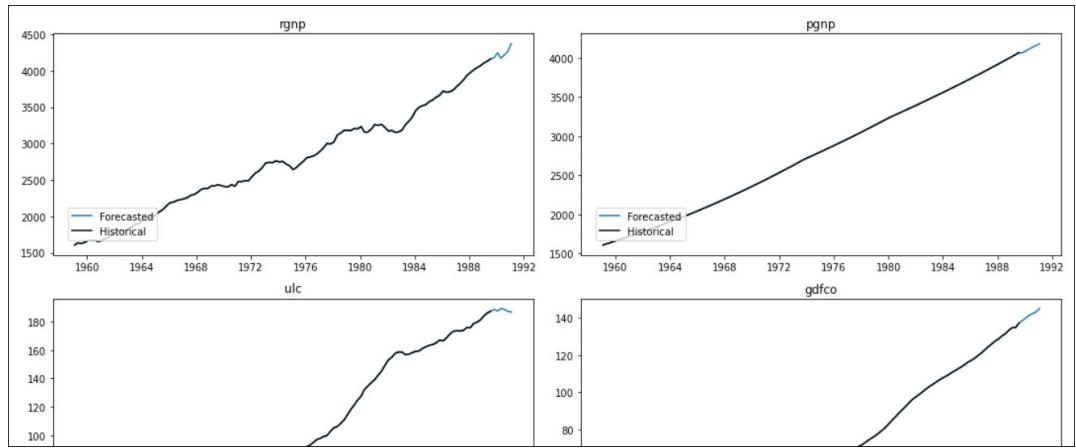
- d) Scroll down and examine the next code cell.

```
1 plot_feats(df, forecast_df, has_preds = True)
```

This calls the plotting function defined earlier, but this time you're supplying the predictions.

- e) Run the code cell.

f) Examine the output.



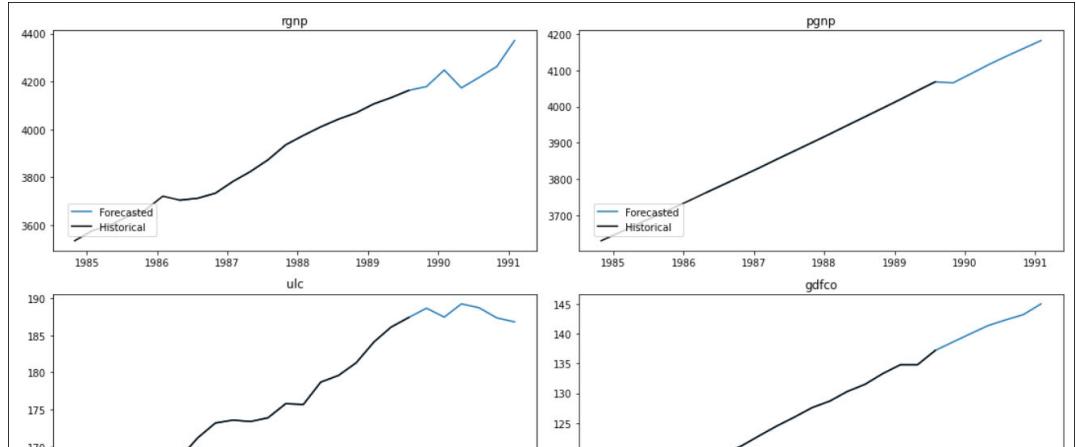
- The historical values are the black lines, whereas the blue lines indicate the forecasts starting with October 1989.
- Most forecasts show an increase, though some start to dip.
- It's difficult to interpret the forecasts in such a long-term context, so you'll zoom in on the graphs.

g) Scroll down and examine the next code cell.

```
1 # Zoom in on tail end of data.
2 plot_feats(df[-20:], forecast_df[-20:], has_preds = True)
```

This time, you'll only plot the last 20 quarters.

- h) Run the code cell.
i) Examine the output.



Now that the graphs are zoomed in, it's easier to see the forecasted trend.

- rgnp is forecasted to fluctuate in the early 1990s, but ultimately start growing.
- pgnp is forecasted to mostly grow, with one hiccup in the late 80s.
- ulc is forecasted to fluctuate and appears to even out.
- gdfco is forecasted to grow.
- gdf is forecasted to mostly grow.
- gdfim is forecasted to significantly decline in the late 80s and the early 90s, but seems to start recovering.
- gdfcf is forecasted to take a slight dip in 1990s, but then start growing.
- gdfce is forecasted to significantly decline.

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 **Forecasting - Wage Growth and Inflation** tab in Firefox, but keep a tab open to **CAIP** in the file hierarchy.
-

Summary

In this lesson, you built models that can forecast future values from data that follows a time series. You started by building a univariate ARIMA model for simple forecasting, then moved onto more complex forecasting of multivariate data using VAR. Forecasting methods like these will enable you to, in a sense, predict the future—and being able to predict the future will give your organization the ability to anticipate and prepare for changes, rather than react to them when they happen.

What type of data in your organization do you think might be conducive to forecasting?

Do you think you'll be more likely to use univariate forecasting or multivariate forecasting? 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.

6

Building Classification Models Using Logistic Regression and k-Nearest Neighbor

Lesson Time: 4 hours

Lesson Introduction

You've built regression models that can tackle numeric 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 common techniques for building classification machine learning models.

Lesson Objectives

In this lesson, you will:

- Train binary classification models using logistic regression.
- Train binary classification models using k -nearest neighbor (k -NN).
- Train multi-class classification models using logistic regression and k -NN.
- Use various metrics to evaluate the performance of classification models.
- Use various techniques to tune the performance of classification models.

TOPIC A

Train Binary Classification Models Using Logistic Regression

To begin with, you'll train binary classification models using one of the common algorithms for doing so: logistic regression.

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:

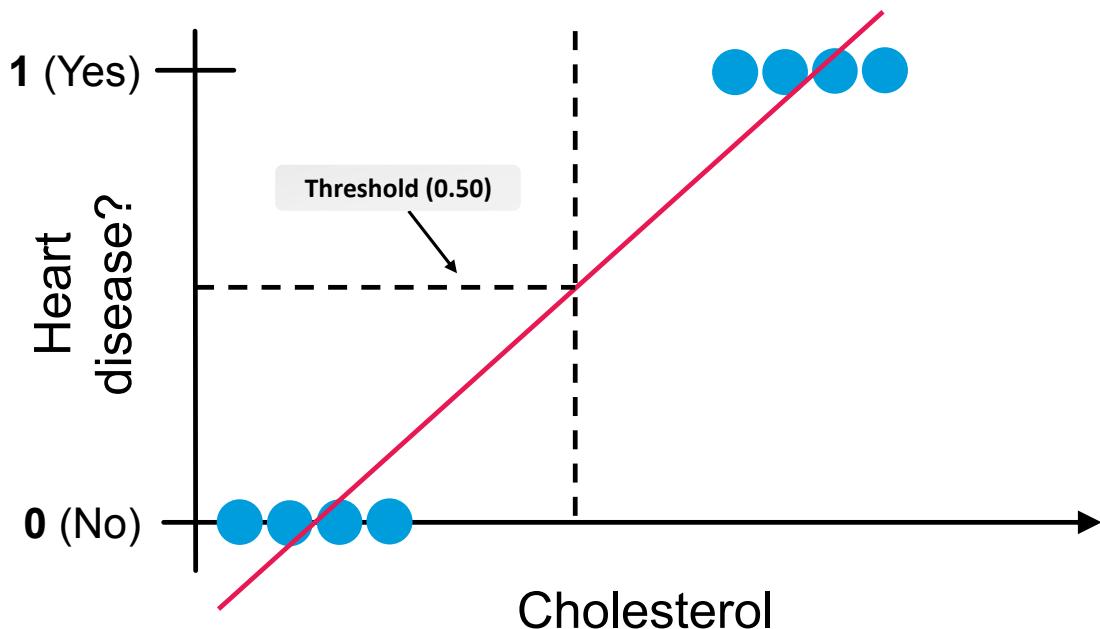


Figure 6–1: Fitting a straight line to a classification problem.

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 classified as 1 (has heart disease), and any predictions below 0.50 (to the left of the threshold) are classified 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?

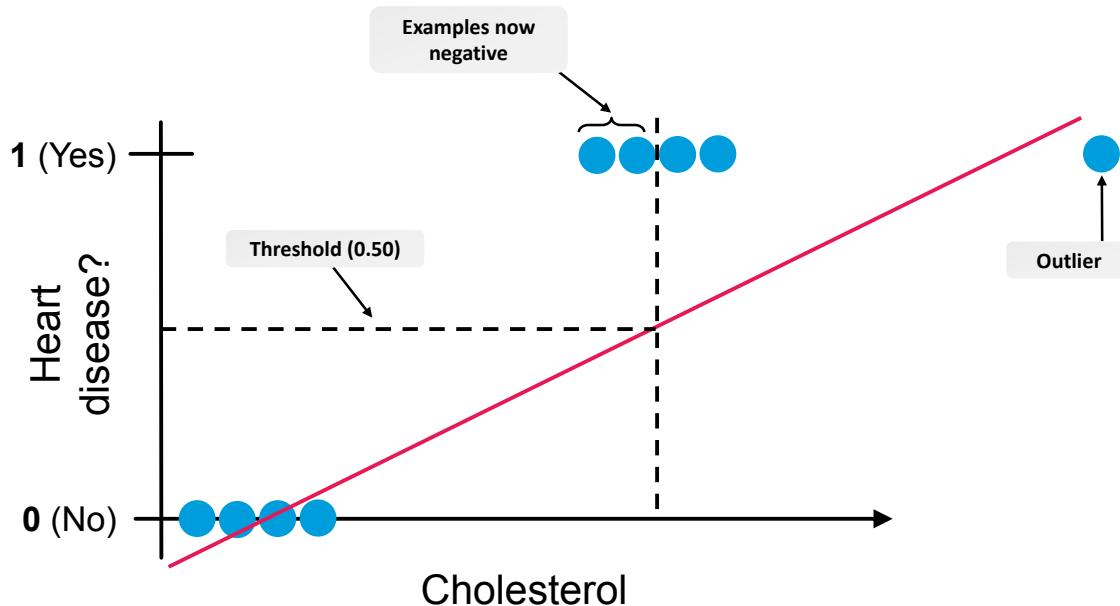


Figure 6–2: An outlier skews the line and changes the classification results.

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 regression in which the output is a classification probability 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. It is also one of the most common machine learning algorithms used in classification because it can produce results relatively quickly on larger datasets.

The term *logistic regression* can be somewhat confusing. It is technically a type of regression analysis since it estimates the relationships between dependent and independent variables. However, it is used to output a classification label and not a continuous value. Therefore, it does not produce the outcome commonly referred to as "regression," but instead produces a classification outcome.

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 like in linear regression, the logistic function takes an *S* shape, as shown here:

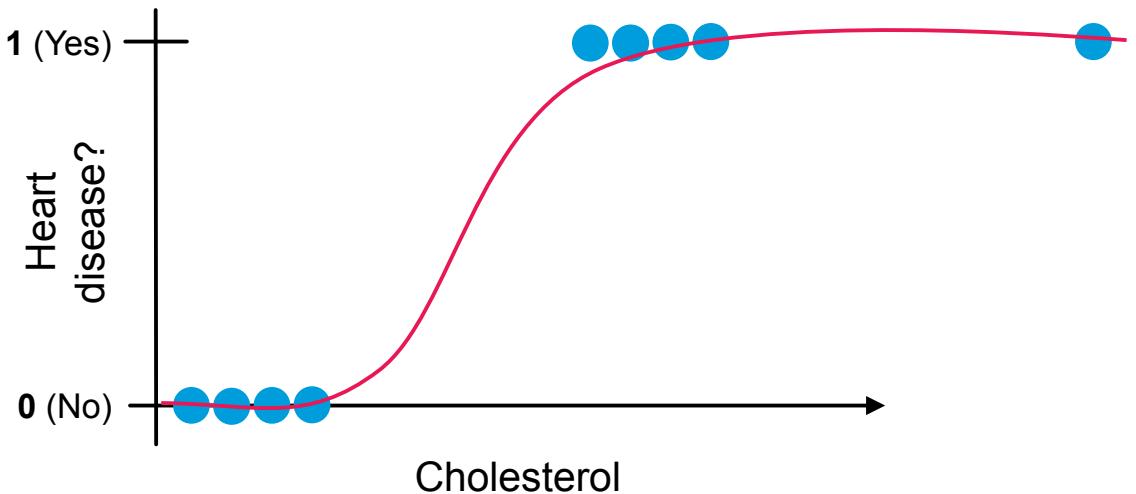


Figure 6-3: A sigmoid function used in logistic regression.

Notice that function is better able to fit the outlier, and that in doing so, it takes an *S* shape.

The Logistic Function

The logistic function can be expressed as the following equation:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Where:

- t is the estimation between negative infinity and positive infinity.
- e is the natural logarithm base.

The estimation 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—the values that the model "learns" in order to enhance its decision-making capabilities.

Decision Boundaries

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 estimation 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:

$\text{if } \hat{p} \geq 0.50, \hat{y} = 1$

$\text{if } \hat{p} < 0.50, \hat{y} = 0$

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:

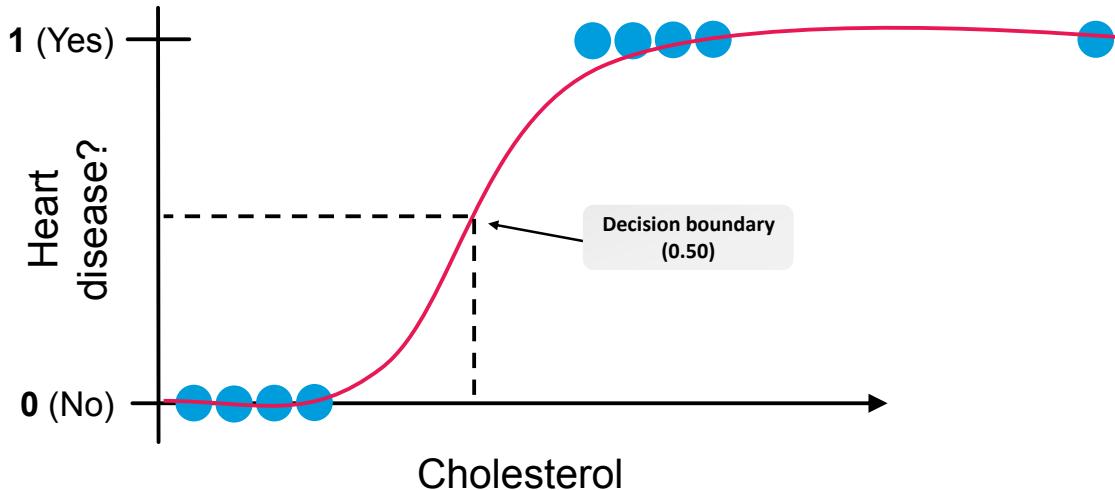


Figure 6–4: A sigmoid function applied to a classification problem. Note that the decision boundary in this case was selected arbitrarily.

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:

$\text{if } y = 1, c(\theta) = -\log(\hat{p})$

$\text{if } y = 0, c(\theta) = -\log(1 - \hat{p})$

Where:

- y is the observed classification.
- $c(\theta)$ is the cost function.

- \hat{p} is the estimated probability.

As the model's estimation (\hat{p}) approaches 0, $-\log(\hat{p})$ increases significantly if the observed classification (y) is in the positive class (1). If y is in the negative class (0), $-\log(1-\hat{p})$ will increase significantly as \hat{p} approaches 1. This is as expected because the farther an estimation is away from the actual observed value, the worse it is at minimizing cost, so the cost function will be very high. When the estimated 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.

Guidelines for Training Binary Classification Models Using Logistic Regression

Follow these guidelines when you are training binary classification models using logistic regression.



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 Using Logistic Regression

When training a binary classification model using logistic regression:

- Consider using logistic regression rather than linear regression for classification tasks.
- Consider using logistic regression when classifications must be quick, and when working with larger datasets.
- Recognize that regression cost functions like MSE and MAE are not suitable for logistic regression.
- Recognize that the normal equation cannot effectively minimize cost with logistic regression, but gradient descent can.
- Time permitting, run a dataset through multiple classification algorithms to determine which performs the best.

Use Python for Binary Classification with Logistic Regression

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)` —Use the model to return estimated classifications on the validation/test set.
- `model.predict_proba(X_test)` —Use the model to provide the raw probability estimates for each classification decision.
- `model.coef_` —Return an attribute that lists the optimal model parameters generated during training.

ACTIVITY 6–1

Training a Binary Classification Model Using Logistic Regression

Data Files

/home/student/CAIP/Logistic Regression and k-NN/Logistic Regression and k-NN - Titanic.ipynb

/home/student/CAIP/Logistic Regression and k-NN/titanic_data/titanic_train_data.csv

/home/student/CAIP/Logistic Regression and k-NN/titanic_data/titanic_new_data.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 classification model 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.



Note: This dataset was retrieved from Kaggle at: <https://www.kaggle.com/c/titanic>.

- From Jupyter Notebook, select **CAIP/Logistic Regression and k-NN/Logistic Regression and k-NN - Titanic.ipynb** to open it.

- Import the relevant libraries and load the dataset.

- View the cell titled **Import software libraries and load the dataset**, and examine the code cell below it.
- Run the code cell.
- Verify that **titanic_train_data.csv** was loaded with 891 records.

There are two datasets you'll work with for this project: the training dataset that you'll split into train and test subsets, and a separate dataset that includes new, unlabeled data that the model hasn't seen. But first you'll get familiar with the data.

- Get acquainted with the dataset.

- Scroll down and view the cell titled **Get acquainted with the dataset**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   PassengerId 891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64 
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare          891 non-null    float64 
 10  Cabin         204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3		female	26.0	0	0	STON/O2.3101282	7.9250	NaN	S
3	4	1	1	Euttaloo, Mrs. Jacques Heath (Ilu Mau Poo)	female	35.0	1	0	113803	53.1000	C123	S

- 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.
- `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.
- 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.
- 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.
- 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.

4. Examine descriptive statistics.

- Scroll down and view the cell titled **Examine descriptive statistics**, and examine the code cell below it.

- b) Run the code cell.
- c) 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?

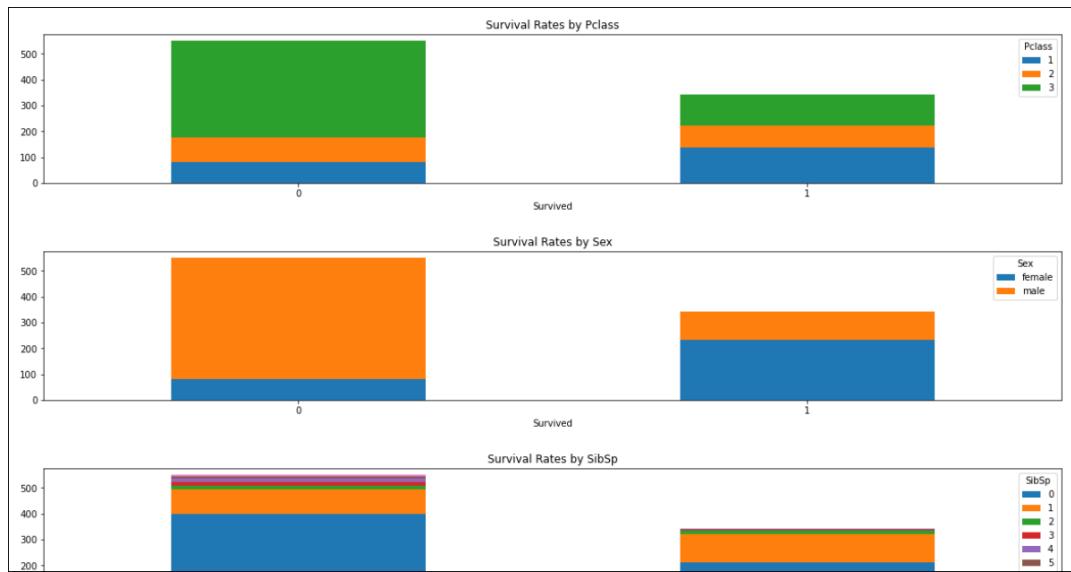
6. Unlike with linear regression, attempting to find a correlation between a feature and a classification label is not useful. In other words, there's no need to run code to compare the *Titanic* features to the `Survived` label.

Why is such a correlation not relevant in classification problems like this one?

7. Use a stacked bar chart to show survival numbers.

- a) Scroll down and view the cell titled **Use a stacked bar chart to show survival numbers**, and examine the code cell below it.
- b) Run the code cell.

c) Examine the output.

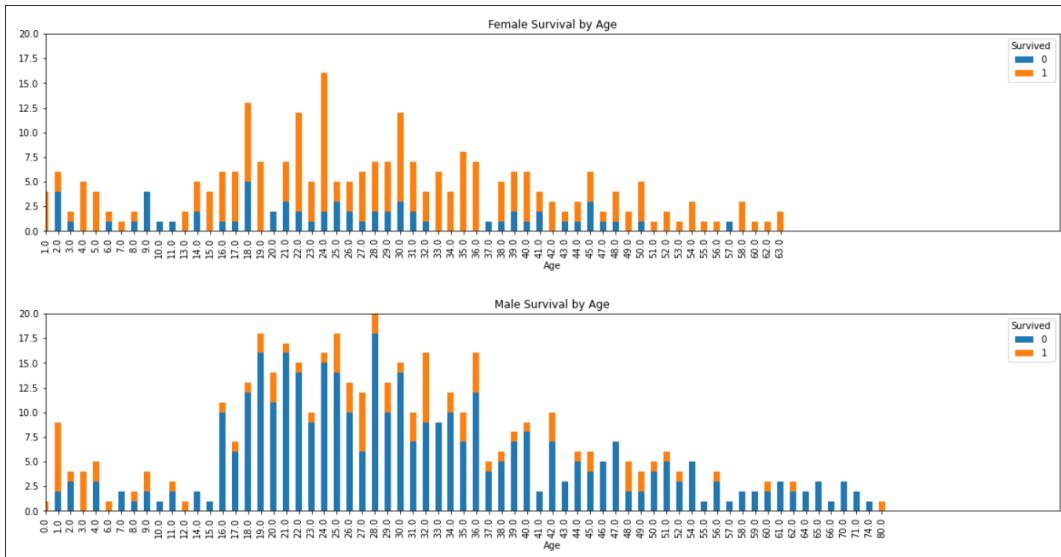


- In general, passengers were more likely to perish than survive.
- The majority of people who perished were in third class. However, keep in mind that these graphs are measuring absolute values and not proportions—there may have been more people in third class than in any other class.
- The majority 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 view the cell titled **Look for relationships between survival, age, and sex**, and examine the code cell below it.
- Run the code cell.

- c) 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. Determine how to handle ticket values.

- Scroll down and view the cell titled **Determine how to handle ticket values**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
array(['110152', '110413', '110465', '110564', '110813', '111240',
       '111320', '111361', '111369', '111426', '111427', '111428',
       '112050', '112052', '112053', '112058', '112059', '112277',
       '112379', '113028', '113043', '113050', '113051', '113055',
       '113056', '113059', '113501', '113503', '113505', '113509',
       '113510', '113514', '113572', '113760', '113767', '113773',
       '113776', '113781', '113783', '113784', '113786', '113787',
       '113788', '113789', '113792', '113794', '113796', '113798',
       '113800', '113803', '113804', '113806', '113807', '11668', '11751',
       '11752', '11753', '11755', '11765', '11767', '11769', '11771',
       '11774', '11813', '11967', '12233', '12460', '12749', '13049',
       '13213', '13214', '13502', '13507', '13509', '13567', '13568',
       '14311', '14312', '14313', '14973', '1601', '16966', '16988',
       '17421', '17453', '17463', '17464', '17465', '17466', '17474',
       '17764', '19877', '19928', '19943', '19947', '19950', '19952',
       '19973', '19980', '19986', '19993', '19997', '19999', '19999'])
```

It seems as though it would be difficult to convert ticket codes into something useful, so they will be dropped from the dataset.

10. Identify all personal titles and embarked port codes.

- Scroll down and view the cell titled **Identify all personal titles and embarked port codes**, and examine the code cell below it.
- Run the code cell.

- c) 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 one-hot encode both of these features. However, because there are so many different titles, and several of them are bound to be rare or even unique, you'll consolidate them. This will hopefully simplify the model, as too many one-hot encoded columns can create a sparse dataset that the model has trouble learning from.

11. Prepare the dataset.

- a) Scroll down and view the cell titled **Prepare the dataset**, and examine the code cell below it.

Common cleaning and feature engineering tasks will be performed on the data.

- On lines 9 through 12, missing values for `Age` and `Fare` are being filled in with the median of those values.
- On line 15, for `Embarked`, you're using the mode to fill in missing values.
- On lines 21 and 22, 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.
- On lines 25 through 28, if a passenger has a personal title, it is being removed from `Name` and placed into its own `Title` column.
- On line 29, missing `Title` values are being filled in with the mode.
- On line 34, only the top 4 most common titles are being kept. Any passenger with a title outside the top 4 is assigned a 5th title called "Spec" (for "special").
- On lines 39 through 42, `Sex`, `Title`, and `Embarked` are all being one-hot encoded.

- b) Run the code cell.

- c) Examine the output.

After prep:

PassengerId	0
Survived	0
Pclass	0
Name	0
Sex_male	0
Sex_female	0
Age	0
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked_S	0
Embarked_C	0
Embarked_Q	0
SizeOfFamily	0
Title_Mr	0
Title_Mrs	0
Title_Miss	0
Title_Master	0
Title_Spec	0

- Except for `Cabin`, each feature no longer has missing values.
- The `Sex` column has been replaced by the `Sex_male` and `Sex_female` encoded columns.
- The `Embarked` column has been replaced by the `Embarked_S`, `Embarked_C`, and `Embarked_Q` encoded columns.
- The `Title` column has been replaced by the `Title_Mr`, `Title_Mrs`, `Title_Miss`, `Title_Master`, and `Title_Spec` encoded columns.

12. Preview the current data.

- a) Scroll down and view the cell titled **Preview the current data**, and examine the code cell below it.
- b) Run the code cell.

- c) Examine the output.

PassengerId	Survived	Pclass	Name	Sex_male	Sex_female	Age	SibSp	Parch	Ticket	...	Cabin	Embarked_S	Embarked_C	Embarked_Q	SizeOf
0	1	0	3 Mr. Owen Harris	1	0	22.0	1	0	A/5 21171	...	NaN	1	0	0	0
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th... ...	0	1	38.0	1	0	PC 17599	...	C85	0	1	0	0
2	3	1	Heikkinen, Miss. Laina	0	1	26.0	0	0	STON/O2, 3101282	...	NaN	1	0	0	0
3	4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	1	35.0	1	0	113803	...	C123	1	0	0	0
4	5	0	Allen, Mr. William Henry	1	0	35.0	0	0	373450	...	NaN	1	0	0	0

5 rows × 21 columns

- As mentioned previously, `PassengerId` and `Name` don't contain information that will be useful for training the model.
- `Cabin` contains data that might be useful, but numerous records are missing this data, so you'll drop it.
- `Ticket` likewise *might* contain useful data, but you don't have enough information to convert it to a number.

13. 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 cell below it.
Unused columns will be dropped from the dataset.
- Run the code cell.
- Examine the output.

```
Columns before drop:
['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex_male', 'Sex_female', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked_S', 'Embarked_C', 'Embarked_Q', 'SizeOfFamily', 'Title_Mr', 'Title_Mrs', 'Title_Miss', 'Title_Master', 'Title_Spec']

Columns after drop:
['Survived', 'Pclass', 'Sex_male', 'Sex_female', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked_S', 'Embarked_C', 'Embarked_Q', 'SizeOfFamily', 'Title_Mr', 'Title_Mrs', 'Title_Miss', 'Title_Master', 'Title_Spec']
```

The aforementioned columns have been removed from the dataset.

- Scroll down and examine the next code cell.

```
1 | df.head()
```

- Run the code cell.

- f) Examine the output.

	Survived	Pclass	Sex_male	Sex_female	Age	SibSp	Parch	Fare	Embarked_S	Embarked_C	Embarked_Q	SizeOfFamil
0	0	3	1	0	22.0	1	0	7.2500	1	0	0	0
1	1	1	0	1	38.0	1	0	71.2833	0	1	0	0
2	1	3	0	1	26.0	0	0	7.9250	1	0	0	0
3	1	1	0	1	35.0	1	0	53.1000	1	0	0	0
4	0	3	1	0	35.0	0	0	8.0500	1	0	0	0

The data should now be ready to train a model.

14. Split the dataset.

- Scroll down and view the cell titled **Split the dataset**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
Original set: (891, 17)
-----
Training features: (668, 16)
Testing features: (223, 16)
Training labels: (668, 1)
Testing labels: (223, 1)
```

The original training dataset has now been split into two: one set to use as training, the other to use in testing. Note that the `Survived` label has been removed from the `X` matrices and placed into its own `y` vector.

15. Create a logistic regression model.

- Scroll down and view the cell titled **Create a logistic regression model**, and examine the code cell below it.
 - On line 3, `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.
- Run the code cell.
- Examine the output.

```
Logistic regression model took 326.88 milliseconds to fit.
Score on test set: 83%
```

The score for this model is around 83%, which is not bad for a first attempt.

- d) Scroll down and examine the next code cell.

```

1 # Use test set to evaluate.
2 results = X_test.copy()
3 results['PredictedSurvival'] = log_reg.predict(X_test)
4 results['ActualSurvival'] = y_test.copy()
5 results['ProbPerished'] = np.round(log_reg.predict_proba(X_test)[:, 0] * 100, 2)
6 results['ProbSurvived'] = np.round(log_reg.predict_proba(X_test)[:, 1] * 100, 2)
7
8 # View examples of the predictions compared to actual survival.
9 results.head(10)

```

- e) Run the code cell.
f) Examine the output.

OffFamily	Title_Mr	Title_Mrs	Title_Miss	Title_Master	Title_Spec	PredictedSurvival	ActualSurvival	ProbPerished	ProbSurvived
1	1	0	0	0	0	0	1	84.59	15.41
4	1	0	0	0	0	0	0	85.74	14.26
11	1	0	0	0	0	0	0	97.19	2.81
1	0	1	0	0	0	1	1	32.66	67.34
1	1	0	0	0	0	0	0	79.92	20.08
3	0	1	0	0	0	1	1	37.22	62.78
1	0	0	1	0	0	1	1	37.59	62.41
1	1	0	0	0	0	0	0	65.78	34.22
1	1	0	0	0	0	0	0	83.78	16.22
1	1	0	0	0	0	0	0	75.15	24.85



Note: You may need to scroll the data frame to the right to see the prediction results.

In addition to the existing columns, the classification that was predicted by the model on the test 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).

16. Use the logistic regression model to make predictions on the new, unlabeled data.

- a) Scroll down and view the cell titled **Use the logistic regression model to make predictions on the new, unlabeled data**, and examine the code cell below it.
b) Run the code cell.
c) Examine the output.

```
Loaded 418 records from ./titanic_data/titanic_new_data.csv
```

418 records are loaded from **titanic_new_data.csv**. Recall that this dataset is separate from the training and test sets. This 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*.

- d) Scroll down and examine the next code cell.

```

1 # Prepare the dataset and drop unneeded columns.
2 print('Preparing new data for prediction.\n')
3 df_new = prep_dataset(df_new_raw.copy())
4 df_new = drop_unused(df_new.copy())

```

- e) Run the code cell.
f) Examine the output.

Preparing new data for prediction.

Before prep:

```

PassengerId      0
Pclass            0
Name              0
Sex               0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64

```

After prep:

```

PassengerId      0
Pclass            0
Name              0
Sex male          0

```

As with the training and test sets, you're performing the same data preparation steps to get the data in an optimal state.

- g) Scroll down and examine the next code cell.

```

1 # Show example predictions with the new data.
2 results_new = df_new.copy()
3 results_new['PredictedSurvival'] = log_reg.predict(df_new)
4 results_new['ProbPerished'] = np.round(log_reg.predict_proba(df_new)[:, 0] * 100, 2)
5 results_new['ProbSurvived'] = np.round(log_reg.predict_proba(df_new)[:, 1] * 100, 2)
6 results_new.head(10)

```

- h) Run the code cell.
i) 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	84.27	15.73
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	1	41.90	58.10
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	0	82.60	17.40
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	0	79.68	20.32
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	1	40.69	59.31
5	897	3	Svensson, Mr. Johan Cervin	male	14.0	0	0	7538	9.2250	NaN	S	0	78.37	21.63
6	898	3	Connolly, Miss. Kate	female	30.0	0	0	330972	7.6292	NaN	Q	1	39.85	60.15
7	899	2	Caldwell, Mr. Albert Francis	male	26.0	1	1	248738	29.0000	NaN	S	0	78.78	21.22
8	900	3	Abrahim, Mrs. Joseph (Sophie Halaut Easu)	female	18.0	0	0	2657	7.2292	NaN	C	1	37.10	62.90
9	901	3	Davies, Mr. John Samuel	male	21.0	2	0	A/4 48871	24.1500	NaN	S	0	84.97	15.03

The logistic regression model has predicted values for this new dataset, fulfilling its ultimate purpose.

17.Keep this notebook open.

TOPIC B

Train Binary Classification Models Using k-Nearest Neighbor

Different algorithms may be ideal for solving certain types of classification problems, so you need to be aware of how they differ. In this topic, you'll explore an alternative to logistic regression.

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, respectively).

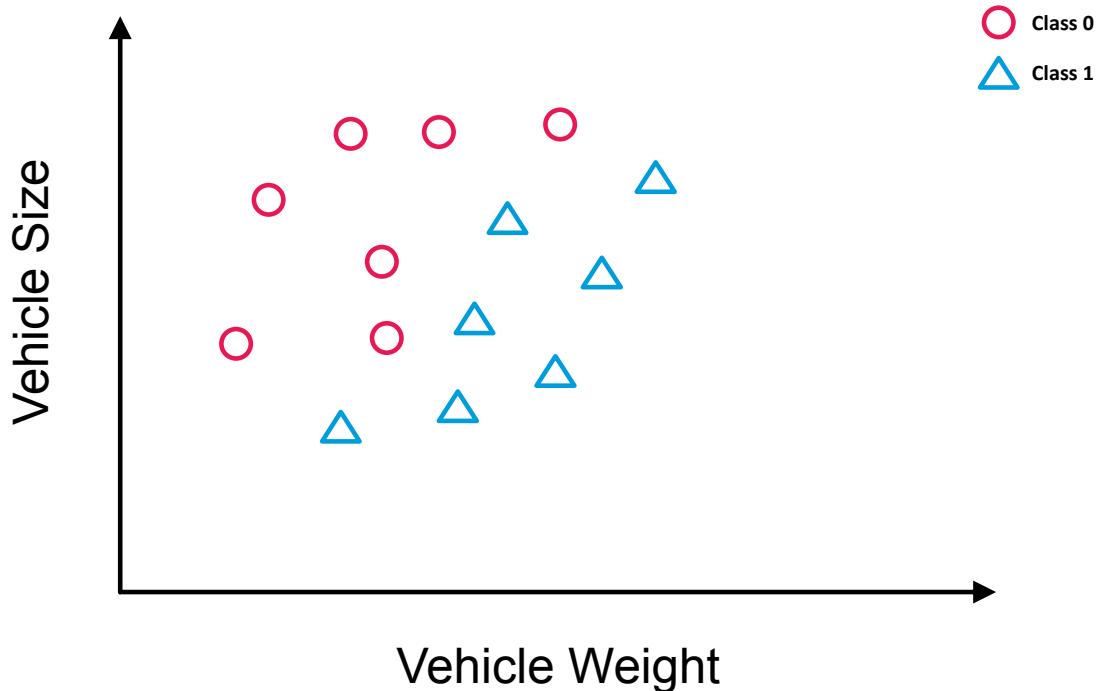


Figure 6-5: Data examples plotted for vehicle weight and vehicle size.

You could use a logistic regression model to determine these classifications. However, consider how you might determine the class for a new test example, depicted in the following figure as a green X.

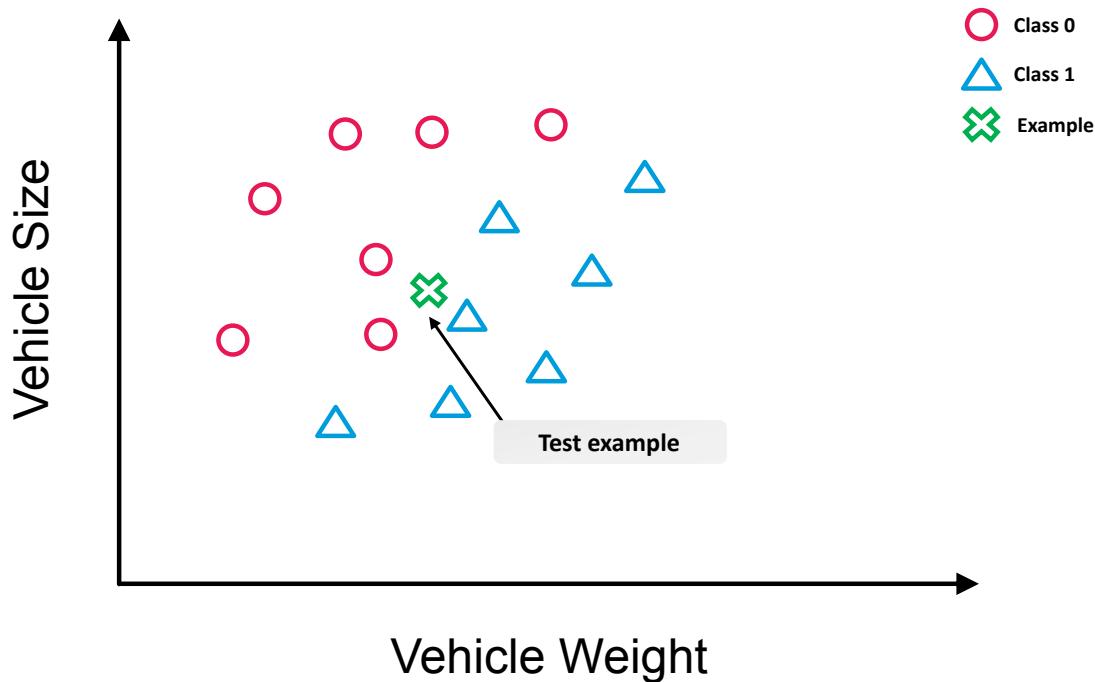


Figure 6–6: A new data example needs a classification.

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)

The **k -nearest neighbor (k -NN)** algorithm is an alternative classification approach in which 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 to the data example (the green X). Since two of the three neighbors are in class 0 (red circles), the data example is classified as a 0.

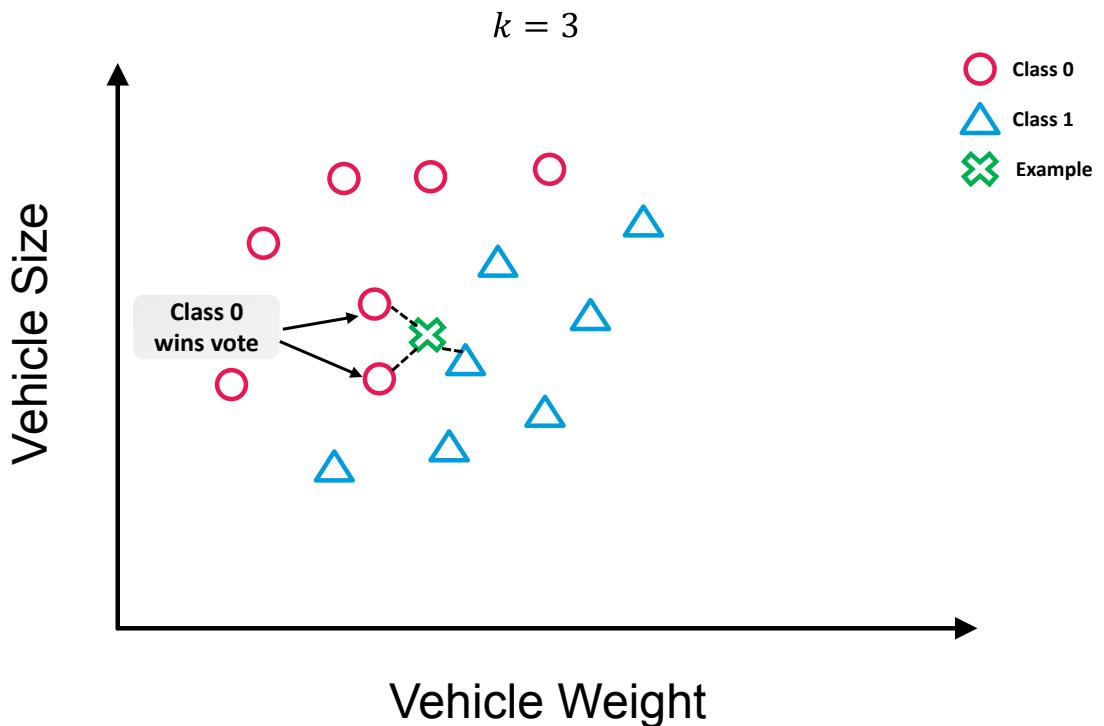


Figure 6-7: k-NN classification where $k = 3$.



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.

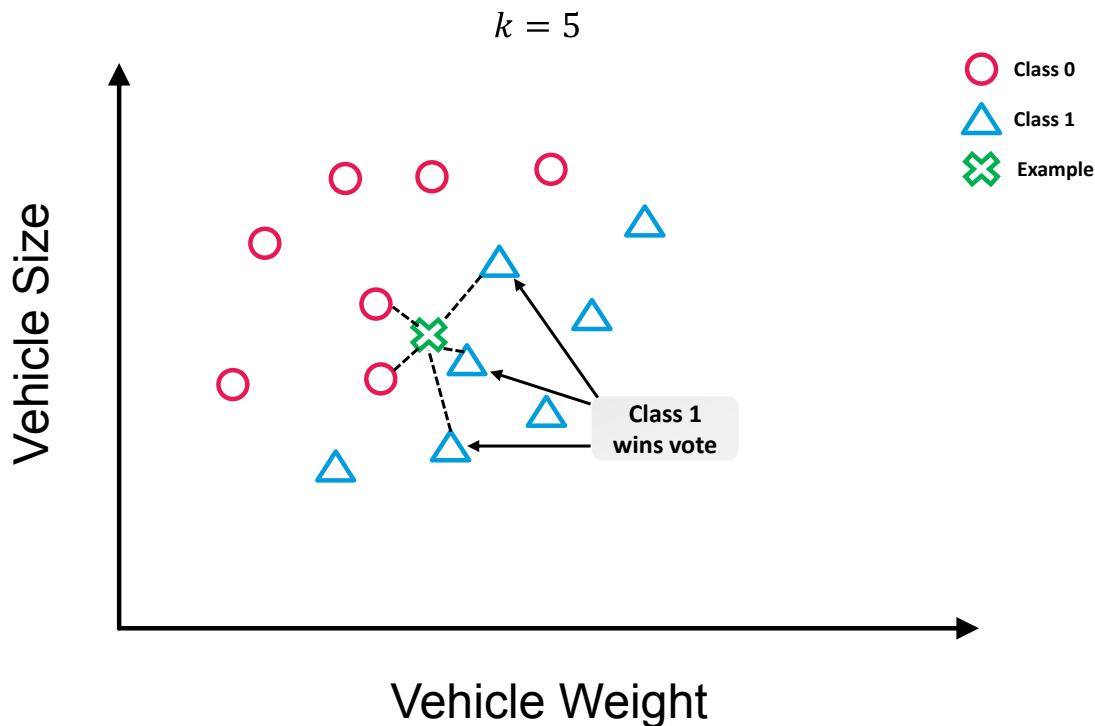


Figure 6-8: Changing k in a k -NN model can lead to different results.

One rule of thumb when selecting k is to always make k odd when you have a binary classification problem; this eliminates the chance of there being a tie in the vote. Beyond that, you can tune k based on evidence of underfitting or overfitting. Lower values tend to lead to high variance (overfitting), whereas higher values tend to lead to high bias (underfitting). A dataset with many outliers or high levels of noise might benefit from higher k values than a dataset without these issues.

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 classify 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 Using k -NN

Follow these guidelines when you are training binary classification models using k -nearest neighbor.

Train a Binary Classification Model Using k -NN

When training a binary classification model using 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 when simplicity of implementation is key and when working with smaller datasets.
- Time permitting, run a dataset through multiple classification algorithms to determine which performs the best.

Use Python for Binary Classification with k -NN

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 6–2

Training a Binary Classification Model Using k -NN

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/Logistic Regression and k-NN/Logistic Regression and k-NN - Titanic.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 **Create a k -nearest neighbor model**. Select **Cell→Run All Above**.

Scenario

Your preliminary logistic regression model did fairly well on the *Titanic* data, but there's no reason why you shouldn't experiment with other algorithms to see if you get better results. One of the simpler alternatives to logistic regression is k -nearest neighbor. You'll train a model using this algorithm to see how it differs from the logistic regression model.

Since you already prepared the data earlier, most of the work is already done. You just need to train and test the new model.

1. Create a k -nearest neighbor model.

- Scroll down and view the cell titled **Create a k -nearest neighbor model**, and examine the code cell below it.
 - This is an alternative to the logistic regression model built in the prior code blocks. Both of these models perform binary classification, but it's useful to build multiple models to see if one performs better than another.
 - On lines 4 through 8, 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.
- Run the code cell.
- Examine the output.

```
Value of k: 27
KNN model took 1.46 milliseconds to fit.
Score on validation set: 72%
```

- The bootstrapping method produced a k value of 27.
- The score for the k -nearest neighbor model is around 72%. This performs worse than the logistic regression model, so you'll stick with the logistic regression model for evaluation and tuning in later activities.
- Still, it might be interesting to see how the results between the two models differ.

- d) Scroll down and examine the next code cell.

```

1 # Use test set to evaluate.
2 results = X_test.copy()
3 results['PredictedSurvival'] = knn.predict(X_test)
4 results['ActualSurvival'] = y_test.copy()
5 results['ProbPerished'] = np.round(knn.predict_proba(X_test)[:, 0] * 100, 2)
6 results['ProbSurvived'] = np.round(knn.predict_proba(X_test)[:, 1] * 100, 2)
7
8 # View examples of the predictions compared to actual survival.
9 results.head(10)

```

- e) Run the code cell.
f) Examine the output.

OffFamily	Title_Mr	Title_Mrs	Title_Miss	Title_Master	Title_Spec	PredictedSurvival	ActualSurvival	ProbPerished	ProbSurvived
1	1	0	0	0	0	0	1	92.59	7.41
4	1	0	0	0	0	0	0	51.85	48.15
11	1	0	0	0	0	0	0	55.56	44.44
1	0	1	0	0	0	0	1	66.67	33.33
1	1	0	0	0	0	0	0	92.59	7.41
3	0	1	0	0	0	0	1	59.26	40.74
1	0	0	1	0	0	0	1	55.56	44.44
1	1	0	0	0	0	1	0	40.74	59.26
1	1	0	0	0	0	0	0	92.59	7.41
1	1	0	0	0	0	0	0	66.67	33.33

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).

2. Use the *k*-NN model to make predictions on the new, unlabeled data.

- a) Scroll down and view the cell titled **Use the *k*-NN model to make predictions on the new, unlabeled data**, and examine the code cell below it.
b) Run the code cell.

c) 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	85.19	14.81
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	0	88.89	11.11
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	0	85.19	14.81
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	0	88.89	11.11
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	0	77.78	22.22
5	897	3	Svensson, Mr. Johan Cervin	male	14.0	0	0	7538	9.2250	NaN	S	0	55.56	44.44
6	898	3	Connolly, Miss. Kate	female	30.0	0	0	330972	7.6292	NaN	Q	0	55.56	44.44
7	899	2	Caldwell, Mr. Albert Francis	male	26.0	1	1	248738	29.0000	NaN	S	1	44.44	55.56
8	900	3	Abraham, Mrs. Joseph (Sophie Halaut Easau)	female	18.0	0	0	2657	7.2292	NaN	C	0	70.37	29.63
9	901	3	Davies, Mr. John Samuel	male	21.0	2	0	A/4 48871	24.1500	NaN	S	0	51.85	48.15

Just like the logistic regression model, the k -NN model puts forth its own predictions on the unlabeled dataset.

3. Keep this notebook open.

TOPIC C

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.
- $\boldsymbol{\theta}$ is the vector of the model parameters.
- \mathbf{x} is the vector of the feature values.



Note: This is very similar to the equation used to estimate values in linear regression.

The scores are computed for every class k for vector \mathbf{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:

$$\sigma(\mathbf{x})_k = \frac{\exp(\mathbf{x})_k}{\sum_{j=1}^K \exp(\mathbf{x})_j}$$

Where:

- $\sigma(\mathbf{x})_k$ is the probability that example \mathbf{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 might generate 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. And, as you can see, all of the percentages add up to 1 (100%).

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 estimation.

Multi-Class k -NN Example

You can also use k -NN to solve a multi-class classification problem. The approach is essentially the same as with a binary problem, only with more "neighbor" categories for the algorithm to consider. In the following example, a third class of vehicle, motorcycles, has been added.

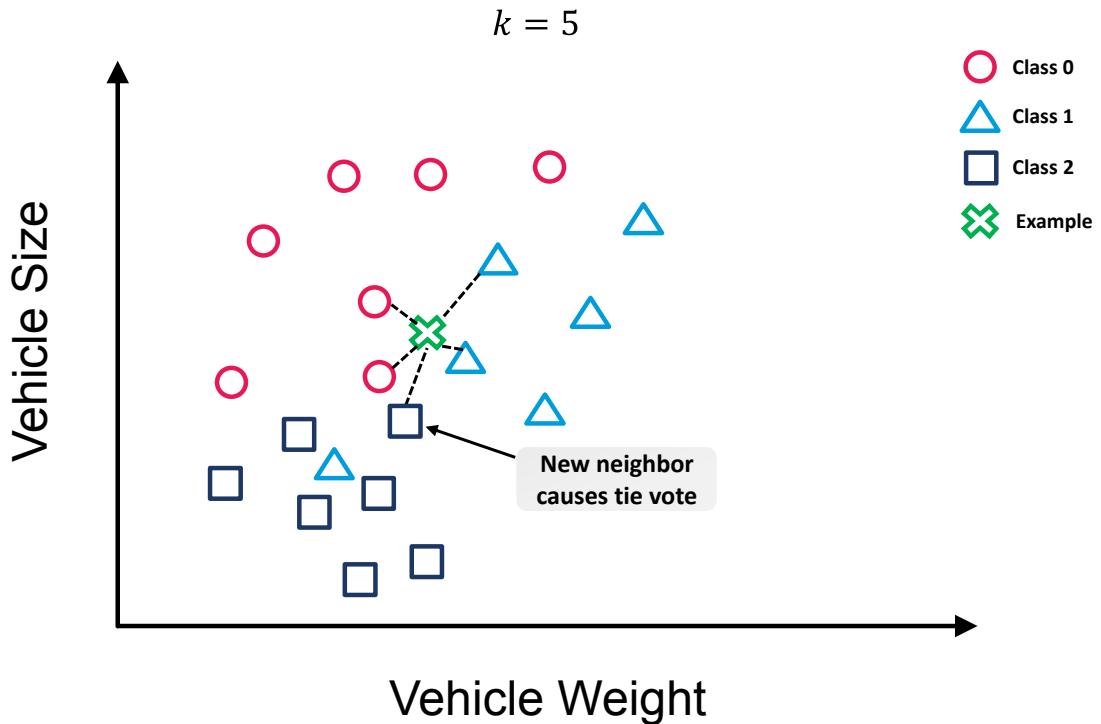


Figure 6-9: The addition of a new neighbor has changed the class vote.

Because one example of the motorcycle class is close enough to be a neighbor when k is 5, the voting result has changed. In fact, it has led to a tie between class 0 and class 1, both of which now have two votes. Unlike with a binary problem, making k odd in a multi-class problem is not guaranteed to prevent tie votes.

There are a few approaches you can take to avoid a tie in such cases:

- **Simply change k by 1 until there is no longer a tie.** Because this is a kind of "hack," and because you may need to increment or decrement k multiple times until there is no longer a tie, you should consider other options before doing this.
- **Randomly select from among the tied classes.** This may seem like a somewhat better option than changing k arbitrarily, but it's still not an optimal approach since there's not much "intelligence" behind the decision.
- **Weight each class by distance.** In other words, the closer a neighbor is to the data example, the more weight it is given in the decision. The class that is, on average, closer to the example, is the one that breaks the tie. In the vehicle scenario, the class 0 (car) labels are, on average, slightly closer than the class 1 (plane) labels. So, the example is classified as a car. This is a smarter approach to breaking ties since k -NN is a distance-based algorithm, and you're incorporating distance even more into the calculation.

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.
- Or, consider training a multi-class k -NN model for simplicity.
- In the case of tie votes for multi-class k -NN, consider weighting the distances for each neighbor to break ties.

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/Logistic Regression and k-NN/Multinomial Logistic Regression - 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. Open a new notebook.

- a) From the Jupyter Notebook toolbar, select **File→Open**.
- b) In the new tab that opens, select **CAIP/Logistic Regression and k-NN/Multinomial Logistic Regression - Wine.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 cell below it.
- b) Run the code cell.
- c) 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 data file stored on the local machine.

3. Get acquainted with the dataset.

- a) Scroll down and view the cell titled **Get acquainted with the dataset**, and examine the code cell below it.
 - Lines 2 and 3 convert the dataset into a pandas data frame so that it's in a familiar format.
 - Lines 6 and 7 shuffle the dataset. By default, the dataset is ordered by class. Mixing the rows will ensure the output of `head()` is more varied.
- b) Run the code cell.

- c) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   alcohol          178 non-null    float64 
 1   malic_acid       178 non-null    float64 
 2   ash              178 non-null    float64 
 3   alcalinity_of_ash 178 non-null    float64 
 4   magnesium        178 non-null    float64 
 5   total_phenols    178 non-null    float64 
 6   flavanoids        178 non-null    float64 
 7   nonflavanoid_phenols 178 non-null    float64 
 8   proanthocyanins  178 non-null    float64 
 9   color_intensity   178 non-null    float64 
 10  hue              178 non-null    float64 
 11  od280/od315_of_diluted_wines 178 non-null    float64 
 12  proline          178 non-null    float64 
 13  target            178 non-null    int64  
dtypes: float64(13), int64(1)
memory usage: 19.6 KB
None

alcohol  malic_acid  ash  alcalinity_of_ash  magnesium  total_phenols  flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity  hue  od280/od315
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      12.41      2.84  2.12          18.8      90.0       2.45      2.62          0.27          1.48          4.28      0.01
```

- 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 descriptive statistics.

- Scroll down and view the cell titled **Examine descriptive statistics**, and examine the code cell below it.
- Run the code cell.

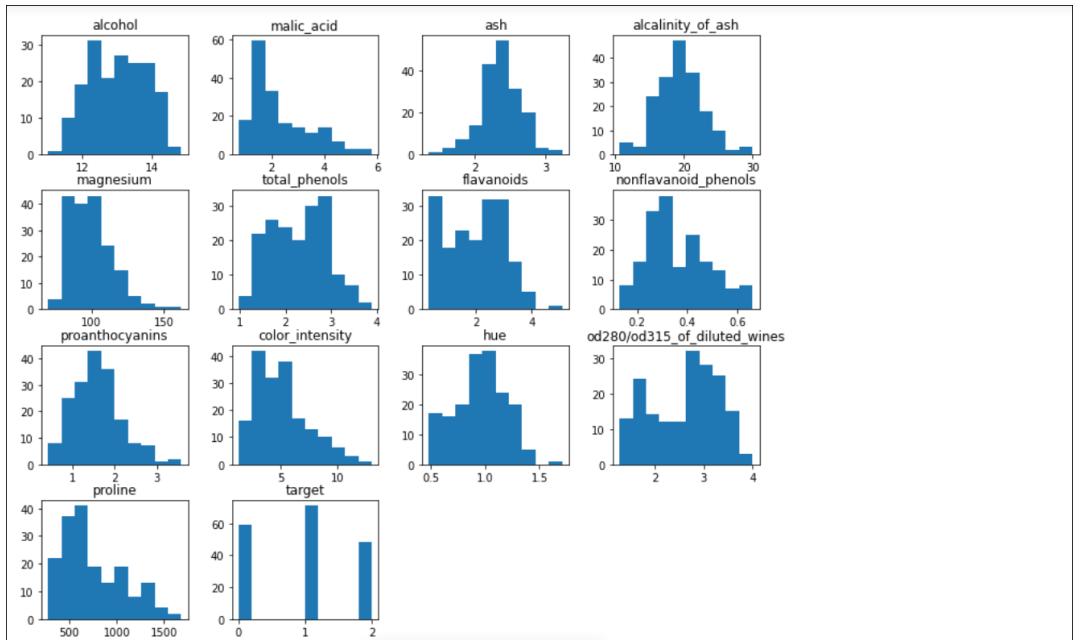
- c) Examine the output.

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	target
count	178.00	178.00	178.00	178.00	178.00	178.00	178.00	178.00	178.00	178.00
mean	13.00	2.34	2.37	19.49	99.74	2.30	2.03	0.36	0.36	0.36
std	0.81	1.12	0.27	3.34	14.28	0.63	1.00	0.12	0.12	0.12
min	11.03	0.74	1.36	10.60	70.00	0.98	0.34	0.13	0.13	0.13
25%	12.36	1.60	2.21	17.20	88.00	1.74	1.21	0.27	0.27	0.27
50%	13.05	1.87	2.36	19.50	98.00	2.35	2.13	0.34	0.34	0.34
75%	13.68	3.08	2.56	21.50	107.00	2.80	2.88	0.44	0.44	0.44
max	14.83	5.80	3.23	30.00	162.00	3.88	5.08	0.66	0.66	0.66

- 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 distributions of the features.

- Scroll down and view the cell titled **Examine the distributions of the features**, and examine the code cell below it.
- Run the code cell.
- Examine the output.



- For the most part, these histograms seem to confirm the idea that there are not many extreme outliers in the dataset.
- A few of the features, such as `malic_acid` and `color_intensity`, exhibit a degree of right skew.
- You can also see that the class labels (`target`) are distributed pretty evenly.

6. Split the label from the dataset.

- Scroll down and view the cell titled **Split the label from the dataset**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
Original set:      (178, 14)
-----
Training features: (178, 13)
Training labels:   (178, 1)
```

Rather than use the holdout method to split the datasets into a training set and a test set, you'll train 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.

- Scroll down and view the cell titled **Transform magnesium and proline**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
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

   h  alcalinity_of_ash  magnesium  total_phenols  flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity  hue  od280/od315_of_diluted_wines  proline
4       20.0        4.672829         1.83       0.56             0.50            0.80           5.88      0.96                  1.82       6.522093
4       18.0        4.595120         2.20       1.94             0.30            1.46           2.62      1.23                  3.16       6.109248
7       16.0        4.584967         2.98       3.15             0.22            1.85           7.22      1.01                  3.55       6.951772
0       22.5        4.615121         3.00       3.25             0.29            2.38           5.70      1.19                  2.71       7.158514
2       18.8        4.499810         2.45       2.68             0.27            1.48           4.28      0.91                  3.00       6.942157
```

- 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 logistic 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.

- Scroll down and view the cell titled **Create a multinomial logistic regression model**, and examine the code cell below it.
 - 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).
- Run the code cell.

- c) Examine the output.

```
Multinomial logistic regression model created.
```

9. Train the model using stratified k -fold cross-validation.

- a) Scroll down and view the cell titled **Train the model using stratified k -fold cross-validation**, and examine the code cell below it.
 This code trains the model using five-fold cross-validation.
- b) Run the code cell.
- c) Examine the output.

```
Multinomial logistic regression model took 324.67 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.
 - The mean score on the tests sets is high at 95%.
- d) Scroll down and examine the next code cell.

```
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 = X.copy()  

6 results['magnesium'] = np.exp(results['magnesium'])  

7 results['proline'] = np.exp(results['proline'])  

8 results['PredictedWine'] = preds  

9 results['ActualWine'] = y.copy()  

10 results['ProbWine0'] = np.round(proba[:, 0] * 100, 2)  

11 results['ProbWine1'] = np.round(proba[:, 1] * 100, 2)  

12 results['ProbWine2'] = np.round(proba[:, 2] * 100, 2)  

13  

14 # View examples of the predictions compared to actual wine.  

15 results.head(10)
```

- e) Run the code cell.

- f) Examine the output.

ins	color_intensity	hue	od280/od315_of_diluted_wines	proline	PredictedWine	ActualWine	ProbWine0	ProbWine1	ProbWine2	
.80	5.88	0.96		1.82	680.0	2	2	1.59	1.94	96.47
.46	2.62	1.23		3.16	450.0	1	1	7.16	92.75	0.10
.85	7.22	1.01		3.55	1045.0	0	0	99.39	0.59	0.03
.38	5.70	1.19		2.71	1285.0	1	0	46.75	52.65	0.61
.48	4.28	0.91		3.00	1035.0	0	0	72.71	26.61	0.68
.73	5.50	0.66		1.83	510.0	2	2	1.02	2.04	96.94
.29	5.60	1.24		3.37	1265.0	0	0	97.71	2.26	0.03
.62	3.05	0.96		2.06	495.0	1	1	1.22	95.47	3.31
.81	5.40	1.25		2.73	1150.0	0	0	99.85	0.15	0.00
.77	3.94	0.69		2.84	352.0	1	1	5.51	94.36	0.12

- 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 at the end of the data frame.
- 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.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **Multinomial Logistic Regression - Wine** tab in Firefox, and return to the **Logistic Regression and k-NN - Titanic** tab.

TOPIC D

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 your classification models to see how they're performing.

Classification 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.

<i>True (Correct) Label</i>	<i>Estimated Label</i>	<i>Model Result</i>	<i>Assessment</i>
Yes	Yes	Success	True positive (TP)
No	No	Success	True negative (TN)
No	Yes	Failure	False positive (FP)
Yes	No	Failure	False negative (FN)

Considerations When Choosing Classification Metrics

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 estimation 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 estimations to be false. 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.

Confusion Matrices

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 estimated data. In a confusion matrix, the number of estimations 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).

		Estimation	
		No	Yes
Actual	No	True negatives	False positives
	Yes	False negatives	True positives

Figure 6–10: 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.

		Estimation	
		No heart disease	Heart disease
Actual	No heart disease	212	6
	Heart disease	3	43

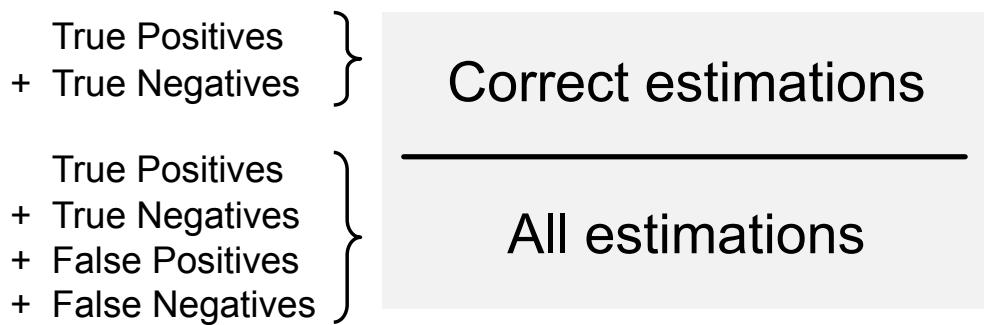
Figure 6–11: A confusion matrix of heart disease predictions.

Using a table like this, it's much easier to identify the prevalence of errors in the model's predictions. To summarize the example:

- 212 patients were correctly predicted to not have heart disease (true negatives).
- 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).
- 43 patients were correctly predicted to have heart disease (true positives).

Accuracy

Accuracy is a measure of how frequently each prediction is correctly deemed positive or negative.

**Figure 6–12: How to calculate accuracy.**

As shown here, accuracy is the number of correct estimations divided by the number of all estimations made. In other words, accuracy can be expressed as:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

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.

		Estimation	
		No collapse	Collapse
Actual	No collapse	512	1
	Collapse	2	1

Figure 6–13: 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.

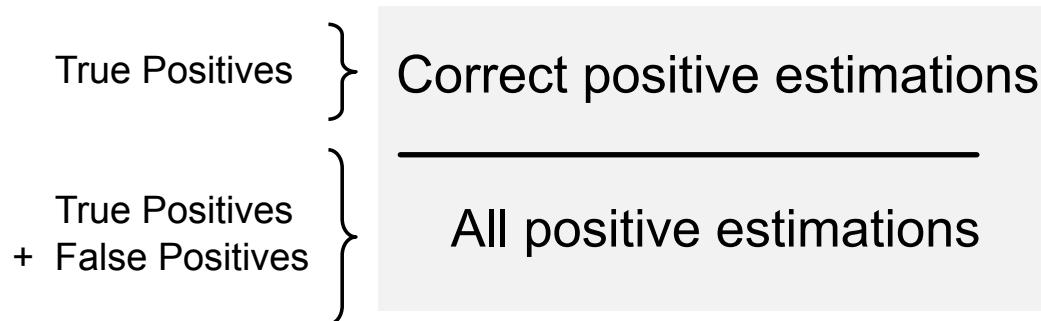


Figure 6-14: How to calculate precision.

As shown here, precision is the number of correct positive estimations divided by the number of all positive estimations 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 bridge collapse, 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 estimation is wrong.

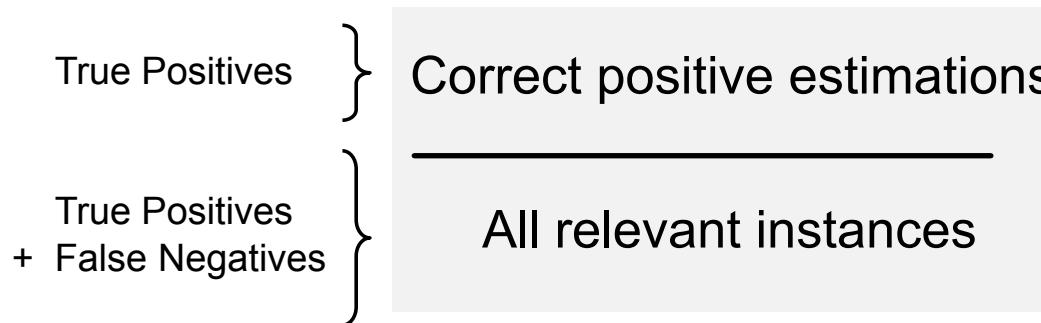


Figure 6-15: How to calculate recall.

As shown here, recall is the number of correct positive estimations divided by the number of correct positive estimations plus the number of incorrect negative estimations. In other words, recall can be expressed as:

$$\frac{\text{TP}}{\text{TP} + \text{FN}}$$

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. This is because recall focuses on false negatives. 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. Also, recall doesn't minimize false positives as well as precision does.

The 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.

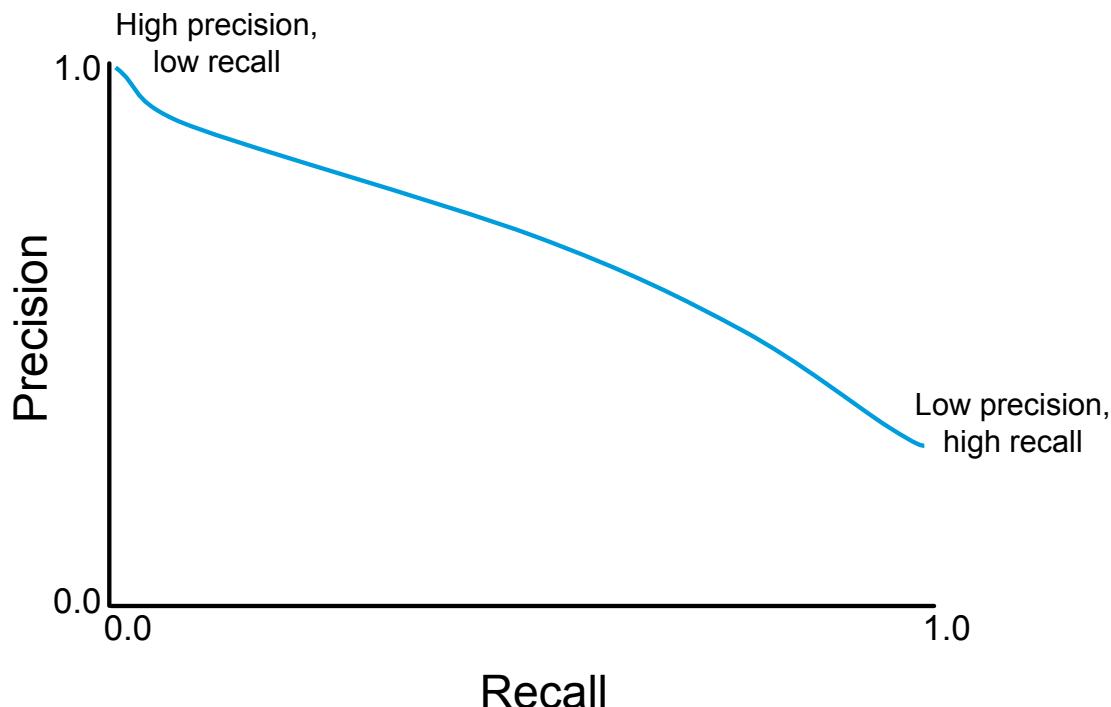


Figure 6–16: The precision and recall tradeoff.

When	False Positives	False Negatives
Precision increases	Tend to decrease	Tend to increase
Recall increases	Tend to increase	Tend to decrease

Which factor you choose to emphasize depends on your business requirements. The bridge collapse model may be more conducive to recall, but there are plenty of other datasets and scenarios where precision (or some other metric) would be the better choice.

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.

Specificity

Specificity, also called the **true negative rate (TNR)**, is a measure of how often the model identifies the actual negatives.

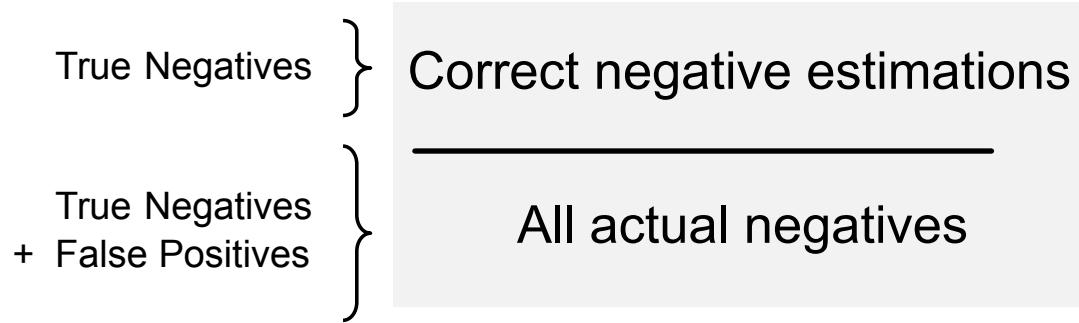


Figure 6-17: How to calculate specificity.

As shown here, specificity is the number of correct negative estimations divided by the total number of actual negatives. In other words, specificity can be expressed as:

$$\frac{\text{TN}}{\text{TN} + \text{FP}}$$

Just like the other classification metrics you've seen thus far, specificity can be expressed as a number from 0 to 1.0 or as a percentage.

In the bridge collapse example, you would calculate specificity as $512 / (512 + 1)$, or around 99%. However, specificity is useful for when you need to maximize the amount of true negatives the model produces. Also, like accuracy, it doesn't do so well when the label in the dataset is imbalanced. So, the bridge collapse example is not really a good candidate for specificity.

Consider a different scenario in which you have a model that predicts whether or not students will pass an exam on their first attempt. The label is positive when the student passes the exam, and negative when they do not. The label seems to be balanced, where roughly half of the students have passed the exam on their first attempt in prior years. In this case, you might want your model to maximize true negatives so that you can more easily offer alternative resources and one-on-one mentoring to the students who are unlikely to pass the exam the first time, or take some other action that will reduce exam failure rates. Therefore, specificity might be a good metric to optimize.

Receiver Operating Characteristic (ROC) Curves

A **receiver operating characteristic (ROC) curve** is a method of plotting the relationship between estimated "hits" versus false alarms. On the y-axis is the **true positive rate (TPR)** (the "hits"), which is 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:

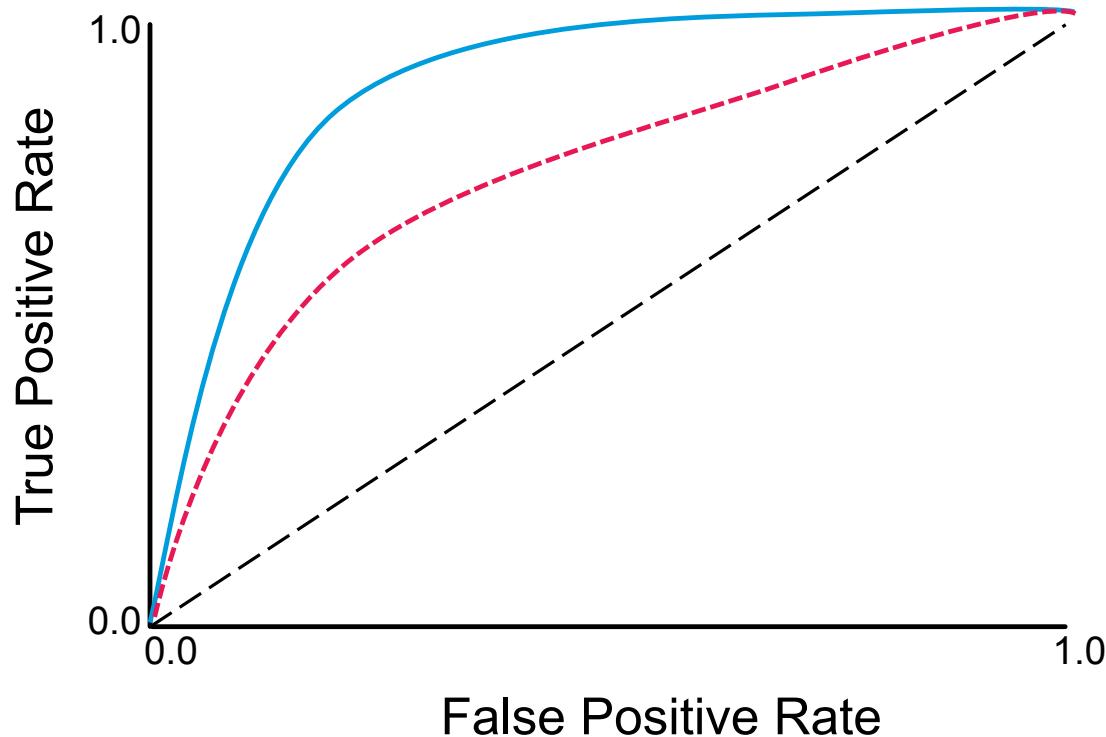


Figure 6-18: ROC curves plotted on a graph.

The diagonal dashed line (0.5, 0.5) represents a random guess. Therefore:

- Any data above and to the left of the dashed line is *better* than a random guess. This is what you want for your model.
- Any data near or at the dashed 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 dashed line is *worse* than a random guess. This typically means there's a problem with the model.

A perfectly estimative 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 dashed 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 (TNR)/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 estimative model.

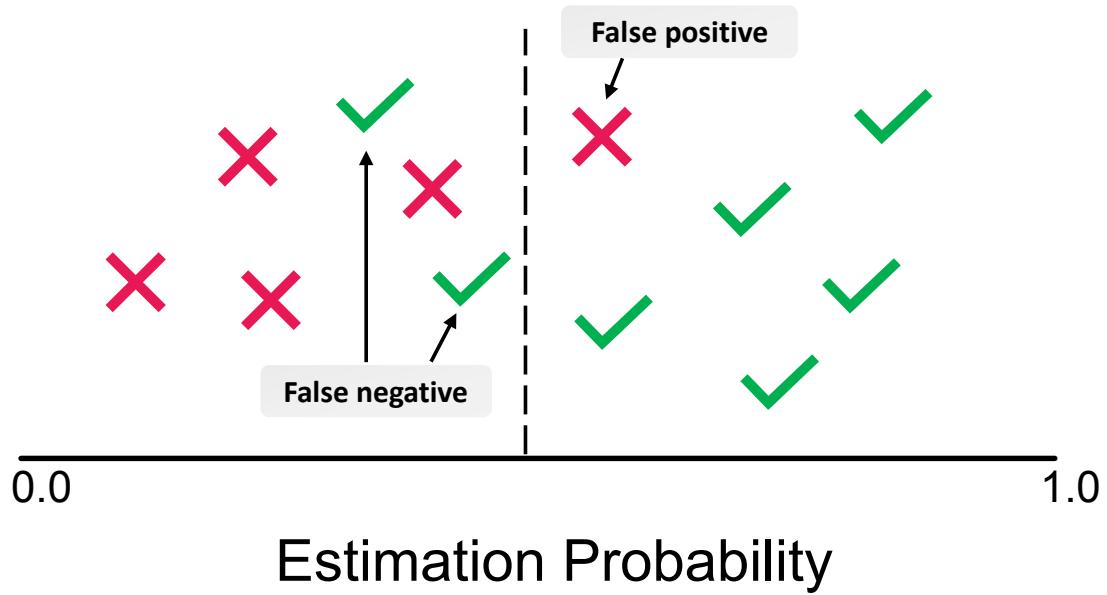


Figure 6–19: Plotting classification values with a threshold of 0.50.

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.

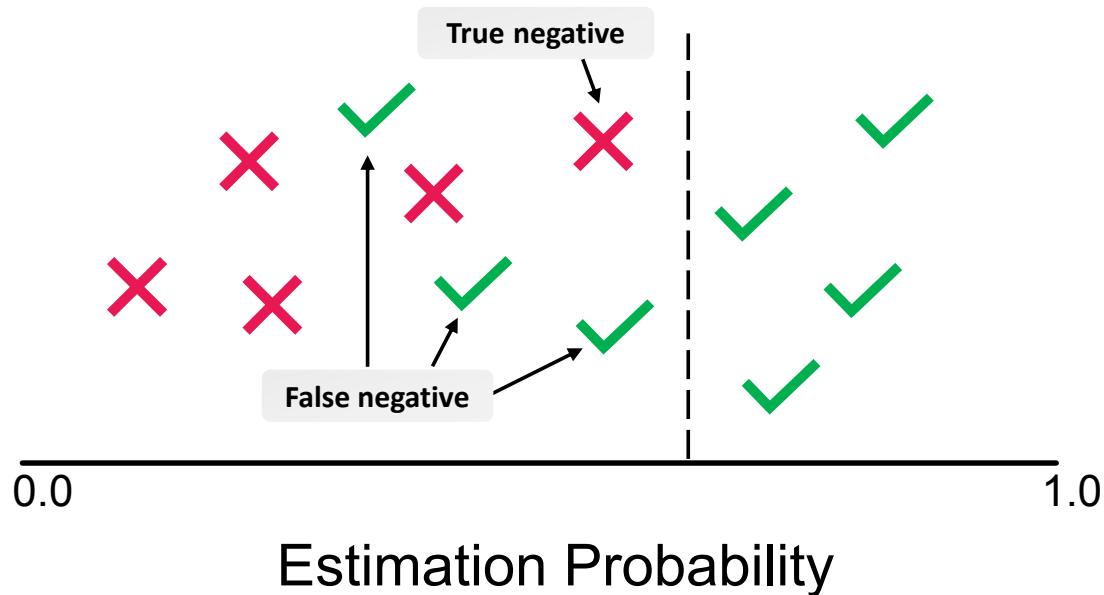


Figure 6-20: Moving the threshold to ~0.70.

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.

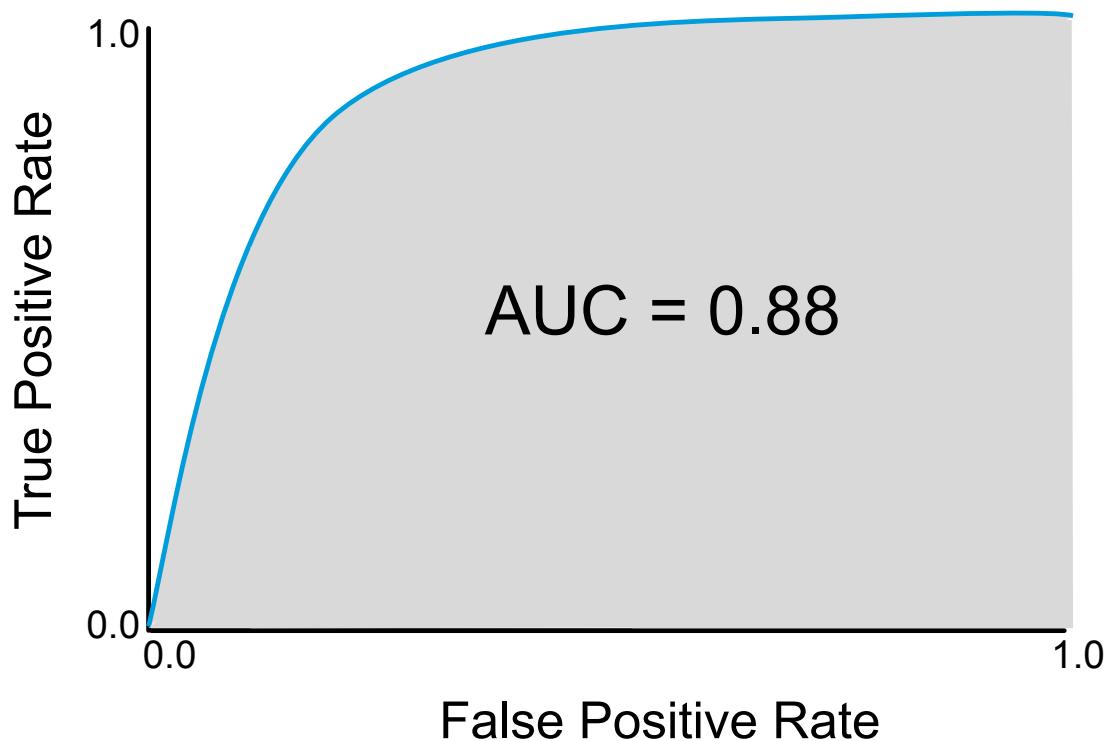


Figure 6-21: The AUC in this example is 0.88.

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.

It's important to note that 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.

Precision-Recall Curves (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:

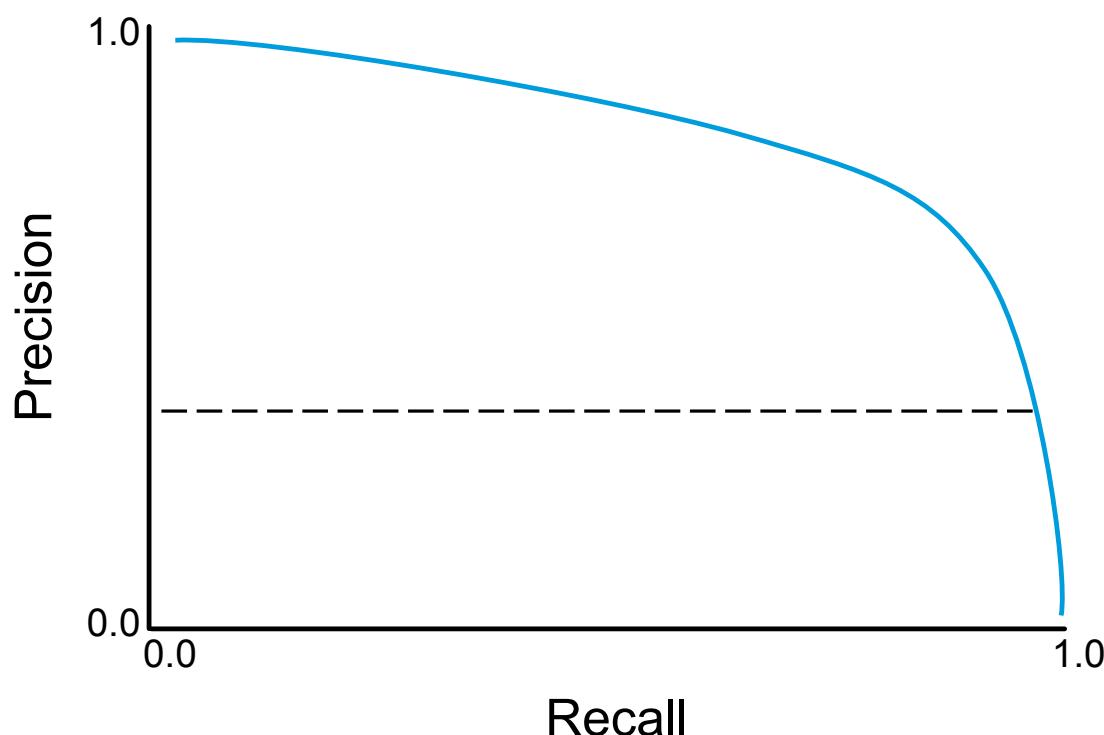


Figure 6–22: A skillful model, with the curve above the line.

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 estimated 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 these truth results for a classification problem.
- For individual metrics:
 - 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.
- Prefer specificity when you need to maximize the amount of true negatives in a balanced dataset.
- 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.
- Overall:
 - 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.



Note: There is no separate method for calculating specificity; however, by supplying the argument `pos_label = 0` to `recall_score()`, you are essentially flipping the label that gets reported, causing the method to report specificity instead of sensitivity (recall).

ACTIVITY 6–4

Evaluating a Classification Model

Before You Begin

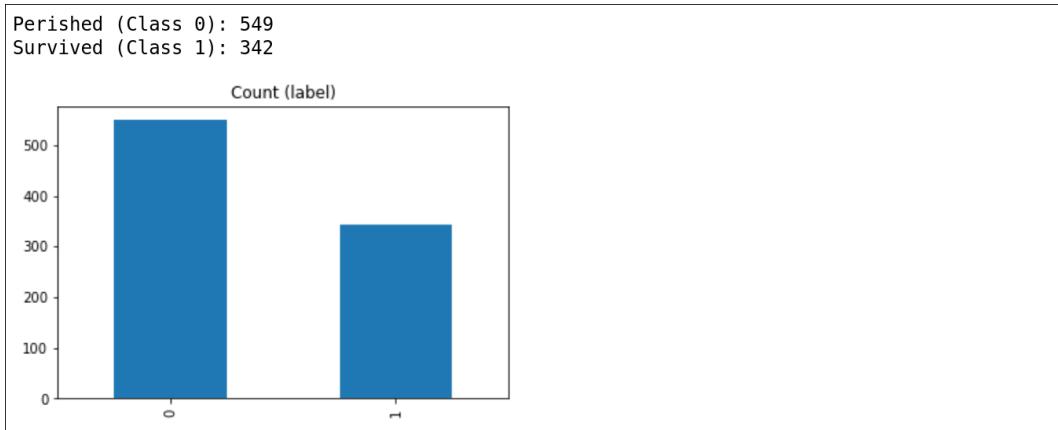
If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the **CAIP/Logistic Regression and k-NN/Logistic Regression and k-NN - Titanic.ipynb** notebook. To ensure all Python objects and output are in the correct state to begin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

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. Identify the class distribution in the dataset.

- Scroll down and view the cell titled **Identify the class distribution in the dataset**, and examine the code cell below it.
- Run the code cell.
- Examine the output.



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.

2. Generate a confusion matrix.

- Scroll down and view the cell titled **Generate a confusion matrix**, and examine the code cell below it.
- Run the code cell.

- c) Examine the output.

The confusion matrix graphing function has been defined.

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.

3. 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 cell below it.
- b) Run the code cell.
- c) Examine the output.

The function to compute scores has been defined.

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.

4. 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 cell below it.
- b) Run the code cell.
- c) Examine the output.

The function to generate a ROC curve and compute AUC has been defined.

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.

5. 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 cell below it.
- b) Run the code cell.
- c) Examine the output.

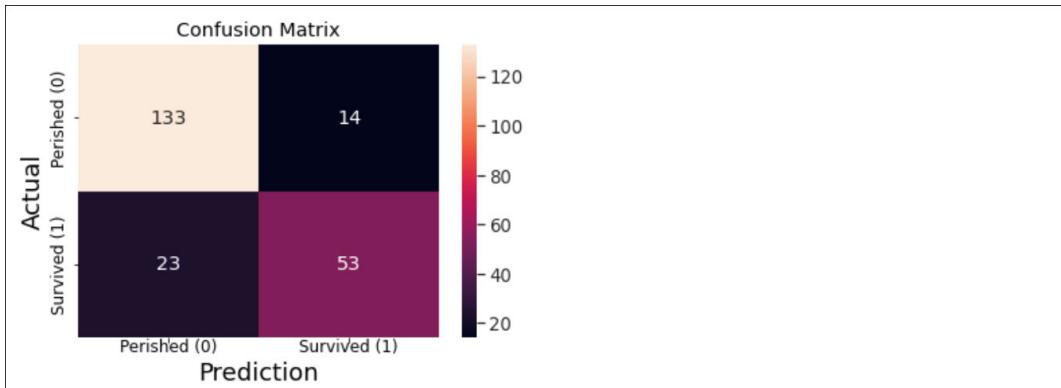
The function to generate a precision–recall curve and compute average precision has been defined.

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.

6. Evaluate the initial logistic regression model.

- a) Scroll down and view the cell titled **Evaluate the initial logistic regression model**, and examine the code cell below it.
- b) Run the code cell.

- c) Examine the output.



To help orient you, each quadrant is plotted as follows:

- Top-left (133): *True negatives*
- Top-right (14): *False positives*
- Bottom-left (23): *False negatives*
- Bottom-right (53): *True positives*

7. 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*?

8. Continue evaluating the initial logistic regression model.

- a) Scroll down and examine the next code cell.

```
1 model_scores(y_test, initial_preds)
```

- b) Run the code cell.
c) Examine the output.

```
Accuracy: 83%
Precision: 79%
Recall: 70%
F1: 74%
```

The familiar accuracy score is printed, as are precision, recall, and F_1 scores for this model.

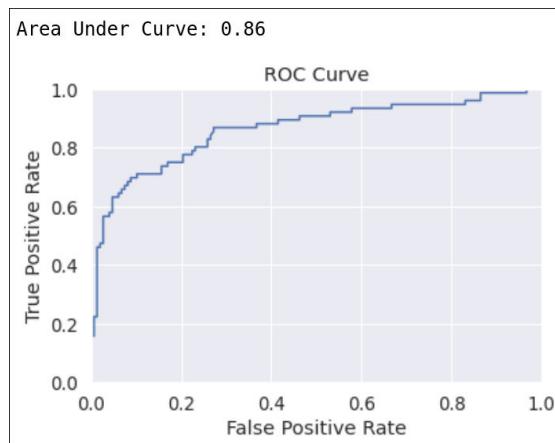
9. In what situation are precision and recall a better measure of a model's skill than accuracy?
10. Think about the problem unique to this *Titanic* dataset. In particular, consider the problem as it pertains to the model's accuracy, precision, recall, and F_1 score.
- Is there any one of these measures you'd be more interested in optimizing than the others? Why or why not?

11. Continue evaluating the initial logistic regression model.

a) Scroll down and examine the next code cell.

```
1 initial_pred_proba = log_reg.predict_proba(X_test)
2
3 roc(y_test, initial_pred_proba[:, 1])
```

b) Run the code cell.
c) Examine the output.

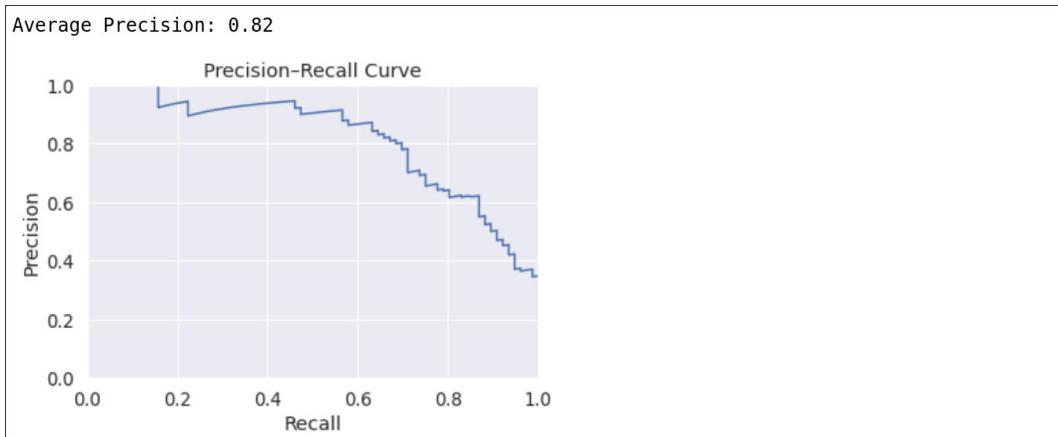


The ROC curve is above and to the left of the graph, indicating that the model is performing better than a random guess. The AUC also confirms this, as it is well above 0.5.

- d) Scroll down and examine the next code cell.

```
1 prc(y_test, initial_pred_proba[:, 1])
```

- e) Run the code cell.
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.

12.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 E

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 estimative 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 is 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.

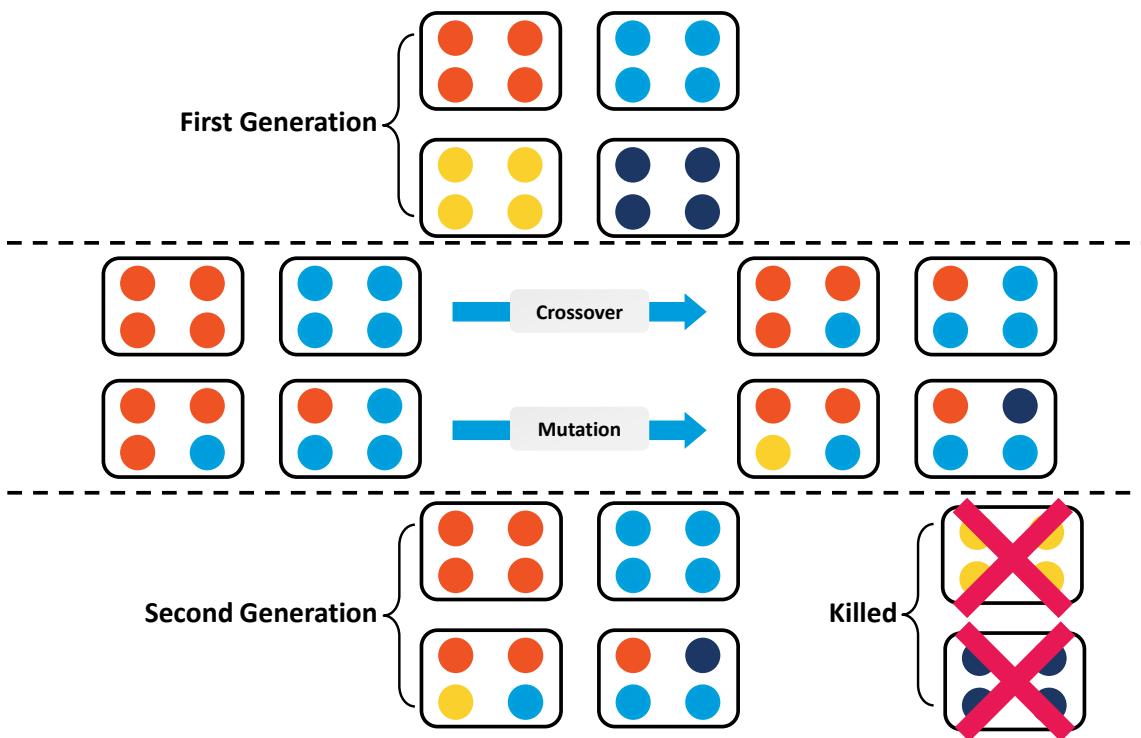


Figure 6-23: The hyperparameter optimization process applied by genetic algorithms.

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

If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the **CAIP/Logistic Regression and k-NN/Logistic Regression and k-NN - Titanic.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 **Fit a logistic regression model using grid search with cross-validation**. Select **Cell→Run All Above**.

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 cell below it.

The `grid` array on lines 6 through 12 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 on line 14:

- 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.

- Run the code cell.

This will take a few moments.

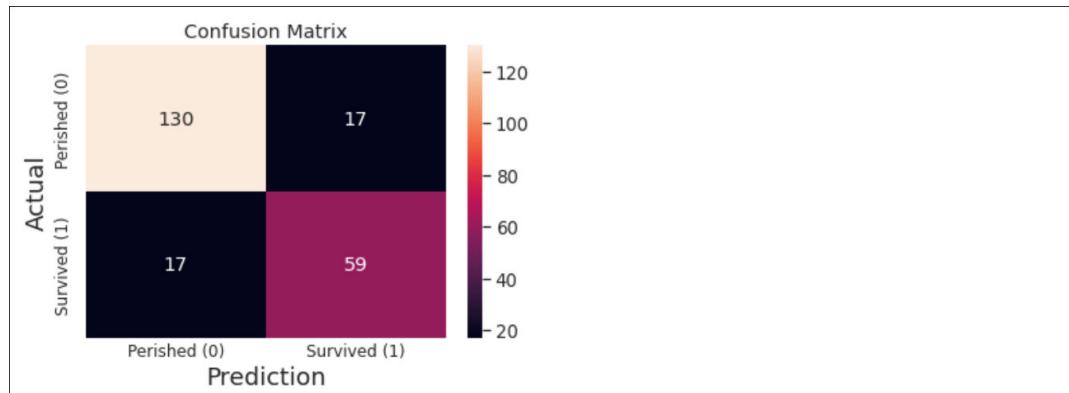
- Examine the output.

```
{'C': 25, 'penalty': 'l1', 'solver': 'liblinear'}
```

Grid search identified these hyperparameters as being the best combination for optimizing the F_1 score on the model.

2. Evaluate the optimized model.

- Scroll down and view the cell titled **Evaluate the optimized model**, and examine the code cell below it.
- Run the code cell.
- 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 examine the next code cell.

```
1 model_scores(y_test, search_preds)
```

- Run the code cell.
- Examine the output.

```
Accuracy: 85%
Precision: 78%
Recall: 78%
F1: 78%
```

As you can see, most of the scores for this optimized model have improved by a modest amount.

- The recall and F_1 have improved the most, with the former increasing by 8%, and the latter increasing by 4%.
- The accuracy improved by 2%.
- The precision actually went down by 1%.

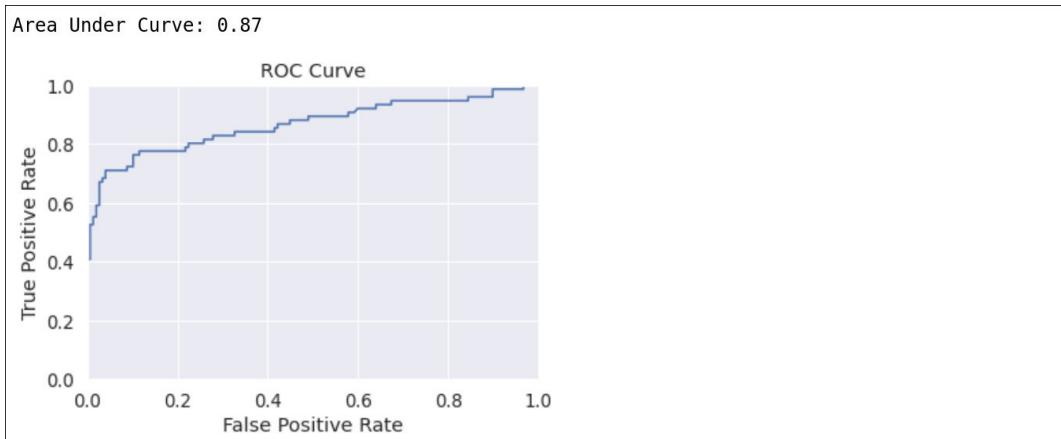
You typically won't be able to improve every metric, which is why it's important to focus on the metrics you think are most relevant.

- Scroll down and examine the next code cell.

```
1 search_pred_proba = search.predict_proba(X_test)
2
3 roc(y_test, search_pred_proba[:, 1])
```

- Run the code cell.

- f) Examine the output.

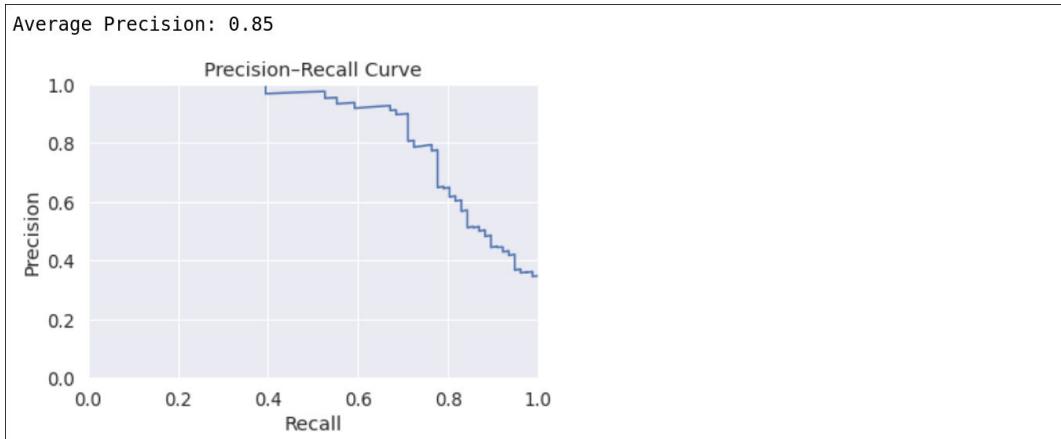


The ROC curve has changed, and the AUC has improved by 1%.

- g) Scroll down and examine the next code cell.

```
1 | prc(y_test, search_pred_proba[:, 1])
```

- h) Run the code cell.
i) Examine the output.



Once again, the curve has changed. The average precision statistic improved by about 3%.

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 unlabeled dataset.

- Scroll down and view the cell titled **Compare the logistic regression models' predictions on the unlabeled dataset**, and examine the code cell below it.
- Run the code cell.
- 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	84.27	15.73
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	1	41.90	58.10
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	0	82.60	17.40
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	0	79.68	20.32
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	1	40.69	59.31
5	897	3	Svensson, Mr. Johan Cervin	male	14.0	0	0	7538	9.2250	NaN	S	0	78.37	21.63
6	898	3	Connolly, Miss. Kate	female	30.0	0	0	330972	7.6292	NaN	Q	1	39.85	60.15
7	899	2	Caldwell, Mr. Albert Francis	male	26.0	1	1	248738	29.0000	NaN	S	0	78.78	21.22
8	900	3	Abrahim, Mrs. Joseph (Sophie Halau Easu)	female	18.0	0	0	2657	7.2292	NaN	C	1	37.10	62.90
9	901	3	Davies, Mr. John Samuel	male	21.0	2	0	A/4 48871	24.1500	NaN	S	0	84.97	15.03

These are the same results you saw earlier for the initial logistic regression model.

- Scroll down and examine the next code cell.

```

1 # Show example predictions with the new data using optimized logistic regression model.
2 results_new_opt = df_new_raw.copy()
3 results_new_opt['PredictedSurvival'] = search.predict(df_new)
4 results_new_opt['ProbPerished'] = np.round(search.predict_proba(df_new)[:, 0] * 100, 2)
5 results_new_opt['ProbSurvived'] = np.round(search.predict_proba(df_new)[:, 1] * 100, 2)
6 results_new_opt.head(10)

```

This code will display results for the model you just optimized.

- Run the code cell.
- 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	93.17	6.83
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	1	31.80	68.20
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	0	89.24	10.76
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	0	86.39	13.61
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	1	27.63	72.37
5	897	3	Svensson, Mr. Johan Cervin	male	14.0	0	0	7538	9.2250	NaN	S	0	82.31	17.69
6	898	3	Connolly, Miss. Kate	female	30.0	0	0	330972	7.6292	NaN	Q	1	37.92	62.08
7	899	2	Caldwell, Mr. Albert Francis	male	26.0	1	1	248738	29.0000	NaN	S	0	83.26	16.74
8	900	3	Abrahim, Mrs. Joseph (Sophie Halau Easu)	female	18.0	0	0	2657	7.2292	NaN	C	1	16.50	83.50
9	901	3	Davies, Mr. John Samuel	male	21.0	2	0	A/4 48871	24.1500	NaN	S	0	94.33	5.67

By comparing both results, you can see that the optimized model made the same predictions for the first ten passengers as the initial model. However, the classification probabilities are different.

- g) Scroll down and examine the next code cell.

```

1 # Return first 10 different predictions only.
2 results_new_opt[['PredictedSurvival']].compare(results_new[['PredictedSurvival']]) \
3 .head(10)

```

This code will compare the predictions in both data frames and return where they are different (for the first 10 instances).

- h) Run the code cell.
i) Examine the output.

PredictedSurvival		
	self	other
11	1.0	0.0
21	1.0	0.0
28	1.0	0.0
41	1.0	0.0
50	1.0	0.0
67	1.0	0.0
73	1.0	0.0
80	1.0	0.0
81	0.0	1.0
89	1.0	0.0

You can see which rows (passengers) had different survival predictions. The `self` column indicates the optimized model's predictions, and the `other` column indicates the initial model's predictions.

7. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel**→**Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **Logistic Regression and k-NN - Titanic** tab in Firefox, but keep a tab open to **CAIP** in the file hierarchy.

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.

7

Building Clustering Models

Lesson Time: 2 hours, 30 minutes

Lesson Introduction

You've built models to tackle regression problems, forecasting 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 unsupervised models using k -means clustering.
- Building unsupervised models using hierarchical clustering.

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.

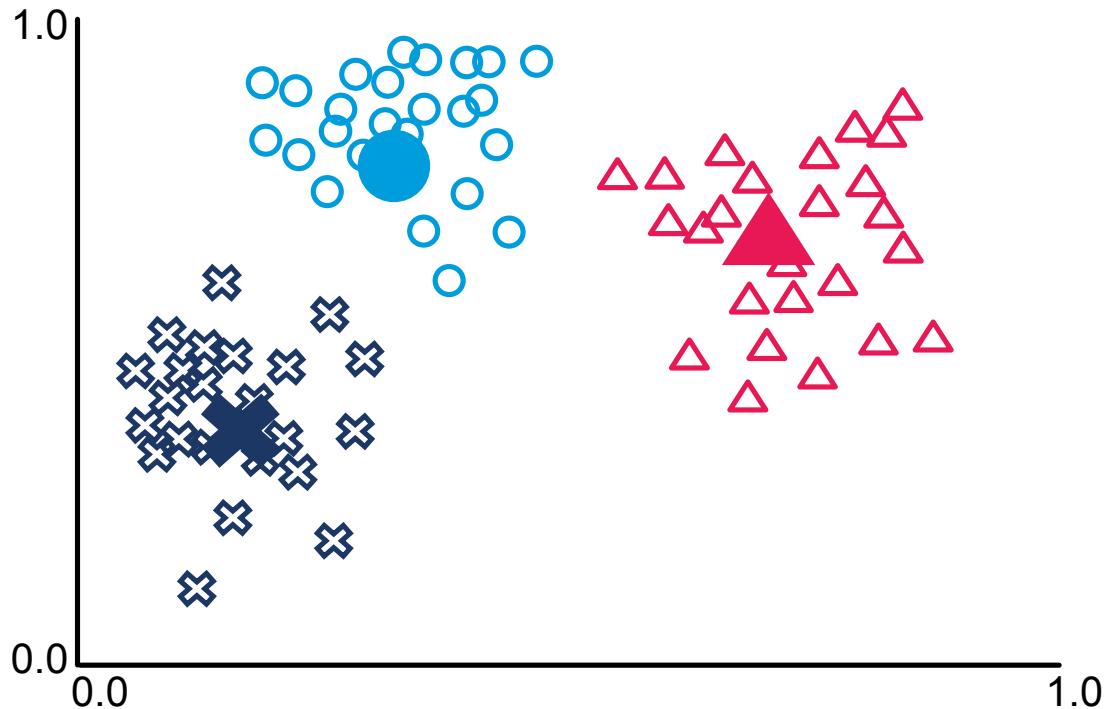


Figure 7-1: Three clusters of data, in which the large, filled-in shapes represent the centroids.

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. The centroids then move to this new mean value, and each data example is reassigned to whatever centroid is now closest. This process repeats 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. You therefore won't need to estimate a "correct" number of clusters.

The 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.

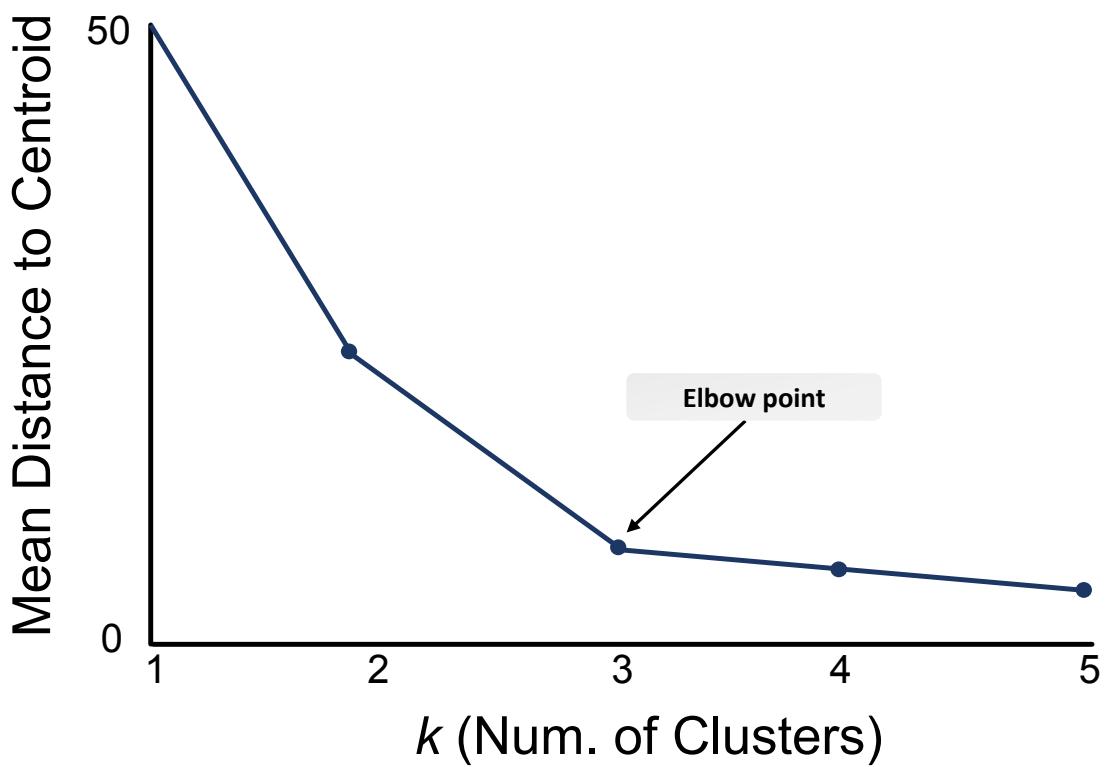


Figure 7-2: The elbow point indicates that k should be 3.

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.

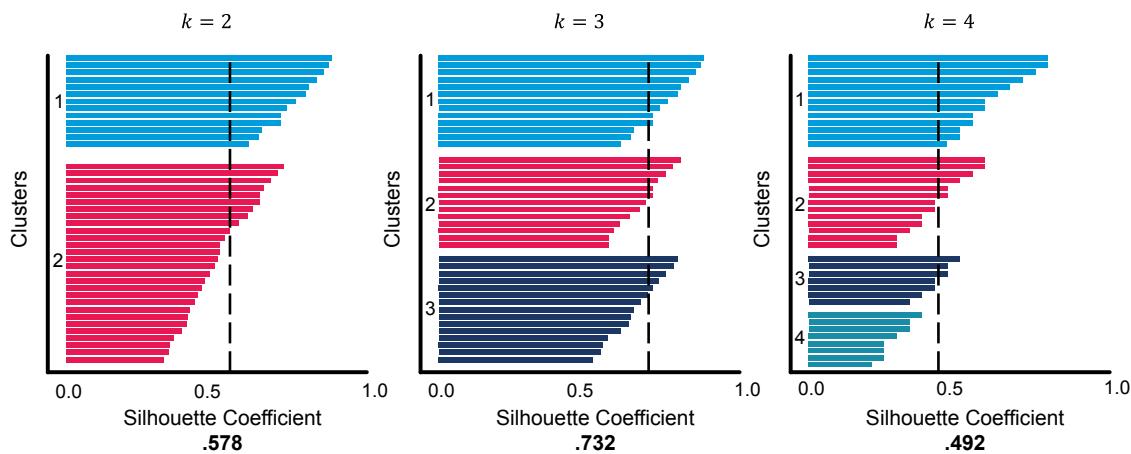


Figure 7-3: A silhouette analysis of three different k values. The dashed line indicates the average coefficient for that model.

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 k -Means Clustering Models

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, or for visualizing silhouette scores, but the third-party Yellowbrick library does:

- `visualizer = yellowbrick.cluster.KElbowVisualizer(model, k = (1, 10))` —This generates an elbow point visualization object by passing in a scikit-learn model and a range of k values to show.
- `visualizer = yellowbrick.cluster.SilhouetteVisualizer(model)` —This generates a silhouette visualization object by passing in a scikit-learn model. It will show the silhouette scores for the number of clusters specified in the model object that you pass in as `model`.
- `visualizer.fit(X_train)` —Fit a set of training data to the visualizer.

ACTIVITY 7–1

Building a *k*-Means Clustering Model

Data Files

/home/student/CAIP/Clustering/Clustering - KC Housing.ipynb
 /home/student/CAIP/Clustering/housing_data/kc_house_data_prep.pickle
 /home/student/CAIP/Clustering/housing_data/kc_house_data_prep_norm.pickle

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. From Jupyter Notebook, select **CAIP/Clustering/Clustering - KC Housing.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 cell below it.
 - b) Run the code cell.
 - c) Verify that **kc_house_data_prep.pickle** and **kc_house_data_prep_norm.pickle** were loaded with 21,609 records each.

These are the pickle files of the King County housing data you saved earlier. You'll use the non-normalized version for preliminary visualization purposes, whereas you'll use the normalized version to train an actual candidate model.
3. Reacquaint yourself with the dataset.
 - a) Scroll down and view the cell titled **Reacquaint yourself with the dataset**, and examine the code cell below it.
 - b) Run the code cell.

- c) Examine the output.

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	yr_built_group	y
0	7129300520	2014-10-13	2219000.0	3	1.00	1180	5650	1.0	0	0	...	1941–1960	
1	6414100192	2014-12-09	5380000.0	3	2.25	2570	7242	2.0	0	0	...	1941–1960	
2	5631500400	2015-02-25	1800000.0	2	1.00	770	10000	1.0	0	0	...	1921–1940	
3	2487200875	2014-12-09	6040000.0	4	3.00	1960	5000	1.0	0	0	...	1961–1980	
4	1954400510	2015-02-18	510000.0	3	2.00	1680	8080	1.0	0	0	...	1981–2001	

5 rows × 25 columns

- Recall that each row is a house with various attributes, like the number of bedrooms, number of bathrooms, the square footage, etc.
- The `price` feature was the target in your supervised learning efforts, but for this unsupervised project, you'll focus on other features.

4. Extract a new feature for price per square foot.

- Scroll down and view the cell titled **Extract a new feature for price per square foot**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

_built_encoded	yr_renovated	yr_ren_group	yr_ren_encoded	zipcode	lat	long	sqft_living15	sqft_lot15	price_per_sqft
2	0	N/A	0	98178	47.5112	-122.257	1340	5650	188.05
2	1991	1975–1994	3	98125	47.7210	-122.319	1690	7639	209.34
1	0	N/A	0	98028	47.7379	-122.233	2720	8062	233.77
3	0	N/A	0	98136	47.5208	-122.393	1360	5000	308.16
4	0	N/A	0	98074	47.6168	-122.045	1800	7503	303.57

The last column in the data frame shows the `price_per_sqft` feature that was created by dividing `price` by `sqft_living`. You'll use this new feature when you visualize the clusters.

5. Use a *k*-means model to cluster every row in the dataset.

- Scroll down and view the cell titled **Use a *k*-means model to cluster every row in the dataset**, and examine the code cell below it.
 - This function, when called, will create a *k*-means clustering model object using the number of clusters specified in the call.
 - On line 5, 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.
 - On line 11, the function will return a data frame of the training dataset with a new column of cluster assignments added.
- Run the code cell.
- Examine the output.

The function to assign clusters using a *k*-means model has been defined.

You'll call this function soon.

6. Generate the cluster assignments and attach them to the original dataset.

- Scroll down and view the cell titled **Generate the cluster assignments and attach them to the original dataset**, and examine the code cell below it.
 - 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 5, the code uses an arbitrary number of clusters (4) to call the clustering function with.
- Run the code cell.
- Examine the output.

sw	...	yr_renovated	yr_ren_group	yr_ren_encoded	zipcode	lat	long	sqft_living15	sqft_lot15	price_per_sqft	cluster
0	...	0	N/A	0	98178	47.5112	-122.257	1340	5650	188.05	0
0	...	1991	1975–1994	3	98125	47.7210	-122.319	1690	7639	209.34	1
0	...	0	N/A	0	98028	47.7379	-122.233	2720	8062	233.77	1
0	...	0	N/A	0	98136	47.5208	-122.393	1360	5000	308.16	0
0	...	0	N/A	0	98074	47.6168	-122.045	1800	7503	303.57	2

At the end of the data frame, the `cluster` column shows which cluster each data example is placed in: 0, 1, 2, or 3.

7. Observe how many houses were distributed to each cluster.

- Scroll down and view the cell titled **Observe how many houses were distributed to each cluster**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
Number of houses in cluster 0 = 5074
Number of houses in cluster 1 = 7437
Number of houses in cluster 2 = 4818
Number of houses in cluster 3 = 4280
```

Each cluster contains several thousand houses, with cluster 1 having the most.

8. Show clusters of homes on the map by location.

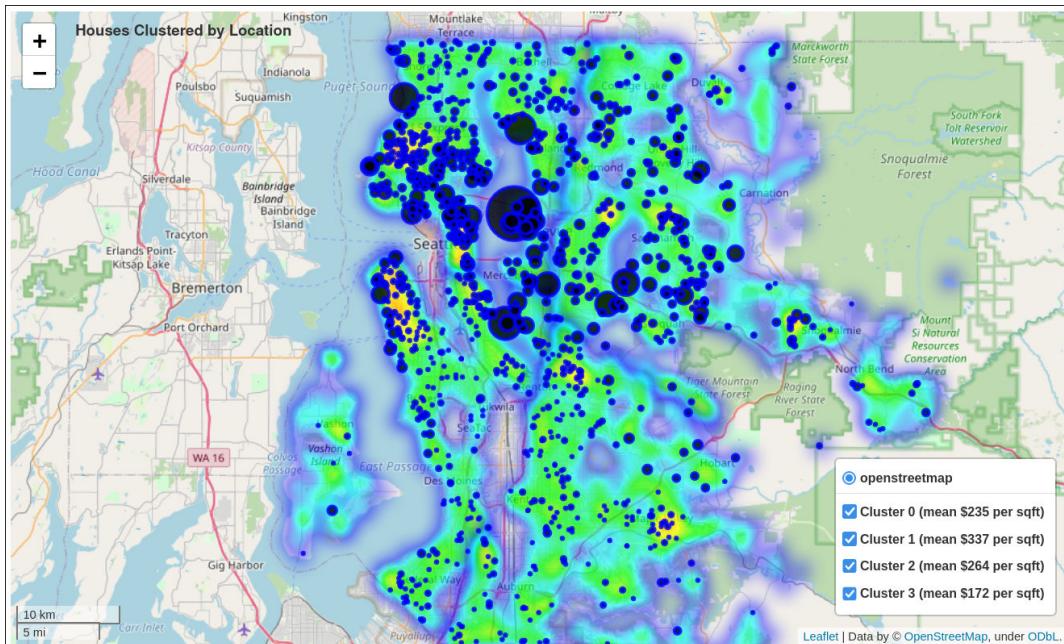
- Scroll down and view the cell titled **Show clusters of homes on the map by location**, and examine the code cell below it.
 - The `show_on_map()` function plots every 20th data point on a geographic map of the King County area, along with the clusters of that data.
 - Lines 12 through 16 use the Folium library to display the geographic map.
 - Line 19 defines the highest price of the houses in the dataset to use in scaling the rest of the points.
 - Line 22 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 25 through 52 create the markers and pop-up text for the data points.
 - The `for` loop that begins on line 58 iterates through each cluster.
 - Lines 60 through 70 create the heatmap 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.
- Run the code cell.
- Examine the output.

```
The function to show the map has been defined.
```

- d) Scroll down and examine the next code cell.

```
1 # View the results on the map.
2 show_on_map(clust_houses, 'Houses Clustered by Location')
```

- e) Run the code cell.
f) 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 heatmaps, respectively.



Note: Drag to adjust the location and zoom in and out as needed using the + and - buttons.

9. How did the clusters form with regard to location?

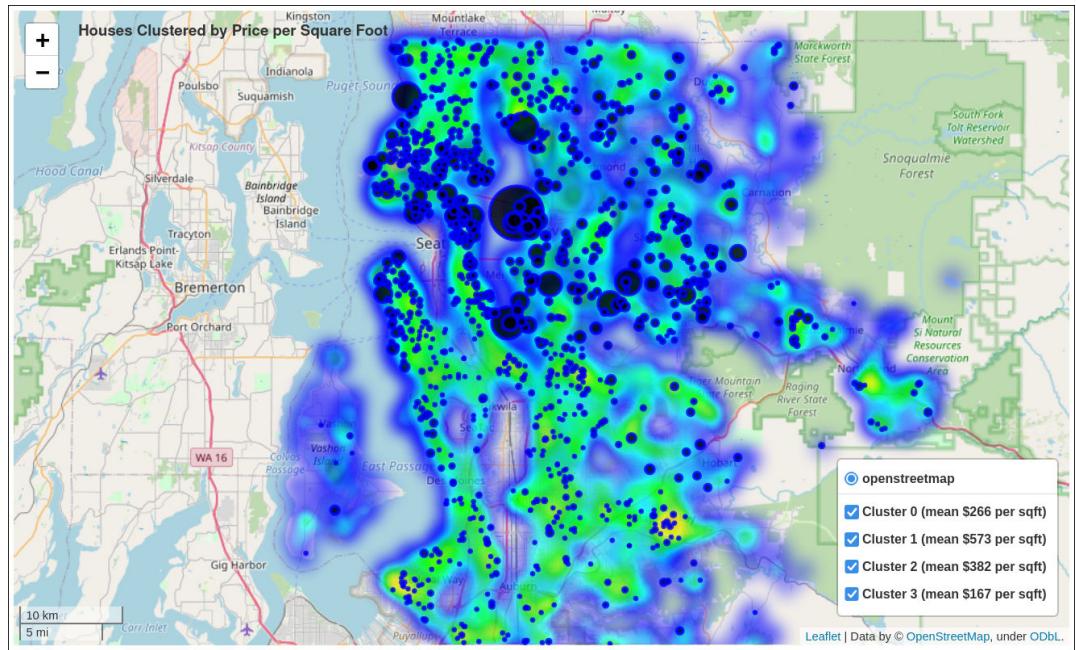
10. Cluster by price per square foot.

- a) Scroll down and view the cell titled **Cluster by price per square foot**, and examine the code cell below it.

This is essentially the same code as before, except it's using `price_per_sqft` rather than `lat` and `long` (latitude and longitude).

- b) Run the code cell.

- c) Examine the output.



11. How did the clusters form with regard to price per square foot?

12. 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 cell below it.

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.

Note that, on line 3, the normalized data is being used to train the clustering models from here on out. Remember, distance-based algorithms like k -means clustering benefit significantly from scaled data.

- b) Run the code cell.
c) Examine the output.

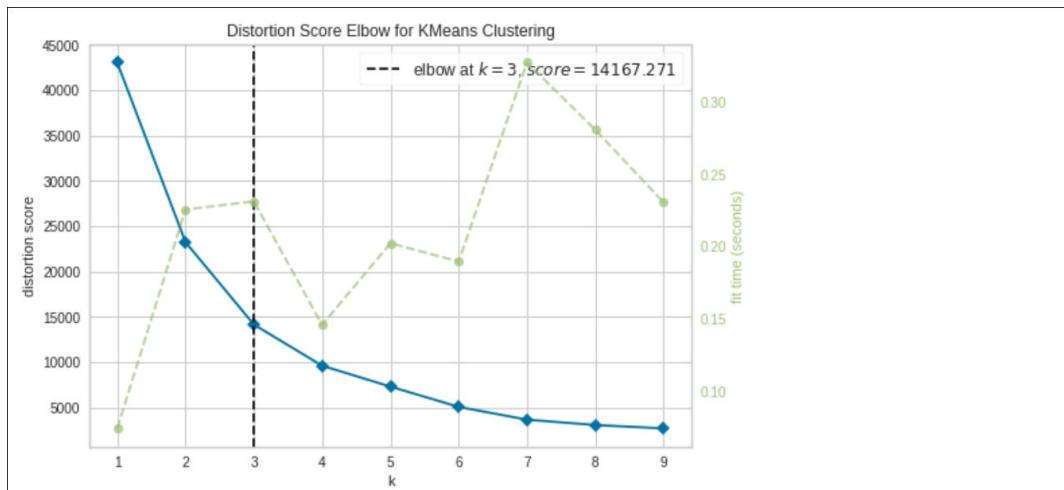
A dataset containing features of interest to customers has been defined.

13. Use the elbow method to determine the optimal number of clusters.

- a) Scroll down and view the cell titled **Use the elbow method to determine the optimal number of clusters**, and examine the code cell below it.

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.

- Run the code cell.
- 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.

14. Use silhouette analysis to determine the optimal number of clusters.

- Scroll down and view the cell titled **Use silhouette analysis to determine the optimal number of clusters**, and examine the code cell below it.
 - 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 25 through 27 keep track of the highest mean coefficient of all samples.
 - Lines 38 and 39 use Yellowbrick to plot the silhouette graphs on the left side of the screen for each k value.
 - Lines 49 through 72 continue the `for` loop, this time plotting the scatter plot that will appear on the right side of the screen for each corresponding silhouette graph.
 - Lines 52 and 53 will use the square foot living space and number of bathrooms as the axes for the scatter plots.
 - Lines 62 through 66 add boxes to the scatter plots representing the cluster centroids.
 - Lines 76 and 77 print the highest silhouette coefficient and the number of clusters that obtained it. This can be used to determine the optimal number of clusters.
- Run the code cell.



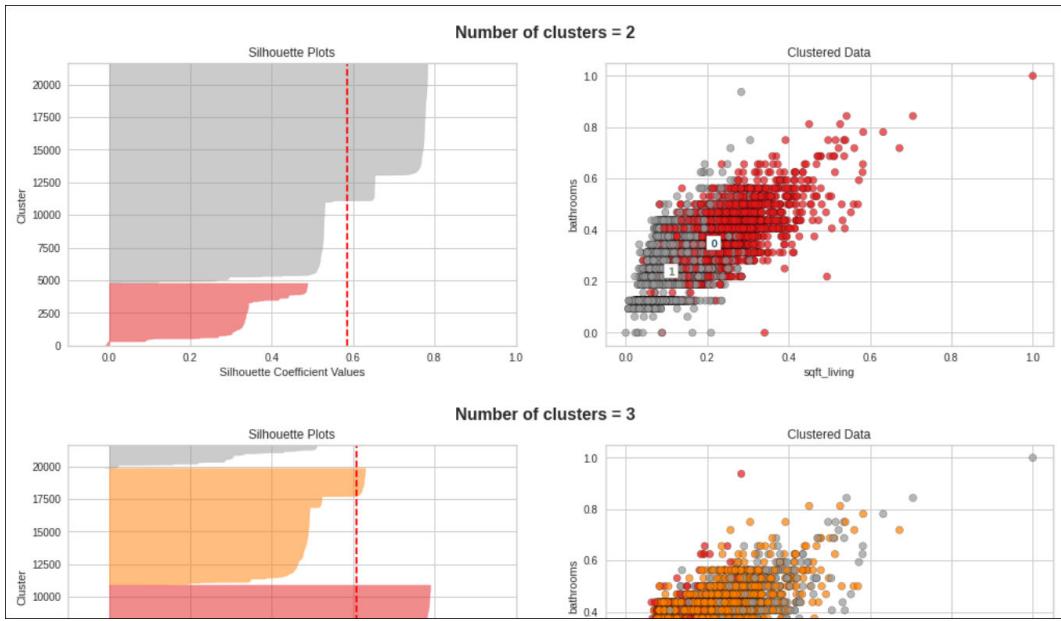
Note: It may take several minutes for the code to finish running.

- c) Examine the first part of the output.

```
With 2 clusters:  
  - Average silhouette score: 0.5846  
Number of houses in cluster 0 = 4794  
Number of houses in cluster 1 = 16815  
  
With 3 clusters:  
  - Average silhouette score: 0.6075  
Number of houses in cluster 0 = 10932  
Number of houses in cluster 1 = 8938  
Number of houses in cluster 2 = 1739  
  
With 4 clusters:  
  - Average silhouette score: 0.6283  
Number of houses in cluster 0 = 5522  
Number of houses in cluster 1 = 10932  
Number of houses in cluster 2 = 1665  
Number of houses in cluster 3 = 3490  
  
With 5 clusters:  
  - Average silhouette score: 0.6459  
Number of houses in cluster 0 = 5522  
Number of houses in cluster 1 = 10883  
Number of houses in cluster 2 = 832  
Number of houses in cluster 3 = 956  
Number of houses in cluster 4 = 3416
```

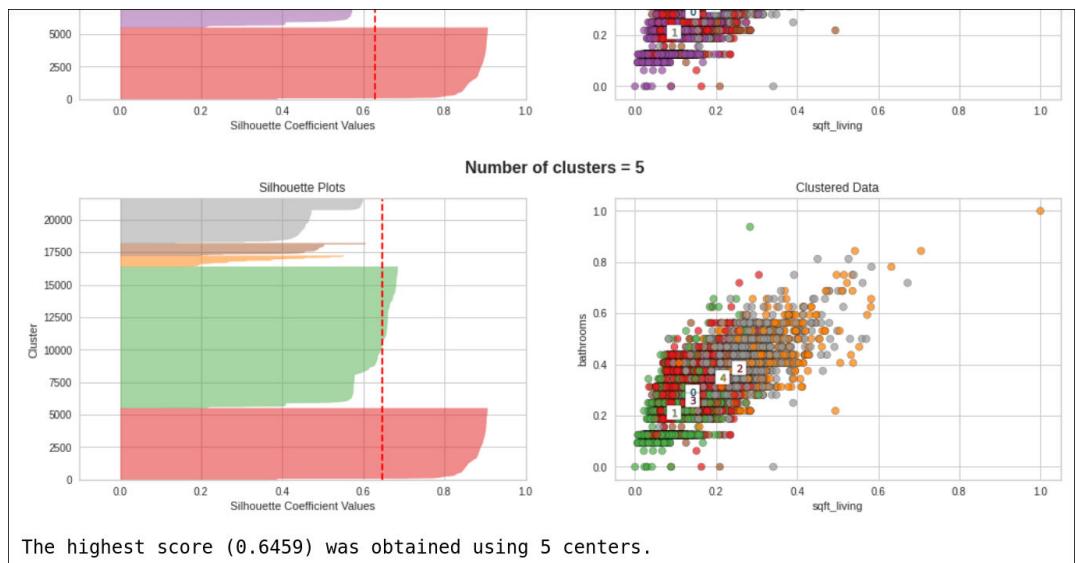
The highest score appears to be for the model with 5 clusters.

d) Examine the silhouette and scatter plots below.



- The overall silhouette coefficient for each round of training (i.e., each value of k) appears as a dashed red line on the silhouette plot.
- The silhouettes themselves show the relative size of each cluster as well as that specific cluster's silhouette coefficient.
- The scatter plot represents the corresponding clusters as they appear for square foot living space compared to number of bathrooms.
- Note that, on the scatter plots, the clusters *appear* to overlap. This is because you are representing a multi-dimensional model (i.e., one that was trained on multiple features) in two-dimensional space. The scatter plots therefore just give you a glimpse into how the clusters appear for these two features only—there are other factors of separation you are not seeing.
- Still, there are at least some patterns exhibited by comparing these two features. For example, in the model where $k = 2$, cluster 1 (gray) is more likely to include houses with a lower amount of living space and fewer bathrooms, whereas cluster 0 (red) usually includes houses with more living space and more bathrooms.

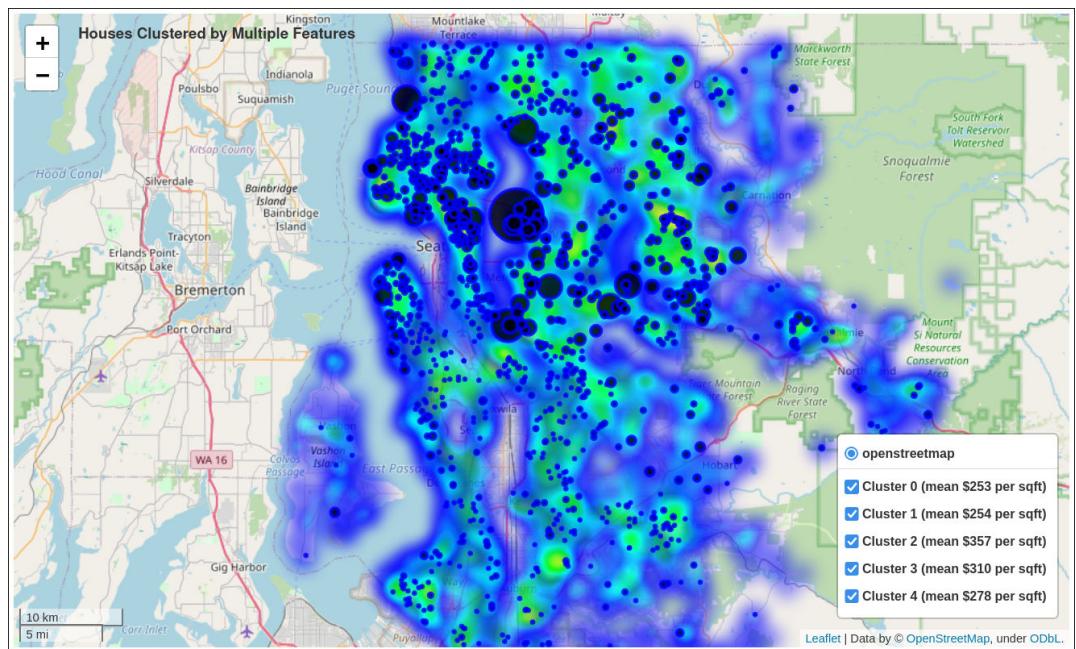
- e) Scroll as needed to see the remaining silhouette and scatter plots.



The silhouette method indicates that 5 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.

15. Use the optimal number of clusters to show the house groups.

- Scroll down and view the cell titled **Use the optimal number of clusters to show the house groups**, and examine the code cell below it.
- Run the code cell.
- Examine the output.



16. 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?

17. What are some reasons why this model may need to be retrained over time?

18. 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 - KC Housing** tab in Firefox, but keep a tab open 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:

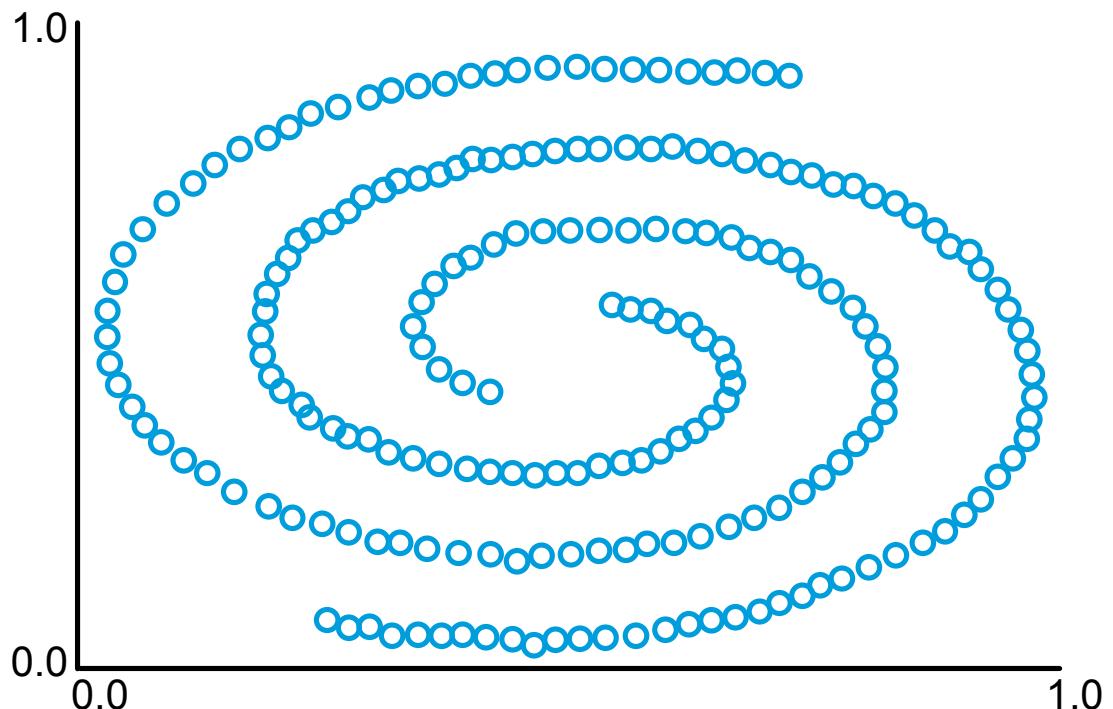


Figure 7–4: A spiral-shaped dataset.

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.

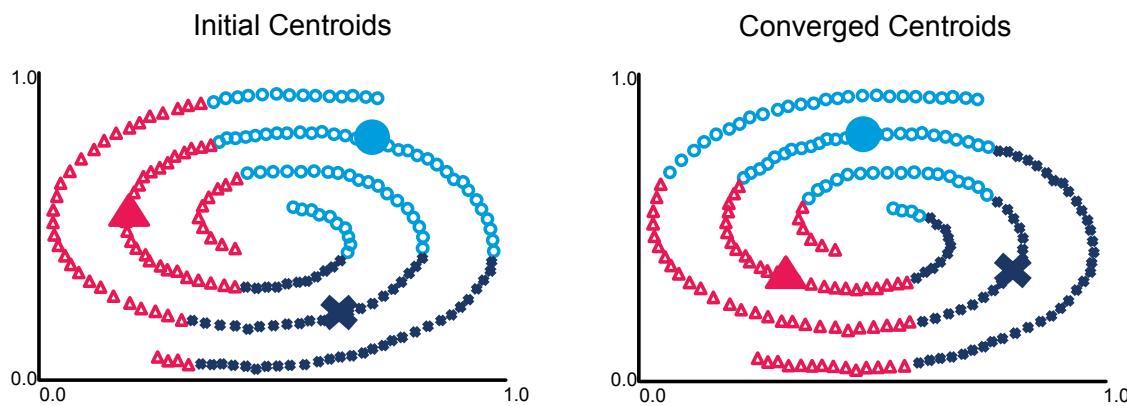


Figure 7-5: The clusters in this spiral-shaped dataset aren't particularly useful.

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.

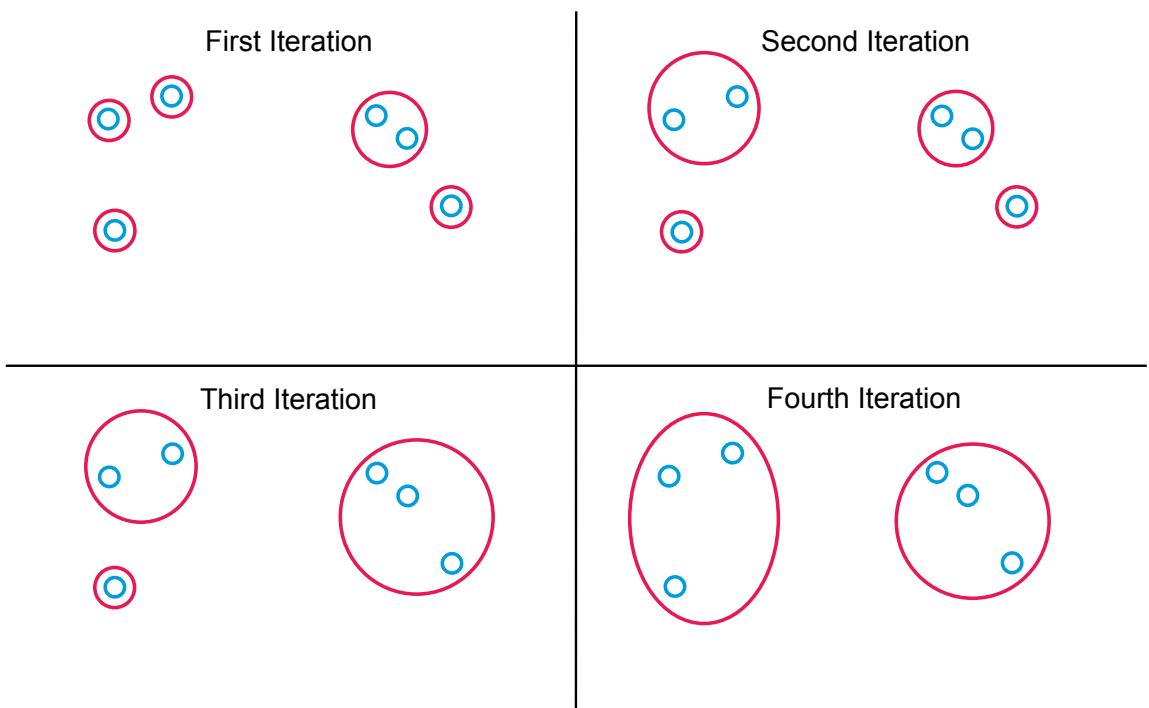


Figure 7-6: HAC starts merging the closest data points into clusters. A fifth iteration (not shown) would merge all of the points into one cluster.

Hierarchical divisive clustering (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.

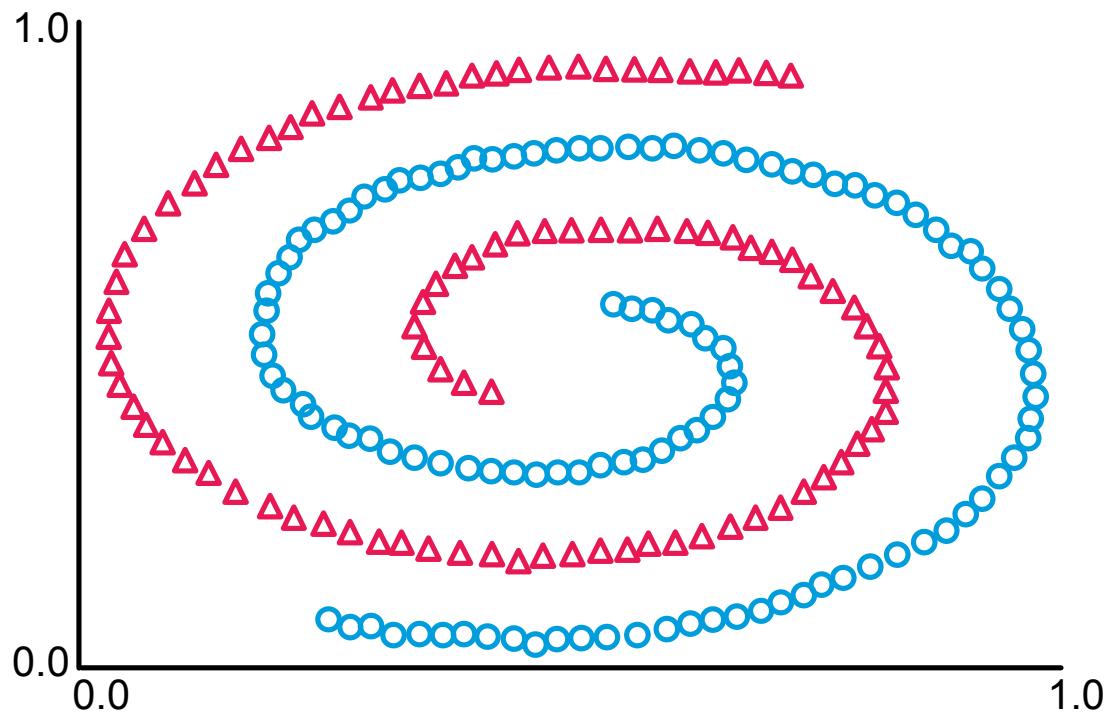


Figure 7-7: Hierarchical clustering applied to spiral data.

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 criteria, 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 is 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.

Dendrograms

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.

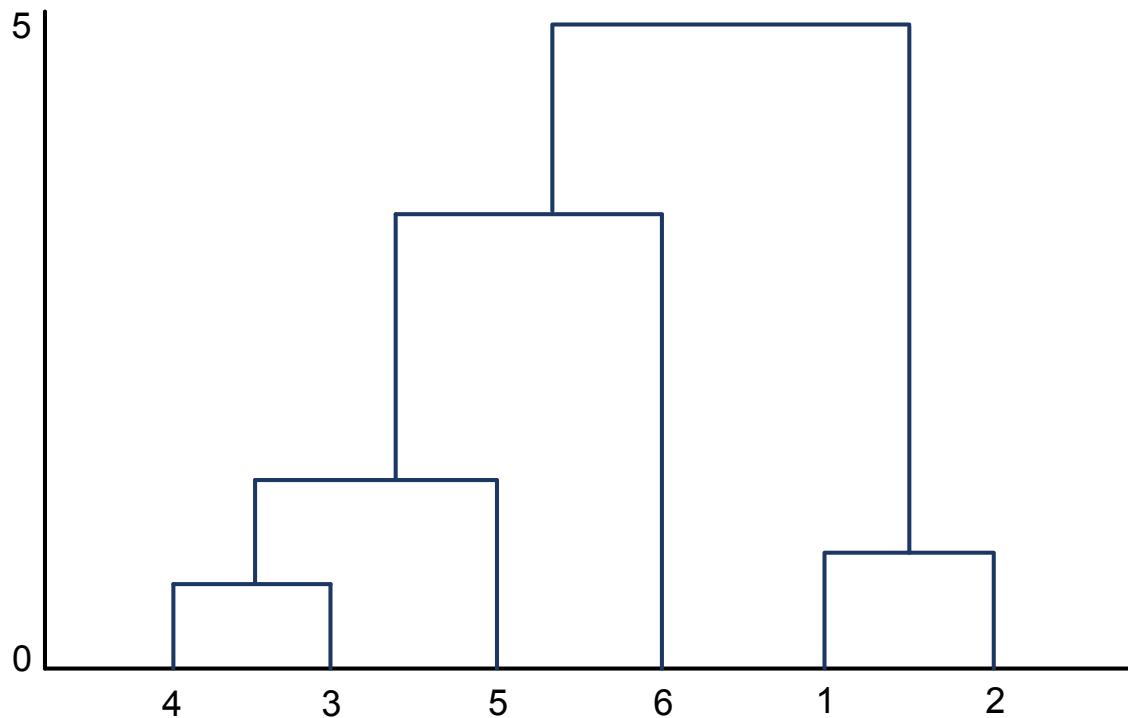


Figure 7-8: A dendrogram of hierarchical clustering data.

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.

You can use a dendrogram to visually analyze how specific data points are being clustered. Typically, you'd plot just a few data examples at a time to determine how they are being merged. If you were to plot all data examples at once, the dendrogram would become very difficult to read in all but the smallest datasets.

Dendrogram Cutoff Lines

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.

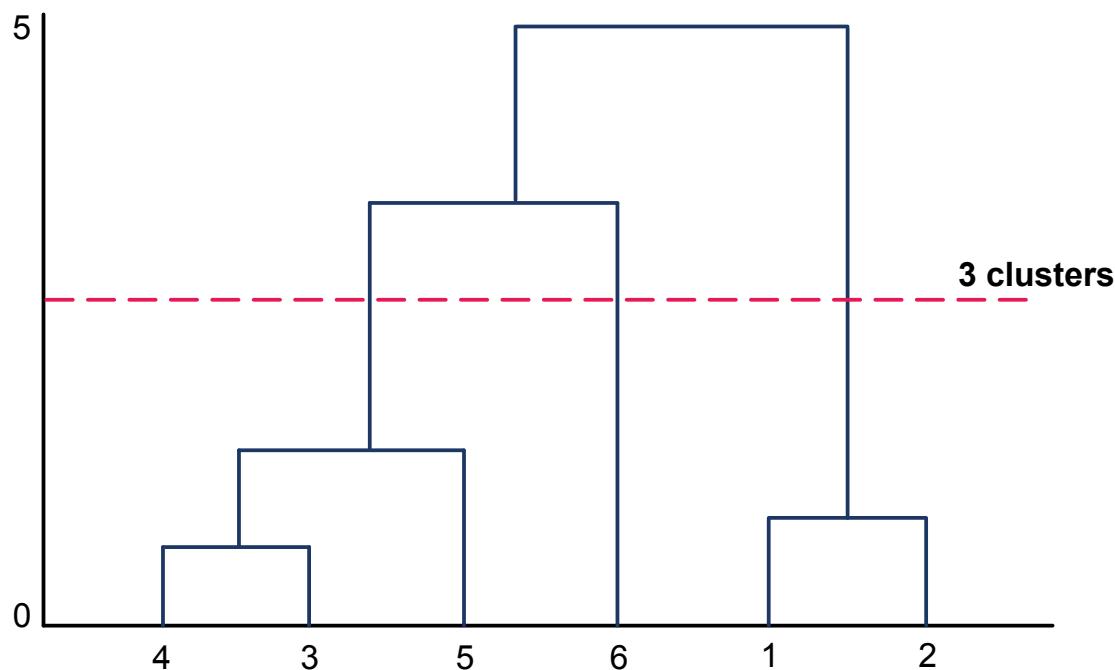


Figure 7–9: A cutoff point indicating that the number of clusters to stop on is 3.

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 Hierarchical Clustering Models

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
- Consider using HDC over HAC 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 and silhouette 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 7–2

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. From Jupyter Notebook, select **CAIP/Clustering/Clustering - Moons.ipynb** to open it.
2. Import the relevant libraries.
 - a) View the cell titled **Import software libraries**, and examine the code cell below it.
 - b) Run the code cell.
3. Generate the dataset.
 - a) Scroll down and view the cell titled **Generate the dataset**, and examine the code cell below it.

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 on line 1:

 - There are 1,000 total samples.
 - Some random noise is added to the sample spread for a bit more realism.
 - b) Run the code cell.

- c) Examine the output.

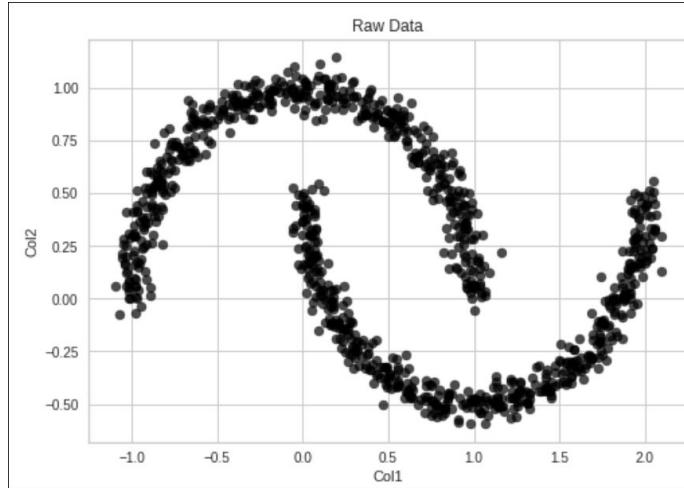
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   Col1    1000 non-null   float64 
 1   Col2    1000 non-null   float64 
dtypes: float64(2)
memory usage: 15.8 KB
None

   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
5  2.052310  0.557111
6  0.062628  0.190827
7  -0.469405  0.881809
8  -0.821266  0.518035
9  1.502068 -0.289701
```

There are two feature columns, both containing float values.

4. Plot the data to identify its shape.

- Scroll down and view the cell titled **Plot the data to identify its shape**, and examine the code cell below it.
- Run the code cell.
- Examine the output.



The data takes the shape of two symmetrical crescent moons.

5. Use a clustering model to cluster every row in the dataset.

- Scroll down and view the cell titled **Use a clustering model to cluster every row in the dataset**, and examine the code cell below it.

This function is similar to the function used to generate the clusters 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.

- Run the code cell.
- Examine the output.

The function to assign clusters has been defined.

6. Plot the clusters.

- Scroll down and view the cell titled **Plot the clusters**, and examine the code cell below it.

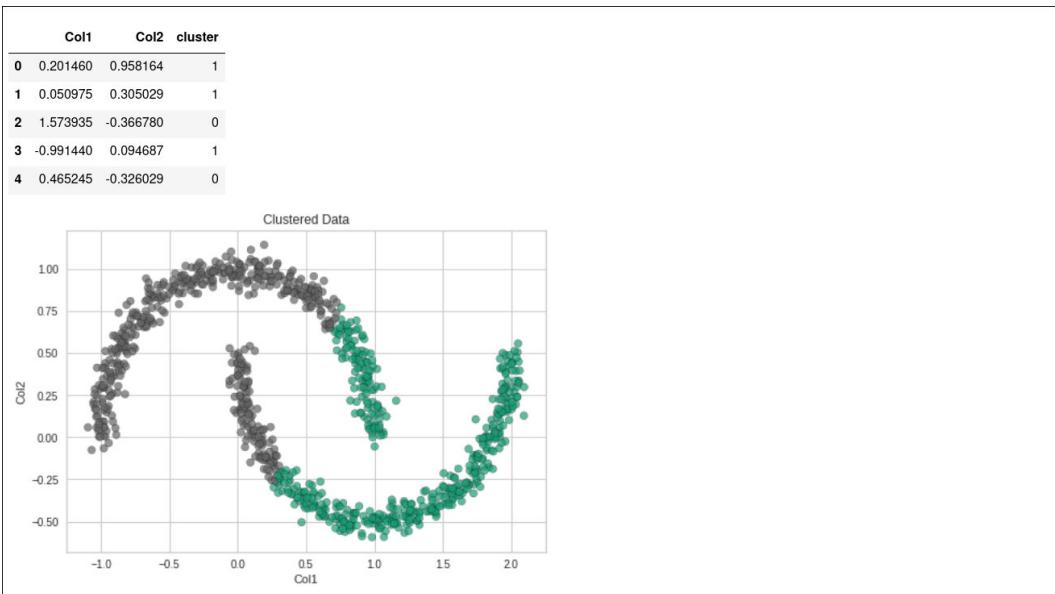
This function, when called, plots the data as before. The difference is on line 11, in which each of the model's clusters is being assigned to a different color (using a built-in Matplotlib color map). Each data point will be plotted in the color of whatever its cluster is.

- Run the code cell.
- Examine the output.

The function to plot the clusters has been defined.

7. Generate a k -means clustering model and plot the results.

- Scroll down and view the cell titled **Generate a k -means clustering model and plot the results**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

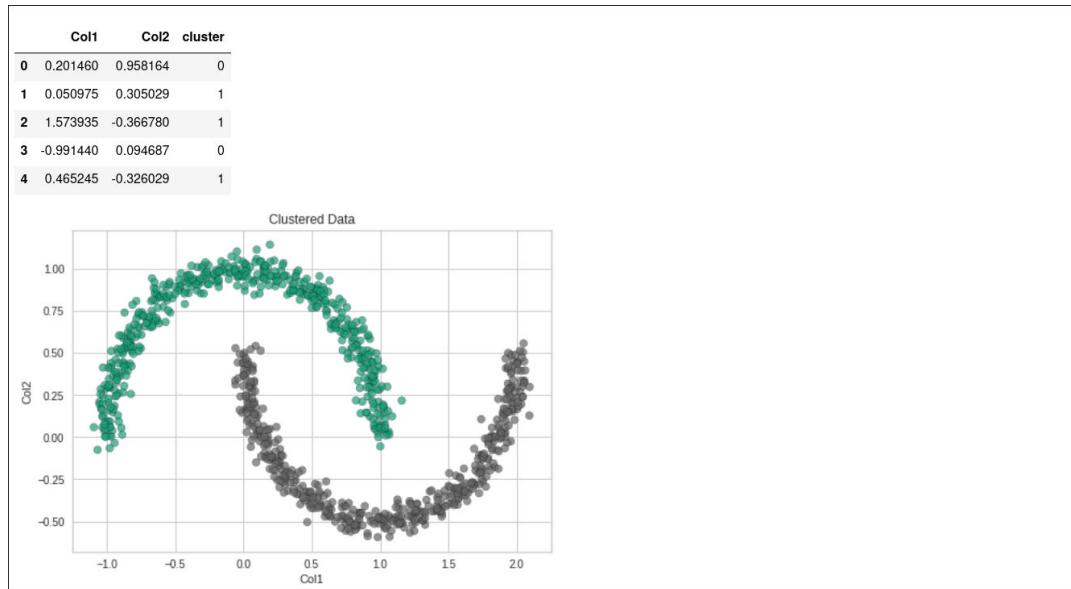


- Two clusters were generated using a k -means clustering model.
- The clusters don't seem to do a good job of supporting the separation between the crescents. Instead, the clusters seem to be defined by dividing the entire feature space in half.

8. Generate a hierarchical agglomerate clustering model and plot the results.

- Scroll down and view the cell titled **Generate a hierarchical agglomerate clustering model and plot the results**, and examine the code cell below it.
- Run the code cell.

- c) Examine the output.



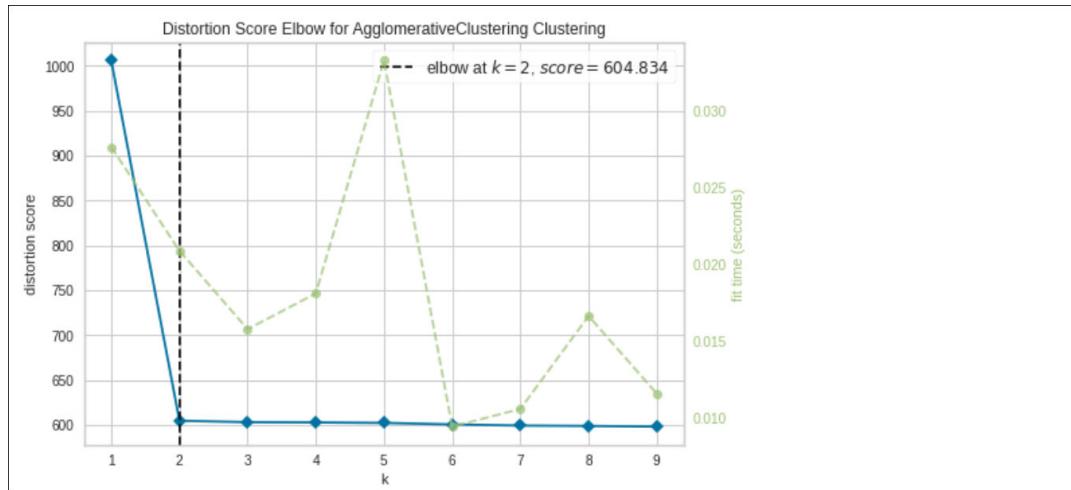
- Two clusters were generated using a hierarchical agglomerative clustering (HAC) model.
- The clusters seem to do a better job of supporting the separation between the crescents.

9. Use the elbow method to determine the optimal number of clusters.

- a) Scroll down and view the cell titled **Use the elbow method to determine the optimal number of clusters**, and examine the code cell below it.

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 gray cluster than they are to their own (green) center.

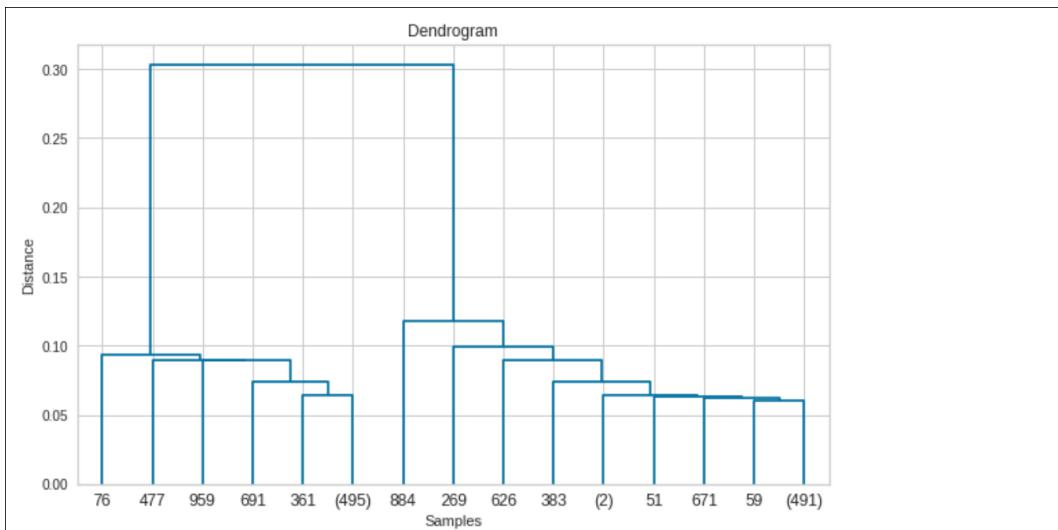
- b) Run the code cell.
c) Examine the output.



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.

10. 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 cell below it.
- 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.
 - On line 14, the `truncate_mode` will truncate the dendrogram based on the value provided by `p` (15), which determines how many examples are plotted.
- b) Run the code cell.
- 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 examine the next code cell.

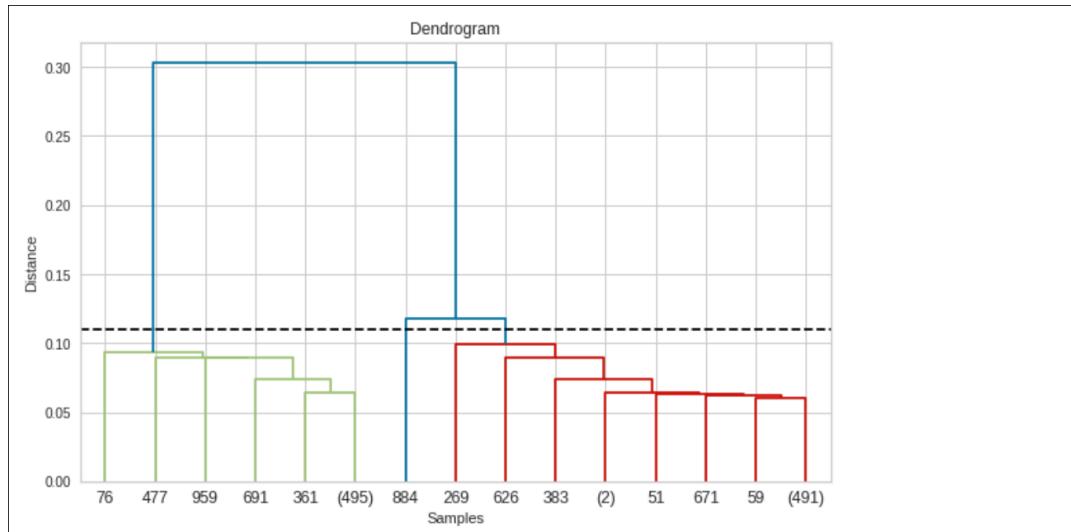
```

1 cutoff = 0.11
2
3 plot_dendro(cutoff)
4
5 # Plot cutoff line based on specified distance.
6 plt.axhline(y = cutoff, color = 'black', linestyle = '--');
```

A cutoff line will be 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.

- e) Run the code cell.

- f) Examine the output.



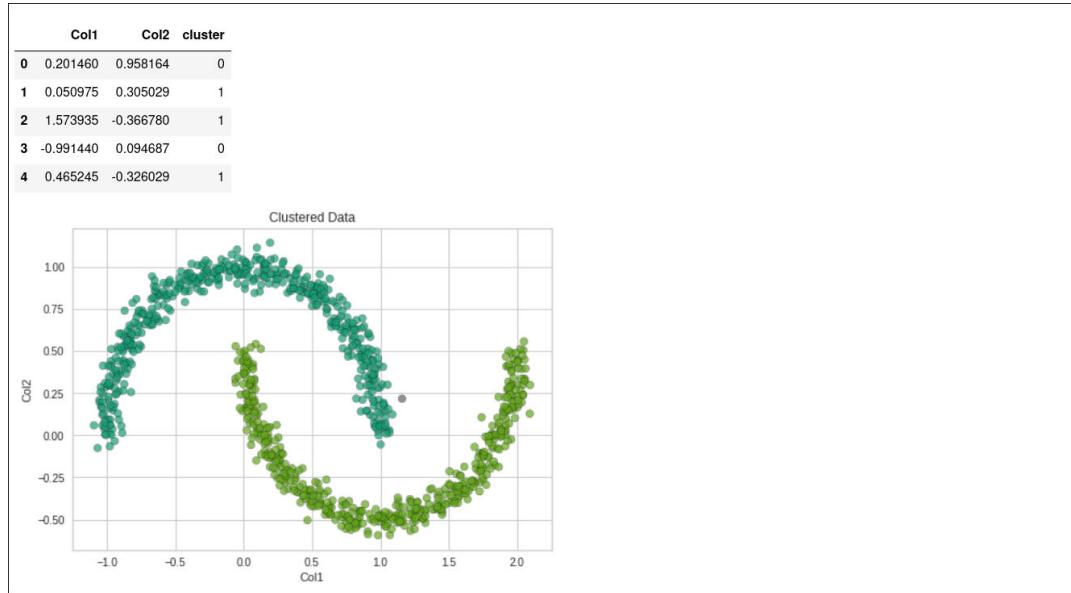
Since the cutoff passes through 3 branches, the number of clusters is 3. This differs from the elbow analysis, but, like with k -means clustering, there is not necessarily one "best" cluster number for a given agglomerative clustering model.

11. Generate one last HAC model, this time with 3 clusters.

- a) Scroll down and view the cell titled **Generate one last HAC model, this time with 3 clusters**, and examine the code cell below it.

Because the dendrogram cutoff suggests 3 clusters, it might be worth taking a look at this as an alternative model.

- b) Run the code cell.
c) Examine the output.



Very little has changed with the 3-cluster model compared to the 2-cluster model. In fact, the only visible example placed into a third cluster is the gray dot at the approximate coordinates (1.2, 0.24).



Note: The addition of a third cluster has changed how the colors are mapped to each cluster. However, the color is not relevant to the results of the model.

12. 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?

13. 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 **CAIP** in the file hierarchy.
-

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 supervised machine learning models from fundamental algorithms like linear regression, logistic regression, and k -nearest neighbor. 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 classification and regression models using decision trees.
- Build classification and regression models using random forests.

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 Trees

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. Consider the following figure in which a bank determines whether or not a customer qualifies for "preferred" status.

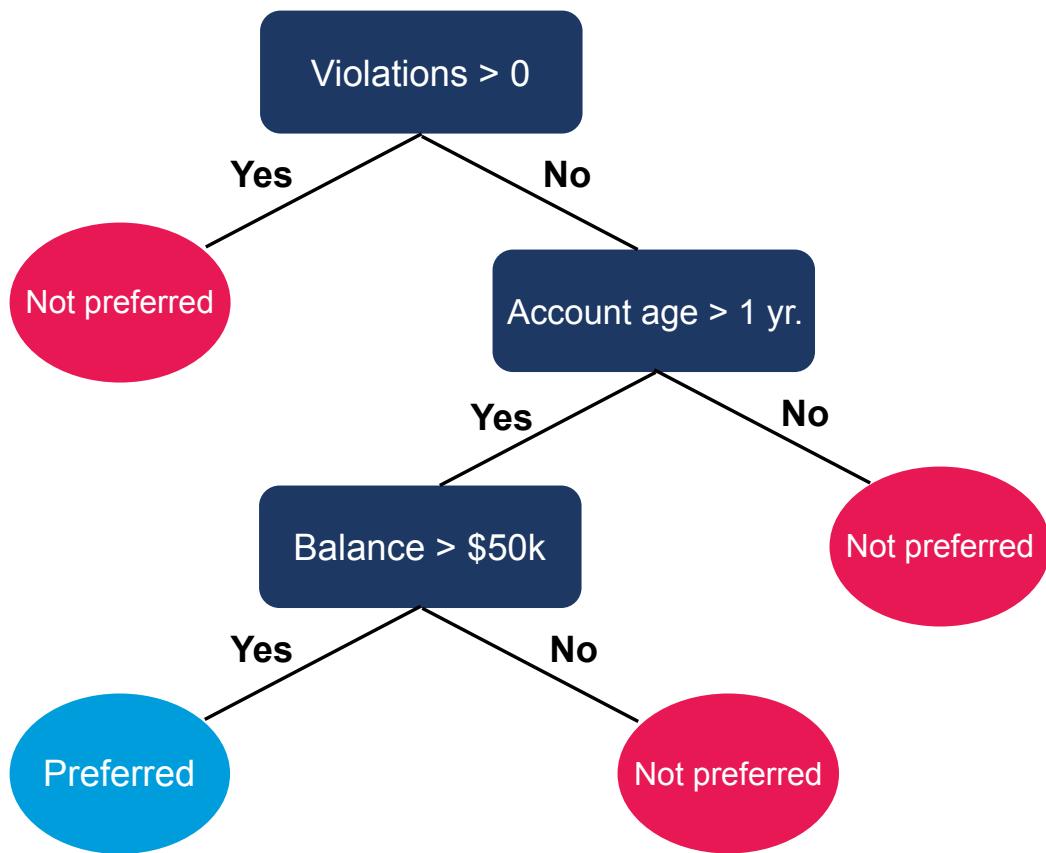


Figure 8-1: A decision tree determining the preferred status of customers.

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 Trees (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 one class, and none belong to the other 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

Account age > 1	Violations > 0	Balance > \$50,000	Preferred
No	No	Yes	No
Yes	No	Yes	Yes
Yes	No	No	No
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:

$$G = (5/8) \cdot 0.48 + (3/8) \cdot 0.44 = 0.465$$

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	0.465
Violations	0.375
Balance	0.430

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.

Classification Tree Example: Customer Preferred Status

The decision tree that was shown earlier was a simplified version. The following is what the tree might look like when the bank customer data is run through the CART algorithm.

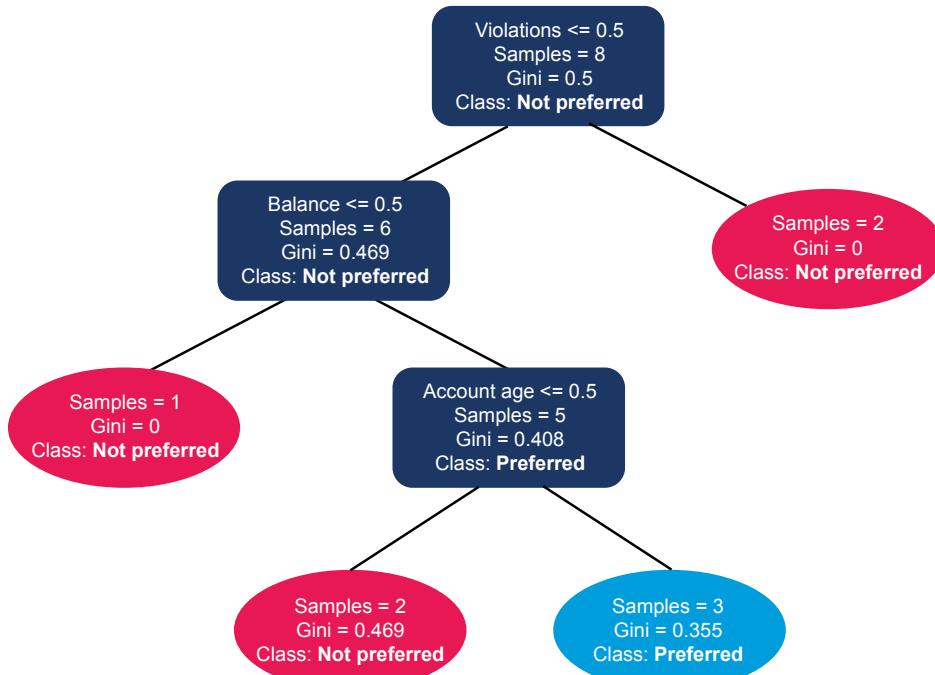


Figure 8-2: A CART model of the bank customer data.

The dataset was encoded so that "Yes" values are 1 and "No" values are 0. So, the root node that says `Violations <= 0.5` is referring to the truth value of the feature (i.e., was the customer's number of violations above 0?), not the actual number of violations. Since the truth value is either a 1 or 0, less than or equal to 0.5 means it's a 0. In other words, the root node splits based on if the customer did *not* have any violations. The node branches off to the left for customers in good standing and to the right for customers with at least one violation. Note that the root node also has a class determination. This isn't particularly meaningful since, at this point, it's just indicating that the majority of customers in the entire dataset do not have preferred status. These classifications will occur at each node, but ultimately, they are most useful at the terminating leaves.

Moving down the tree, the decision node on the left checks if the customer's balance is *not* over \$50,000. The Gini index for this node has also decreased from the Gini index at the root node. For customers who have less than the threshold balance, the terminating leaf determines that they are not a preferred customer. The Gini index at this leaf is 0, indicating a totally pure node. For customers who have more than the threshold balance, the tree continues to another decision node, this one concerning account age.

The remaining decisions on this side of the tree can be summed up like so:

- **Question 1:** Is a customer with no violations, who has an account balance greater than \$50,000, and whose account is *more than* a year old, a preferred customer?
 - **Answer:** Yes.
- **Question 2:** Is a customer with no violations, who has an account balance greater than \$50,000, and whose account is *less than* a year old, a preferred customer?
 - **Answer:** No.

Note that the Gini indices for the leaves that answer these questions are above 0, indicating that the tree was not able to reach total purity at this point. That means, at each leaf, there is a mix of "Yes" and "No" classifications. A tree does not need to reach total purity in order to be useful.

Lastly, if you move back to the right side of the tree after the root node, you can see that customers who have had one or more violations are automatically rejected for preferred status. That side of the tree simply terminates at a completely pure leaf.

Any new data samples that you put through the model will descend this tree and be evaluated at each relevant decision node.

Regression Tree Example: Vehicle Value

CART, if you recall, stands for classification and *regression* trees. So, as you'd expect, it can be used for regression tasks in order to make estimations about continuous numerical variables. The general structure of a decision tree regressor is similar to its classification counterpart, but one of the key differences is that regression trees use different splitting metrics. Reducing Gini impurity is not really suitable for continuous variables, so CART regressors must instead attempt to reduce the error at each decision node. There are different ways to define "error" based on the metric used. Common error metrics include the mean squared error (MSE) and mean absolute error (MAE).

The other key difference between decision tree classifiers and regressors is that regressors output a continuous numeric estimation rather than a class value. In the figure, you can see a CART regressor modeled from the vehicle value dataset.

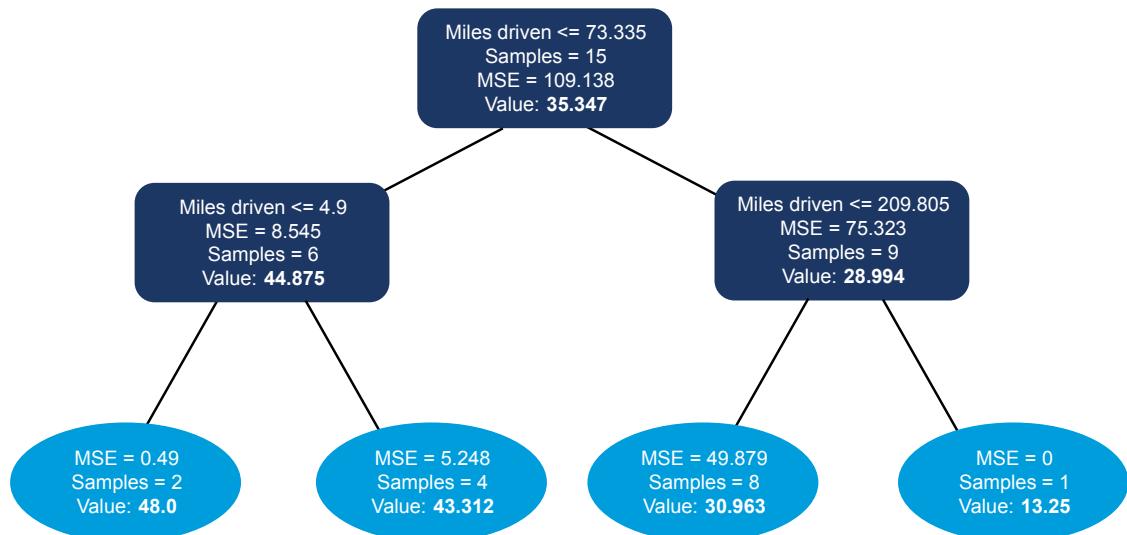


Figure 8-3: A simplified decision tree regressor.

Unlike linear regression, regression using CART does not make assumptions about the relationship between dependent and independent variables. So, CART regressors are the more common choice when linear regression is too restrictive to the problem at hand. As a result, they usually require much more training data to learn from in order to be effective.

CART Hyperparameters

The following are some of the common hyperparameters associated with CART.

Hyperparameter	Description
max_depth	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 decision nodes are less than the value of min_samples_split.
	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 max_depth 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.
min_samples_split	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 underfit it.
min_samples_leaf	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.
splitter	The default value for this parameter is 'best', which calculates the Gini index as described previously. However, if you specify splitter = 'random', 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.



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>
Cost-complexity pruning	<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> <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 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.

Decision Trees and Continuous Variables

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.

Therefore, it's usually a good idea to bin your continuous variables if you plan to use them in decision trees. You'd typically do this as part of the feature engineering process before developing the decision tree, but discretization can also be done by algorithms like C4.5 during tree generation.



Note: Categorical variables, including those that are created from the binning process, may work better with decision trees if they are encoded (e.g., using one-hot encoding). However, keep in mind that a significant amount of encoding can be computationally expensive.

Decision Tree Algorithm Comparison

The following table compares CART with C4.5 and how they are used in building decision trees.

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

Algorithm	Splitting Metric	Pruning Method	Supports Classification and Regression?	Supports Multi-Class Splitting?	Supports Missing Values?
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 support-vector machines (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 Decision Tree Models

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 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.plot_tree(model, feature_names = X_train.columns.values.tolist(), class_names = ['0', '1'], filled = True)` —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/Decision Trees and Random Forests - Titanic.ipynb

/home/student/CAIP/Decision Trees and Random Forests/titanic_data/titanic_train_data.pickle

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/Decision Trees and Random Forests - 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 cell below it.
 - b) Run the code cell.
 - c) Verify that **titanic_train_data.pickle** was loaded with 891 records.
This is the same labeled *Titanic* data you worked with earlier, all prepared and ready to use for training.
3. Preview the current training data.
 - a) Scroll down and view the cell titled **Preview the current training data**, and examine the code cell below it.
 - b) Run the code cell.

- c) Examine the output.

	Survived	Pclass	Sex_male	Sex_female	Age	SibSp	Parch	Fare	Embarked_S	Embarked_C	Embarked_Q	SizeOfFamil
0	0	3	1	0	22.0	1	0	7.2500	1	0	0	
1	1	1	0	1	38.0	1	0	71.2833	0	1	0	
2	1	3	0	1	26.0	0	0	7.9250	1	0	0	
3	1	1	0	1	35.0	1	0	53.1000	1	0	0	
4	0	3	1	0	35.0	0	0	8.0500	1	0	0	

Most of the data is already in a good state for training. However, the continuous features `Age` and `Fare` can cause the decision tree to split too many times, increasing its complexity and potentially leading to overfitting. So, you'll discretize these two features before you proceed.

4. Bin and encode Age and Fare.

- Scroll down and view the cell titled **Bin and encode Age and Fare**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
Youngest passenger: 0.42
Oldest passenger: 80.0
-----
Lowest fare: 0.0
Highest fare: 512.3292
```

Seeing the highest and lowest values for both features can help you determine how to bin them.

- Scroll down and examine the next code cell.

```
1 # Define Age bins and labels.
2 age_bins = [0, 20, 40, 60, 80, 100]
3 age_labels = ['0-20', '21-40', '41-60', '61-80', '81-100']
4
5 age_group = pd.cut(df['Age'], bins = age_bins,
6                      labels = age_labels, right = False)
7 df.insert(5, 'Age_grp', age_group)
8
9 age_encoded = df['Age_grp'].cat.codes
10 df.insert(6, 'Age_code', age_encoded)
11
12 # Define Fare bins and labels.
13 fare_bins = [0, 100, 200, 300, 400, 500, 600]
14 fare_labels = ['0-100', '101-200', '201-300',
15                 '301-400', '401-500', '501-600']
16
17 fare_group = pd.cut(df['Fare'], bins = fare_bins,
18                      labels = fare_labels, right = False)
19 df.insert(10, 'Fare_grp', fare_group)
20
21 fare_encoded = df['Fare_grp'].cat.codes
22 df.insert(11, 'Fare_code', fare_encoded)
```

- Lines 2 through 10 bin and then label encode the `Age` feature. There are 5 bin labels, each 20 years apart.
- Lines 13 through 22 bin and then label encode the `Fare` feature. There are 6 bin labels, each £100 (pounds sterling) apart.

- Run the code cell.

- f) Scroll down and examine the next code cell.

```
1 df.head()
```

- g) Run the code cell.
h) Examine the output.

	Survived	Pclass	Sex_male	Sex_female	Age	Age_grp	Age_code	SibSp	Parch	Fare	...	Fare_code	Embarked_S	Em
0	0	3	1	0	22.0	21–40	1	1	0	7.2500	...	0	1	
1	1	1	0	1	38.0	21–40	1	1	0	71.2833	...	0	0	
2	1	3	0	1	26.0	21–40	1	0	0	7.9250	...	0	1	
3	1	1	0	1	35.0	21–40	1	1	0	53.1000	...	0	1	
4	0	3	1	0	35.0	21–40	1	0	0	8.0500	...	0	1	

5 rows × 21 columns

The encoded bins appear for both `Age` and `Fare`.

5. Split the datasets.

- a) Scroll down and view the cell titled **Split the datasets**, and examine the code cell below it.

The `SizeOfFamily` feature (the combination of `SibSp` and `Parch`) is not being included in the training features. A decision tree may be able to generate useful splits from each feature individually.

- b) Run the code cell.
c) Examine the output.

```
Original set:      (891, 21)
-----
Training features: (668, 15)
Testing features: (223, 15)
Training labels:  (668, 1)
Testing labels:   (223, 1)
```

6. Create a basic decision tree model.

- a) Scroll down and view the cell titled **Create a basic decision tree model**, and examine the code cell below it.

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.

- b) Run the code cell.
c) Examine the output.

```
Decision tree model took 2.10 milliseconds to fit.
Accuracy: 83%
```

Like with other classification methods, you can evaluate the model using the familiar metrics. The default model has an accuracy of 83%.

7. Visualize the decision tree structure.

- a) Scroll down and view the cell titled **Visualize the decision tree structure**, and examine the code cell below it.

This function, when called, will use scikit-learn's built-in `plot_tree()` function to visually generate the structure of the decision tree.

- b) Run the code cell.
c) Examine the output.

A function to plot the decision tree structure has been defined.

8. 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 cell below it.

This function, when called, computes the four named statistical measures.

- b) Run the code cell.
c) Examine the output.

The function to compute scores has been defined.

9. 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 cell below it.

This function, when called, generates a ROC curve.

- b) Run the code cell.
c) Examine the output.

The function to generate a ROC curve and compute AUC has been defined.

10. 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 cell below it.

This function, when called, generates a precision–recall curve.

- b) Run the code cell.
c) Examine the output.

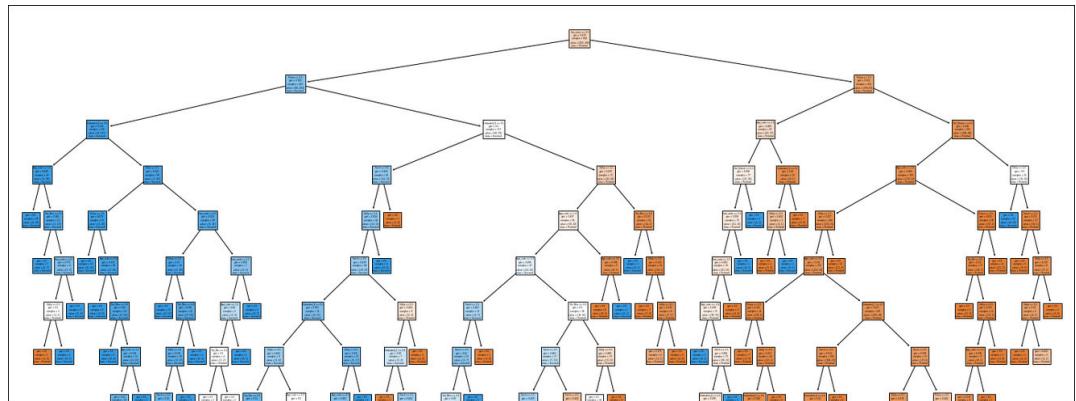
The function to generate a precision–recall curve and compute average precision has been defined.

11. Evaluate the initial decision tree model.

- a) Scroll down and view the cell titled **Evaluate the initial decision tree model**, and examine the code cell below it.

- b) Run the code cell.

- 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 examine the next code cell.

```
1 model_scores(y_test, initial_preds)
```

- e) Run the code cell.
f) Examine the output.

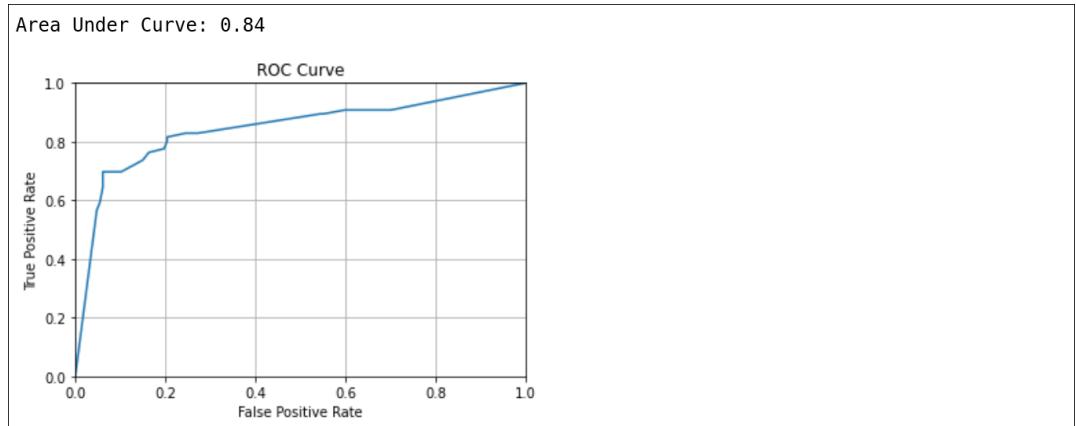
```
Accuracy: 83%
Precision: 78%
Recall: 70%
F1: 74%
```

There's room to improve these scores, which you'll do shortly.

- g) Scroll down and examine the next code cell.

```
1 initial_pred_proba = first_tree.predict_proba(X_test)
2
3 roc(y_test, initial_pred_proba[:, 1])
```

- h) Run the code cell.
i) Examine the output.

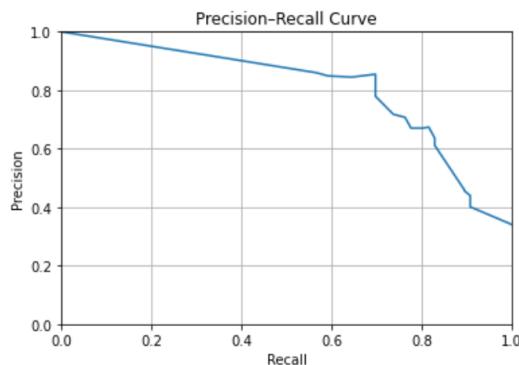


- j) Scroll down and examine the next code cell.

```
1 prc(y_test, initial_pred_proba[:, 1])
```

- k) Run the code cell.
l) Examine the output.

Average Precision: 0.76



Both the AUC and the average precision can likely be improved as well.

12. 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 cell below it.

The only difference here is that the decision tree is being constrained to a maximum depth of 3. 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) Run the code cell.
c) Examine the output.

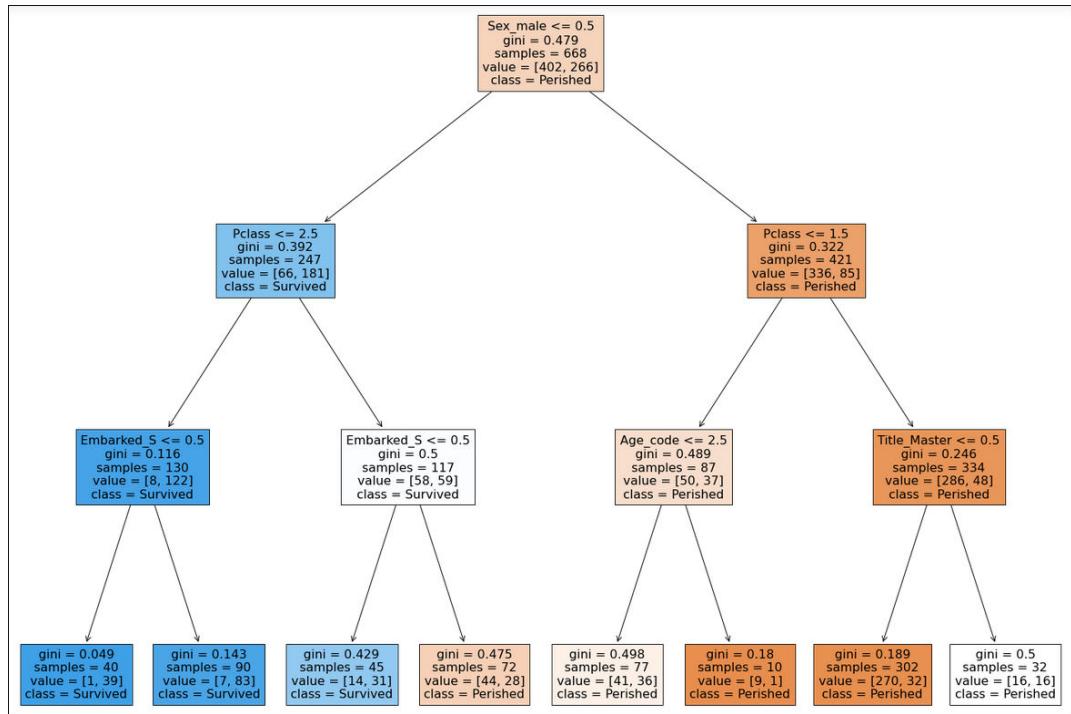
```
Decision tree model took 10.02 milliseconds to fit.
Accuracy: 85%
```

The accuracy has increased slightly.

13. Evaluate the pruned decision tree model.

- a) Scroll down and view the cell titled **Evaluate the pruned decision tree model**, and examine the code cell below it.
b) Run the code cell.

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* male (in other words, female).
- 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 the split of each class prediction. In other words, 402 passengers perished (0), and 266 passengers survived (1).
- The overall class prediction at this point is perished, since that is the most common (402 passengers). This prediction will get much more useful as you descend the tree.

For the first "True" branch on the left:

- The decision node here is evaluating if the passenger was in first or second class.
- The Gini index is 0.392, which, as expected, is becoming more pure.
- The number of samples is 247.
- 66 passengers are predicted to perish at this point; 181 are predicted to survive.
- The class prediction here is survived.

For the first "False" branch on the right:

- The decision node here is evaluating if the passenger was in first class.
- The Gini index is 0.322.
- The number of samples is 421.
- 336 passengers are predicted to perish at this point; 85 are predicted to survive.
- The class prediction here is perished.

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 some of these terminating leaves are low, but none of them are totally pure (0). This is because you specified a maximum depth as an early stopping criterion.

- d) Scroll down and examine the next code cell.

```
1 model_scores(y_test, pruned_preds)
```

- e) Run the code cell.
f) Examine the output.

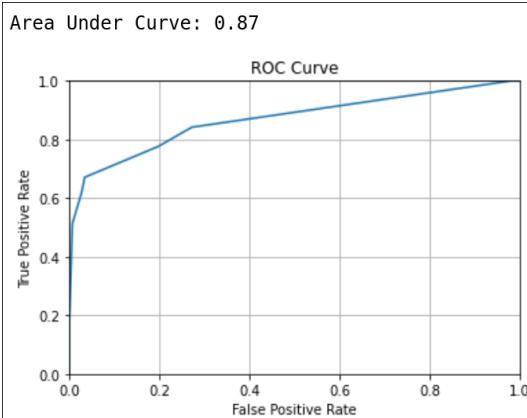
```
Accuracy: 85%
Precision: 92%
Recall: 62%
F1: 74%
```

The recall seems to have decreased by several percent, whereas precision has increased by a lot, and accuracy and F_1 score have increased by a few.

- g) Scroll down and examine the next code cell.

```
1 pruned_pred_proba = pruned_tree.predict_proba(X_test)
2 roc(y_test, pruned_pred_proba[:, 1])
```

- h) Run the code cell.
i) Examine the output.

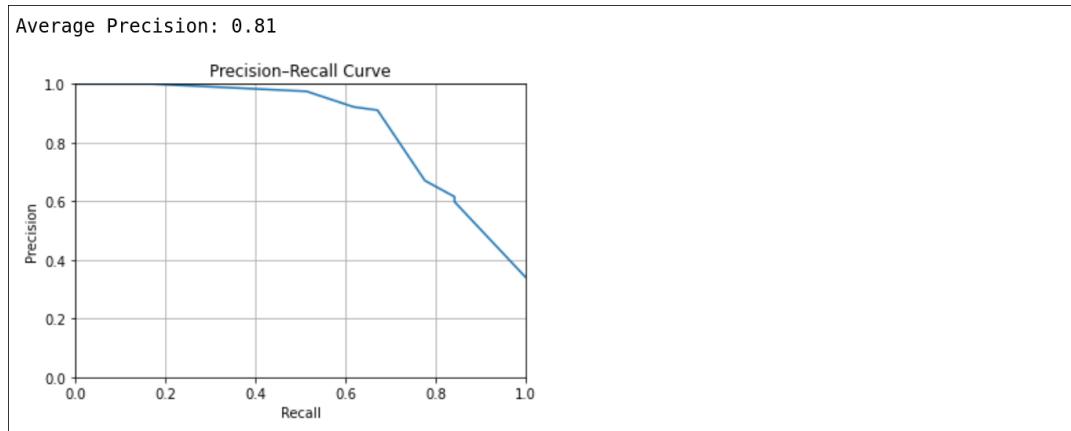


- j) Scroll down and examine the next code cell.

```
1 prc(y_test, pruned_pred_proba[:, 1])
```

- k) Run the code cell.

- I) Examine the output.



Both the AUC and the average precision have improved by a moderate amount.

14. Compare the results to a model that used a different algorithm.

- a) Compare the pruned decision tree model with the results of the best logistic regression model you created earlier.



Note: "Best," in this case, refers to the best model you created—not the best possible model.

For reference, here are the results:

Model	Accuracy	Precision	Recall	F_1	ROC AUC	Average Precision
Best logistic regression	85%	78%	78%	78%	87%	85%
Best decision tree	85%	92%	62%	74%	87%	81%

- b) Use the results table to answer the following question.

15. 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?

16. Keep this notebook open.

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 estimations of multiple models are considered in combination. The purpose is to obtain a more skillful estimation 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 estimations from multiple models, especially a diverse set of models, tends to lead to better estimations.

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 determined by the most models. For example, the following models determine 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 classification 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. 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 estimators between all decision trees (regression).

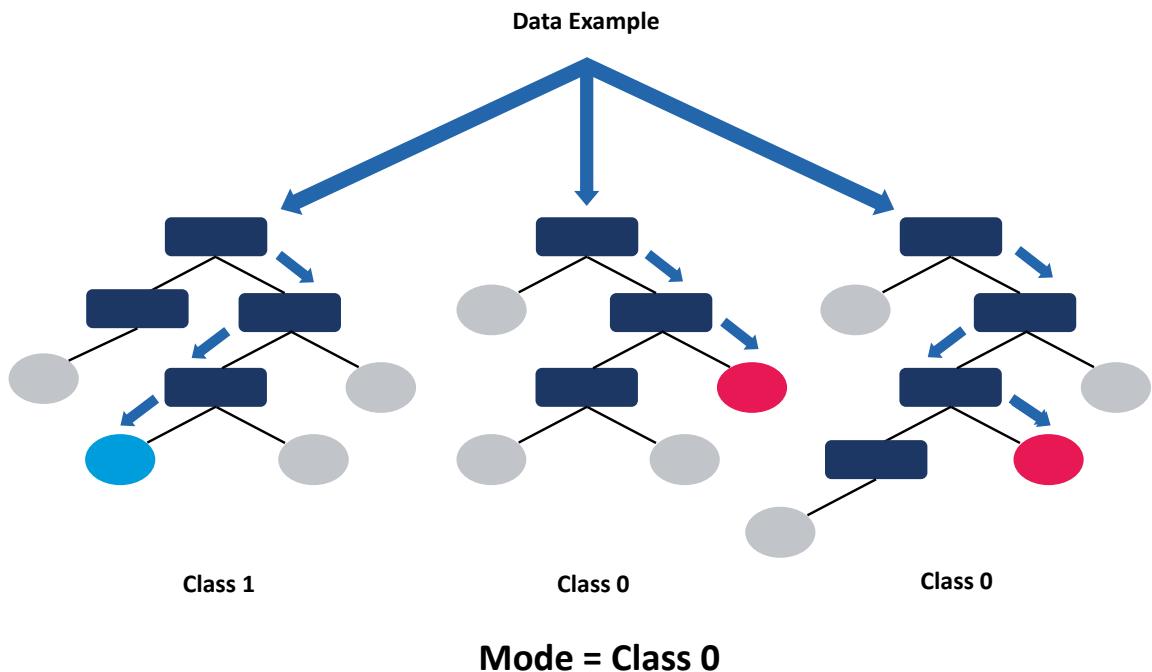


Figure 8–4: A random forest selecting a classifier based on the mode of its constituent trees.
Note that light blue ovals indicate class 1, whereas red ovals indicate class 0.

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.



Note: Random forests, like other tree-based algorithms, don't require features to be scaled for training.

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 determines 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 estimations to correct majority vote estimations, when considering all out-of-bag data examples. For instance, an out-of-bag error of 0.19 indicates that 19% of the estimated classifications 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 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%

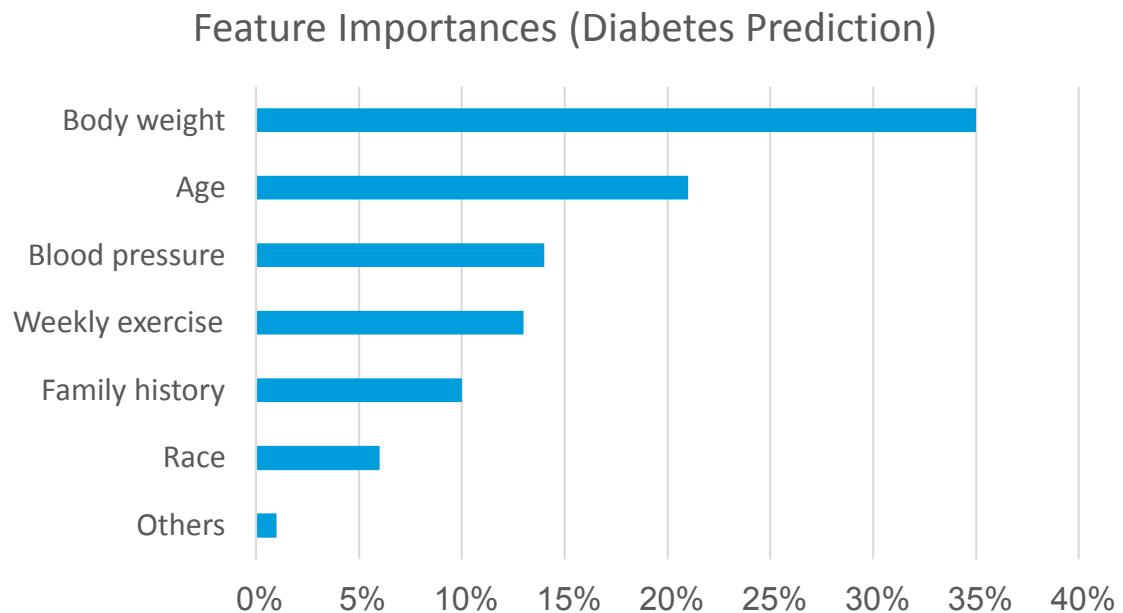


Figure 8-5: Graphing feature importances on a bar chart.

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.

Gradient Boosting

Another ensemble method that incorporates multiple decision trees is called gradient boosting. **Gradient boosting** algorithms build models in stages, where past decision trees influence how later decision trees are built. This differentiates them from random forests, where each tree is built independently and then randomly sampled, concluding in the majority vote. With gradient boosting, the trees are built in an additive fashion, where each tree attempts to correct the errors of the tree before it. That means its conclusions are developed iteratively rather than determined all at once. The algorithm starts by generating decision trees that have low classification skill (called weak learners) and then combines those weak learners into an eventual decision tree with high skill (called a strong learner).

Gradient boosting applied to classification problems starts by generating a base probability for the initial decision tree. Then, it calculates the residuals (errors) between actual classes and estimated classes. When the next tree is built, it attempts to estimate these residuals while also generating new probabilities. This process repeats until the residuals have been reduced to a value close to zero, or until the maximum number of trees are produced (according to a hyperparameter). For regression tasks, gradient boosting does not calculate residuals based on probabilities, but rather calculates them from direct numeric estimations.

Gradient boosting models can lead to better performance as compared to random forests, particularly for unbalanced datasets. However, they are highly susceptible to overfitting in cases where the data is noisy, and they can also take longer to train than random forests since the trees are built iteratively instead of independently. Gradient boosting models are also difficult to tune. In terms of real-world applications, gradient boosting has found success in tasks like page ranking for search engines, real-time risk assessment, and scientific data analysis.

XGBoost

One of the most popular open-source libraries that implements gradient boosting is XGBoost. It is available not only for Python®, but for other languages such as R, C++, Java, and more. The scikit-

learn package has its own gradient boosting methods, but XGBoost tends to be faster and can lead to more skillful models. XGBoost is also adept at handling missing values without requiring them to be imputed.

Guidelines for Building Random Forest Models

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 estimative 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.
- Consider experimenting with gradient boosting as an alternative ensemble method to random forests.

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 estimations 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

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/Decision Trees and Random Forests/Decision Trees and Random Forests - Titanic.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 **Create a random forest model**. Select **Cell→Run All Above**.

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. Why is cross-validation typically not necessary when training a random forest model?

2. Create a random forest model.

- Scroll down and view the cell titled **Create a random forest model**, and examine the code cell below it.

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 `max_depth` hyperparameter is set to 4, and the other parameters that aren't being explicitly defined are at the same defaults (e.g., `min_samples_split` and `min_samples_leaf`). 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. This is also the default in the most recent versions of scikit-learn.
- `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.

- Run the code cell.
- Examine the output.

```
Random forest model took 197.57 milliseconds to fit.
Accuracy: 84%
```

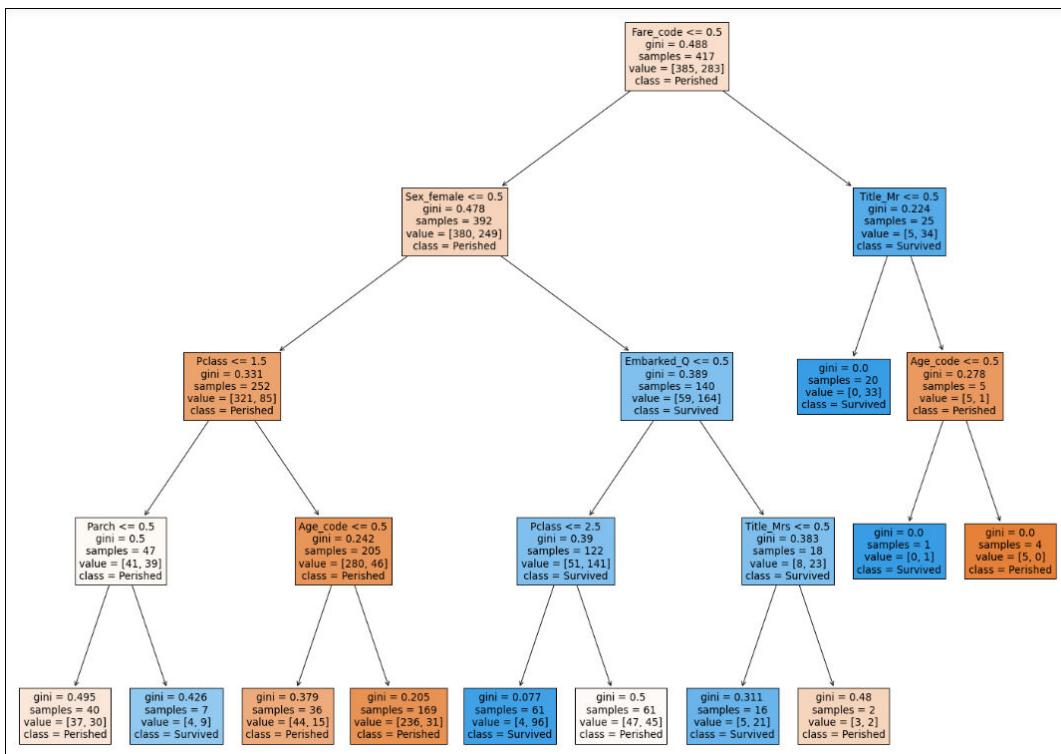
This initial random forest model has an accuracy of 84%, which is slightly lower than your best lone decision tree. You'll evaluate the model using other metrics shortly.

3. Examine a couple of trees in the forest.

- a) Scroll down and view the cell titled **Examine a couple of trees in the forest**, and examine the code cell below it.

As before, this function call 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. The `estimators_` attribute refers to each tree in the forest, so the tree at index 0 will be shown.

- b) Run the code cell.
c) Examine the output.



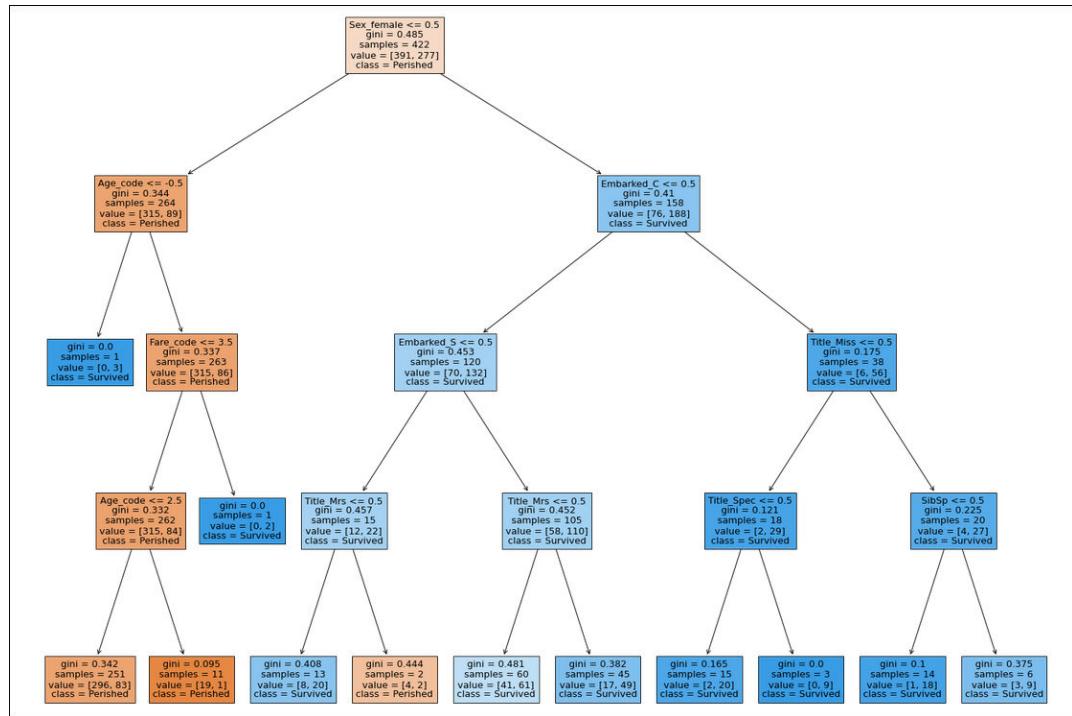
- d) Scroll down and examine the next code cell.

```
1 | plot_tree(forest.estimators_[1])
```

This code will show the tree at index 1.

- e) Run the code cell.

f) Examine the output.



4. Compare the two trees.

How do the first two branches of each tree differ in terms of their splitting criteria?

5. Evaluate the random forest model.

- a) Scroll down and view the cell titled **Evaluate the random forest model**, and examine the code cell below it.
 - b) Run the code cell.
 - c) Examine the output.

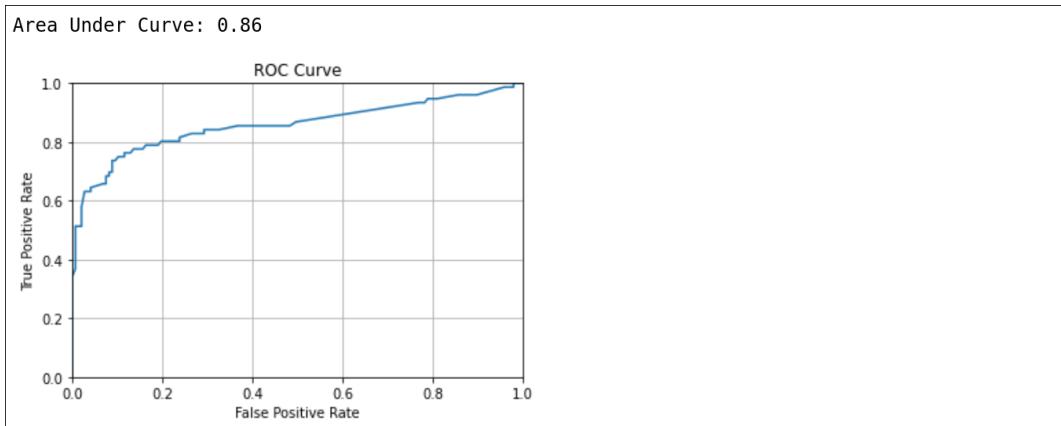
Accuracy: 84%
Precision: 80%
Recall: 70%
F1: 75%

d) Scroll down and examine the next code cell.

```
1 forest_pred_proba = forest.predict_proba(X_test)
2
3 roc(y_test, forest_pred_proba[:, 1])
```

e) Run the code cell.

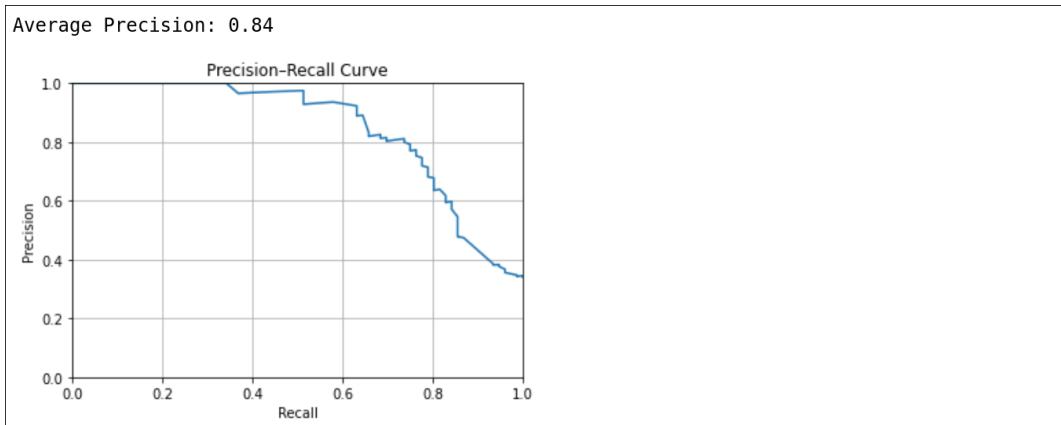
- f) Examine the output.



- g) Scroll down and examine the next code cell.

```
1 | prc(y_test, forest_pred_proba[:, 1])
```

- h) Run the code cell.
i) Examine the output.



- j) Compare the random forest model with the results of the best lone decision tree model you created earlier.

For reference, here are the results:

Model	Accuracy	Precision	Recall	F_1	ROC AUC	Average Precision
Best lone decision tree	85%	92%	62%	74%	87%	81%
Random forest	84%	80%	70%	75%	86%	84%

Some of these metrics changed slightly (e.g., accuracy and F_1), whereas others changed 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 grid or 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.

6. What advantage does a random forest have over a single decision tree?

7. Generate the out-of-bag error and decision function.

- Scroll down and view the cell titled **Generate the out-of-bag error and decision function**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
Out-of-bag error: 0.189
```

In this case, around 19% of the predictions were incorrect.

- Scroll down and examine the next code cell.

```
1 forest.oob_decision_function_
```

- Run the code cell.
- Examine the output.

```
array([[0.843856, 0.156144],
       [0.3221827, 0.6778173],
       [0.83743877, 0.16256123],
       ...,
       [0.79151481, 0.20848519],
       [0.33316286, 0.66683714],
       [0.19295393, 0.80704607]])
```

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.

8. Verify that the random forest took the majority vote of all trees in the forest.

- Scroll down and view the cell titled **Verify that the random forest took the majority vote of all trees in the forest**, and examine the code cell below it.
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.
- Run the code cell.
- Examine the output.

```
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]
```

As you can see, manually taking this vote gives you the same results as automatically calling the forest's `predict()` method.

9. Identify important features.

- Scroll down and view the cell titled **Identify important features**, and examine the code cell below it.
On line 1, 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.
- Run the code cell.

c) Examine the output.

```
Title_Mr      0.225546
Sex_female   0.202298
Sex_male     0.164754
Pclass       0.139245
Title_Miss   0.059383
SibSp        0.047429
Title_Mrs    0.037961
Parch        0.028922
Embarked_C   0.021483
Title_Master 0.017712
Embarked_S   0.016527
Age_code     0.015216
Fare_code    0.013519
Title_Spec   0.005438
Embarked_Q   0.004567
dtype: float64
```

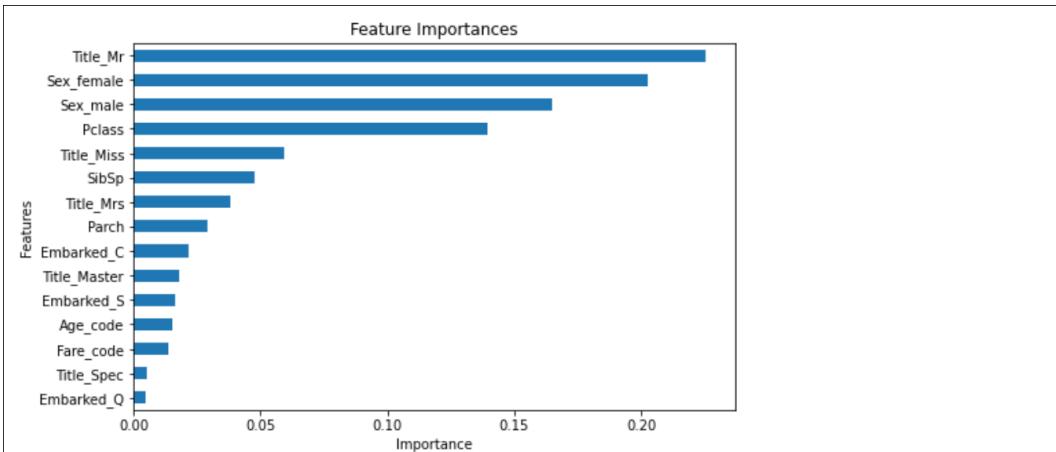
- It looks like whether or not a passenger had the title "Mr." contributed the most to the predictions.
- The sex of the passenger, particularly if they were female, also contributed a good deal to the predictions. This makes sense, as the title "Mr." is essentially a proxy for a person's sex/gender.
- The class of the passenger also seemed to have some importance.
- Analyzing this data visually might be more helpful.

d) Scroll down and examine the next code cell.

```
1 plt.figure(figsize = (8, 5))
2
3 # Reverse series so.plot is in descending order.
4 feat_importances.iloc[::-1].plot(kind = 'barh')
5
6 plt.xlabel('Importance')
7 plt.ylabel('Features')
8 plt.title('Feature Importances');
```

e) Run the code cell.

f) 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.

- Title_Mr, Sex_female, Sex_male, and Pclass all seem relatively important, so you'll keep them.
- Most of the rest of the features may still have some importance, though perhaps dropping Title_Spec and Embarked_Q won't impact the model all that much. The model may even do a better job of avoiding overfitting if those features are removed.

10. Drop the unimportant features for a new round of training.

- Scroll down and view the cell titled **Drop the unimportant features for a new round of training**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

	Pclass	Sex_male	Sex_female	Age_code	SibSp	Parch	Fare_code	Embarked_S	Embarked_C	Title_Mr	Title_Mrs	Title_
439	2	1	0	1	0	0	0	1	0	1	1	0
617	3	0	1	1	1	0	0	1	0	0	0	1
242	2	1	0	1	0	0	0	1	0	1	0	0
82	3	0	1	1	0	0	0	0	0	0	0	0
398	2	1	0	1	0	0	0	1	0	0	0	0

The training and test sets now only include the features that were identified as having some importance, reducing the datasets' dimensionality.

11. Train a new random forest model on the dataset with reduced dimensionality.

- Scroll down and view the cell titled **Train a new random forest model on the dataset with reduced dimensionality**, and examine the code cell below it.
This is the same training code as before, but now it's using the newly reduced datasets.
- Run the code cell.
- Examine the output.

```
Random forest model took 240.03 milliseconds to fit.
Accuracy: 85%
```

The model's accuracy has increased slightly.

12. Evaluate the new random forest model.

- Scroll down and view the cell titled **Evaluate the new random forest model**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
Accuracy: 85%
Precision: 81%
Recall: 74%
F1: 77%
```

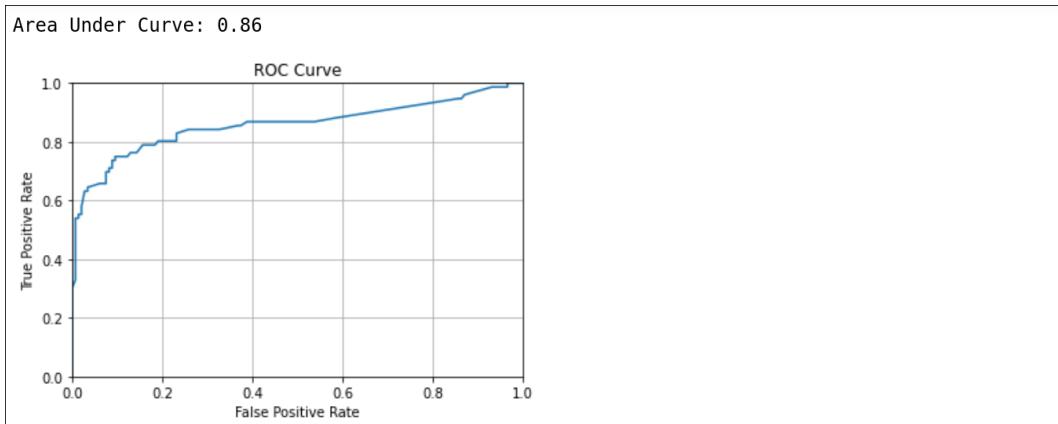
Compared to the previous model, the new model's accuracy, precision, and F_1 score have improved slightly.

- Scroll down and examine the next code cell.

```
1 pred_proba = forest.predict_proba(X_test_reduce)
2
3 roc(y_test, pred_proba[:, 1])
```

- Run the code cell.

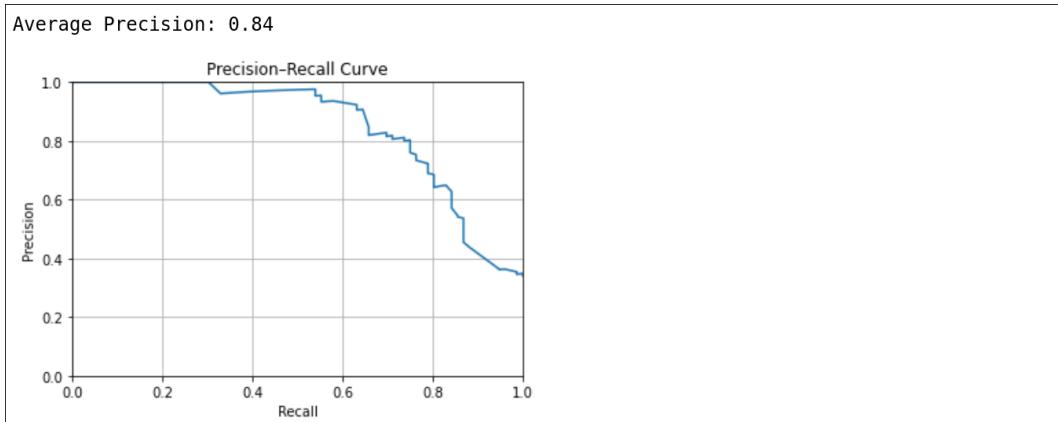
- f) Examine the output.



- g) Scroll down and examine the next code cell.

```
1 | prc(y_test, pred_proba[:, 1])
```

- h) Run the code cell.
i) Examine the output.



The ROC AUC and average precision both stayed the same.

- j) Scroll down and examine the next code cell.

```
1 | oob_error = 1 - forest.oob_score_
2 | print('Out-of-bag error: {}'.format(round(oob_error, 3)))
```

- k) Run the code cell.
l) Examine the output.

```
Out-of-bag error: 0.181
```

The out-of-bag error slightly improved on this reduced model.

13. 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?

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 **Decision Trees and Random Forests - Titanic** tab in Firefox, but keep a tab open to **CAIP** in the file hierarchy.
-

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.

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 classification models using support-vector machines (SVMs).
- Build regression models using support-vector machines (SVMs).

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 called a support–vector margin. The functions on either edge of these margins are called the support vectors.

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.

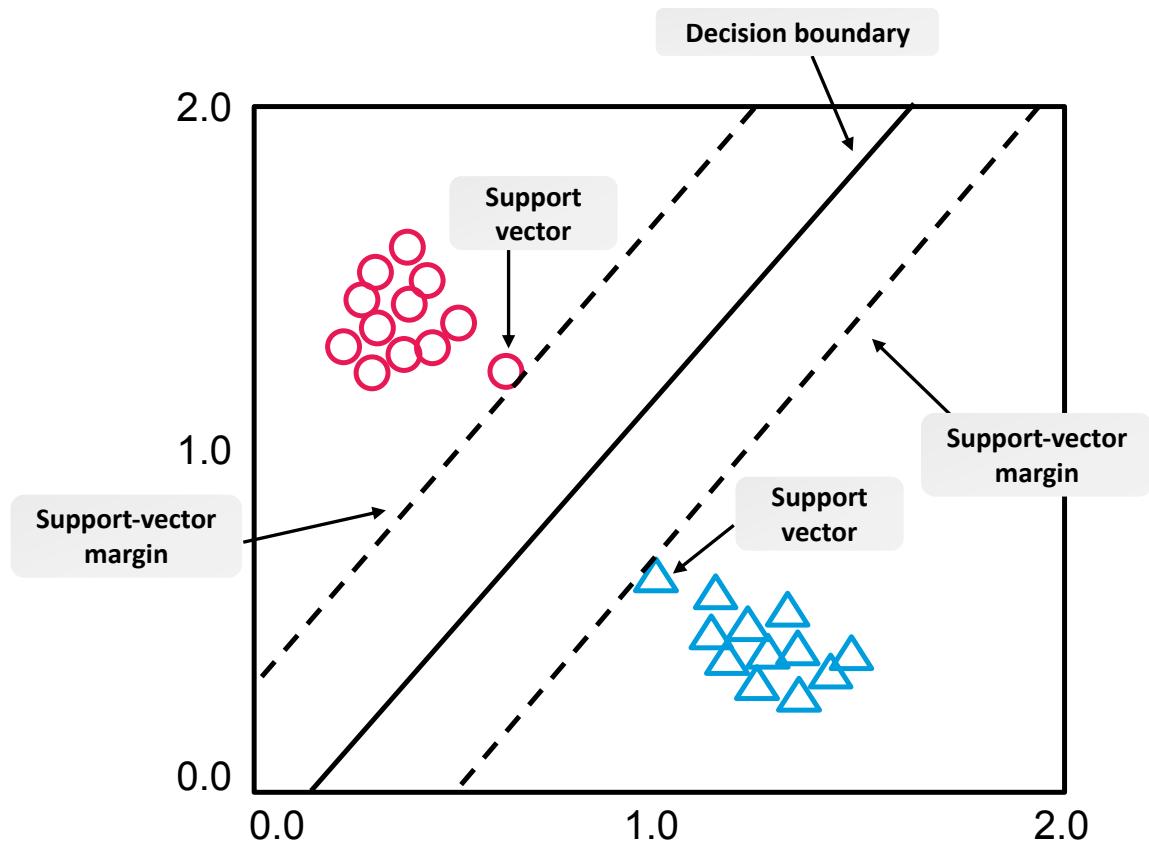


Figure 9–1: An example of a hyperplane used in SVMs.

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.

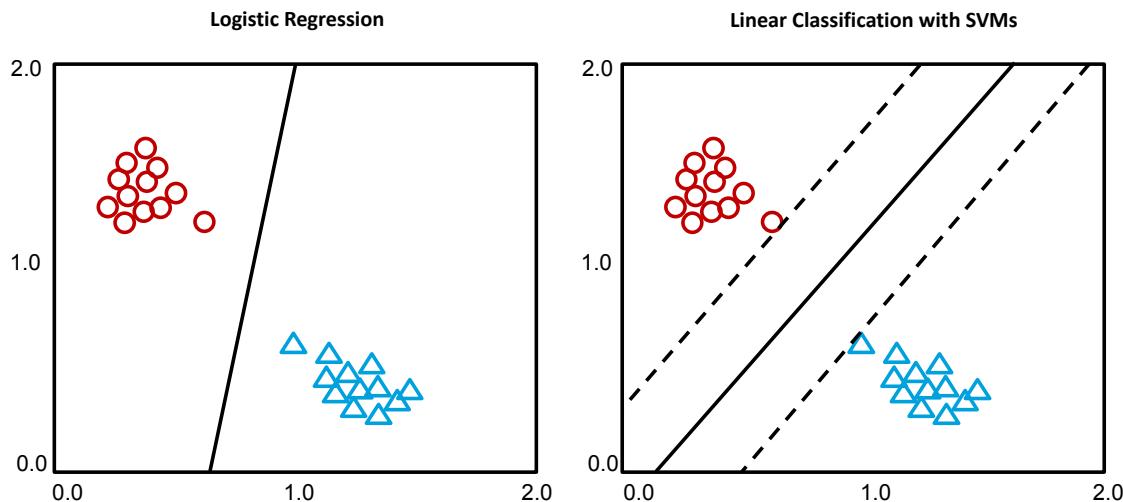


Figure 9-2: A linear classifier without SVMs (left) compared to a linear classifier with SVMs (right).

Now, observe what happens when a new test example is evaluated on the model.

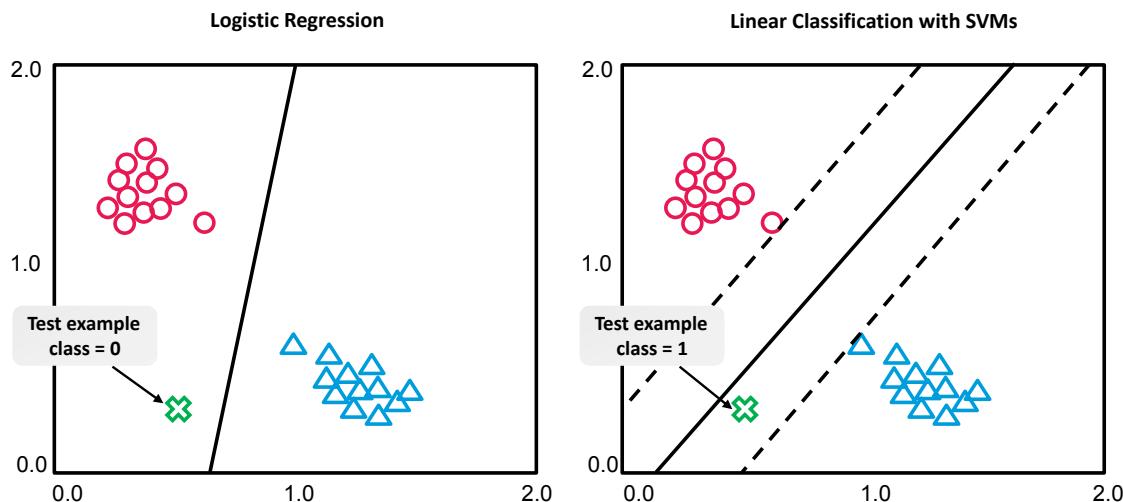


Figure 9-3: Classifying a test example for a model without SVMs (left) and a model with SVMs (right).

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.

However, when you use SVMs, it's very important to scale your data—more so than other classification algorithms. If one feature has much larger values than another, it will have an undue influence on the distances that are calculated between the support vectors and the decision

boundary. The distribution of the data points is more important in this case than the range of each feature.

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.

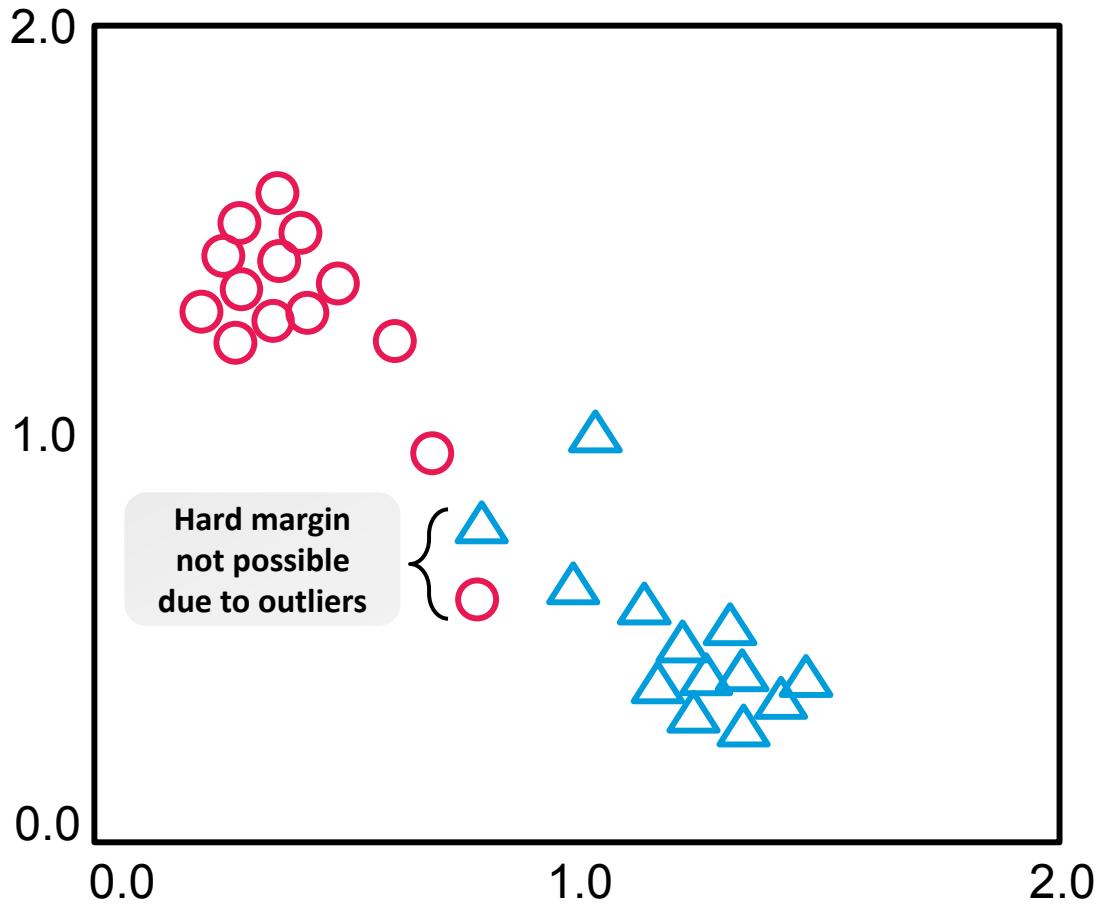


Figure 9–4: Extreme outliers in the training may lead to hard-margin classification failing.

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 to new test data.

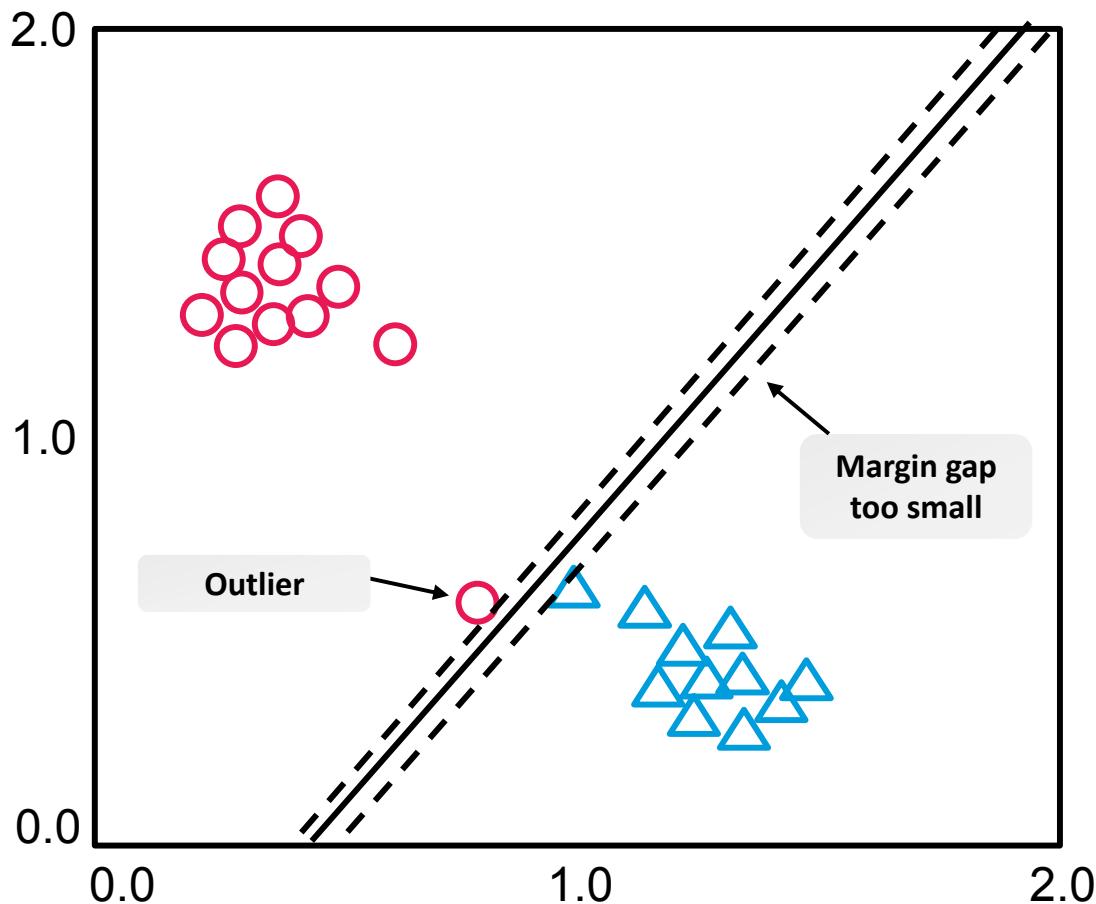


Figure 9–5: A single outlier leading to a poorly fit hyperplane.

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. 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. It helps the model avoid overfitting to the training data.

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). So, the difference between hard and soft margins is not an absolute; instead, there is a degree of "softness" or "hardness" that you, the practitioner, specify before training.

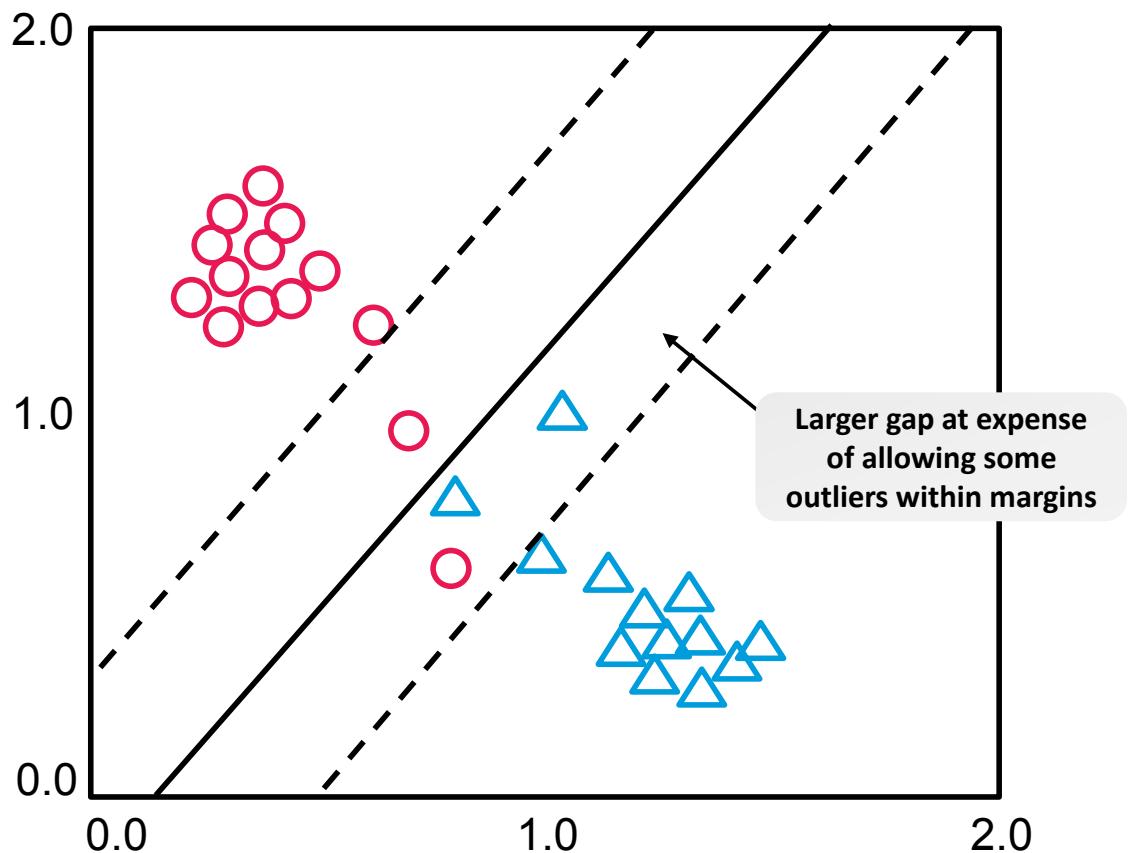


Figure 9–6: A hyperplane plotted using soft-margin classification. The margin gap remains wide at the expense of keeping some outliers within the margins.

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.

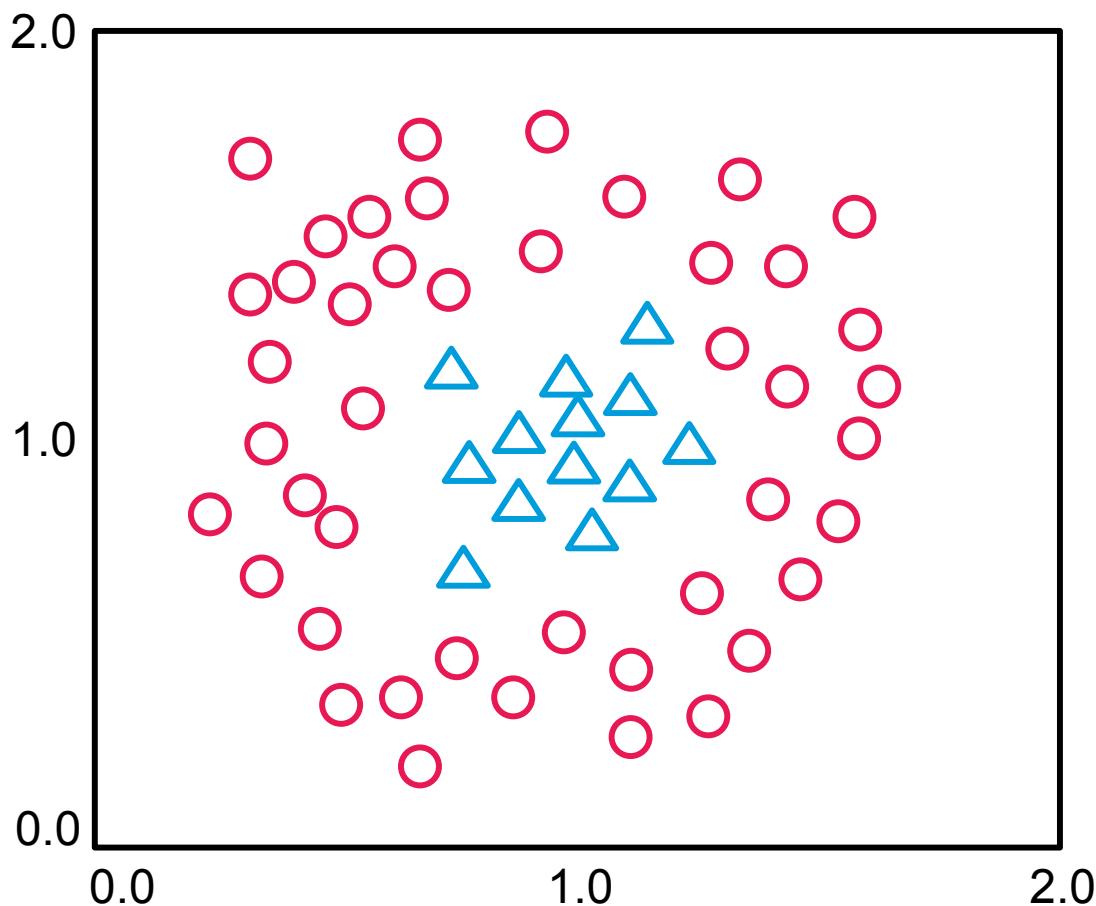


Figure 9–7: A dataset that cannot be fit well using a straight line.

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.

The 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.

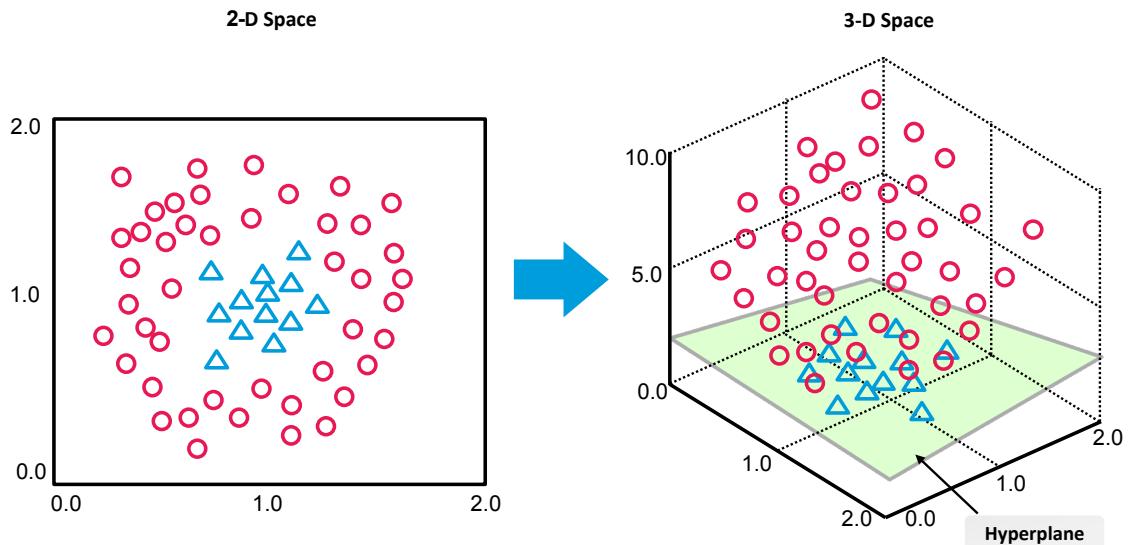


Figure 9-8: The kernel trick representing features from two-dimensional space in three-dimensional space.

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:

- K 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.
- $\langle \cdot, \cdot \rangle$ (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 at 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	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. 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.

Kernel Method	Description
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.
- Recognize that the goal of an SVM classification 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.
- Scale the data used to train an SVM model so the margins are calculated based on the distribution of the feature spaces and not the range of values.
- 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. From Jupyter Notebook, select **CAIP/SVMs/SVMs - Iris.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 cell below it.
 - b) Run the code cell.
 - 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.
3. Get acquainted with the dataset.
 - a) Scroll down and view the cell titled **Get acquainted with the dataset**, and examine the code cell below it.
 - b) Run the code cell.

- c) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   sepal length (cm)    150 non-null   float64 
 1   sepal width (cm)     150 non-null   float64 
 2   petal length (cm)    150 non-null   float64 
 3   petal width (cm)     150 non-null   float64 
 4   target              150 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 6.0 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.

4. Examine descriptive statistics.

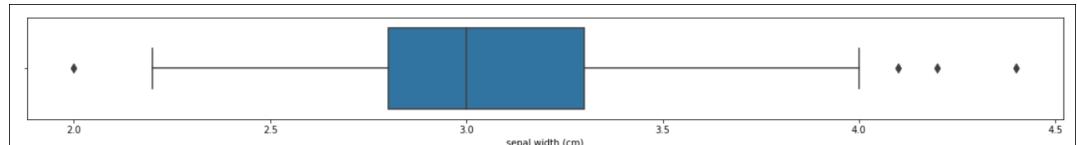
- Scroll down and view the cell titled **Examine descriptive statistics**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.00	150.00	150.00	150.00	150.00
mean	5.84	3.06	3.76	1.20	1.00
std	0.83	0.44	1.77	0.76	0.82
min	4.30	2.00	1.00	0.10	0.00
25%	5.10	2.80	1.60	0.30	0.00
50%	5.80	3.00	4.35	1.30	1.00
75%	6.40	3.30	5.10	1.80	2.00
max	7.90	4.40	6.90	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.

5. Identify outliers.

- Scroll down and view the cell titled **Identify outliers**, and examine the code cell below it.
- Run the code cell.
- Examine the output.



There appear to be a few outliers for sepal width.

6. Why are SVMs often better than other algorithms at handling datasets with outliers?

7. Reduce the dimensionality of the dataset.

- Scroll down and view the cell titled **Reduce the dimensionality of the dataset**, and examine the code cell below it.

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.

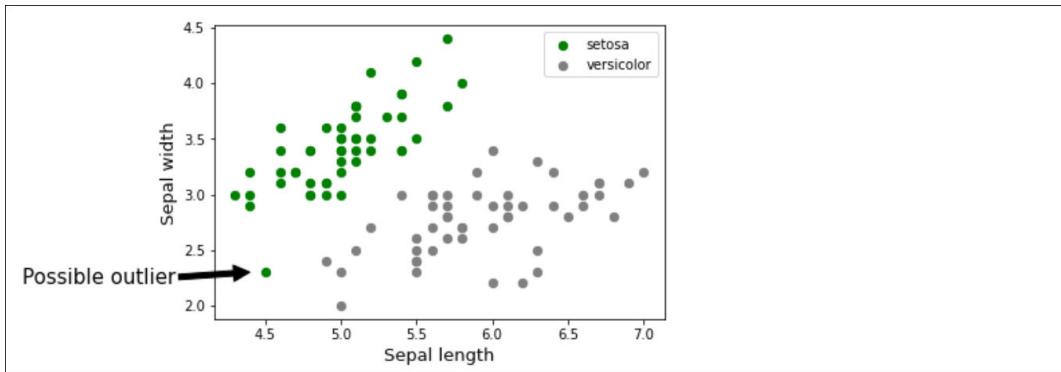
- Run the code cell.
- Examine the output.

```
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,)
```

8. Examine the separation between classes using a scatter plot.

- Scroll down and view the cell titled **Examine the separation between classes using a scatter plot**, and examine the code cell below it.
- Run the code cell.

- c) Examine the output.



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 gray 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.

9. 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 cell below it.

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 3 through 19 create the scatter plot, same as before.
- Lines 21 through 32 begin creating a grid on which to plot the model's decision function.
- Lines 34 through 41 plot the decision boundary line, then plot the support-vector margins for SVM models.
- Lines 42 and 43 just plot the decision boundary for non-SVM models.

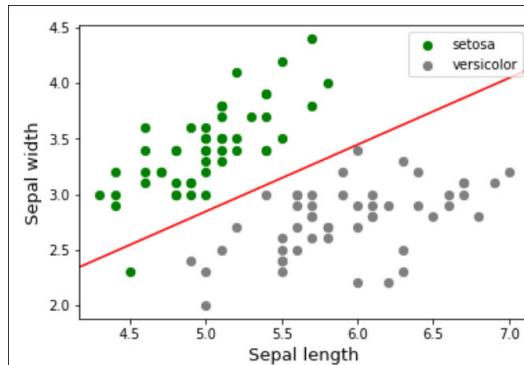
- b) Run the code cell.
c) Examine the output.

The function to plot the decision boundary has been defined.

10. Train a basic logistic regression model and plot its decision boundary.

- a) Scroll down and view the cell titled **Train a basic logistic regression model and plot its decision boundary**, and examine the code cell below it.
b) Run the code cell.

- c) 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.

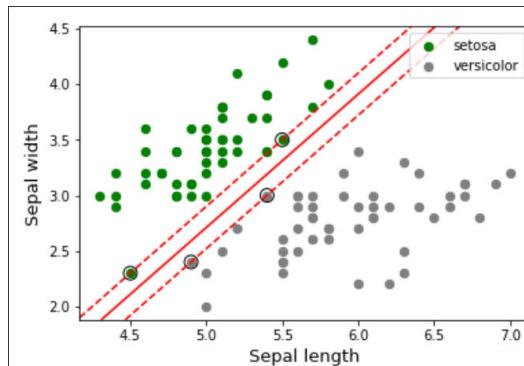
11. Train an SVM model and plot its decision boundary plus margins.

- a) Scroll down and view the cell titled **Train an SVM model and plot its decision boundary plus margins**, and examine the code cell below it.

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) Run the code cell.
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.

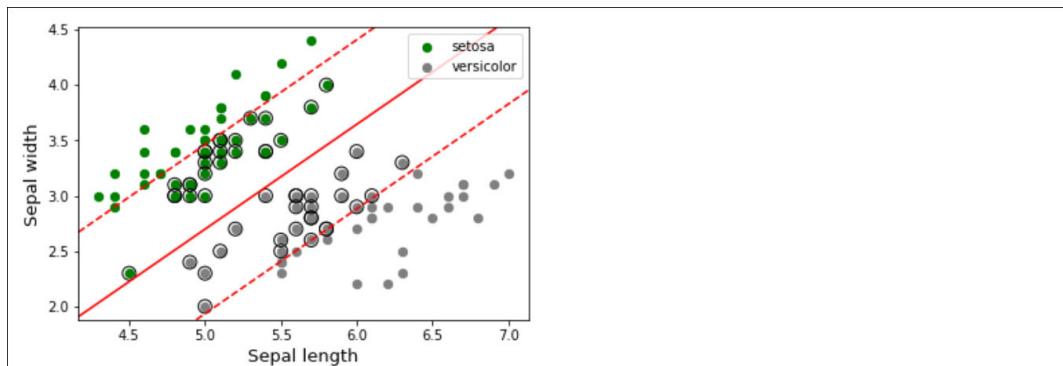
12. How does this SVM boundary fit to the data as compared to the logistic regression boundary?

13. Reduce the regularization penalty to soften the margins.

- a) Scroll down and view the cell titled **Reduce the regularization penalty to soften the margins**, and examine the code cell below it.

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) Run the code cell.
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.

14. Split the dataset.

- a) Scroll down and view the cell titled **Split the dataset**, and examine the code cell below it.

This splits the dataset in the usual way, incorporating the entire dataset this time (i.e., all four features and all three labels).

- b) Run the code cell.
c) Examine the output.

Training and test datasets and their labels have been split.

15. 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 cell below it.

To start with, you'll just use the arbitrary value of 100 for the regularization penalty.

- b) Run the code cell.
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.

16. 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 cell below it.

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) Run the code cell.
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 examine the next code cell.

```
1 # Score using the test data.
2 score = search.score(X_test, y_test)
3
4 print('Accuracy: {:.0f}%'.format(score * 100))
```

- e) Run the code cell.
f) Examine the output.

```
Accuracy: 95%
```

The model's accuracy increased.

17. Examine the optimized SVM model's predictions.

- a) Scroll down and view the cell titled **Examine the optimized SVM model's predictions**, and examine the code cell below it.
b) Run the code cell.
c) Examine the output.

	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

This seems to confirm the high accuracy that was just reported. There are no misclassifications within the first 10 records.

18. 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 regression problems.

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. Estimations 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 estimations. Likewise, small ε values penalize errors more when calculating the cost function, but may lead to overfitting.



Note: ε is the Greek letter epsilon.

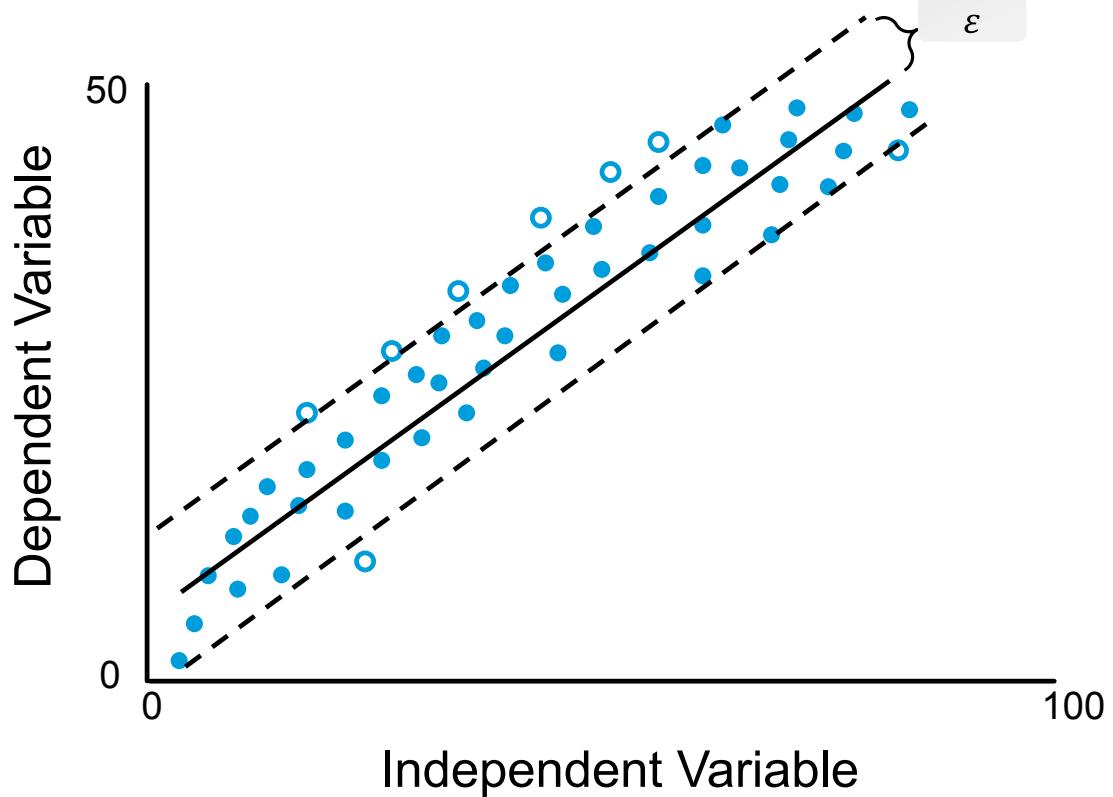


Figure 9–9: SVMs used in regression. The hollow circles indicate data examples that are outside of the margins (i.e., support vectors).

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 Files

/home/student/CAIP/SVMs/SVMs - California Housing.ipynb
 /home/student/CAIP/SVMs/housing_data/cali_house_data.pickle

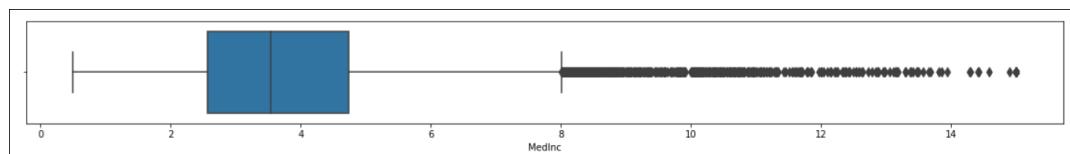
Before You Begin

Jupyter Notebook is open.

Scenario

Thus far, you've trained the California 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. From Jupyter Notebook, select **CAIP/SVMs/SVMs - California Housing.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 cell below it.
 - b) Run the code cell.
 - c) Verify that **cali_house_data.pickle** was loaded with 20,640 records.
3. Identify outliers.
 - a) Scroll down and view the cell titled **Identify outliers**, and examine the code cell below it.
 - b) Run the code cell.
 - c) Examine the output.



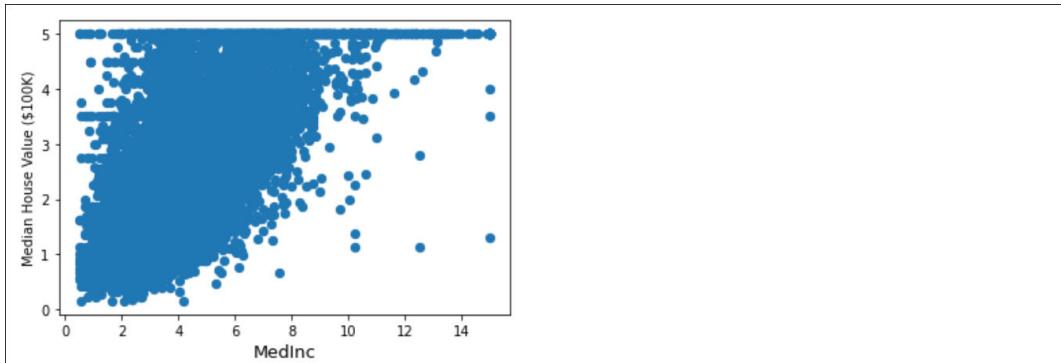
- **MedInc**, if you recall, is the median income value of residents in a census block group.
- There appear to be several outliers toward the higher end of the range of values.
- The median income value is usually below 8 for a given area.

4. Separate **MedInc** from **target** for demonstration purposes.
 - a) Scroll down and view the cell titled **Separate MedInc from target for demonstration purposes**, and examine the code cell below it.
 You'll start by looking at only one feature (**MedInc**) as compared to the **target** label (median house value). Later, you'll train the model using more features.
 - b) Run the code cell.

\mathbf{x} contains the feature data, and \mathbf{y} includes the label data.

5. Examine a scatter plot of MedInc and target.

- Scroll down and view the cell titled **Examine a scatter plot of MedInc and target**, and examine the code cell below it.
- Run the code cell.
- Examine the output.



- As you might expect, a house's value tends to increase as the income of the surrounding residents 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.

- Scroll down and view the cell titled **Plot a regression line for a given model**, and examine the code cell below it.

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 3 through 5 create the scatter plot, same as before.
- Lines 9 through 24 plot the regression line, then plot the support-vector margins for SVM models.
- Lines 25 through 28 just plot the regression line for non-SVM models.

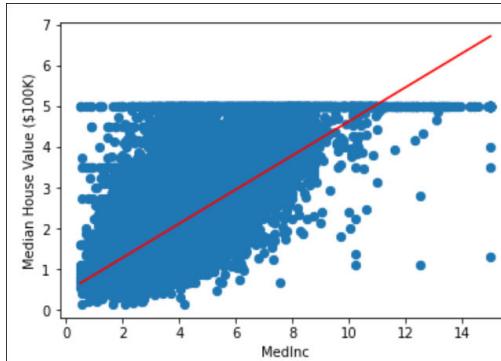
- Run the code cell.
- Examine the output.

The function to plot the regression line has been defined.

7. Train a basic linear regression model and plot its line of best fit.

- Scroll down and view the cell titled **Train a basic linear regression model and plot its line of best fit**, and examine the code cell below it.
- Run the code cell.

- c) 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 cell below it.

The `SVR()` class implements support-vector regression. In this case, there are two hyperparameters being set:

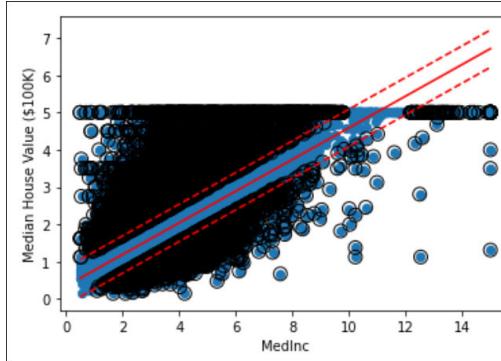
- `kernel` specifies the kernel method to use; in this case, you'll start with a linear kernel.
- `epsilon (ε)` determines the space between the margins. A higher value leads to wider margins. You're starting with a value of 0.5.

- b) Run the code cell.



Note: This may take a few minutes to complete.

- 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 downward slightly but with a similar slope. This may help the model generalize better with the presence of outliers in the training.

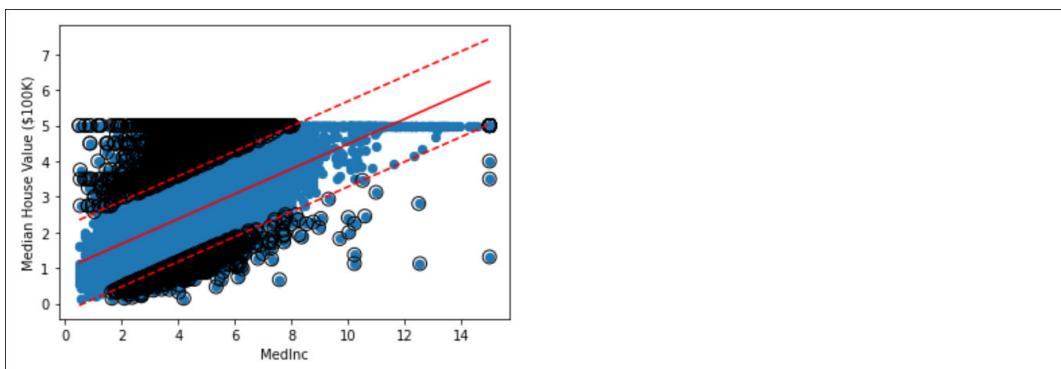
9. How does SVM regression differ from SVM classification in terms of how the data examples are included or not included within the margins?

10. Adjust the margins using a different `epsilon` value.

- Scroll down and view the cell titled **Adjust the margins using a different `epsilon` value**, and examine the code cell below it.

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.

- Run the code cell.
- Examine the output.



The margins are much wider now, and more data examples are placed within those margins. The regression line has also shifted upward and the slope is somewhat smaller. 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 dataset.

- Scroll down and view the cell titled **Split the dataset**, and examine the code cell below it.

This splits the datasets in the usual way, incorporating three features from the dataset this time: `MedInc`, `AveRooms`, and `Latitude`. SVMs can take a while to train, so for classroom purposes, you won't use the entire dataset.

- Run the code cell.
- Examine the output.

Training and test datasets and their labels have been split.

12. Standardize the features.

- Scroll down and view the cell titled **Standardize the features**, and examine the code cell below it.

Recall that you're scaling the features on this dataset so that the regularization penalty is applied equally. Also, SVMs tend to do better with scaled data.

- Run the code cell.
- Examine the output.

The features have been standardized.

13. Evaluate an SVM model using a holdout test set.

- Scroll down and view the cell titled **Evaluate an SVM model using a holdout test set**, and examine the code cell below it.
To start with, you'll just use the arbitrary value of 1.2 for `epsilon`.
- Run the code cell.
- Examine the output.

```
Cost (mean squared error): 0.8046
```

The mean squared error (MSE) for this model is 0.8046. This is somewhat worse than your previous linear regression models (the best of which produced an MSE of around 0.73), but there's still more opportunity for improvement.

14. Optimize the SVM model with grid search and cross-validation.

- Scroll down and view the cell titled **Optimize the SVM model with grid search and cross-validation**, and examine the code cell below it.

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 three `C` values: 0.01, 0.1, and 1. 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 three values for `epsilon`, which determines the width of the margins.
- On line 9, the grid search will be optimizing for MSE and will perform three-fold cross-validation on the training data.

- Run the code cell.
This will take around 5 to 10 minutes to complete.
- Examine the output.

```
{'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}
```

The grid search identified a `C` value of 1 and an `epsilon` value of 0.5, along with the RBF kernel, as the optimal hyperparameter combination.

- Scroll down and examine the next code cell.

```
1 preds = search.predict(X_test)
2
3 cost = mse(y_test, preds)
4
5 print('Cost (mean squared error): {:.4f}'.format(cost))
```

- Run the code cell.
- Examine the output.

```
Cost (mean squared error): 0.6110
```

The model's error decreased, and is now lower than the error of your best linear regression models.

15. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.

- b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
 - c) Close the **SVMs - California Housing** tab in Firefox, but keep a tab open to **CAIP** in the file hierarchy.
-

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.

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 the simplest form of artificial neural network, the multi-layer perceptron (MLP).
- Build a type of neural network well suited for computer vision tasks, the convolutional neural network (CNN).
- Build a type of neural network well suited for natural language processing (NLP) tasks, the recurrent neural network (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 Networks (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.

Perceptrons

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)**.

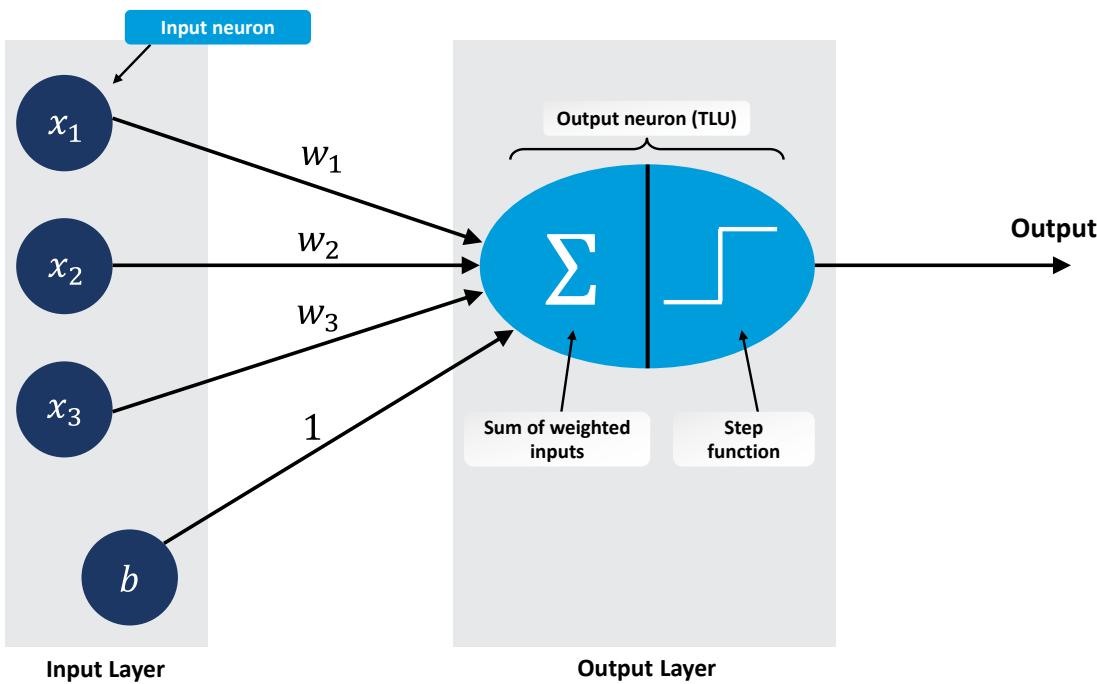


Figure 10-1: A simple perceptron with several inputs and one TLU output neuron.



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 Perceptrons

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 next layer, as in the following figure.

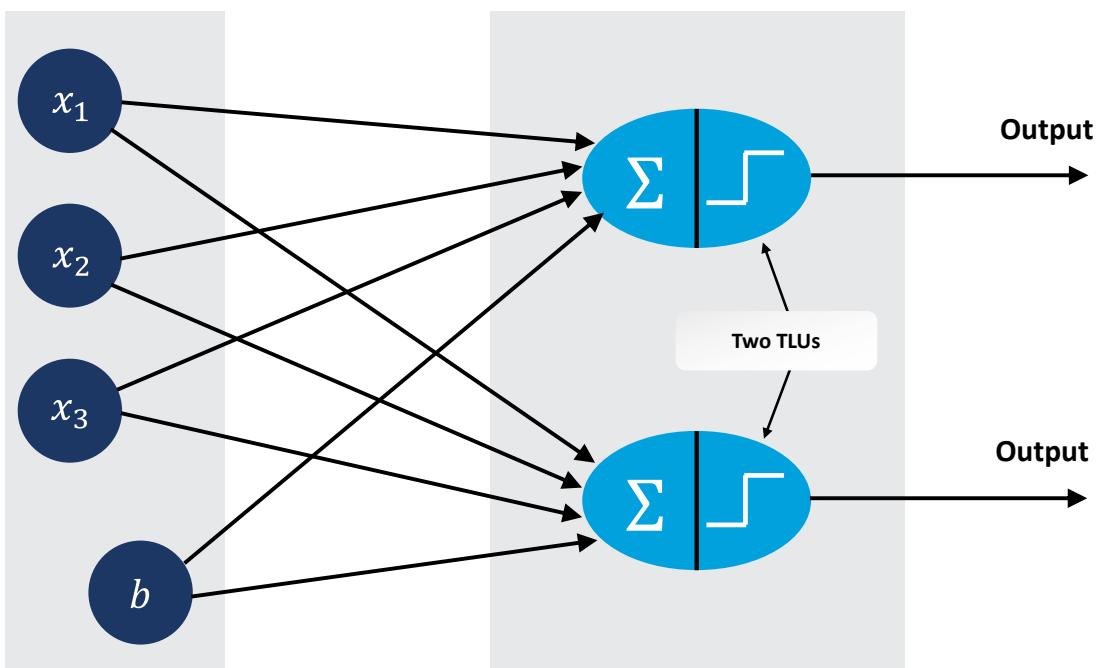


Figure 10-2: A perceptron that solves a multi-label classification problem.

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 estimations. This effectively strengthens the connection between neurons. When inputs lead to the wrong estimation —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. An estimation is made for this data example.
3. For an output neuron that made an incorrect estimation, the input neurons that would have led to the correct estimation 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 estimated value of the j^{th} output neuron.
- x_i is the value of the i^{th} training example input.

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.



Note: This equation is very similar to the equation for stochastic gradient descent.



Note: η is the Greek letter eta, and ' is the prime symbol.

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.

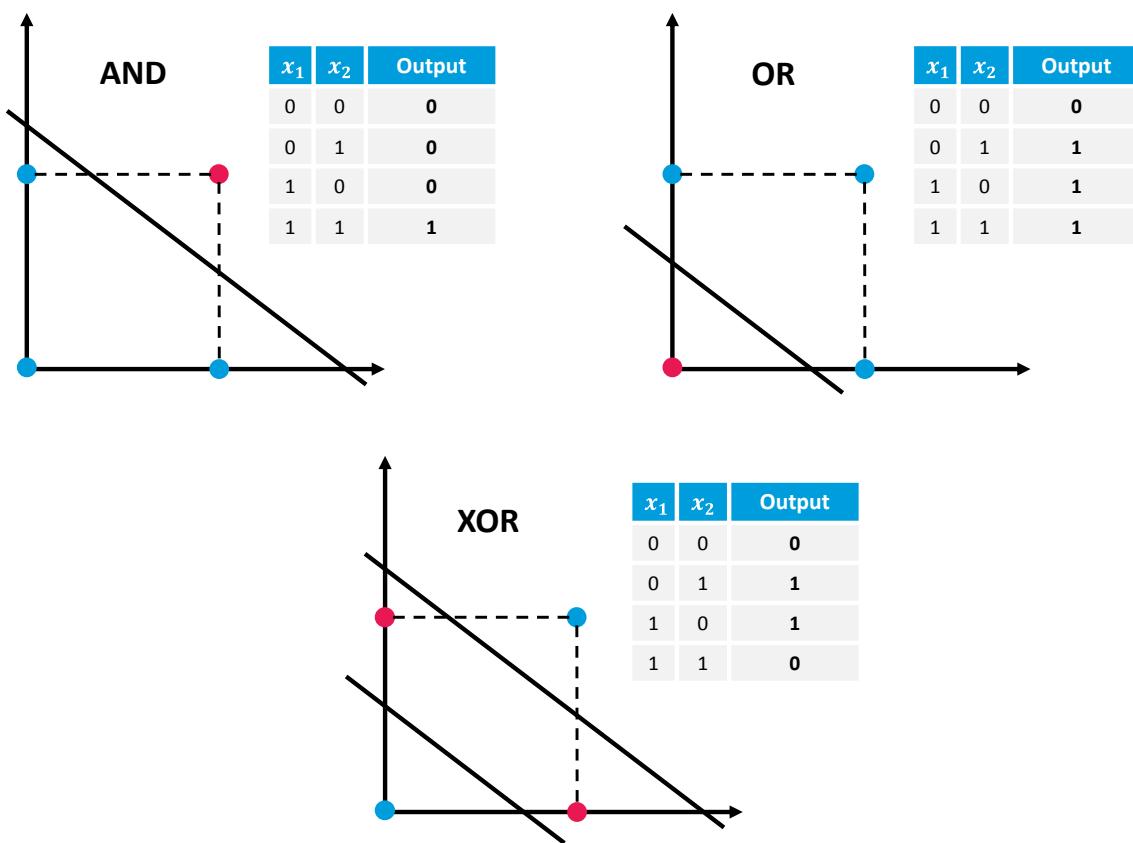


Figure 10-3: The XOR operation is not linearly separable.

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 Perceptrons (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 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.

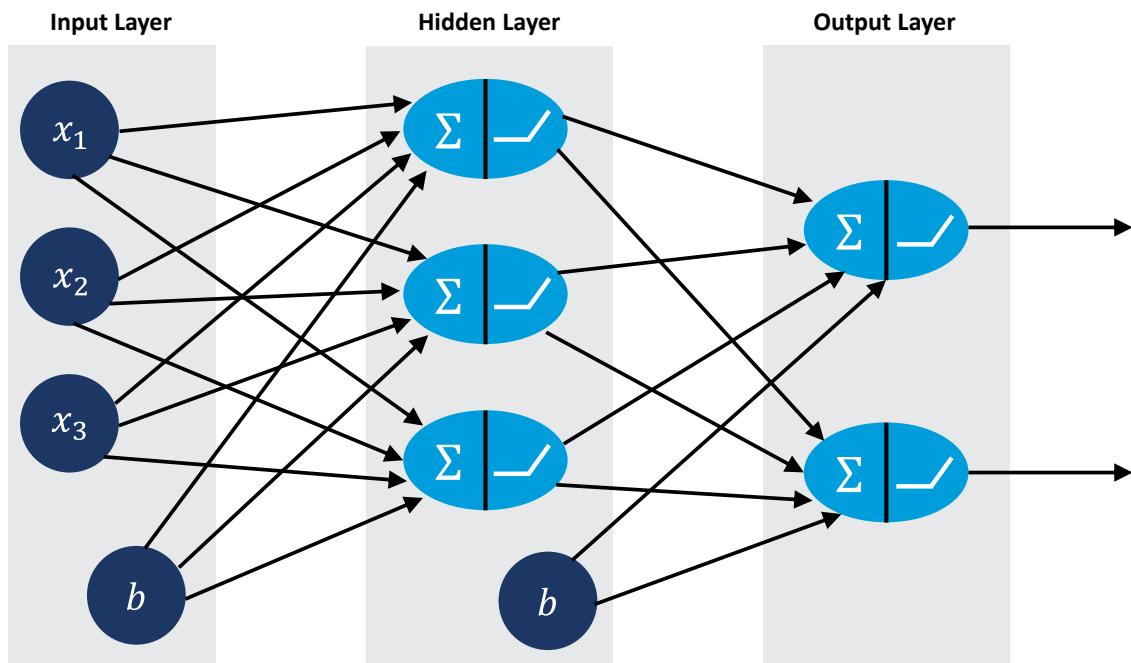


Figure 10-4: The presence of at least one hidden layer defines a multi-layer perceptron.

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. An estimation is made for this data example.
3. The error between actual and estimated 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.

Epochs and Iterations

Neural networks can be trained over several epochs, just like gradient descent in a standard machine learning algorithm. In a neural network, an **epoch** is a single pass (both forward and backward) through the *entire* training dataset. The neural network may not learn the optimal weights from a single epoch, so training over multiple epochs can lead to better results as the weights are updated to more effectively minimize loss. The tradeoff is that every epoch increases training time.

Note that an epoch is not to be confused with an **iteration**, which is a single pass through just one batch of data—i.e., a random subset of the overall training dataset. A single epoch can comprise multiple iterations.

For example, if you had 100 total training examples, and a batch size of 25, a single epoch would comprise 4 iterations ($100 / 25 = 4$). If you trained over 3 epochs, the network would go through 12 iterations ($4 \times 3 = 12$).

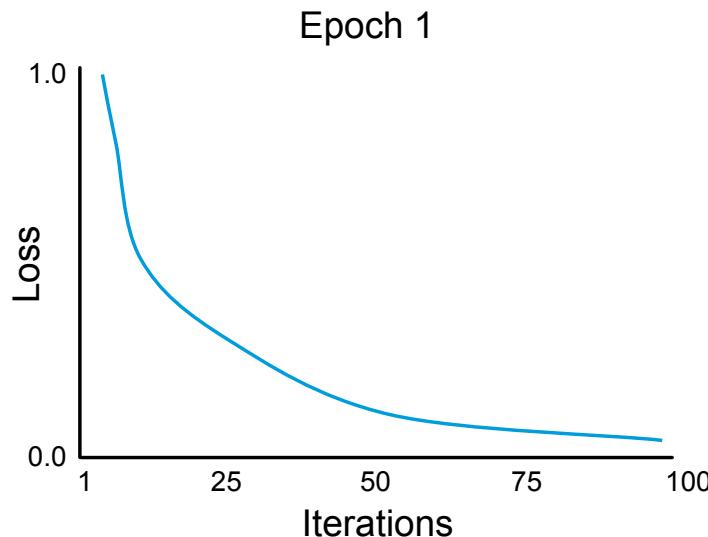
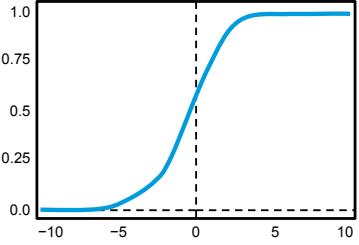
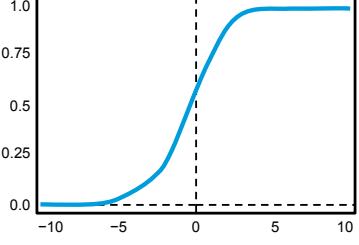
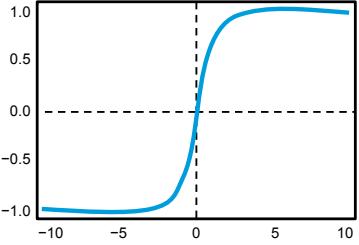


Figure 10-5: Gradual loss minimization over multiple iterations for one epoch.

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	<p>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 binary classifier. If applied to hidden layers, the backpropagation of the gradient error may become too slow.</p> 
Softmax function	<p>Just like softmax is used in multinomial logistic regression to perform multi-class classification, so too can it be used as an activation function in the output layer of a neural network. It outputs probability scores for each class between 0 and 1, where all scores add up to 1. You would therefore choose softmax instead of sigmoid for multi-class problems.</p> 
Hyperbolic tangent (tanh) function	<p>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. It is typically preferred over sigmoid when applied to hidden layers. However, its usefulness in hidden layers has largely been superseded by ReLU.</p> 

Activation Function	Description
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 binary classification problem.
- Consider using the softmax activation function for the output layer of a multi-class classification problem.
- Consider using the ReLU activation function for hidden layers instead of the tanh function.
- 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 = 'relu')` —This constructs an MLP to use for regression. In this case, two hidden layers will be constructed, each with two 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/Neural Networks - Occupancy.ipynb
 /home/student/CAIP/Neural Networks/VisualizeNN.py
 /home/student/CAIP/Neural Networks/occupancy_data/occupancy_train.csv
 /home/student/CAIP/Neural Networks/occupancy_data/occupancy_test.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. From Jupyter Notebook, select **CAIP/Neural Networks/Neural Networks - Occupancy.ipynb** to open it.
2. Import the relevant libraries and load the datasets.
 - a) View the cell titled **Import software libraries and load the datasets**, and examine the code cell below it.
 - b) Run the code cell.
 - c) Verify that **occupancy_train.csv** and **occupancy_test.csv** were loaded with 8,143 and 2,665 records, respectively.

This dataset already comes pre-split for training and testing.
3. Get acquainted with the dataset.
 - a) Scroll down and view the cell titled **Get acquainted with the dataset**, and examine the code cell below it.
 - b) Run the code cell.

- c) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8143 entries, 0 to 8142
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Date              8143 non-null    object  
 1   Temperature       8143 non-null    float64 
 2   RelativeHumidity  8143 non-null    float64 
 3   Light             8143 non-null    float64 
 4   CO2               8143 non-null    float64 
 5   HumidityRatio     8143 non-null    float64 
 6   Occupancy          8143 non-null    int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 445.4+ KB
None

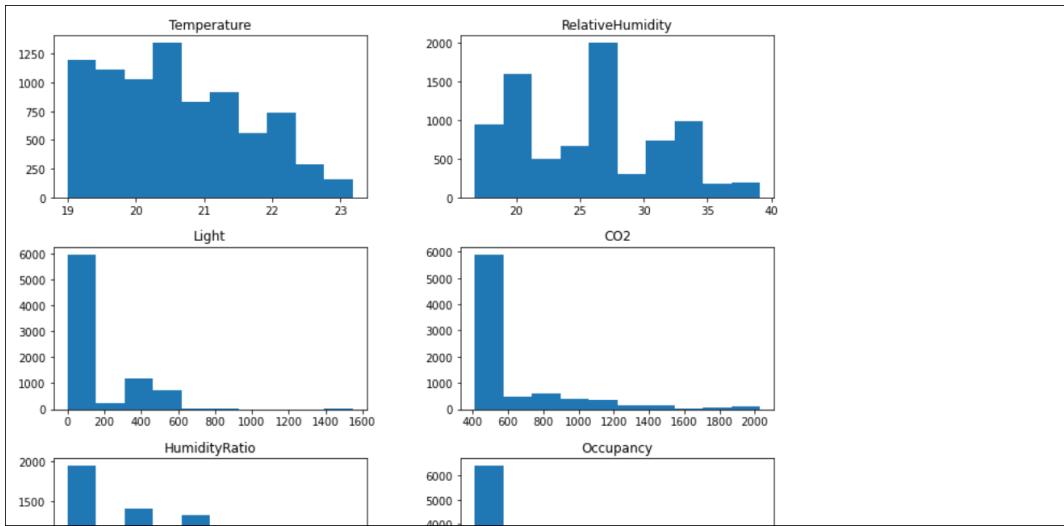
   Date  Temperature  RelativeHumidity  Light      CO2  HumidityRatio  Occupancy
0  2/5/2015 19:37    21.200        19.840   0.00  525.333333  0.003082    0
1  2/4/2015 22:11    21.390        25.700   0.00  475.000000  0.004046    0
2  2/9/2015 13:51    21.245        32.925  474.50 1126.500000  0.005146    1
3  2/5/2015 20:36    21.200        19.390   0.00  472.500000  0.003012    0
4  2/10/2015 2:39    20.290        32.900   0.00  460.000000  0.004846    0
5  2/6/2015 21:20    20.200        19.000   0.00  447.500000  0.002774    0
6  2/8/2015 20:17    19.390        27.500   0.00  437.000000  0.003825    0
7  2/8/2015 7:44     19.200        31.200   0.00  435.500000  0.004291    0
8  2/5/2015 6:06     20.890        23.290   0.00  448.000000  0.003553    0
9  2/8/2015 9:39     19.500        30.600   42.25  430.750000  0.004288    0
```

- 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.

4. Examine the distributions of the features.

- Scroll down and view the cell titled **Examine the distributions of the features**, and examine the code cell below it.
- Run the code cell.

c) Examine the output.



- Several features are right skewed, especially CO₂ and Light.
- The distribution for RelativeHumidity has alternating peaks.

5. Examine descriptive statistics.

- Scroll down and view the cell titled **Examine descriptive statistics**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

	Temperature	RelativeHumidity	Light	CO2	HumidityRatio	Occupancy
count	8143.000	8143.000	8143.000	8143.000	8143.000	8143.000
mean	20.619	25.732	119.519	606.546	0.004	0.212
std	1.017	5.531	194.756	314.321	0.001	0.409
min	19.000	16.745	0.000	412.750	0.003	0.000
25%	19.700	20.200	0.000	439.000	0.003	0.000
50%	20.390	26.223	0.000	453.500	0.004	0.000
75%	21.390	30.533	256.375	638.833	0.004	0.000
max	23.180	39.117	1546.333	2028.500	0.006	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.

6. Split the label from the datasets.

- Scroll down and view the cell titled **Split the label from the datasets**, and examine the code cell below it.
- Run the code cell.

- c) Examine the output.

```
Original set: (10808, 7)
-----
Training features: (8143, 6)
Testing features: (2665, 6)
Training labels: (8143, 1)
Testing labels: (2665, 1)
```

The labels for both training and testing sets were split from the rest of the data.

7. Convert the Date column to datetime format for processing.

- Scroll down and view the cell titled **Convert the Date column to datetime format for processing**, and examine the code cell below it.
- Run the code cell.
- 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 Date column has been converted to a pandas datetime format.
- 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.

8. Determine which datetime components have unique values.

- Scroll down and view the cell titled **Determine which datetime components have unique values**, and examine the code cell below it.
- Run the code cell.
- 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.

9. Split the relevant datetime features.

- Scroll down and view the cell titled **Split the relevant datetime features**, and examine the code cell below it.

- This function, when called, will split out the relevant date and time components from the `Date` column.
 - On lines 4 through 11, 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) Run the code cell.
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	19	37
1	2015-02-04 22:11:00	21.390	25.700	0.0	475.000000	0.004046	4	22	11
2	2015-02-09 13:51:00	21.245	32.925	474.5	1126.500000	0.005146	9	13	51
3	2015-02-05 20:36:00	21.200	19.390	0.0	472.500000	0.003012	5	20	36
4	2015-02-10 02:39:00	20.290	32.900	0.0	460.000000	0.004846	10	2	39

Now that you've extracted the relevant date and time components, you can drop the `Date` column.

10. Drop the original Date column.

- a) Scroll down and view the cell titled **Drop the original Date column**, and examine the code cell below it.
b) Run the code cell.
c) Examine the output.

```
Columns before drop:  
['Date', 'Temperature', 'RelativeHumidity', 'Light', 'CO2', 'HumidityRatio', 'Day', 'Hour',  
'Minute']  
  
Columns after drop:  
['Temperature', 'RelativeHumidity', 'Light', 'CO2', 'HumidityRatio', 'Day', 'Hour', 'Minute'  
'']
```

The `Date` column was dropped.

11. Standardize the features.

- a) Scroll down and view the cell titled **Standardize the features**, and examine the code cell below it.
As you've seen before, this function scales the data using the *z*-score.
b) Run the code cell.
c) Examine the output.

```
The features have been standardized.
```

- d) Scroll down and examine the next code cell.

```
1 with pd.option_context('float_format', '{:.2f}'.format):  
2     display(X_train.describe())
```

- e) Run the code cell.

- f) Examine the output.

	Temperature	RelativeHumidity	Light	CO2	HumidityRatio	Day	Hour	Minute
count	8143.00	8143.00	8143.00	8143.00	8143.00	8143.00	8143.00	8143.00
mean	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
min	-1.59	-1.62	-0.61	-0.62	-1.39	-1.84	-1.61	-1.70
25%	-0.90	-1.00	-0.61	-0.53	-0.92	-0.64	-0.90	-0.89
50%	-0.23	0.09	-0.61	-0.49	-0.07	-0.05	-0.06	-0.03
75%	0.76	0.87	0.70	0.10	0.57	0.55	0.93	0.84
max	2.52	2.42	7.33	4.52	3.07	1.75	1.64	1.70

The features have been scaled.

12. Train an MLP model.

- a) Scroll down and view the cell titled **Train an MLP model**, and examine the code cell below it.

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 10 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 (0.0001).
- `no_iter_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) Run the code cell.

- c) Examine the output.

```

Iteration 140, loss = 0.05206668
Iteration 141, loss = 0.05198519
Iteration 142, loss = 0.05191858
Iteration 143, loss = 0.05184728
Iteration 144, loss = 0.05175652
Iteration 145, loss = 0.05176086
Iteration 146, loss = 0.05165869
Iteration 147, loss = 0.05161021
Iteration 148, loss = 0.05154285
Iteration 149, loss = 0.05148312
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.

MLP model took 4.55 seconds to fit.
Accuracy: 88%
```

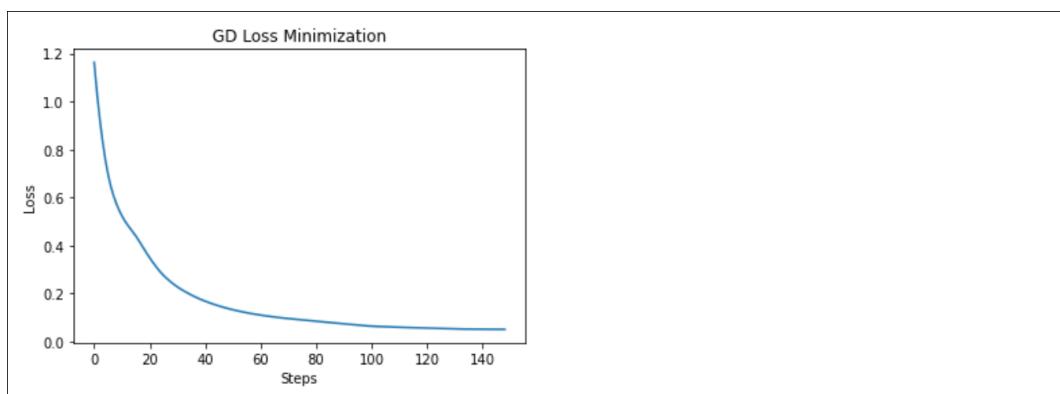
- The loss at each iteration is output.
- Each iteration minimizes the loss more than the previous iteration.
- Iteration 149 was the 10th iteration for which the loss did not improve by more than `tol`, so the solver stopped there.
- You'll plot this loss minimization in the next step to get a better 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.

13. 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 cell below it.

The `loss_curve_` attribute conveniently returns an array of the loss value at each iteration.

- b) Run the code cell.
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.

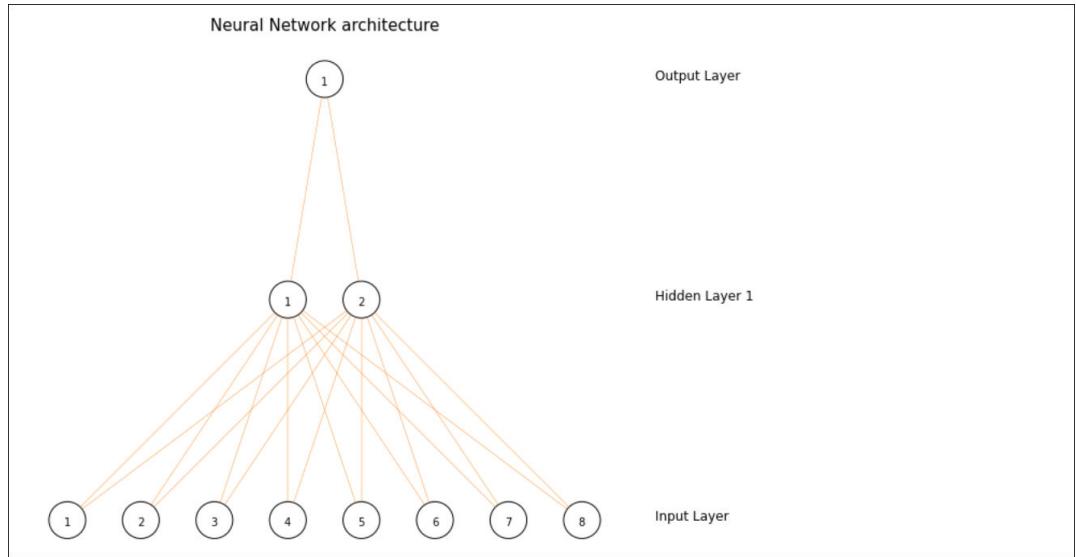
14. Visualize the neural network architecture.

- a) Scroll down and view the cell titled **Visualize the neural network architecture**, and examine the code cell below it.

This function uses the `VisualizeNN.py` module to draw a visual representation of the neural network.

- Lines 4 through 6 create 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 9 through 12 determine whether to draw the diagram with or without weights, according to the user's preference.

- Line 16 calls the function without weights.
- Run the code cell.
 - 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.

15. Retrieve the neuron weights and bias terms and redraw the network architecture.

- Scroll down and view the cell titled **Retrieve the neuron weights and bias terms and redraw the network architecture**, and examine the code cell below it.
- Run the code cell.

c) Examine the output.

```
Weights between input layer and hidden layer:  
[[ -0.34384442  0.25458622]  
[ 0.07705166 -0.35899729]  
[ 1.27567401 -1.05073071]  
[ 0.62436879 -1.29790752]  
[-0.42360002 -0.35968313]  
[ 0.09716226  0.53567013]  
[-0.17196756  0.10182009]  
[-0.09947617 -0.0033433 ]]  
  
Weights between hidden layer and output layer:  
[[ 2.06144361]  
[ -2.24575533]]  
  
Bias terms between input layer and hidden layer:  
[0.48033091 1.30779524]  
  
Bias terms between hidden layer and output layer:  
[-2.74117582]
```

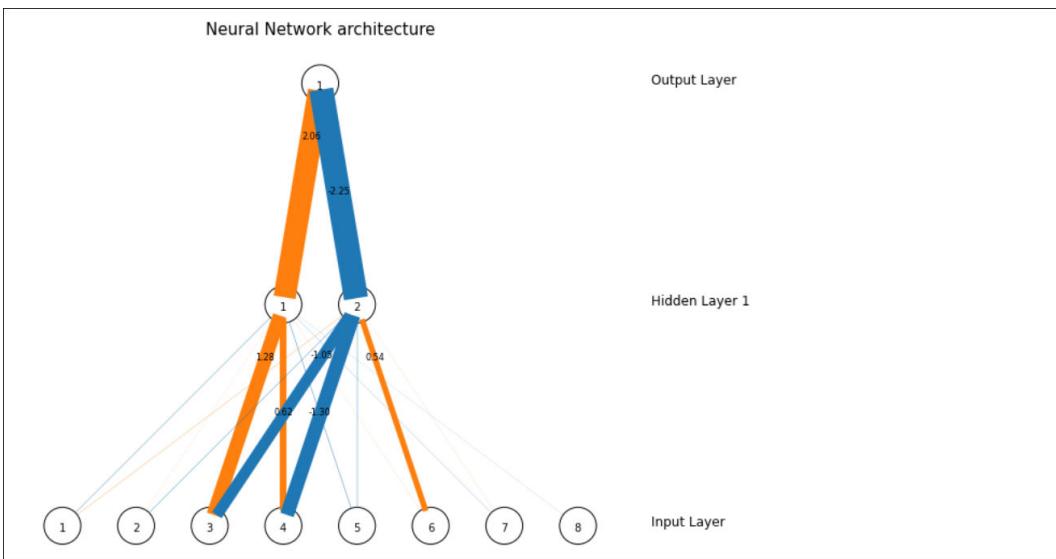
- The first array lists the weights of each neuron in the connections between the input layer and the hidden layer.
- The second array does likewise for the connections between the hidden and output layer.
- The third array lists the bias terms for the connections between the input layer and the hidden layer.
- The fourth array does likewise for the connection between the hidden and output layer.

d) Scroll down and examine the next code cell.

```
1 nn_diagram(X_train, y_train, mlp, True)
```

e) Run the code cell.

f) 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).

16. How does backpropagation generate the weights between the neurons of different layers in an MLP neural network?

17. What can you tell about the weights of this particular network structure?

18. 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 cell below it.

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.

- Run the code cell.



Note: It may take up to 10 minutes for the search to complete.

- Examine the output.

```
Grid search took 99.27 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.

- Scroll down and examine the next code cell.

```
1 score = search.score(X_test, y_test)
2
3 print('Accuracy: {:.0f}%'.format(score * 100))
```

- Run the code cell.

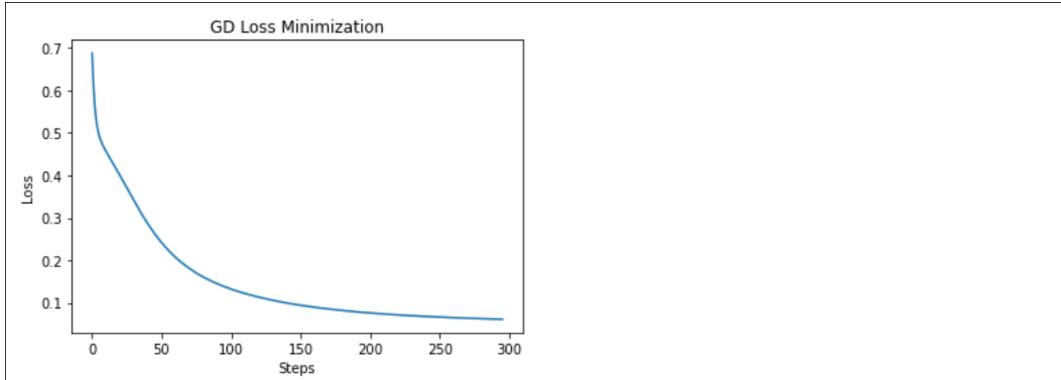
- f) Examine the output.

```
Accuracy: 94%
```

The model's accuracy has increased.

19. Visualize the loss minimization of the optimized model.

- Scroll down and view the cell titled **Visualize the loss minimization of the optimized model**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

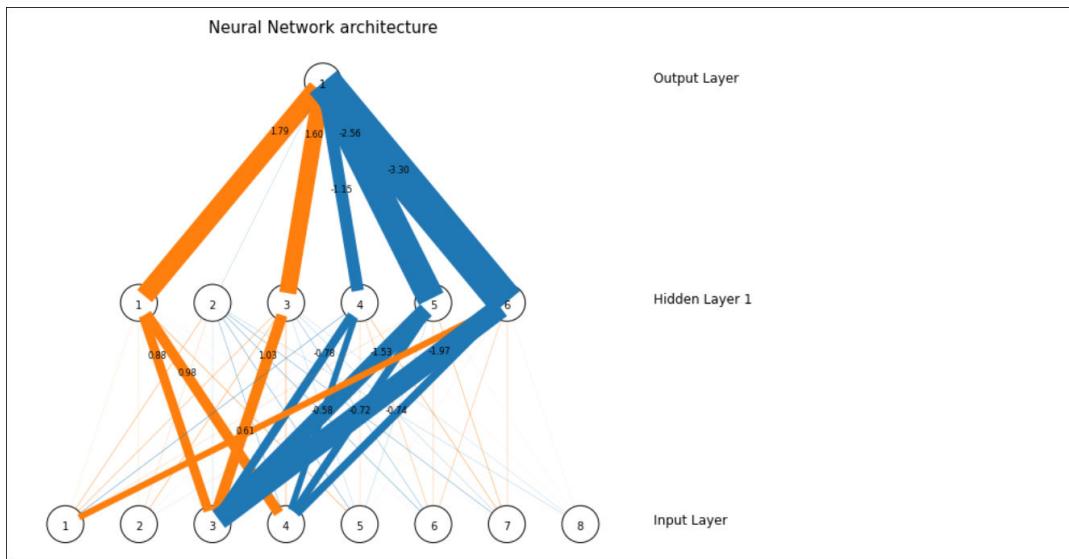


It takes about 300 steps for the solver to converge before failing to minimize more than the tolerance threshold.

20. Visualize the network structure of the optimized model.

- Scroll down and view the cell titled **Visualize the network structure of the optimized model**, and examine the code cell below it.
- Run the code cell.

- c) Examine the output.



- The input layer and output layer have the same number of neurons as before, 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.

21. Examine the model's predictions on the test set.

- Scroll down and view the cell titled **Examine the model's predictions on the test set**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

	Date	Temperature	RelativeHumidity	Light	CO2	HumidityRatio	ActualOccupancy	PredictedOccupancy
0	2/2/2015 23:49	20.650000	22.2450	0.0	443.000000	0.003342	0	0
1	2/2/2015 21:10	20.890000	23.0000	0.0	491.666667	0.003508	0	0
2	2/3/2015 19:56	21.245000	27.7450	0.0	770.750000	0.004331	0	0
3	2/3/2015 5:51	20.290000	22.6500	0.0	431.000000	0.003328	0	0
4	2/3/2015 2:12	20.525000	22.2675	0.0	442.750000	0.003320	0	0
5	2/4/2015 8:50	21.200000	25.1800	454.0	740.200000	0.003917	1	0
6	2/2/2015 17:19	22.500000	24.8650	433.0	816.500000	0.004189	1	1
7	2/4/2015 9:56	23.200000	25.5000	722.0	1011.400000	0.004485	1	1
8	2/4/2015 8:44	21.083333	25.2000	453.0	719.500000	0.003892	1	0
9	2/3/2015 13:28	23.200000	25.5500	171.0	918.000000	0.004494	0	0

22. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.

- b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
 - c) Close the **Neural Networks - Occupancy** tab in Firefox, but keep a tab open to **CAIP/Neural Networks/** 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 Networks (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.

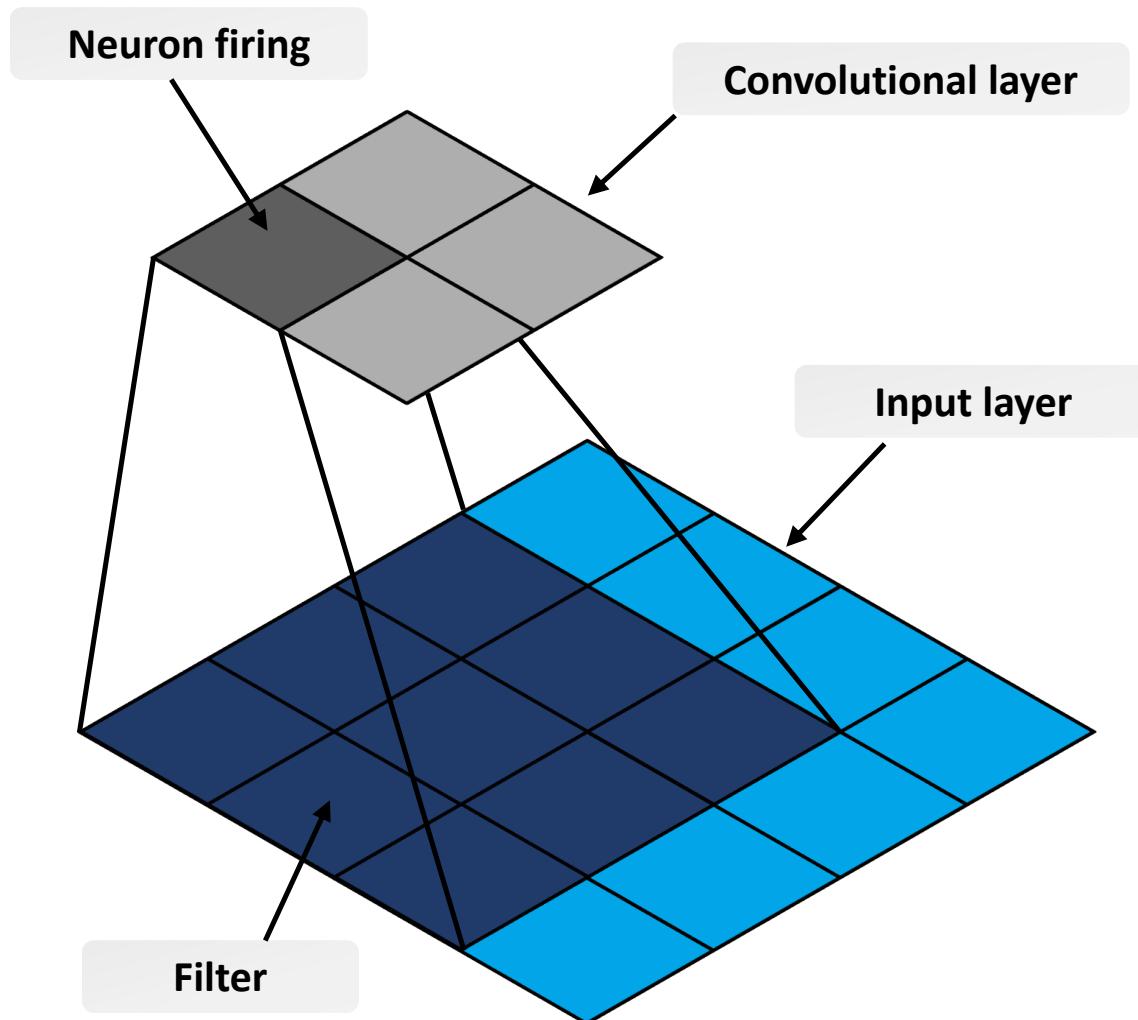


Figure 10-6: A convolutional layer using a filter to scan an input image.

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.

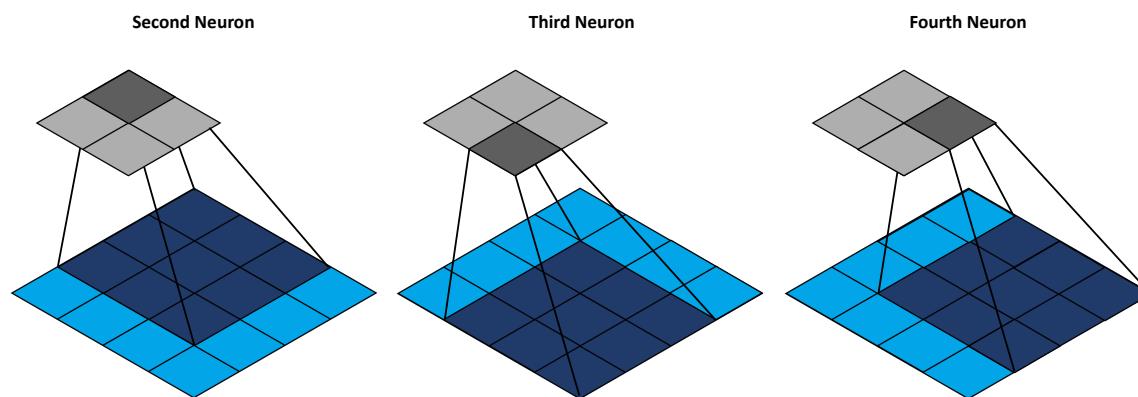


Figure 10-7: The remaining three neurons in the convolutional layer scan the image.

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.

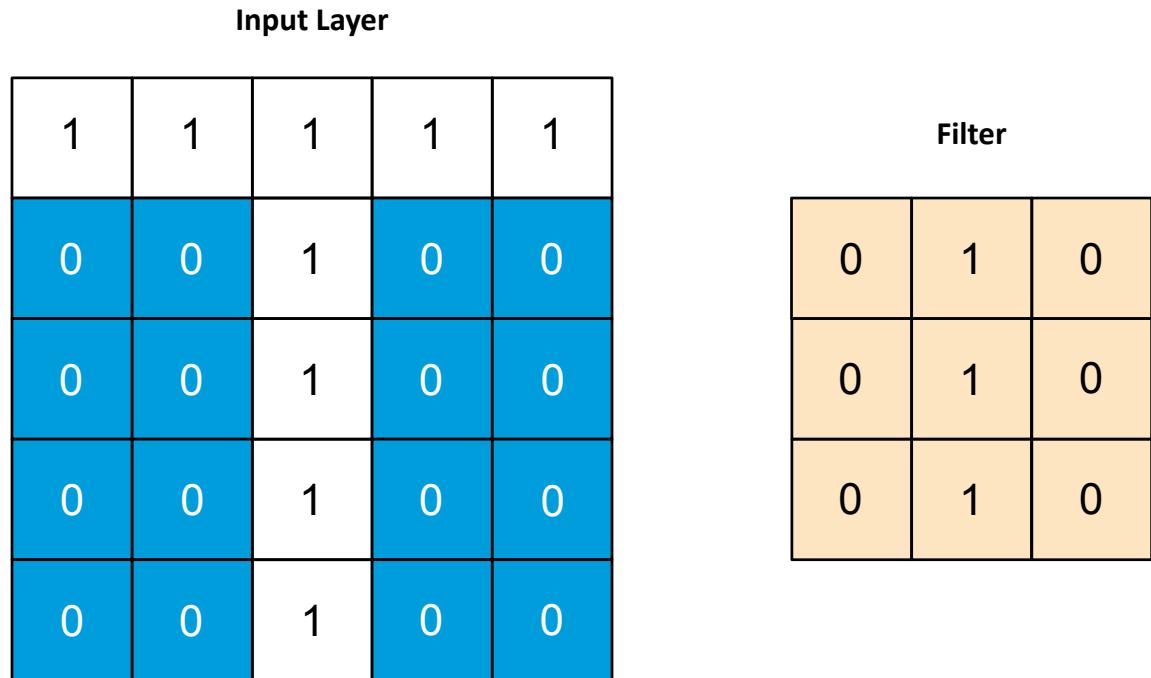


Figure 10–8: An input image (left) and a vertical line filter (right).

Now, consider how this filter would be projected on top of the input image.

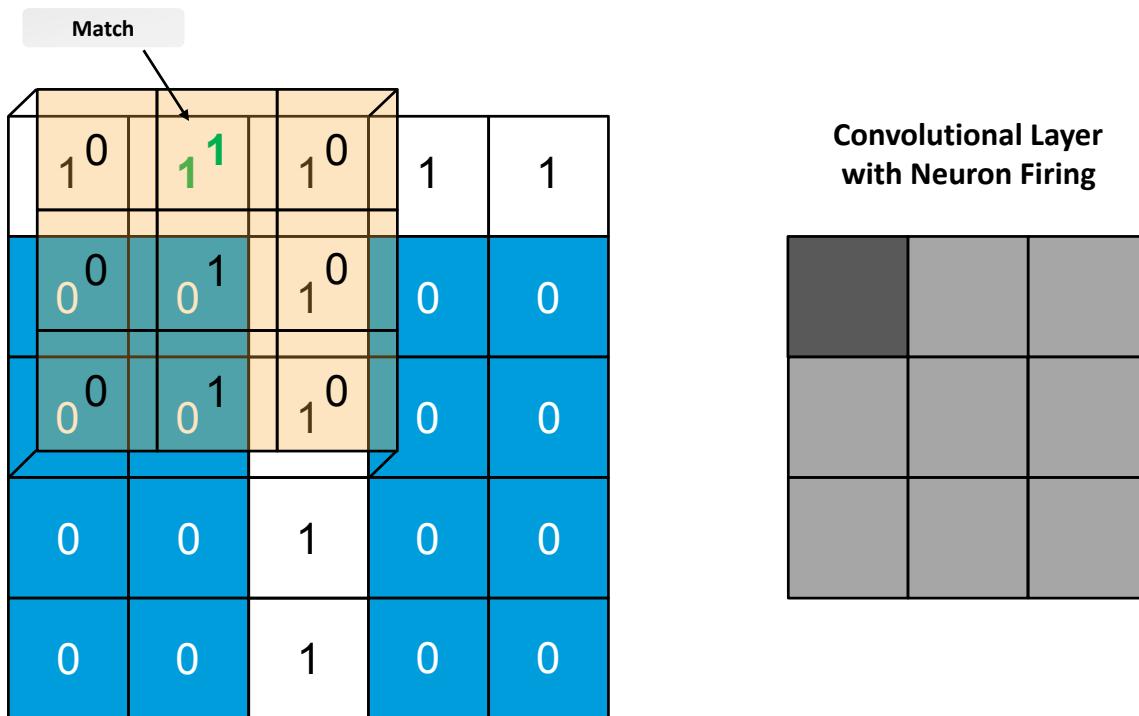


Figure 10-9: The vertical line filter applied to the image for the first neuron.

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.

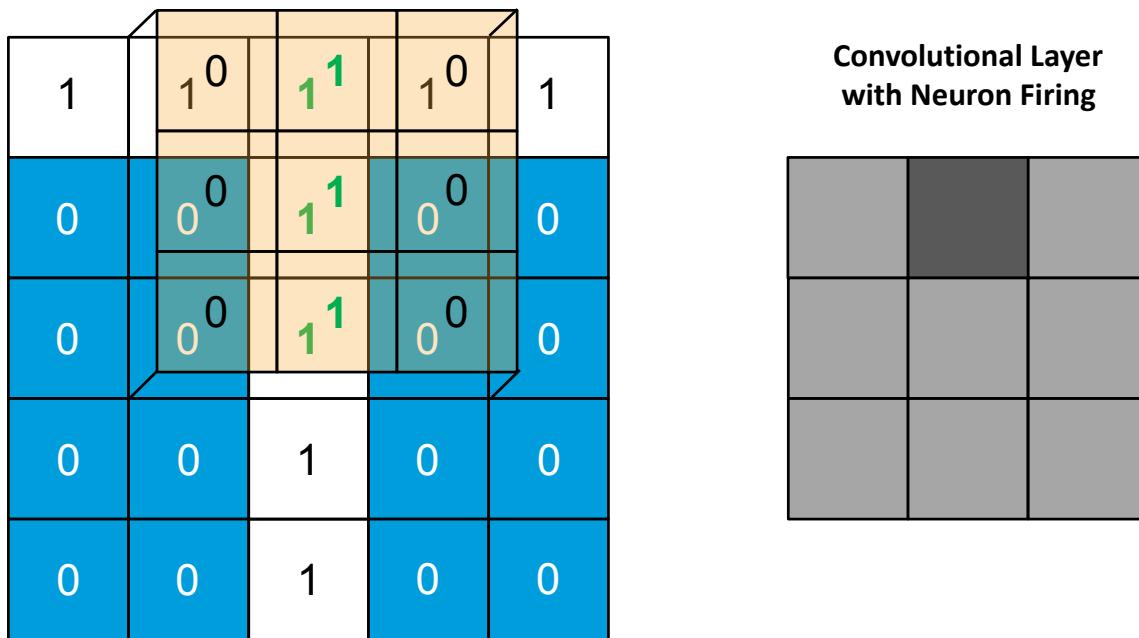


Figure 10-10: The next neuron applying the filter.

In this case, the filter found three 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 10-11: 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.

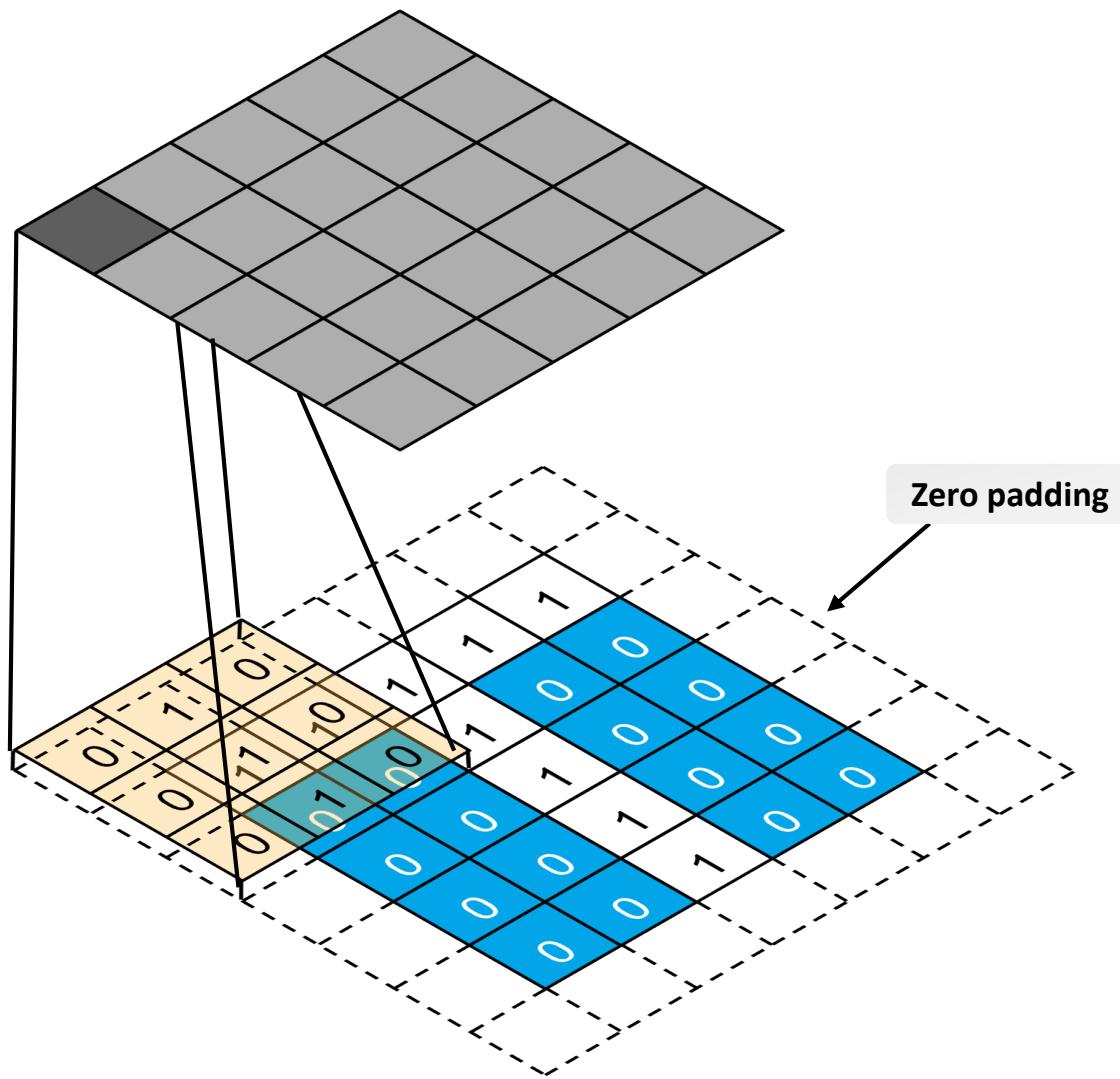


Figure 10–12: Padding an input image with zeros so that the convolutional neurons' filter can cover every part of the image.

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.

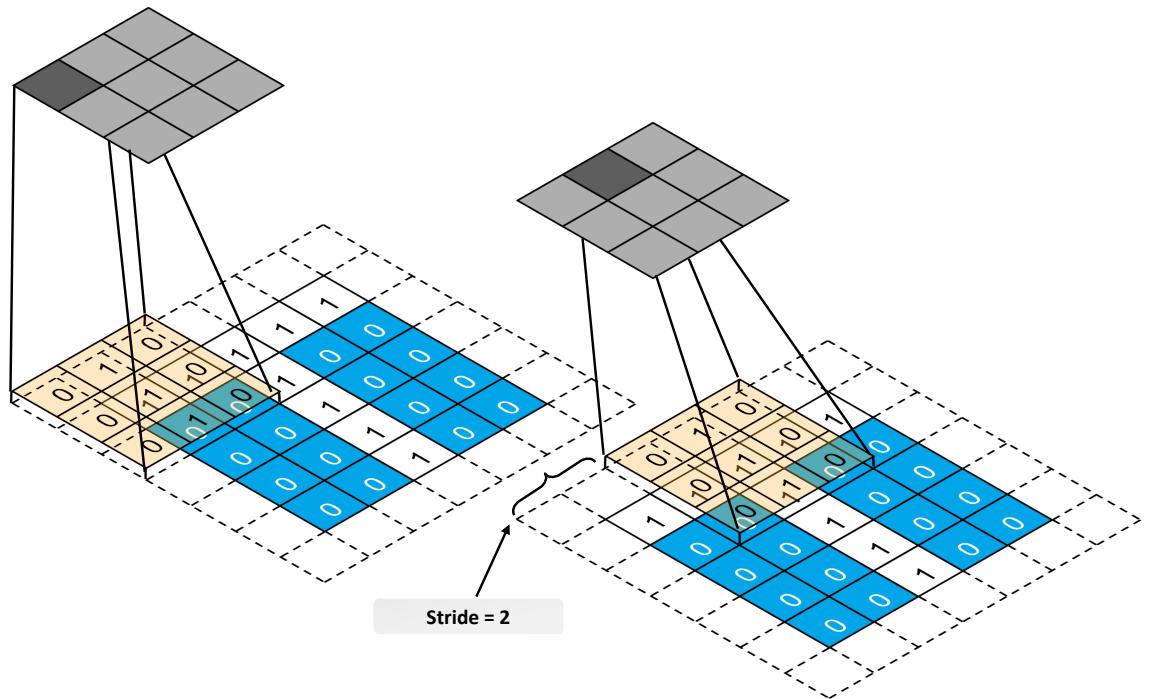


Figure 10-13: A stride of 2 enables this smaller convolutional layer to perform a downsampled scan of a large input layer.

Pooling Layers

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.

Previous Feature Map

1	3	1
0	3	0
0	3	0

1	3	1
0	3	0
0	3	0

New Feature Map

3	3
3	3

1	3	1
0	3	0
0	3	0

1	3	1
0	3	0
0	3	0

Figure 10-14: 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.

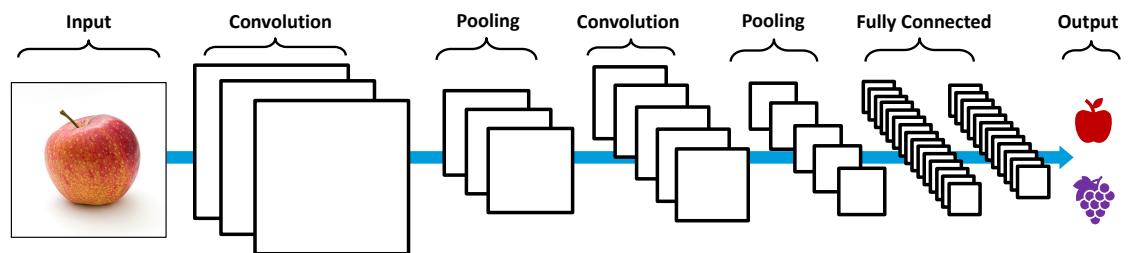


Figure 10–15: An example CNN architecture. Note that the image gets smaller as it goes through the network, while also generating more feature maps.

Generative Adversarial Networks (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 estimates 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 estimates 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.

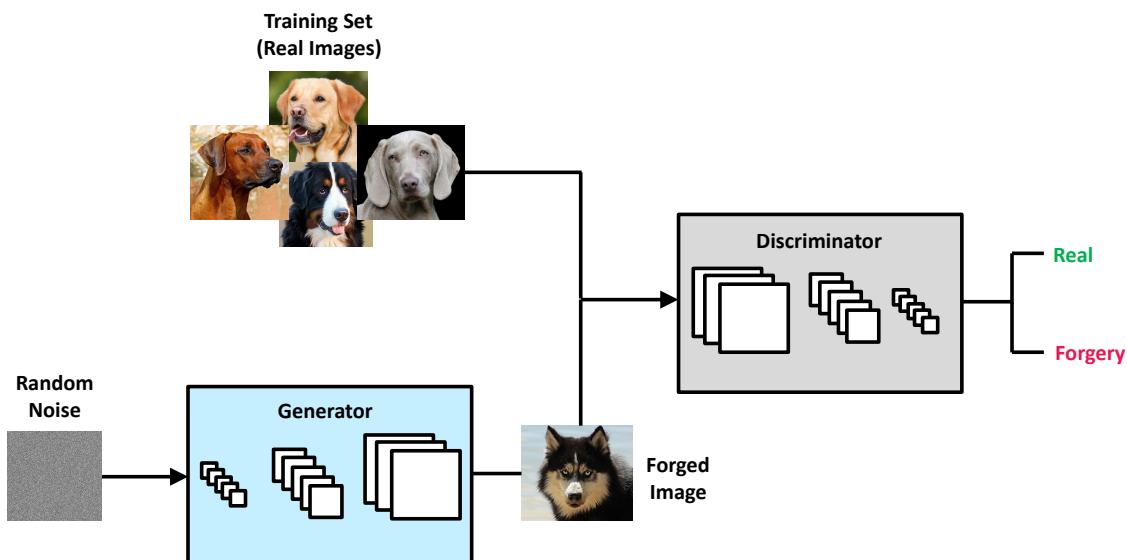


Figure 10–16: A general GAN architecture used for image generation.

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')` —Create a more visual representation of the network structure and save 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/Neural Networks - Fashion.ipynb

Before You Begin

Jupyter Notebook is open.

Scenario

You work for an online clothing retailer that wants to develop an automated system for classifying new clothing products that the business begins to sell. The product management team should be able to just feed the system an image of a new article of clothing and have it output a category that can be assigned to the article's database entry. The category will make it easier for users to find what they're looking for in the online store.

You have a rather large database of existing product images that have already been categorized and preprocessed. 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. From Jupyter Notebook, select **CAIP/Neural Networks/Neural Networks - Fashion.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 cell below it.
 - b) Run the code cell.
 - c) 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. This is similar to the clothing dataset you preprocessed earlier in the course, though the data itself is from a different source.

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, follow this link: https://github.com/zalandoresearch/fashion-mnist .
	Note: As the warning states, you can ignore it because the VM doesn't have a GPU.

3. Get acquainted with the dataset.
 - a) Scroll down and view the cell titled **Get acquainted with the dataset**, and examine the code cell below it.
 - b) Run the code cell.

- c) 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.
- d) Scroll down and examine the next code cell.

```
1 class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
2                 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
3
4 for i in range(10):
5     print('{} {}'.format(class_names[i], np.unique(y_train)[i]))
```

- e) Run the code cell.
f) Examine the output.

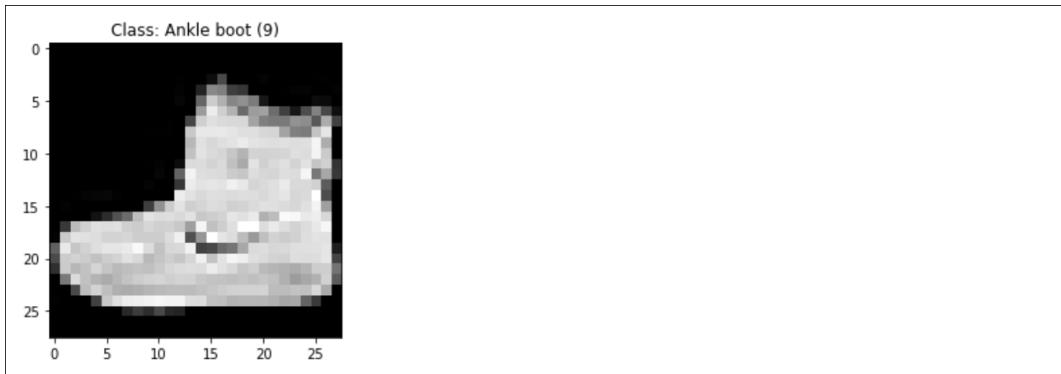
```
T-shirt/top (0)
Trouser (1)
Pullover (2)
Dress (3)
Coat (4)
Sandal (5)
Shirt (6)
Sneaker (7)
Bag (8)
Ankle boot (9)
```

Each class label is mapped to its actual class name (i.e., the type of clothing). There are 10 total classes.

4. Visualize the data examples.

- a) Scroll down and view the cell titled **Visualize the data examples**, and examine the code cell below it.
b) Run the code cell.

- c) 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).

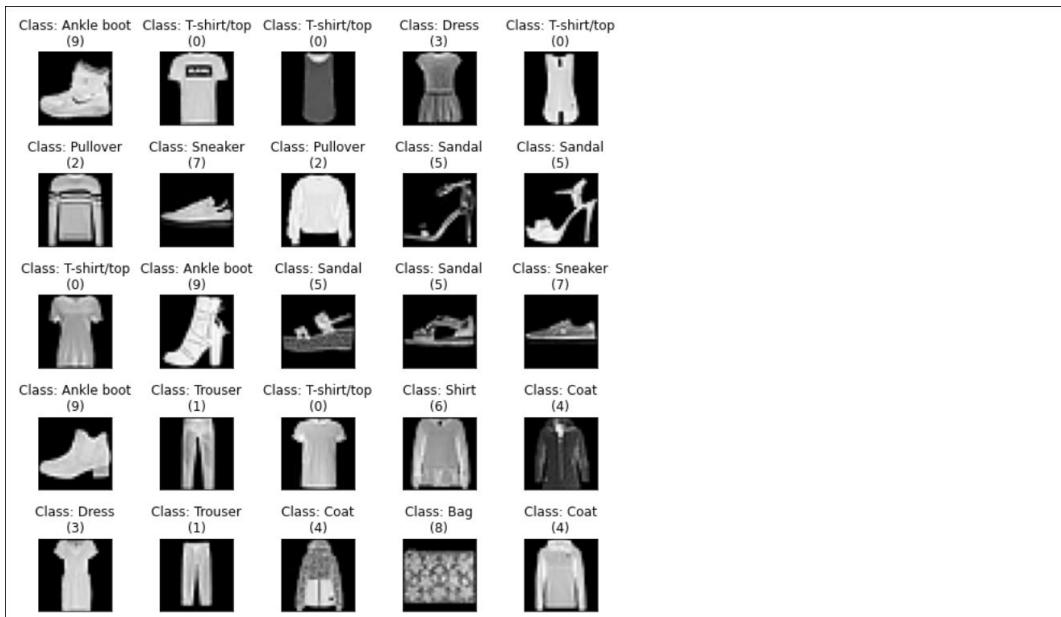
- d) Scroll down and examine the next code cell.

```

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()

```

- e) Run the code cell.
f) Examine the output.



The first 25 images from the training set are plotted in a grid.

5. 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 cell below it.

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) Run the code cell.
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.

6. Split the dataset.

- a) Scroll down and view the cell titled **Split the dataset**, and examine the code cell below it.

You're splitting the initial training dataset in order to have a validation holdout. The test set you loaded at the start will be treated as the ultimate test case.

- b) Run the code cell.
c) Examine the output.

```
Training features: (45000, 28, 28, 1)
Validation features: (15000, 28, 28, 1)
Training labels: (45000, 10)
Validation labels: (15000, 10)
```

7. Build the CNN structure.

- a) Scroll down and view the cell titled **Build the CNN structure**, and examine the code cell below it.

This code builds the actual structure of the CNN with Keras.

- On line 5, 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) Run the code cell.
c) Examine the output.

The CNN structure has been built.



Note: As before, you can ignore the TensorFlow warnings. They are just pointing out that certain hardware drivers are unavailable in the VM.

8. 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 cell below it.

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) Run the code cell.

- c) Examine the output.

Model: "sequential"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	320
leaky_re_lu (LeakyReLU)	(None, 28, 28, 32)	0
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18496
leaky_re_lu_1 (LeakyReLU)	(None, 14, 14, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_2 (Conv2D)	(None, 7, 7, 128)	73856
leaky_re_lu_2 (LeakyReLU)	(None, 7, 7, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 10)	20490

Total params: 113,162
 Trainable params: 113,162
 Non-trainable params: 0

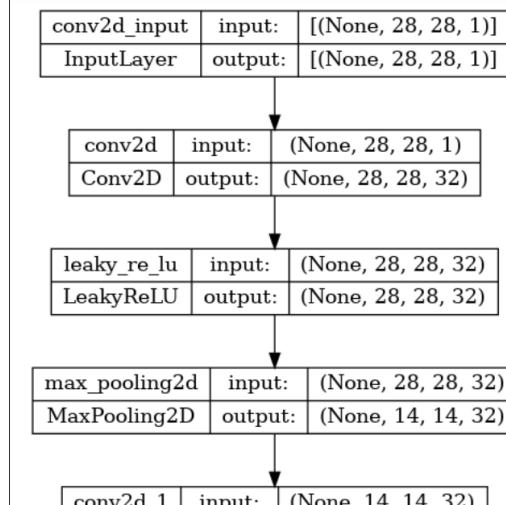
This provides an overview of the CNN's structure. It lists each layer's type, its output shape, and the number of parameters at each convolution. The number of parameters is defined as: number of output filters \times (number of input filters \times filter size + 1).

- d) Scroll down and examine the next code cell.

This code will produce a more visual method for displaying the different layers of the CNN.

```
1 from keras.utils import plot_model
2 plot_model(cnn, show_shapes = True)
```

- e) Run the code cell.
 f) Examine the output.



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).

9. Train the model.

- a) Scroll down and view the cell titled **Train the model**, and examine the code cell below it.

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) Run the code cell.
c) Examine the output.

```
245/1407 [====>.....] - ETA: 1:14 - loss: 1.3290 - accuracy: 0.7662
```

Keras provides a way to observe the progress of the training while it is underway.



Note: It will take a few 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.

```
1407/1407 [=====] - 104s 74ms/step - loss: 0.6168 - accuracy: 0.837  
9 - val_loss: 0.3093 - val_accuracy: 0.8855
```

10. 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 cell below it.

Since your test set is labeled, you'll evaluate the model on that set for the final fit.

- b) Run the code cell.
c) Examine the output.

```
Loss: 0.34  
Accuracy: 88%
```

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.

11. Make predictions on the test data.

- a) Scroll down and view the cell titled **Make predictions on the test data**, and examine the code cell below it.
b) Run the code cell.
c) Examine the output.

```
313/313 [=====] - 9s 27ms/step  
Actual class: [9 2 1 1 6 1 4 6 5 7]  
Predicted class: [9 2 1 1 6 1 4 6 5 7]
```

For the first 10 images, all of the predictions align with the actual class labels.

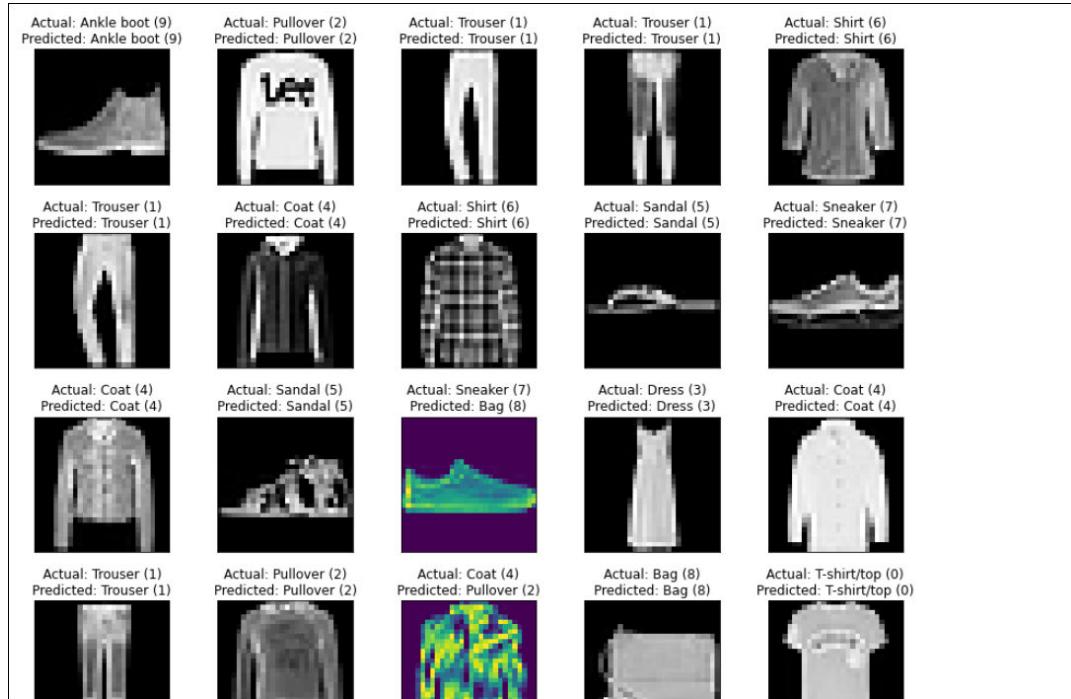
12. Visualize the predictions for several examples.

- a) Scroll down and view the cell titled **Visualize the predictions for several examples**, and examine the code cell below it.

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.

- Run the code cell.
- Examine the output.



Of the first 25 images, 3 were incorrectly classified.

13. 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?

14. What are some ways you might retrain this CNN model to improve its skill?

15. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **Neural Networks - Fashion** tab in Firefox, but keep a tab open 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 Networks (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.

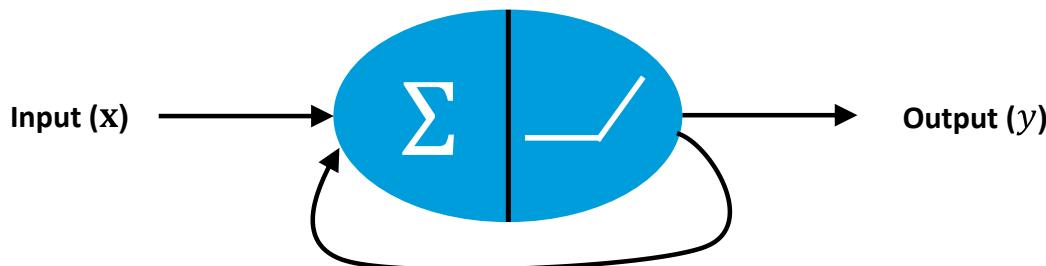


Figure 10-17: A single recurrent neuron.

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.

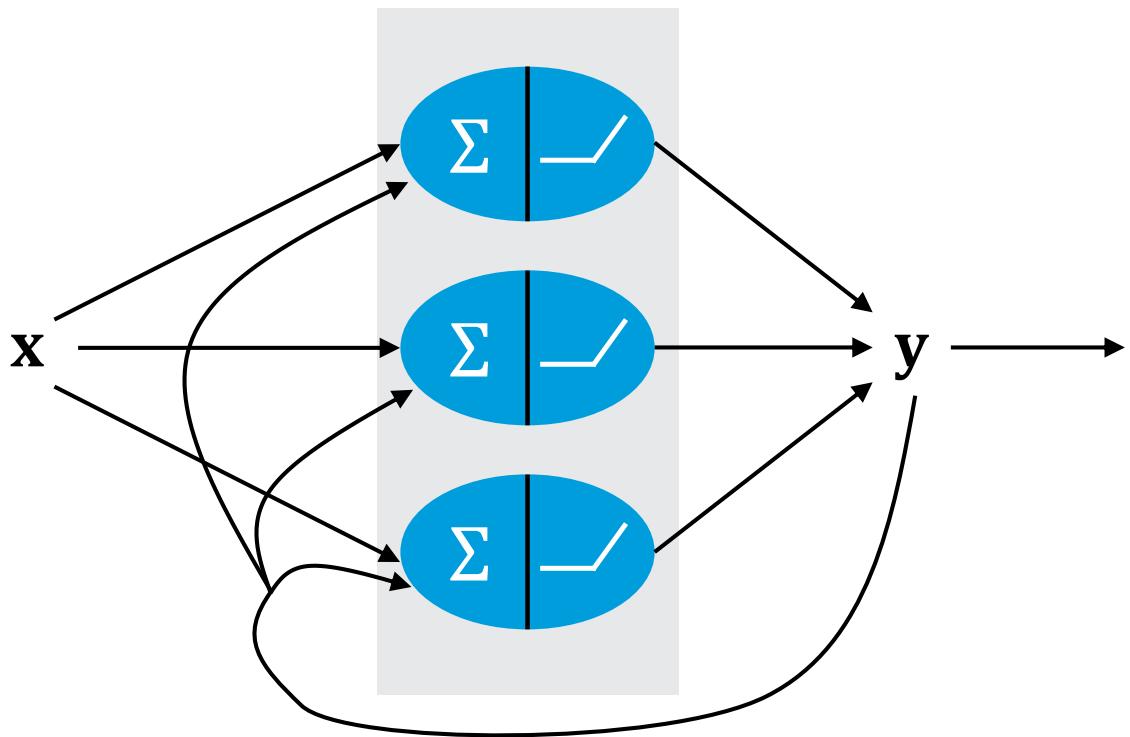


Figure 10-18: Several recurrent neurons in a layer.

When multiple RNN layers are represented in a time sequence—a process that is also called *unrolling*—you get something like the following figure.

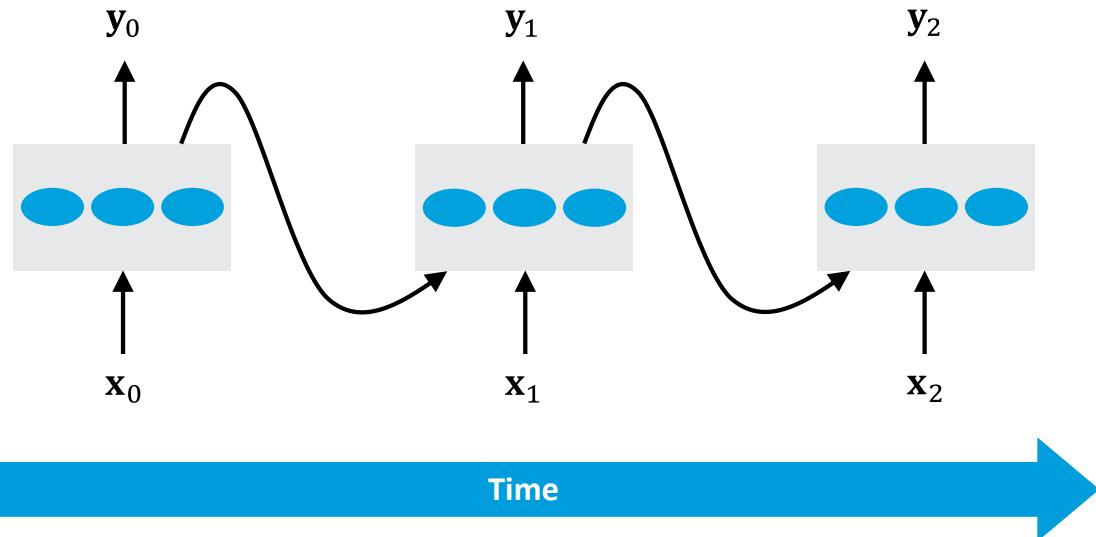


Figure 10-19: Unrolling several layers of recurrent neurons through time.

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 Cells

A recurrent neuron or a layer of recurrent neurons has something akin to memory—it is able to take the state of a neuron/layer in one time step and incorporate that state into the calculations of the

next time step. 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.

A memory cell has a current state at some time step t called a *hidden state*. This hidden state is often represented as the vector \mathbf{h}_t . A cell also has an output vector at time step t represented as \mathbf{y}_t . A cell's hidden state is a function of the previous hidden state (\mathbf{h}_{t-1}) and the current input (\mathbf{x}_t). Likewise, the output is also a function of the previous hidden state and the current input.

In the most basic memory cells, like the one shown in the figure, the hidden state of the cell is the same as its output. In more complex cells, the two may be different.

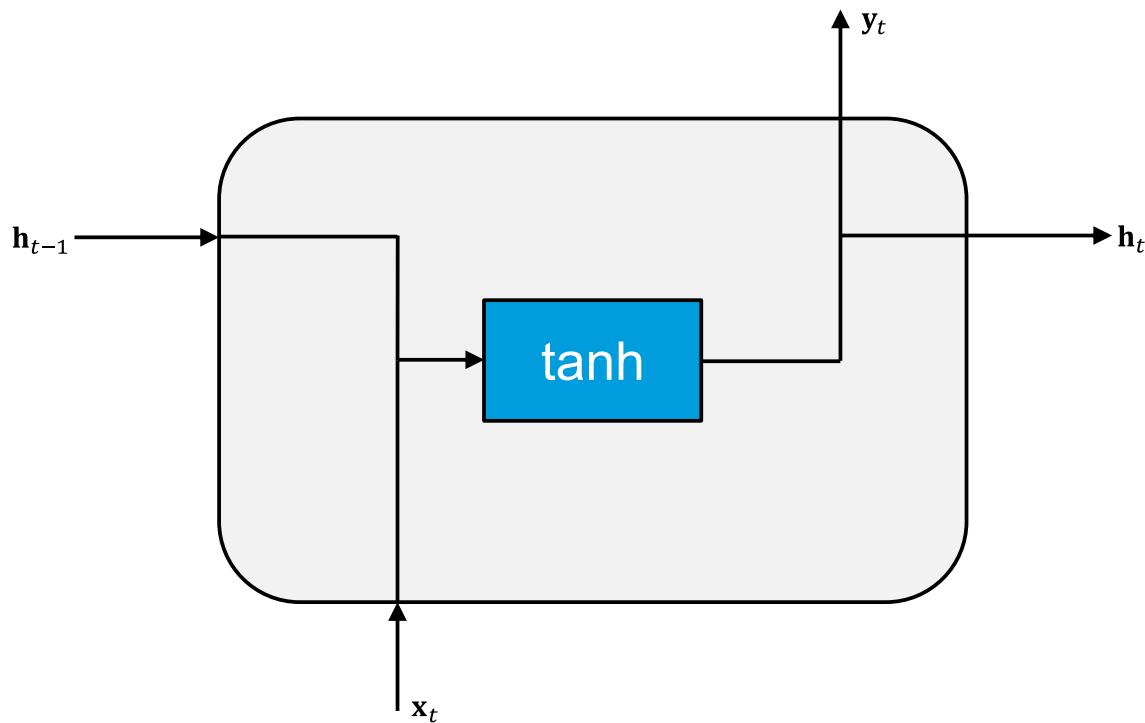


Figure 10-20: A basic memory cell with input, output, and hidden state values. The tanh activation function is applied.

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 estimated 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) Cells

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.

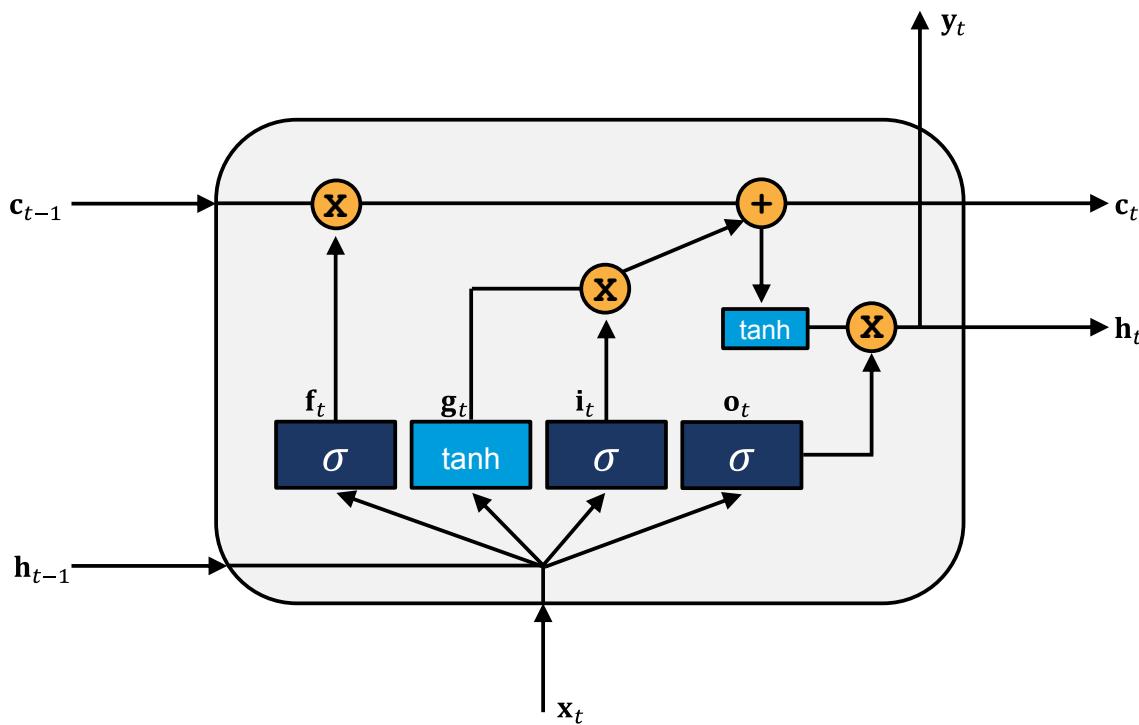


Figure 10-21: The architecture of an LSTM cell.

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.

Text Data Processing for RNNs

The text transformation and processing techniques discussed earlier in the course are commonly applied to RNNs. This is particularly true in the case of word embeddings, as the reduction in vector

dimensions significantly eases the computational burden of training on text data. As the RNN trains on these embeddings, it is able to arrange these embeddings so that similar words are grouped together, increasing the network's efficiency and effectiveness at whatever task it was designed to do. The most skillful RNNs can group words according to semantics, grammatical function, and contextual meaning. For example, an RNN might place all words describing color together; it might separate nouns from verbs; it might divide words describing positive emotions from those describing negative emotions; and so on.

Remember, you can rely on the word embeddings created by others in the machine learning field if you'd rather not create them yourself. In either case, you should consider applying preprocessing techniques like bag of words, stop words, and lemmatization to your text data before building an RNN.

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 long short-term memory (LSTM) cells 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.
- Use word embeddings to condense the language vocabulary into relatively small vectors.
- Use preprocessing techniques like bag of words and lemmatization before inputting text data to an RNN for training.

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 10-3

Building an RNN

Data File

/home/student/CAIP/Neural Networks/Neural Networks - IMDb.ipynb

Before You Begin

Jupyter Notebook is open.

Scenario

Another major component of your online storefront is user 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.

You want to perform sentiment analysis on these reviews to determine how customers feel about a product. Sentiment analysis is 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. From Jupyter Notebook, select **CAIP/Neural Networks/Neural Networks - IMDb.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 cell below it.
 - b) Run the code cell.
 - c) Verify that the training and testing sets were both loaded with 25,000 records.
 - This is the full version of the IMDb movie review dataset you worked on earlier.
 - The Keras version of the dataset has already been preprocessed. The approach taken is somewhat different than the approach you took earlier.
 - 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.
3. Get acquainted with the dataset.
 - a) Scroll down and view the cell titled **Get acquainted with the dataset**, and examine the code cell below it.
 - b) Run the code cell.

- c) 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, 1
00, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4,
172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 46
13, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 6
26, 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, 1
4, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 47
6, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141,
6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 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, 5
345, 19, 178, 32]
```

Label: 1

- This output shows the features from just the first 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. A bag of words was likely created from the initial dataset to identify the frequency of each word. 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.

- d) Scroll down and examine the next code cell.

```
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
5 # Replace unknown words with '?'.
6 decode = ' '.join([index_dict.get(i - 3, '?') for i in X_train[0]])
7 decode
```

- Lines 2 and 3 will translate each numerical feature value to the word it represents. This is done by mapping the values to an existing dictionary for the dataset.
- Line 6 will replace a word that is not known with a question mark (?).

- e) Run the code cell.

- f) Examine the output.

```
"? this film was just brilliant casting location scenery story direction everyone's really suited the part they played and 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 throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for ? and would 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 also ? 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 that 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 all that was shared with us all"
```

- 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.
- You could continue to process the text by removing stop words or extracting the lemmas, but this should suffice for now.
- Reading the text, you can see why it was labeled as being positive.

4. Examine some statistics about the reviews.

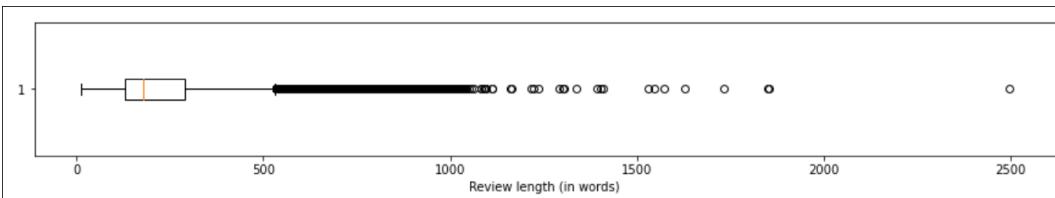
- Scroll down and view the cell titled **Examine some statistics about the reviews**, and examine the code cell below it.
- Run the code cell.
- Examine the output.

```
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.
- Scroll down and examine the next code cell.

```
1 plt.figure(figsize = (15, 2))
2 plt.boxplot(result, vert = False)
3 plt.xlabel('Review length (in words)')
4 plt.show()
```

- Run the code cell.
- 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.

5. Add padding to the data.

- Scroll down and view the cell titled **Add padding to the data**, and examine the code cell below it.
 - 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 that same size. This will make all of the input the same shape, simplifying the model.
- Run the code cell.
- 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.

6. Split the dataset.

- Scroll down and view the cell titled **Split the dataset**, and examine the code cell below it.
You're splitting the initial training dataset in order to have a validation holdout. The test set you loaded at the start will be treated as the ultimate test case.
- Run the code cell.
- Examine the output.

```
Training features: (18750, 500)
Validation features: (6250, 500)
Training labels: (18750,)
Validation labels: (6250,)
```

7. Build the RNN structure.

- Scroll down and view the cell titled **Build the RNN structure**, and examine the code cell below it.
This code builds the actual structure of the RNN with Keras.
 - On line 5, 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.
- Run the code cell.
- Examine the output.

```
The RNN structure has been built.
```

8. What is word embedding, and why might it be beneficial to use in this case?

9. 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

10. 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 cell below it.

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) Run the code cell.
c) Examine the output.

```
Model: "sequential"
-----  

Layer (type)      Output Shape       Param #  

=====  

embedding (Embedding)    (None, 500, 100)   1000000  

lstm (LSTM)        (None, 64)          42240  

leaky_re_lu (LeakyReLU)  (None, 64)          0  

dense (Dense)      (None, 128)         8320  

leaky_re_lu_1 (LeakyReLU) (None, 128)         0  

dense_1 (Dense)    (None, 1)           129  

=====  

Total params: 1,050,689  

Trainable params: 1,050,689  

Non-trainable params: 0
```

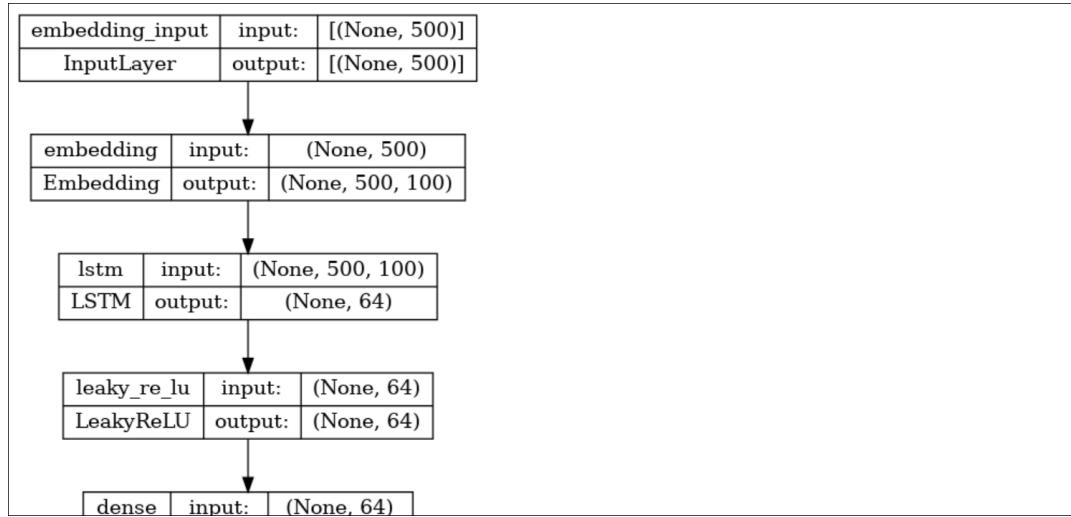
This provides an overview of the RNN's structure. As with the CNN, each layer is listed, along with its output shape and number of parameters.

- d) Scroll down and examine the next code cell.

```
1 from keras.utils import plot_model  
2 plot_model(rnn, show_shapes = True)
```

- e) Run the code cell.

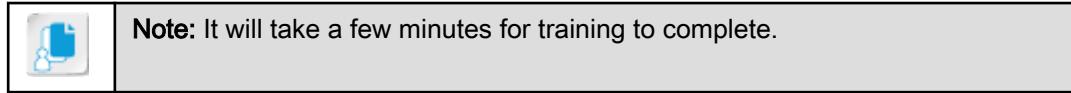
- f) Examine the output.



11. Train the model.

- Scroll down and view the cell titled **Train the model**, and examine the code cell below it.
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.
- Run the code cell.
- Examine the output.

```
64/586 [==>.....] - ETA: 1:58 - loss: 0.6586 - accuracy: 0.5977
```



- While you wait, observe how the training loss decreases over time, while the accuracy gradually increases.
- When training is complete, examine the final scores on the validation set.

```
586/586 [=====] - 152s 258ms/step - loss: 0.4438 - accuracy: 0.7955
- val_loss: 0.3338 - val_accuracy: 0.8648
```

12. Evaluate the model on the test data.

- Scroll down and view the cell titled **Evaluate the model on the test data**, and examine the code cell below it.
Since your test set is labeled, you'll evaluate the model on that set for the final fit.
- Run the code cell.

- c) Examine the output.

Loss: 0.34
Accuracy: 86%

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.

13. Make predictions on the test data.

- a) Scroll down and view the cell titled **Make predictions on the test data**, and examine the code cell below it.
 - b) Run the code cell.
 - c) Examine the output.

```
4/4 [=====] - 1s 56ms/step  
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.

14. Examine a review that was correctly classified.

- a) Scroll down and view the cell titled **Examine a review that was correctly classified**, and examine the code cell below it.
 - b) Run the code cell.
 - c) Examine the output.

- 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).

15. Examine a review that was incorrectly classified.

- a) Scroll down and view the cell titled **Examine a review that was incorrectly classified**, and examine the code cell below it.
 - b) Run the code cell.

- c) Examine the output.

- 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.

16. Why do you think the model incorrectly classified this review as positive?

17.What are some ways you might retrain this RNN model to improve its skill?

18. Shut down this Jupyter Notebook kernel.

- 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.

11

Operationalizing Machine Learning Models

Lesson Time: 2 hours

Lesson Introduction

You've developed many different machine learning models throughout the course, but you've worked on them in isolation. In the real world, you'll need to incorporate your models into a wider context so that they can actually provide value to your business. So, in this lesson, you'll learn how to take the models you've created (and will create in the future) and put them into operation. That way, they'll truly be able to start solving your business problems.

Lesson Objectives

In this lesson, you will:

- Deploy machine learning models so that they are accessible to a wider set of users and developers.
- Automate the preparation of data and the building of machine learning models using pipelines.
- Integrate machine learning models into existing systems that can take advantage of the models' intelligent decision-making capabilities.

TOPIC A

Deploy Machine Learning Models

In order for you and your stakeholders to take advantage of the models you've created, you'll need to deploy your models to an environment that makes them available for use.

Deployment of Machine Learning Models

In the field of machine learning, **deployment** refers to taking a model and putting it into a production environment so that users or services can feed the model input, as well as receive output from the model. In other words, they are able to acquire resources produced by the model in a streamlined way, without having to interface directly with the underlying model code.

"User," in this case, refers not just to a typical end user, but to anyone who is given permission to leverage the model to make business or personal decisions. A user might be a customer browsing an online store, completely unaware that a machine learning model is serving them product suggestions; or it could be a business professional who uses an app that relies on a machine learning model to project sales. Likewise, a service can be any software that receives output from a model without requiring direct human intervention, like a mobile weather app that constantly updates forecasts in the background and sends a notification to the user if a storm is on the way. There are many such use cases for deployment.



Note: Users and services can be thought of as "consumers" of a model.

Offline vs. Online Models

When it comes to model training during deployment, there are two major paradigms: offline and online. **Offline models** are trained every so often on batches of new data. The new data is input to the model, which then learns new parameters. The model is updated on the new data all at once.

Online models, on the other hand, are continuously trained on new data. The model learns from each new training example as it flows in. So, it is constantly updating its parameters.

There are some clear advantages and disadvantages to each approach:

- Online models are best at keeping a model's assumptions about the target domain as fresh as possible. Its parameters are always updating to account for the latest data. Therefore, they may outperform equivalent offline models.
- Offline training can finish much more quickly since it just happens once per cycle rather than continuously. However, very large training datasets might actually slow the training process down in an offline setting.
- Offline training usually requires less computational power for much the same reasons. Online training needs to constantly use computing resources.
- Offline models are easier to manage since machine learning engineers have time to review and evaluate the new data coming in before it gets used in training. It's not so easy to do this with online models since they're constantly accepting new data.
- Online models are usually exposed to greater risk from corrupted or malicious data since they are constantly being trained on incoming data. For example, some navigation apps collect user data in order to determine optimal routes. Users might be able to report false data about increased traffic in an area so that the app routes traffic to other neighborhoods.
- Offline models may be the only realistic choice in certain scenarios. For example, a model that forecasts the weather can constantly receive new weather data, so it makes sense for it to be online. On the other hand, if your model performs sentiment analysis on text, then you may not

be able to obtain a constant stream of new, labeled text examples with which to update the model. Therefore, it must be offline.

Deployment Methods for Model Output

Offline and online training are used to describe the *training* portion of deployment. There is also the *output* aspect of a deployed model—how the model makes predictions and other estimations.

There are a few deployment approaches you can use to service model output. The simplest is not really a "deployment" per se, as it just involves creating a model that produces a one-off output whenever it is requested. This ad hoc output method is most applicable to scenarios based around a single event, or in scenarios where simple predictions are adequate. For example, your company is looking to move offices, and you've built a model to take data about an office and assign it a rank or score. The CEO retrieves these scores from the deployed model, which only really needs to happen once. This is the kind of task that can happen in a closed environment, rather than something that needs to be deployed for a wider audience.

As for actual deployment methods, they can be categorized like so:

- Batch deployment
- Real-time serving
- Streaming

Batch Deployment

Models deployed in a batch format generate outputs on a predefined schedule. For example, if the business wants to predict how much product they'll have in stock for the following month, they could take advantage of batch deployment to make that prediction on a monthly schedule. The model might be trained in an online fashion, where it continuously receives inventory data for the current month that it uses to update its parameters. Then, at the end of each month, the model outputs a prediction.

Batch deployment is therefore most applicable to scenarios where predictions and other decisions need be repeated according to some cycle.

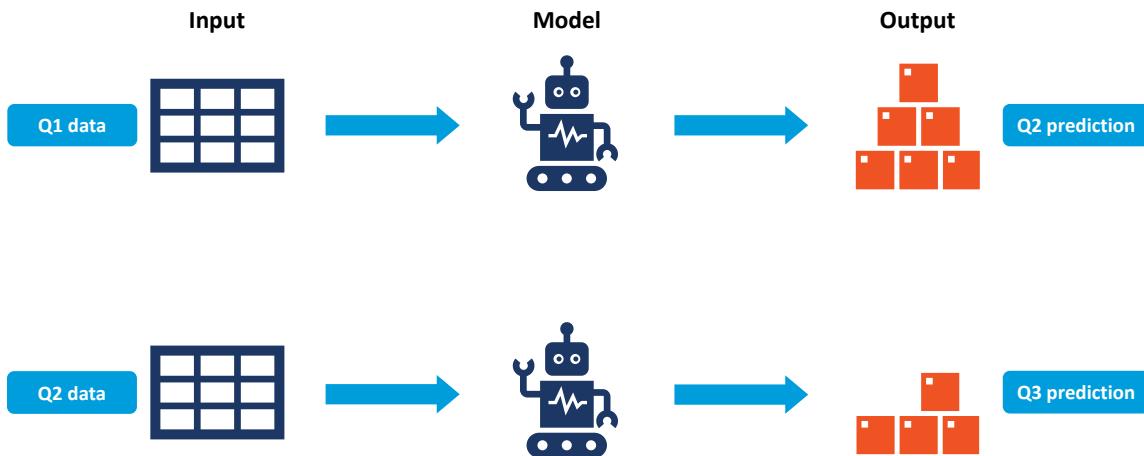


Figure 11-1: A batch deployment process.

Real-Time Serving

Real-time serving involves users or services who access the model's outputs on demand at a high, consistent rate. Typically, the model will be accessible through an application programming interface (API) layer, perhaps using HTTP-formatted calls. The user or service issues an HTTP request, and

the deployment environment instructs the model to generate the necessary outputs, which then get sent to the requesting user or service in real time. The model may be trained in an offline or online fashion depending on the needs of the business and its users.



Note: Batch deployment methods can also use APIs to handle input, though they are not as necessary as they are in real-time serving.

Real-time serving is most appropriate in situations where requests for machine learning outputs are expected constantly, and there needs to be a layer of abstraction in front of the model for the deployment to work effectively. For example, every time a user logs onto an online store, they will be suggested new products to buy. The store needs to make a call to the recommendation system for each user in real time so that the users don't log off before they see the recommendations.

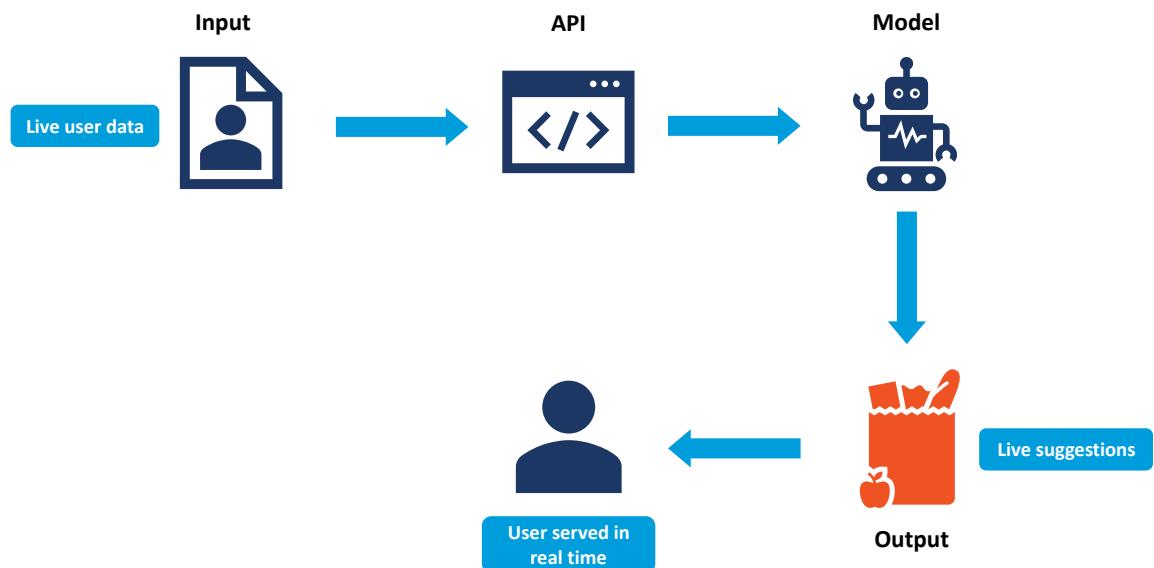


Figure 11-2: A real-time serving process.

Streaming

Streaming is similar to real-time serving, but instead of working synchronously with each request, it works asynchronously using a queuing system. A user or service sends a request to a piece of middleware called a message broker, which queues up the request along with any others that are waiting to be processed. The model is continuously processing the request data as it clears the queue, and delivers outputs to the data stream as quickly as it is able to. The requesting user or service may not receive the output instantly, but streaming can alleviate performance issues that real-time deployments often face. Models in streaming deployments may be trained offline or online.

Streaming is most appropriate for situations where the demand for model outputs is high and fluctuates, and when those outputs are not needed right away. For example, a bank may routinely check transactions for evidence of fraud. Many transactions are sent to the model as they are initiated, but the bank doesn't need a fraud report instantly, so it can wait for the streaming server to process the request.

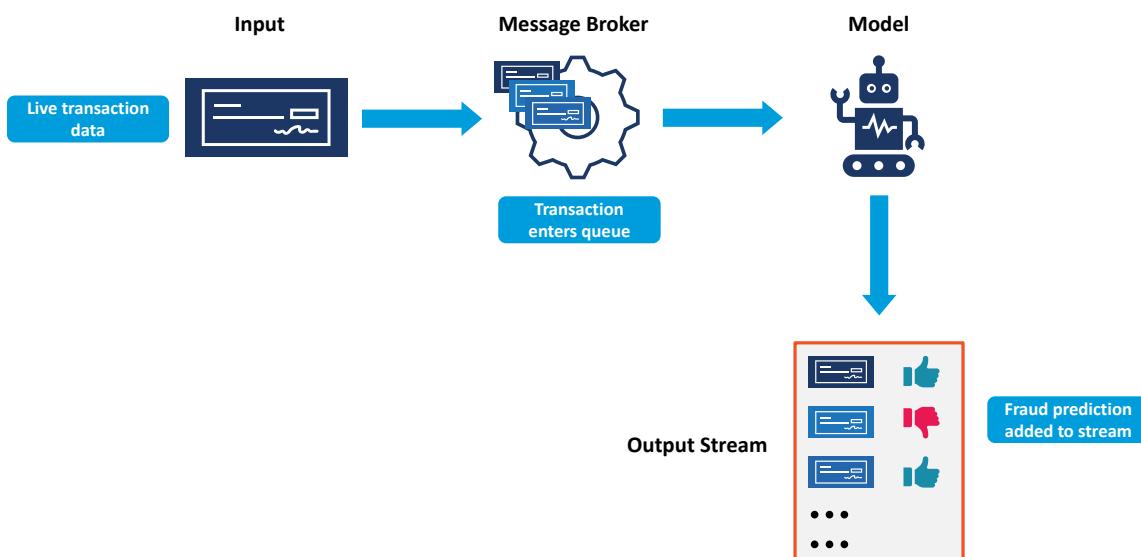


Figure 11–3: A streaming process.

Deployment of Deep Learning Models

Although the deployment paradigms and methods just mentioned will work with deep learning models, these models often require some special considerations during the deployment phase. Highly complex neural networks take up a great deal of processing power and storage space, sometimes orders of magnitude larger than a traditional machine learning model. This is especially the case during the training phase, so if your deep learning models are online and need to be retrained constantly, you should be aware of how this can impact the availability of outputs.

In order to keep neural networks efficient in deployment, they must have access to adequate hardware—everything from multi-core CPUs to high-volume RAM, and in many cases, GPUs or equivalent processors. This can be a difficult challenge for some organizations to meet, which is where a discussion of infrastructure requirements comes into play.

Distributed Artificial Intelligence (DAI)

The deployment needs of deep learning models in some business contexts has given rise to the concept of **distributed artificial intelligence (DAI)**, in which learning "agents" (independent software resources that implement one or more capabilities in a machine learning workflow) run parallel operations from geographically dispersed locations. The agents can communicate with one another and facilitate necessary changes without requiring a centralized platform to change. This results in an overall deployment that is more flexible and able to adapt to change, while also seeing gains in performance.

Infrastructure Requirements

With traditional software, your deployment strategy might be to just give your users a link to where they can download the software. Or, you might publish the software to an app store. Deployment of machine learning and deep learning models is not as straightforward as simply giving users direct access to the product you're providing. In most cases, you'll need to host the model on an infrastructure comprising one or more servers that can process incoming requests and deliver the desired outputs. This shifts the burden of processing away from the consumer and onto the provider, where the models can be more easily centralized and controlled.

Every project's infrastructure requirements will be different. Nevertheless, it's better to be proactive about designing the infrastructure at the beginning stages, rather than stay reactive and just wait for

a problem to arise before addressing it. Consider asking yourself the following questions when planning your deployment infrastructure:

- What will our model need to output to the consumer? Predictions? Classifications? Recommendations? Something else? What format should the output be in?
- How much data will the model need to accept as input from a consumer in order to produce an adequate output? What if we need to accept less data than what the model was trained on? Can we fill in the gaps somehow?
- How often will the model need to be retrained, and what will be the source of the data used in training?
- What deployment method fits our needs and the needs of the consumers the best?
- Will the model be accessible by the public at large, or will it be internal to one or more organizations?
- How many consumers do we expect to use the model in a given time period? How crucial is it that they receive instant results?
- What sort of data replication strategy will we need to ensure the model is able to serve multiple users concurrently, without leading to loss of availability?
- What capabilities does the platform or software that runs the deployment environment need to have?
- What happens when outputs (e.g., recommendations) are not available? Is there a fallback strategy in place?
- What monitoring and alerting strategy needs to be in place to detect issues in deployment?
- Are we capable of hosting the infrastructure ourselves, or do we need to outsource it to a third party?



Note: The answer to the last question might be decided by your answers to the prior questions.

Stakeholder Requirements

Just as stakeholders have requirements for the beginning stages of a machine learning project, so too do they have requirements for the final stages of the project. Consider how each of the following stakeholders may provide input regarding deployment:

- **Customers/end users** will be the ones actually using the product or service that your model supports, even if they're not aware of it. Whether they understand the machine learning process or not, they'll likely expect timely and accurate results. Business clients may have more well-defined requirements, like expecting results to take no longer than a certain number of seconds to appear.
- **Sponsors/champions** will want to ensure that the deployment phase is progressing on time and within budget. The latter is of particular concern, as a project can quickly become expensive when deployed to an infrastructure.
- **Program/project managers** will want to ensure that the machine learning engineers who deploy the model have all of the resources they need to get the job done.
- **Team members** who actually apply their technical skills to deploy the model will want to ensure they are working with the correct platform and toolset for the job, and that they are given all they need from their colleagues, especially if the practitioners who develop the models are not the same ones who deploy it.
- **Business partners** may be involved in ensuring there is an adequate deployment infrastructure, and therefore shoulder some of the financial or operational burden.
- **Governments** may impose laws or regulations on how personal data is used, which becomes a significant issue during the deployment phase when the organization starts accepting new data inputs for the model to work on. Also, in the future, there may be laws that mandate explainable results in AI systems.

- **Societies** may demand that the organization make a commitment to deploy its machine learning product ethically, especially now that the model will start impacting a wide audience.

Deployment Endpoints

In model deployment, an **endpoint** acts as an intermediary between the consumer and one or more deployment environments. Consider that, in order to alleviate performance issues with batch or real-time deployment, you spread out your processing capabilities among multiple servers or compute units. Server A has a copy of the model, as does Server B. When a consumer requests the model's resources, it must first go through the endpoint, which determines where to route the traffic and how. In this sense, the endpoint acts similar to a load balancer in computer networking. It ensures that no one server is overwhelmed with traffic or processing tasks.

Another application of an endpoint is to manage multiple versions of a model. Version A might be trained on one set of data, and Version B might be on another set of data. Or, Version 1 might be an older model that consumers still need access to, and Version 2 is the newer model. The endpoint in this case must make a decision regarding where to route the request based on more than just server load. The request itself might contain information that tells the endpoint where to send it, or the endpoint may need to infer the optimal destination.

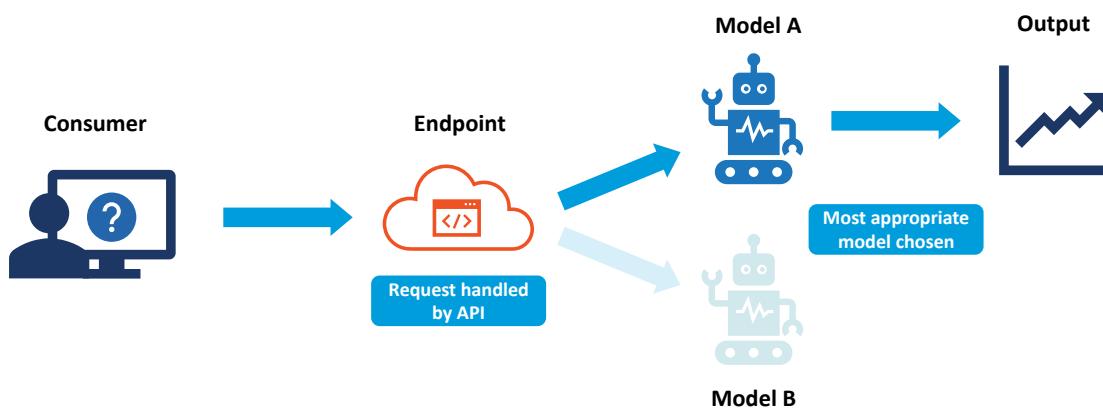


Figure 11–4: A deployment endpoint that routes input requests to one of two models. In this case, the request is routed to Model A.

In any case, deployment endpoints are usually accessed through HTTP requests to an API that has a specific Uniform Resource Identifier (URI). Endpoints may also be set up differently depending on the deployment method being used. For example, in a real-time deployment, the endpoint will route the consumer request to the most appropriate and available server. On the other hand, in a batch deployment, the endpoint might split up a processing job from a single request across multiple servers, which can help minimize the time it takes to work on large-scale tasks.

The Role of Cloud Services in Model Deployment

Given the potentially massive infrastructure requirements for deploying a model, it's no wonder that cloud services have become such a popular choice for organizations looking to operationalize their machine learning projects. It's no small feat to acquire the latest and most powerful hardware, physically maintain that hardware, run multiple servers in multiple clusters, troubleshoot issues with those servers, and so on. Even large organizations are leery of taking on this responsibility alone.

The primary advantage of the cloud is that all of the underlying resources required for a deployment infrastructure are hosted, maintained, and secured by a third-party platform. The infrastructure is provided to you, the client, through a layer of virtualization, rather than giving you direct access to the hardware. Another major advantage is the elasticity of the cloud. An elastic service is one that can scale up or scale down to meet demand. So, instead of running all servers 24/7 at full power,

you can add or remove servers to meet the current demand, and reduce or enhance the processing power of each server as needed. This ensures you are not wasting money on unnecessary infrastructure.

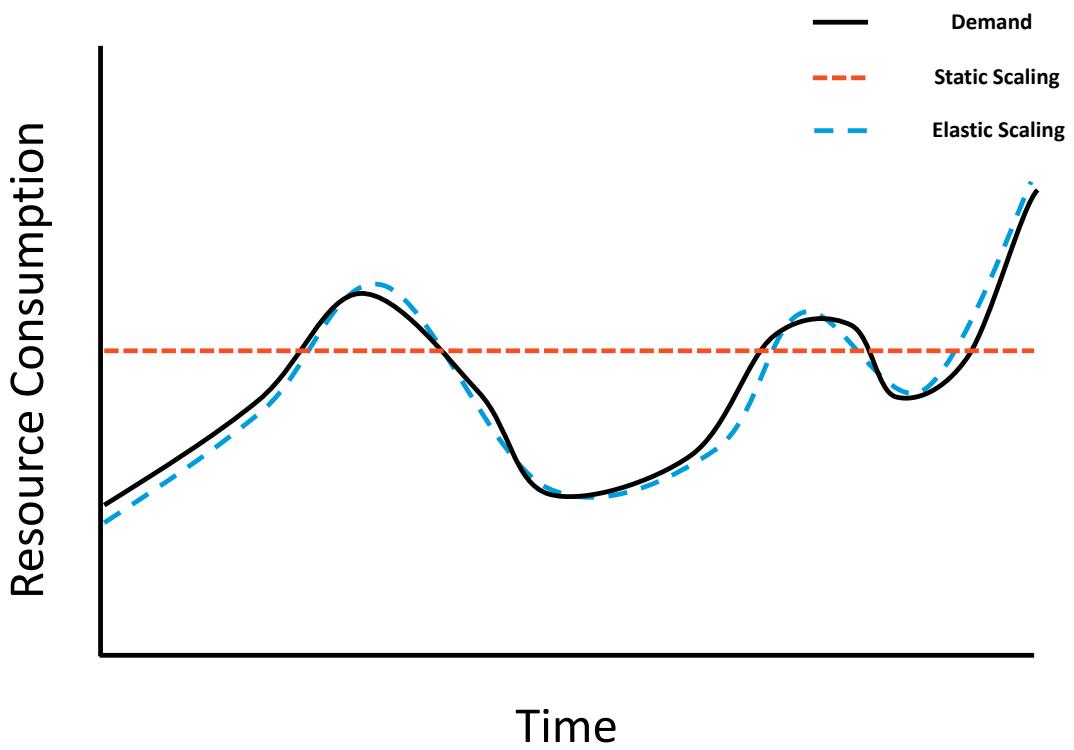


Figure 11-5: The elastic nature of cloud services compared to static resource scaling.

However, there are some clear disadvantages to outsourcing your machine learning deployment to the cloud, including:

- You no longer have complete control over the data, the models, or the infrastructure resources.
- You and your customers are at the mercy of the cloud service should it go down.
- You must trust the security practices of the cloud platform.
- Depending on certain laws and regulations you're beholden to, the level of security offered by the cloud platform may not be adequate for the data you work with.
- Instead of traditional system administrators, you'll need to hire or train cloud engineers. The two disciplines overlap in many ways, but still have distinct skills and responsibilities.
- The cost of using the cloud services, even with the benefits of elasticity, can quickly get out of hand.

Managed vs. Unmanaged Cloud Services

Cloud services can be managed, unmanaged, or some combination thereof. A managed service automatically provides you with the tools you need to deploy a machine learning model in the cloud. You're given stock deployment environments, configured endpoints, data pipelines, and so on. Many platforms allow you to configure these components to your liking. Managed services are therefore a more "plug-and-play" approach to deployment. Though, one disadvantage of managed services is that they are not fully customizable.

An unmanaged service, on the other hand, is. You are essentially given the virtual environments upon which to build your deployment architecture. You determine what software goes on the servers, how the servers will communicate with each other, how they will accept input from and

provide output to consumers, and so on. This approach requires more technical knowledge and IT investment (both upfront and in the long run), but is much more flexible.

There are also cloud platforms that provide a combination of both managed and unmanaged services for the best of both worlds.

Example Cloud Services for Model Deployment

There are several cloud platforms that provide many different services to machine learning professionals interested in deploying models. One platform or service is not necessarily better for all projects or organizations than the others. It's up to you to do more research to find out which one is right for you, should you choose to outsource your deployment to the cloud.

The three major players are Amazon Web Services (AWS), Microsoft® Azure®, and Google Cloud™. The main infrastructure as a service (IaaS) offerings for each are as follows:

- Amazon Elastic Compute Cloud (EC2)
- Azure Virtual Machines (VMs)
- Google Compute Engine™ (GCE)

For the most part, these are unmanaged services, so you'd have to set up the deployment architecture yourself. However, the three major cloud companies also offer managed services specific to machine learning that you should consider. The comprehensive services that offer a deployment component are:

- Amazon SageMaker
- Azure Machine Learning
- Vertex AI™ (Google Cloud)

All three of these services help practitioners build, deploy, and maintain machine learning models in the cloud. They also support many of the most popular Python® data science libraries, including scikit-learn, PyTorch, and TensorFlow.



Note: Amazon SageMaker was launched in 2017, Azure Machine Learning was launched in 2015, and Vertex AI was launched in 2021. Google launched Vertex AI to unify the disparate ML services that already existed on the platform.

Docker

[Docker](#) is an open-source platform for building and maintaining virtual containers. A virtual container provides a virtual environment in which an application can run without disrupting the host system. Although a form of virtualization, containers differ from virtual machines (VMs) in that they do not virtualize an entire operating system. Instead, an application in the container shares resources (e.g., binaries and libraries) provisioned by the host operating system's kernel while also being isolated from that OS or other containers.

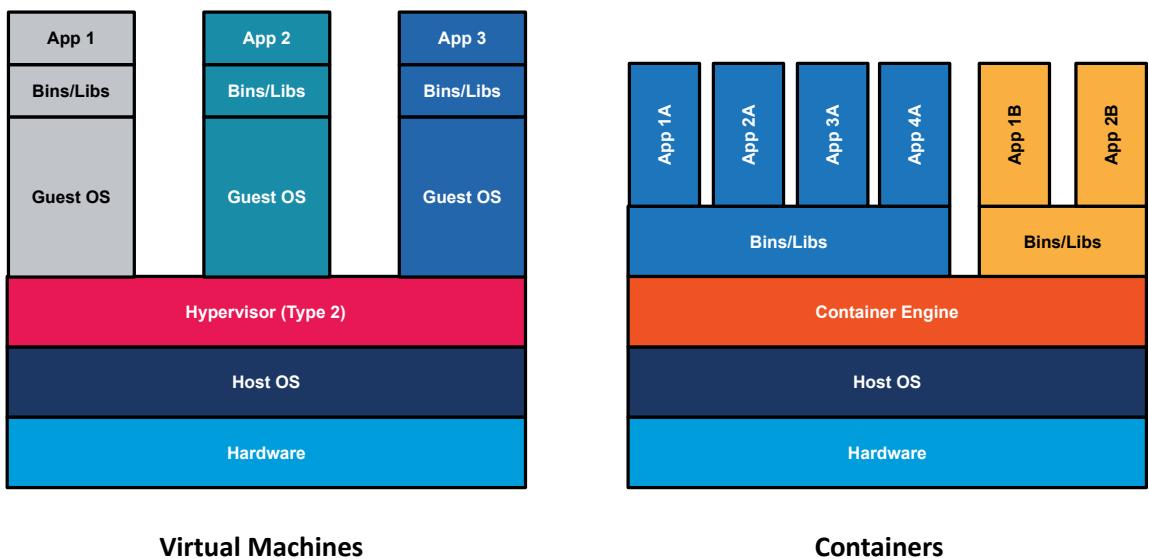


Figure 11-6: Virtual machines vs. containers.



Note: This figure demonstrates a type 2 hypervisor, like VirtualBox. In a type 1 hypervisor, the hypervisor runs directly on the hardware.

Because they only access resources when needed and don't contain an entire OS, containers are faster, more lightweight, and more portable than a traditional VM. It's common to run multiple containers side by side on a single host, where each container includes its own standalone application.

Although not specifically a machine learning deployment platform, Docker can facilitate deployment for a machine learning project due to its ability to run code in an isolated environment. You can deploy a single container, multiple containers that each hold a copy of the entire service, or you can split up different tasks across several containers. In any case, containerization ensures that each major task can be abstracted from both the underlying OS and the other containers, making it easier to configure that task without disrupting the others. Docker also helps make a production environment portable, since you can package each container and share them or integrate them into other systems as needed.

Docker can be installed locally, or it can be incorporated in a cloud platform. You may be able to configure Docker containers in unmanaged services, whereas managed services may use Docker containers as an underlying technology.

Guidelines for Deploying Machine Learning Models



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

Follow these guidelines when deploying machine learning models.

Deploy Machine Learning Models

When deploying machine learning models:

- Consider whether your model needs to be trained in an offline or online mode depending on the model's purpose, your available infrastructure, and other factors.
- Consider which deployment method best suits the needs of the business and the needs of the model's consumers.

- Recognize that deep learning models will require more processing power (either CPU and GPU) in deployment compared to traditional machine learning models.
- Assess your model's infrastructure needs to anticipate performance and data availability issues in production.
- Consult with the appropriate stakeholders to determine if their requirements will influence your deployment strategy.
- Ensure stakeholders understand what is being deployed, and if needed, how to activate and obtain results from the deployed model.
- Ensure you are able to access and retrieve the data that is necessary for the model.
- Establish deployment endpoints in your environment to balance the load on the deployment servers, as well as route consumers to the appropriate model versions.
- Consider relying on cloud services to alleviate much of the burden of setting up your own deployment infrastructure.
- Review offerings from cloud platforms like AWS, Azure, and Google Cloud to determine which is most appropriate for your needs.
- If ease of setup and maintenance is paramount, consider relying on a managed cloud service for model deployment.
- If customization and flexibility is paramount, consider using an unmanaged cloud service for model deployment.
- Evaluate the downsides of using cloud services for deployment, and whether or not the risks and costs outweigh the benefits.
- Consider using Docker containers for deployment, where appropriate.

ACTIVITY 11-1

Deploying a Machine Learning Model

Data Files

All files in /home/CAIP/MLOps

Before You Begin

Docker has already been installed in the Linux VM. A Docker container has already been created.

Scenario

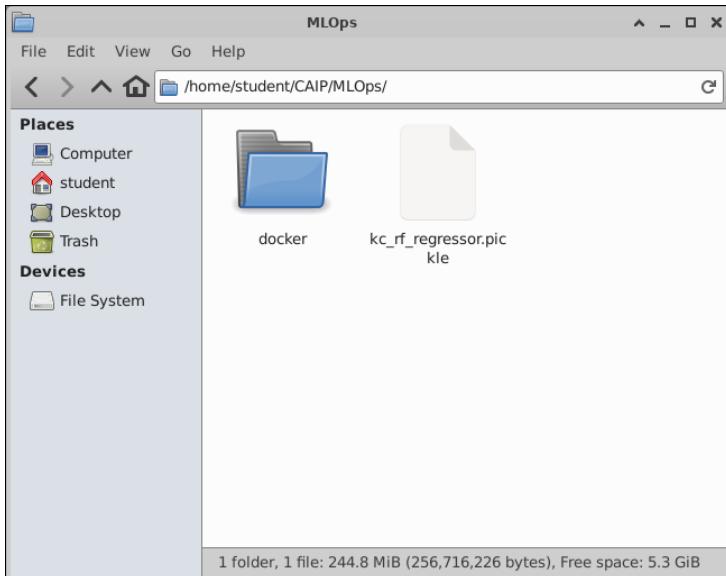
You and your fellow machine learning practitioners at CapitalR Real Estate have been hard at work experimenting with models that predict house prices. The end goal of this project is to enable the company's real estate agents to obtain these predictions for houses that are newly on the market. That way, they can set realistic expectations during listings and negotiations.

The team has generated a model that they believe is ready for production. The big question is: How do they deploy it? The model needs to be continuously accessible to hundreds of real estate agents, and it must deliver predictions to the agents in real time whenever requested. The team needs an environment that is up to the task. In the future, the team might scale up operations using cloud services, but for now, a local environment running virtualized Docker containers should suffice.

The first step is to deploy the model, just to make sure it generally fulfills its purpose.

-
1. Examine the model file and the directory used in the Docker container.
 - a) From the Linux desktop, double-click **Home** to open the file browser to the **/home/student** directory.
 - b) Navigate to **CAIP/MLOps**.

- c) Verify that there is a **kc_rf_regressor.pickle** file as well as a **docker** subdirectory.



- The **kc_rf_regressor.pickle** file is a saved version of a model that is similar to one you created earlier in the course. It is a random forest trained on the King County housing dataset. The target variable it predicts is the price of a house.
 - The **docker** directory includes data that will be accessible by the deployed Docker container.
 - Docker is open-source software that creates virtualized containers. Containers are isolated from the host system and other containers, and are commonly used to manage deployment environments in all fields that employ software, not just machine learning.
- d) Right-click **kc_rf_regressor.pickle** and select **Copy**.

2. Deploy the model in the **docker** directory.

- Open the **docker** directory and verify its subdirectories:
 - housing_data** contains the dataset that was used to train the model.
 - logs** will contain logged event data pertaining to the pipeline.
 - model** will contain the model itself, once you copy it there.
 - scripts** contains preliminary scripts that you'll run to test your deployment and other aspects of a machine learning pipeline.
 - web_app** contains files that create a frontend GUI for working with the deployed model.
- Open **model** and paste the **kc_rf_regressor.pickle** file into it.
- Return to the **/home/student/CAIP/MLOps/docker** directory.

3. Deploy the Docker container.

- Right-click in a blank spot in the directory and select **Open Terminal Here**.

The Docker container was already built for you. It contains all of the Python libraries necessary to perform the machine learning tasks you're used to. However, you still need to start the container to ensure it can run.

- At the terminal, enter **docker run -it --rm -v \$PWD:/home/caip -p 5000:5000 caip/python**

	Caution: Commands in Unix-like systems are case sensitive. Be mindful of how you type.
---	---

- docker run** is the command used to start a pre-built container.
- it** instructs Docker to create an interactive shell within the container.
- rm** will automatically remove the container once it exits.

- `-v $PWD:/home/caip` tells Docker to bind and mount a volume to the container. A volume can be any directory or storage device. In this case, `$PWD` is a shell variable that indicates the *present working directory*. So, `/home/student/CAIP/MLOps/docker` is the directory on the Linux host that will be mounted in the container. On the other side of the colon (`:`), `/home/caip` indicates that the host volume will be mounted *within* the container at that location.
 - `-p 5000:5000` maps network port 5000 from the Linux host to port 5000 in the container. This ensures that network communication using that port is enabled. You'll use this port later to access the web app.
 - `caip/python` indicates the name of the container that Docker should run.
 - Ultimately, the container will have access to the data files in this directory.
- c) Verify that you are presented a shell prompt that starts with `caip`.

```
(base) student@debian:~/CAIP/MLOps/docker$ docker run -it --rm -v $PWD:/home/caip -p 5000:5000 caip/python
caip@2df25bf70594:~$ █
```

The container is running and you are able to interact with it like any other Linux shell.

4. Test the model's prediction capabilities while in the deployment environment.

- a) At the container shell, enter `ls` to get a directory listing.
- b) Verify that the files and folders you looked at earlier are accessible in the container.

```
caip@86c55342ad68:~$ ls
housing_data  logs  model  scripts  web_app
caip@86c55342ad68:~$ █
```

- c) Enter `cd scripts` to change to that directory.
- d) Enter `cat kc_predict.py`
- e) Verify that you see the code in this Python script.

It's not important to review all of this code in detail; just know that it calls predictions from the pickled model file using features that are passed in as command-line arguments. For now, you'll use this file to test your predictions. Later, you'll deploy a more user-friendly interface for doing so.

- f) Enter `python kc_predict.py 3 2.5 1550 3890 2 0 1 3 7 1230 5 0 0 1 0 0 0 0`

This executes the Python script with arguments. Each argument after the script name, separated by a space, maps to a model feature.

- 3 is the number of bedrooms.
- 2.5 is the number of bathrooms.
- 1550 is the square footage of the living space.
- 3890 is the square footage of the lot.
- 2 is the number of floors.
- 0 indicates that this is not a waterfront house.
- 1 indicates that the view is "fair."
- 3 indicates that the condition of the house is "fair."
- 7 indicates that the house grade is "average."
- 1230 is the square footage of the basement.
- 5 is the bin that represents when the house was built or last renovated. This bin is 2001–2020.
- The remaining seven values are binary encodings of the ZIP Code™. The only 1 is in the third binary column, which means the binary value is 0010000, or 16 in decimal.

- g) Verify that the model was loaded and output a prediction.

```
caip@2df25bf70594:~/scripts$ python kc_predict.py 3 2.5 1550 3890 2 0 1 3 7 1230
5 0 0 1 0 0 0 0
Loading model from file...
Prediction: $537,553.52
caip@2df25bf70594:~/scripts$ █
```

The model has been successfully deployed in the container and is outputting predictions as expected.

- h) Keep the terminal window open and connected to the container.
5. The model is deployed in a single container. But, consider that hundreds of real estate agents will need to access the model quite frequently.

What infrastructure concerns might you have regarding deployment, and how might you address them?

TOPIC B

Automate the Machine Learning Process with MLOps

Operationalizing a model means more than just deploying it. The entire machine learning process from beginning to end can be automated, leading to increased efficiency and reliability.

MLOps

MLOps is the combination of machine learning (ML) and DevOps, which is itself a combination of the disciplines of software development and IT operations. More specifically, MLOps is a set of practices for automating the development, testing, deployment, and maintenance of machine learning models in an operational environment. This distinguishes MLOps from more isolated or "siloeed" machine learning processes, where individuals or groups of practitioners create models without much thought put into how the models will be used in production to benefit the business. The purpose of MLOps, therefore, is to ensure that machine learning is integrated into business operations like any other computing technology.

The practitioners who engage in MLOps are typically called machine learning *engineers*.

Google observes three levels of MLOps maturity in organizations:

- **MLOps 0**—The machine learning process is almost entirely manual, from data collection all the way to deployment. This very basic level of maturity can lead to poor planning, lost time, lack of communication between team members, models breaking in production, models "rotting" as they become more and more out of date, and so on.
- **MLOps 1**—The machine learning process is now starting to be automated through the use of pipelines. Key processes like data cleaning, model training, and model deployment are applied continuously, and without manual intervention. This makes these tasks repeatable and reliable. However, the pipeline itself must still be tested and tweaked manually.
- **MLOps 2**—The machine learning pipeline itself is automatically updated and tested to fit an experimental approach. This level applies the concepts of continuous integration and continuous delivery (CI/CD) to enable the business to quickly and easily adapt to new and changing circumstances surrounding their models. This is the ideal level of MLOps maturity, although it can be the most challenging to adopt.



Note: For more information on these maturity levels, see <https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>.

Machine Learning Pipelines

A machine learning **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 the deployment of those models for wider consumption, to the ultimate retirement and safe disposal of data. In other words, the machine learning process can follow a repeatable pattern where the project can move from phase to phase without a practitioner needing to manually perform every task or subtask. This does not mean that you can be completely hands off. It just means that your job is made easier so that you can focus more on the aspects of the project that require human judgment, rather than getting bogged down in tasks that a machine should handle. For example, removing duplicate data can be automated in a pipeline, but drawing conclusions and formulating hypotheses based on an exploratory analysis of that data is still yours to do.

Automated pipelines enable you to optimize a project for simplicity, speed, portability, and reusability. Consider some examples of how automation might apply to each step of the process:

- **Problem formulation**—This phase is mostly conceptual and involves human assessment, but you can actually use machine learning models to identify problems that the business may be interested in solving. Perhaps a model detects some sort of shortcoming in the manufacturing process that results in delays, and you therefore set out to improve manufacturing speeds through a new project.
- **Data collection**—Development endpoints can be configured to constantly ingest data from services like web APIs. For example, if your project involves analyzing weather patterns, you can retrieve today's weather data from an existing resource.
- **Data processing**—Many issues with data can be identified and corrected by automated processes, including duplicates, missing values, etc. For example, you might create a filter to automatically remove missing values from weather data as it comes in.
- **Model training**—Training basic models on prepared data can follow a repeatable process. For example, the pipeline might train the model on one algorithm, then another algorithm, and then automatically compare their scores to determine how to proceed.
- **Model deployment**—The optimal model can now be deployed to a staging environment where its ability to accept inputs and create outputs can be tested. Then, it can be automatically deployed to a final environment where it is accessible to the audience that needs it.

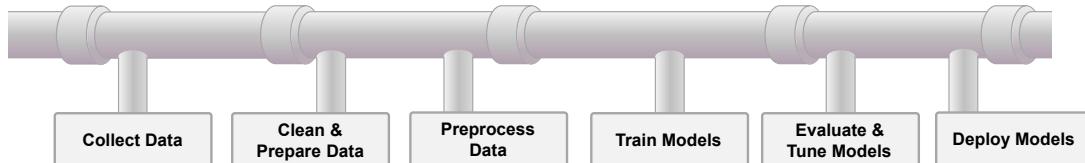


Figure 11-7: A machine learning pipeline.

Also, when you set up a pipeline to automate deployment processes, you should consider how online models will need to reuse the pipeline, whereas offline models may just go through the pipeline one time.

scikit-learn Pipeline

The scikit-learn `Pipeline` module is an example of a simple automation tool. It enables a developer to define a list of transformation steps that will be executed sequentially. A transformer is any data preparation function available in scikit-learn, like `KBinsDiscretizer()` to bin a continuous variable, or `OneHotEncoder()` to one-hot encode a categorical variable. These transformations are added to a `Pipeline` object. The final step is to specify an algorithm to train the model from. When you call `fit()` on the `Pipeline` object, it will perform all of the transformations on the data and train a model. So, all of these tasks can be repeated on other datasets by calling a single function.

Automation of Data Collection

As you construct a pipeline for your machine learning projects, you need to consider how data will be handled within that pipeline. For online models, you need to be able to regularly ingest data that the model will eventually train on. The source of that ingestion will depend on several factors. In the example of a model that forecasts the weather for users, your collection sources are the weather services that actually measure and record the surrounding environment. It's likely that these services have APIs that enable you to pull in the weather data at a consistent rate. So, you'd write scripts to call these APIs and fetch the data, which you can then send down the pipeline.

On the other hand, the source of your data might be something less stable and predictable, like user data. A recommendation system, for example, needs to constantly ingest data about users in order to get an accurate picture of what each user likes and what they're most interested in purchasing. So, you could set up the collection process to automatically grab session data as the user browses the site, such as what product pages they're on, what searches they performed, and what purchases they made.

Wherever your data comes from, you need to make sure your pipeline's collection processes are resilient. If the weather API is temporarily down, or its endpoint changes, or the call format changes, or the service is permanently retired, your pipeline needs a way to react to these conditions. The same goes for any source of data—a change in that data or how it's delivered can "break" the collection process if it's not resilient. To make the collection process resilient, you need to:

- Modularize your collection code so that a failure in one collection task does not cause a failure in the entire collection process.
- If possible, rely on multiple sources of data rather than just one.
- Implement a failover contingency if the primary source of data becomes unavailable, but other sources are still available.
- Validate the data as it comes in to ensure it is in a state that you deem acceptable.
- Trigger alerts if something goes wrong during collection.

Automation of Data Preparation

Once a resilient data collection process is in place, the data needs somewhere to go next. As you know, raw data extracted from most sources is going to need cleaning before it can even be considered ready for training. An effective pipeline is able to apply the most appropriate data preparation tasks in a repeatable and reliable way.

Creating an automated data preparation process can range from being simple to incredibly complex. Consider the weather data that you pull in through weather service APIs. If you pull the data in from one primary source, it's likely in a consistent format. You can easily configure the preparation process to apply the same basic tasks in the same ways—always impute missing temperatures this way, always check for air pressure values that go above this threshold, and so on. But consider how the complexity of this process increases as you add new and disparate sources of weather data. Each service might format its data differently—maybe the temperature column has a different name, or records the temperature in a different scale—so you'll need to account for any discrepancies in your automated code.

Just like the collection process needs to be resilient, so too does the preparation side of the pipeline. Again, writing modular code is one of the best defenses against breakage. If a preparation task cannot immediately proceed with a dataset, it should not just fail or throw out the entire source of data and move on. It should instead be able to gracefully handle multiple conditions.

Also, the code should be able to perform some rudimentary decision making. For example, you might choose to scale the features only if the values of each feature are on significantly different scales. You would therefore write code to first detect this scaling issue, then apply the feature scaling *only* if the issue was detected. If no such issue is detected, the code does not perform scaling and simply moves on to the next task.

Automation of Model Training

After the data is prepared, the pipeline needs to automatically be able to generate multiple candidate models from that data and select the "best" one. This means fitting a model using multiple algorithms, as well as multiple combinations of hyperparameters for each algorithm. Then, you need to write code for the pipeline to automatically test and evaluate each model and determine which one performs the best according to your criteria.

The task your model is meant to perform dictates how the pipeline should select the optimal model. In the weather example, the model needs new data, and that new data may lead to new models that perhaps have a lower test score than older models. Still, you might need to proceed anyway, as long as the best model for that particular instance of data is chosen. On the other hand, with the recommendation system, you may only tolerate a model that scores just as good as or better than models in the past. You don't want your models to get worse over time, negatively impacting the user experience and potentially leading to lost business opportunities.

As you're designing this aspect of a pipeline, you need to pay special attention to the factors of time and compute performance. In an ideal world, you'd be able to train hundreds or thousands of different models—everything from simple regression models to deep neural networks. In the real world, this is simply not feasible for most organizations. You have a deadline you need to meet, not to mention a budget that can quickly run dry if you're consuming a lot of hardware or cloud resources. You need to select a few algorithms you think are most appropriate, and run something like a randomized search to find good enough hyperparameters. As long as the models are meeting whatever baseline level of performance you've set for them, then the tradeoff between model skill and training time/resource availability will be worthwhile.

Lastly, as part of automating model training, the pipeline should be able to validate that the chosen model is ready for deployment—i.e., able to receive the expected inputs and produce the expected outputs, at scale.

Pipeline Triggers

As you construct each section of your pipeline, you also need to think about what exactly will prompt the pipeline to "activate." In other words, what will trigger the collection process so that you can eventually deploy a newly trained model?

There are several ways the pipeline might be triggered:

- **On demand**, where a machine learning engineer explicitly requests that the pipeline start up again. This trigger is common when none of the other triggers really take precedence in a particular situation.
- **On a schedule**, which can be daily, weekly, monthly, yearly, or really any frequency you want. This trigger is common for machine learning projects whose data sources produce new data on a consistent basis, or domains that are inherently "seasonal" (e.g., weather data).
- **On availability of new data**, which is more likely in situations where data cannot be collected consistently. For example, if your model is built on product data, and your organization is currently developing new products, you may need to wait for development to finish before you can use the new product data to train a model.
- **On the model becoming stale or degrading in performance**, which can happen when the model's assumptions about the knowledge domain are no longer up to date, a concept called model drift or concept drift. For example, your pipeline might not regularly ingest new data about house prices, but the housing market can change dramatically from year to year. A model trained on outdated data will perform poorly when it is asked to predict the most current house prices.

Continuous Integration and Continuous Delivery (CI/CD)

Continuous integration and continuous delivery (CI/CD) are two separate, but related concepts that are key in establishing a mature, level 2 MLOps environment.

Continuous integration (CI) is a software engineering term in which developers write, test, and merge code into a centralized repository on a consistent basis, usually at least once a day or several times a day. A new build of the environment is created after every commit so that any errors introduced can be spotted early, rather than later down the line. The use of a central code repository also means that each component can be under version control, enabling the developers to roll back to earlier versions if need be.

In the realm of MLOps, CI involves engineers writing code to collect and prepare data, train models, deploy models, and anything else that might be part of their machine learning workflow. Since identifying and addressing bugs quickly is one of the main benefits of CI, testing becomes very important. This means writing unit tests for modules at each part of the pipeline, like testing a module that performs feature scaling, testing another module that performs encoding, and so on. It's also crucial to test the machine learning model itself, including its ability to cope with unusual inputs, its ability to integrate with deployment endpoints, and its overall performance.

Continuous delivery (CD) is another software engineering term that refers to short, repeated cycles in which the development team can automate the delivery of each software component for rapid adoption.

In MLOps, this means implementing new and/or updated parts of the pipeline in a timely and efficient manner. Once code is pushed to the target environment, engineers can test that each aspect of the pipeline, as well as the pipeline as a whole, is meeting an acceptable level of performance. This can be everything from verifying the latency of calls to an API and responses from the model, to verifying that the form those calls and responses take is as expected. And, engineers can automate the deployment of a pipeline into a testing environment or a pre-production staging environment so that they can validate the pipeline (including any changes to code or data) against realistic use cases. Then, finally, they can push the pipeline to production.

The Pipeline Automation Process with CI/CD

The process of automating a machine learning pipeline while leveraging CI/CD can be summarized as a series of stages, in which a subprocess produces an output, which then leads to the next subprocess, and so on.

1. **Development**—In this initial stage, you develop pipeline components involved in the machine learning process.
 - Output: **Source code** of the pipeline components pushed to a centralized repository.
2. **Continuous integration**—Next, you *build* the source code and run tests on it.
 - Output: **Software packages** like executables and compiled libraries that will be used to deploy the pipeline components.
3. **Continuous delivery**—Next, the software packages are deployed to a production environment.
 - Output: **Automated pipeline** with a preliminary model capable of producing output.
4. **Training**—The pipeline executes in response to any of the previously mentioned triggers, retraining the model.
 - Output: **Trained model** capable of producing outputs based on the latest data.
5. **Model deployment**—In this stage, the model is put into operation so that it can be used to make predictions or other intelligent decisions.
 - Output: **Consumer service** that provides model outputs to API consumers who initiate requests.
6. **Monitoring**—The last stage continuously evaluates the performance of the pipeline and its components.
 - Output: **Trigger** that either prompts another round of training or an entirely new pipeline development cycle.

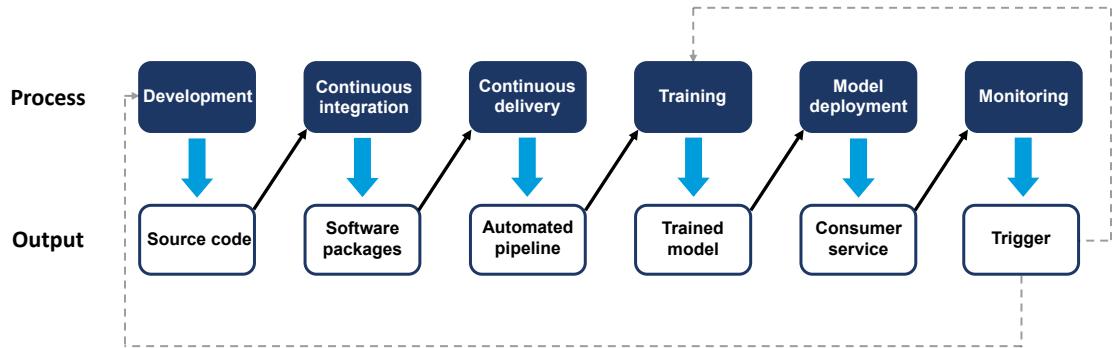


Figure 11-8: The CI/CD process for automating a machine learning pipeline.

Enrichment of ML Frameworks and Libraries

For the most part, the frameworks and libraries you use to create machine learning models—including those used in this course—will be fairly easy to integrate into your pipeline. Python libraries like scikit-learn, TensorFlow, and PyTorch, as well as web frameworks like Flask and Django, are well supported within the popular ML platforms.

However, development on data science and machine learning libraries tends to be very active, especially those that are open source (as many are). New versions are pushed out frequently, and can include new features, deprecated functions, bug fixes, security patches, and so on. Although the versions you started with may be working for the pipeline, you should consider updating the frameworks and libraries it uses to keep them from becoming outdated. And, any new features in updated versions can help you streamline the pipeline or even unlock new capabilities that were not possible before.

The ability to enrich your machine learning tools gives you a greater deal of control over the architecture of your pipeline and how data flows through it.

Guidelines for Automating the Machine Learning Process with MLOps

Follow these guidelines when automating the machine learning process with MLOps.

Automate the Machine Learning Process with MLOps

When automating the machine learning process with MLOps:

- Aim for at least an MLOps level 1 approach, where you construct pipelines that can automate each step of the machine learning process.
- Consider adopting CI/CD practices into your operational environment if having a robust and well-tested pipeline is of paramount importance.
- Ensure your automated processes are resilient and can handle unexpected inputs gracefully, rather than break the whole pipeline.
- Implement a framework and workflow to help process, save, and track versions of your data.
- Identify the most appropriate pipeline triggers for your specific models.
- Put a framework in place for easily training models and tracking the results of model experimentation.
- Clearly define what you consider "good" and "bad" results from a model.
- Monitor pipelines in production and produce alerts when pipeline behavior strays from acceptable baselines.
- Use tools that support collaboration, especially if you're working in larger teams.
- Write code in such a way that makes it easy for teams to collaborate, run and understand each other's work, and iterate on previous work.
- Start with a simple pipeline that you can test thoroughly, then move on to more complex pipelines when you're ready.
- Assess the time and resource requirements of your pipeline so that you can keep the operation of your models on schedule and under budget.
- Adhere to standard software development best practices whenever possible.

ACTIVITY 11–2

Automating the Machine Learning Process with MLOps

Data Files

All files in /home/CAIP/MLOps

Before You Begin

The Docker container is running.

Scenario

Your test of the Docker deployment environment was a success, but now it's time to build out a more robust solution. It won't do you any good to keep manually running Jupyter Notebook to prepare your data and train a model, then move the model to the Docker directory, and so on. It'd be better for you to develop a pipeline that can automate the entire data preparation, model training, and model deployment process. In this activity you'll do just that.

1. Open the automated pipeline script.

- Using the file browser, open **MLOps/docker/scripts**.
- Double-click **kc_train.py** to open it in Mousepad.
- From the menu, check the **View→Line Numbers** check box.
- Scroll through the file and note the four major sections:
 - Constants.
 - Helper functions called by pipeline functions.
 - Pipeline functions.
 - Pipeline execution.

2. Observe the pipeline helper functions.

- Observe lines 38 through 60.

This function is not directly part of the pipeline, but is instead meant to draw a residual plot after the model is trained so that there is some visual artifact for a data scientist or machine learning practitioner to analyze.

- Observe lines 63 through 69.

This function will obtain the first few predictions from the model on the test set after it's been trained.

- Observe lines 72 through 76.

This function retrieves human-readable function names for logging purposes. Each step of the pipeline is sent to a log file for analysis after the fact.

- Observe lines 79 through 81.

This function hashes input data for the purposes of integrity verification.

3. Observe the main pipeline functions.

- Observe lines 88 through 104.

This begins the pipeline proper. The code is set up in such a way that each successive function will be called in order, applying tasks to the data in that order. The key argument is `context`, which takes the results of the previous function in the pipeline and applies it to the current function. Since this is the first function (which loads the data), the `context` is essentially empty.

b) Observe lines 107 through 139.

This function will perform various encoding tasks on the data, including converting `bedrooms` to an integer, `date` to a datetime, and `condition` to a label-encoded integer. Notice that it also takes `context` as its argument, meaning it will take the results of the previous function as input. And, the function returns `context` after it has updated it. This pattern repeats until the last function in the pipeline.

c) Observe lines 142 through 209.

These functions conduct more data preparation steps, such as removing duplicates and binning/encoding variables. The encoder object used to binary encode ZIP Codes is being saved as a pickle file so it can be applied to raw input later.

d) Observe lines 211 through 251.

These functions obtain the relevant list of features for the model to train on, then perform a holdout split to get training and testing subsets.

e) Observe lines 254 through 280.

These functions train a random forest model using a hardcoded number of estimators (trees), then get predictions from the model.

f) Observe lines 282 through 317.

These functions train a random forest model using grid search and get its predictions. This is something the user (i.e., the engineer) can choose to use as an alternative to the standard random forest model. However, the search takes a very long time to complete, which is why it's optional.

g) Observe lines 320 through 343.

These functions get the error score (RMSE) of the chosen model and then generate any relevant plots.

h) Observe lines 346 through 380.

These functions determine which model to use, the default forest or the grid search forest, and then save it as a pickle file.

i) Observe lines 383 through 407.

In this last function of the main pipeline functions, the model in memory is used to make predictions, then the saved model is loaded and also makes predictions on the same data. The two sets of predictions are compared in order to validate that the model was saved properly.

4. Observe the pipeline execution functions.

a) Observe lines 414 through 476.

- This function is used to run each step of the pipeline.
- On lines 435 through 453, the `pipeline` list includes the name of each function, in order of execution.
- Lines 462 through 473 use a `for` loop to iterate through each function/step in the pipeline.
- Line 468 sets the context of the current function equal to the context of the returned value from the previous function.

b) Observe lines 480 through 508.

This code establishes the command-line arguments for calling the function, determining whether or not to conduct a grid search based on the user's choice. It also sets up the logging configuration used throughout the pipeline. Lastly, it calls the `runner()` function to execute the pipeline.

c) Close the script file.

5. Run the pipeline training script.

a) Return to the terminal window open to the Docker container.

b) Enter `python kc_train.py fast`

In the interest of time, you'll just run the "fast" model (i.e., not conducting a grid search).

- c) Verify that the pipeline executed without any errors.

```
caip@2df25bf70594:~/scripts$ python kc_train.py fast
Skipping creation of grid search model

---STARTING PIPELINE---
Step load_dataset
Step encode_types
Step drop_outliers
Step remove_duplicates
Step bin_years
Step encode_zip_codes
Step get_training_cols
Step split_into_train_test
Step train_forest_model
Step forest_predict
Step show_results
Step find_which_model_trained
Step save_trained_model
Step validate_saved_model
Step generate_plots
---PIPELINE COMPLETE!---
caip@2df25bf70594:~/scripts$
```

6. Test the deployed model by obtaining a prediction.
 - a) At the terminal, enter `python kc_predict.py test`
You'll use the same prediction script from the previous activity. The `test` argument tells the script to use the same example values as before so that you don't have to type them out again.
 - b) Verify that the prediction returned is the same as before: \$537,553.52.
You have successfully automated the deployment of your model into the pipeline environment.
 - c) Keep the terminal window open and connected to the container.
7. Compared to how Jupyter Notebook has been used throughout the course, how is this training script more appropriate for automating production tasks?
8. How might you make these automated tasks more robust and resilient to failure?

TOPIC C

Integrate Models into Machine Learning Systems

When you build pipelines for your models, there's a good chance you'll be deploying them as support for an existing system that your customers or other end users can take advantage of. It won't be enough to just deploy them in a vacuum; you'll need to integrate them into a larger system so that the business can achieve its goals.

Machine Learning Systems

In most cases, when you deploy a machine learning model meant for public consumption, you'll need to do more than just build that model in something like scikit-learn and then call it a day. Most models are not products or services in themselves, but merely support the actual products and services that a business provides. Consider the model that predicts whether or not a patient needs immediate medical attention based on their vital signs. This model is integrated into a larger ecosystem that includes the monitoring application, the notification system, and the device hardware itself. All of these components must work well together, as a deficiency in one can degrade the entire experience.

The following are some factors to consider when integrating models into machine learning systems:

- **Compatibility issues**—The system and the model must agree on a format that the inputs and outputs take. APIs and endpoints go a long way in bridging the gap between system and model. Still, you must test the API thoroughly to ensure it is processing requests and responses adequately.
- **Bandwidth limitations**—Assuming your model is stored somewhere in the cloud, users and devices who access it will need to do so over a network. This can lead to problems if there is not enough bandwidth on the server end to handle all requests in a timely manner. It can also be a problem for the consumer, who may have a poor connection or none at all. The machine learning system must account for these issues and deal with them gracefully.
- **Hardware limitations**—Depending on who will use the system and how, you may be able to alleviate bandwidth concerns by having the model operate locally on the device. This is less feasible on relatively low-powered devices like smartphones and IoT devices due to hardware limitations, especially if any retraining needs to be done locally. Still, some models are built to run on high-powered desktops and workstations.
- **Intellectual property**—Many businesses will want to keep their models proprietary. If a model is stored locally, this becomes a security issue, as a user may be able to extract the model and use it outside the system. Although less likely, savvy users may be able to derive a model's weights and parameters by sending it a wide array of input and performing inference on the resulting output. This may enable the user to reconstruct the model even if it's stored remotely.
- **End-user experience**—It's not enough for a model to exchange data with an app or other system; it must do so in a way that fits the user base. For example, if you're marketing an app as being powered by AI, and that's a primary selling point, the user may expect the interface to overtly mention AI. On the other hand, some apps are better designed to make the model "invisible" to the user, particularly if the model is only one small part of a larger system.

Model APIs

As mentioned earlier, in many deployments, models are not directly accessible to users or services, but rather are accessible through endpoints that implement a layer of abstraction. This is what an API does.

Most machine learning endpoints are implemented using RESTful web services, which refers to representational state transfer (REST), the dominant standard for interacting with web services. API

calls can send and retrieve data in file formats like Extensible Markup Language (XML) and JavaScript Object Notation (JSON), like any other web service.

For example, a consumer may call the model API by sending it an HTTP request, like a POST request. In the body of this request is the pertinent data that the model will use to make a prediction. The API receives this request and sends it as input to code you've written that will call a prediction function using an existing model. After the model finishes its prediction, it will return the output to the API, which will initiate a response to the consumer using the same web-based format as the request. The consumer receives the response and the transaction completes.

Model APIs are therefore an important part of the deployment end of a machine learning pipeline; they enable your model to interface with the outside world and actually deliver the output it was designed for, all in an efficient and controllable manner.

Model API Example

Consider an app that runs a health monitor for a patient in a hospital. The monitoring device measures and records vital signs like heart rate, blood pressure, body temperature, etc. The model uses this data to determine if the patient is likely to need medical attention. So, the app collects all of the patient's data and compiles it into a format that it can send as a request to the API. JSON data sent in the request might look something like this:

```
[
  {
    'time': '02:38:21',
    'patientName': 'John Smith',
    'locationFloor': 5,
    'locationRoom': 12,
    'vitals': [
      {
        'heartRate': 90,
        'systolicPres': 140,
        'diastolicPres': 90,
        'bodyTemp': 98.9,
        'respRate': 28
      }
    ],
  }
]
```

The API accepts this request, assuming it's formatted properly, and then passes the vital sign data along to the deployed model code. The code converts the JSON data to a format that the model can understand (e.g., a data frame) and then the model makes a prediction (1 for needs medical attention, 0 for does not). The code sends the prediction to the API using a similar JSON format, which then forwards the response to the health monitoring app. The app is programmed to log the response and send a notification to a nurse if the prediction returned a 1, whereas the app just logs the response if it's a 0.

Thus, the consumer (an app, in this case) is able to use the model to get what it needs—not by directly interfacing with the model, but by sending and receiving data through a web-based API.

Documentation for Model/System Handoff

In some scenarios, you'll be providing a model or an entire machine learning system to your target audience for *them* to operate. So, your end users are not necessarily the average person, but a client such as another organization that wishes to either incorporate your machine learning capabilities into their existing product, or to buy an entire system from you to use in their business.

For example, your client is a food delivery service that wants to incorporate machine learning into their app. Specifically, they want the app to be more intelligent at estimating wait times for their customers. The client may need to operate the pipeline or perhaps just certain parts of it to get this functionality working. Therefore, it's a good idea to draft thorough documentation of whatever it is you'll be providing.

Some key points to include in this document are:

- **Purpose**—State the overall purpose of the model or system; i.e., what it was designed for. That way, there's less chance it gets applied to problems that are not appropriate for it to solve. In the food delivery example, the purpose of the model is to estimate wait times for deliveries.
- **Required data**—State the data that is required for the model to make an estimation, or for it to retrain (or both). This includes the content of the data, the format, and the volume. Any other assumptions made about the data should also be listed. The food delivery model needs the customer's location, the restaurant's location, the deliverer's location, current traffic data, the type and amount of food being ordered, etc.
- **API access**—Describe the protocols and formats that API calls will take, if applicable. Also describe how input requests from the API consumer will be initiated, as well as any caveats involved in the model outputs, such as potential delays or incomplete results due to missing data. For the food delivery app, the data might be sent to the model as a RESTful request, and if too many are received at once, the request might go into a queue while a simple average wait time estimation is displayed to the customer. Later, when the queue clears, the customer's wait time can be updated.
- **Developer access**—Assuming the situation calls for the client to have direct access to parts of the pipeline platform, then individual developers or groups of developers need to be granted certain permissions. And, if the client or someone else needs to take over the system at some point, there should be steps for how to retrain the model. The food delivery app developers may be responsible for training and deploying different models for different areas that the service covers.
- **Ownership and responsibilities**—A statement of who owns what portion of the system, as well as who is responsible for providing and/or upkeeping that portion, must be in written form. This upholds the principle of accountability should something go wrong, or some dispute arise. The food delivery service's IT manager might be given ultimate ownership over the app itself, whereas the company that designed the model might be responsible for ensuring the model stays working and up to date.
- **Training methodology**—It might be useful to justify or at least record the approach you used to train your models. In particular, what algorithm you chose and why, as well as what hyperparameters you chose and why. For example, the delivery wait time model might be built using a regression forest with a certain number of trees deemed optimal.
- **Performance expectations**—The client organization will undoubtedly have some expectations for performance, otherwise they would not have enlisted the help of machine learning. So, their expectations must be well documented so that there is no doubt as to whether or not the model/system is providing value to the business. For example, the delivery service might track the actual time of delivery and calculate the error between the estimated time. If the average of these error values is below a certain threshold (e.g., five minutes), then the client considers this a success.

Design Patterns to Avoid

As you begin integrating your models into a larger ecosystem, you need to be conscious of some of the design patterns that could lead to issues.

Design Pattern	Description and Mitigation Tactics
Boundary erosion	<p>Boundary erosion can occur in any software project when the behavior of certain systems is not clearly defined, or becomes murkier over time. Machine learning is especially susceptible to this because of its stochastic nature; i.e., there is not necessarily a consistent set of inputs that match to a consistent set of outputs.</p>
Entanglement	<p>Boundary erosion is not totally unavoidable in machine learning, so you should focus on identifying where this erosion could happen and whether or not the risk is acceptable.</p>
Hidden feedback loops	<p>Entanglement embodies the CACE principle—<i>changing anything changes everything</i>. In other words, changes to the feature set, the transformations done on the data, the hyperparameters the model uses, and many other factors will necessarily alter the model. The model is therefore entangled with the data, and its performance can suffer as a result of these changes.</p>
	<p>The use of ensemble methods can help mitigate some of the effects of entanglement, but these come at the cost of computing time and performance. Tools that provide model interpretability make it easier to understand a model's behavior from a high level, but only if the model is not a black box. Another potential solution is to apply more robust regularization techniques to a model.</p>
	<p>In machine learning, a feedback loop might occur if a new model prediction relies on past model predictions. In a recommendation system, the model will probably recommend certain products to a user based on several factors, including past recommendations. This kind of feedback loop is expected and can be addressed as needed. However, hidden feedback loops are trickier. They are not immediately evident and can affect the model in ways that are not always clear.</p>
	<p>For example, you have a feature that tracks how many recommended products the user purchased in the past month, and you update the model with this new data every month. The model still gives recommendations every day in the meantime. So, the model will continue to give recommendations based on older data, which itself affects how many recommended products users purchase, which gets fed back into the model at a later time, and so on. Ultimately, the model undergoes subtle changes that aren't necessarily easy to identify.</p>
	<p>The best defense against hidden feedback loops is to brainstorm ways in which they can crop up for your given domain problem, and then monitor for any effects they are likely to produce.</p>

Design Pattern	Description and Mitigation Tactics
Poorly managed data dependencies	<p>Data that a model pipeline consumes as input can be unstable, meaning it changes form over time due to several possible factors. This can lead to negative effects in the model or in the pipeline overall. Another data dependency issue is features and other components of data that provide little value to the model, but which are still included in training. The presence of irrelevant data, especially if it's updated over time, can lead to model degradation.</p> <p>A mitigation strategy for unstable dependencies is using versioned copies of data so that you can revert to a more stable copy as needed. To address irrelevant data dependencies, you should assess how the removal of certain features does or does not impact the skill of the model.</p>
Poorly written code	<p>Glue code is code that is written to bridge the gap between two different components that are not inherently compatible. This is common in machine learning, as practitioners often develop code as self-contained solutions. But for these solutions to work with external environments, a lot of time and effort must be spent writing glue code.</p> <p>Another poor coding practice is creating pipeline "jungles," in which data preparation code is written in such a way that it becomes very dense, overly complex, and hard to understand. This is common because it can be difficult to bootstrap different sources of data and get it all in a format that can be used to train a model.</p> <p>The defense against both glue code and pipeline jungles is to, whenever possible, rely on tools that can be integrated easily into the overall environment, rather than rely on whatever tools are most common for a specific, isolated task. And, you should incorporate best practices from the software engineering industry.</p>

Guidelines for Integrating Models into Machine Learning Systems

Follow these guidelines when integrating models into machine learning systems.

Integrate Models into Machine Learning Systems

When integrating models into machine learning systems:

- Consider the factors that can influence how you integrate a model into a larger machine learning system, e.g., compatibility issues, bandwidth limitations, hardware limitations, etc.
- Construct your deployment around APIs that can automate model input and output for your consumers.
- Ensure your APIs are adequate for the task(s) at hand. At larger scales, a typical API meant for smaller-scale applications may struggle to keep up with the frequency and volume of requests.
- Document important aspects of the model and system that will be handed off to a third party.

- Communicate with third parties regularly to ensure everyone is on the same page regarding changes or to discuss status updates.
- Avoid design patterns like entanglement and hidden feedback loops when integrating models into machine learning systems.
- Whenever possible, design the system to be more flexible at handling changes. You should be able to modify one component without breaking the entire system or the pipeline it's backed by.
- Monitor your system and its components during production to ensure they continue to meet expectations.

Ethical Considerations in Model Operationalization

Ethics is just as important in the deployment of a machine learning model as it is in the design and development of that model. There are some ethical challenges to consider in the context of model operationalization.

Ethical Issue	Considerations
Privacy	<p>A model may take private input as data in order to make an estimation, or the model itself might be built on private data and therefore requires it for retraining. In both cases, the security of that data becomes paramount.</p> <p>For example, a customer's location is PII and, if not handled properly, may be exposed to the wider world. Even if you secured this information during initial training, you may fail to account for the fact that this information gets sent to the model many times per second.</p>
Accountability	<p>As mentioned, documenting who is responsible for what is crucial for operating machine learning systems. Still, some issues are difficult to anticipate, so it may not be clear which individual or organization must accept responsibility for an issue regarding the system.</p> <p>For example, a recommendation engine may suggest a set of products that are potentially harmful to the customer if used together. Without these recommendations, the customer may have never been exposed to this danger. Who is responsible should something go wrong? The customer? The store? The company that developed the system? It's not always clear cut.</p>
User choice	<p>There are many cases where users perceive that some AI-driven system is involved in the products and services they use. They may not agree with this notion, or they may think that the system simply doesn't work very well. In either cases, they may want to be able to opt out, or at least have some way to directly affect the outcomes. You must determine whether or not you should (or are required to) accommodate users in this way and the impact it will have on your system as a whole. Fewer participants means less data, which could lead to worse models.</p> <p>For example, users of social media or content delivery services like YouTube and Netflix often complain about "the algorithm" giving them bad recommendations, or even steering them away from the type of content they're most interested in. They may feel that the service is ethically obligated to give them a better experience.</p>

Guidelines for Addressing Ethical Risks in Model Operationalization

Use the following guidelines when addressing ethical risks in model operationalization.

Address Ethical Risks in Model Operationalization

To address ethical risks in model operationalization:

- **Privacy**

- Take a *privacy by design* approach to your machine learning projects, in which you account for privacy at every phase of development.
- Ensure all endpoints between the consumer and the pipeline are using robust transmission encryption protocols.
- If any data must be stored for a long period of time, ensure it is encrypted at rest.
- Establish a cycle for disposing of any unneeded data that has been stored for a period of time.

- **Accountability**

- During documentation, brainstorm as many scenarios as you can that could raise questions of accountability.
- Ensure all parties sign off on documentation that stipulates who will be responsible for an incident, and how they must respond to such an incident.
- Be transparent with users; don't just tell them you're acting ethically, but actually demonstrate how you're taking ethics seriously.

- **User choice**

- Perform extensive user testing of the machine learning system to identify areas where users may push back.
- Open a dialogue with users and solicit their feedback.
- Determine the potential costs and implications of enabling an opt-out feature in a machine learning system.
- Determine the potential costs and implications of allowing users to "tune" their experience—i.e., giving them more control over the outcomes that affect them.

ACTIVITY 11–3

Integrating a Model into a Machine Learning System

Data Files

All files in /home/CAIP/MLOps

Before You Begin

The Docker container is running.

Scenario

You've tested your pipeline and have verified it can automatically train a model to produce the desired outputs. But this is not how your audience—particularly real estate agents—are going to interact with the model. Instead of running Python scripts, they'll need a more user-friendly interface from which to request predictions.

One of your colleagues has begun prototyping a web app portal that multiple types of users will have access to. For now, you're interested in its primary functionality: presenting real estate agents with a form they can fill out to get the predicted sale price of a house. You'll take this opportunity to review your colleague's work.

1. Examine the web app files.

- Using the file browser, open **MLOps/docker/web_app**.
- Note the files and folders here:
 - static** contains images and a Cascading Style Sheets (CSS) file for formatting the web pages.
 - templates** contains the Hypertext Markup Language (HTML) files that structure the web pages. These files also contain JavaScript code that implements client-side functionality.
 - uploads** will temporarily hold data files that data scientists upload to the web app for the purposes of testing and retraining models.
 - app.py** is a Python script used to initiate the main web app. It uses Flask, a lightweight web framework, that acts as a web server.
 - authentication.py** is a Python script that implements sign-in capabilities in the web app.
 - model_operations.py** is a Python script that uses data from web forms to trigger prediction and training tasks.

2. Start the web app.

- At the container terminal, enter `cd .. /web_app` to change to the web app directory.
- Enter `FLASK_ENV=development flask run --host=0.0.0.0 --debugger`

- c) Verify that the web app environment is running.

```
caip@2df25bf70594:~/web_app$ FLASK_ENV=development flask run --host=0.0.0.0 --de
bugger
* Environment: development
* Debug mode: on
* Debugger is active!
* Debugger PIN: 213-329-331
```



Note: As long as the web app is running, the terminal will not present you a prompt.

3. Observe the CapitalR home page and sign in.

- a) From the Linux desktop menu, select Applications→Internet→Firefox ESR.
 b) In the Firefox URL bar, type <http://localhost:5000> and press Enter.

Remember, you set up the container to communicate over port 5000. This is the port used by the web app.

- c) Verify that you can see the CapitalR home page.

Predictor	Trainer	Admin
Use the machine learning model to predict the sale price of a house.	Retrain or test the machine learning model using new data.	Monitor logged activity in the machine learning pipeline.
Access: Real estate agents	Access: Data scientists	Access: ML engineers

There are three main pages linked from this home page: **Predictor**, **Trainer**, and **Admin**. Each one has a short description of what the page does.

- d) Select the **Sign in** button.
 e) In the pop-up window, sign in as **agent@capitalr.co** with a password of **secret**



Note: The email domain is .co, not .com.

- f) Verify that you have successfully signed in.

Successfully signed in.



Note: If prompted to save your log in, select **Don't save**.

4. Fill out a form to get a house sale price prediction.

- a) Select the **Predictor** icon.
- b) Verify that you are taken to a web form where you can fill out housing information.




Fill out the following form for a house on the market and our machine learning model will predict the sale price.

ZIP Code™: <input type="text"/>	Year built / renovated: <input type="text"/>	Bedrooms: <input type="text"/>
Enter the year the house was built or the year it was last renovated, whichever is most recent.		
Bathrooms: <input type="text"/>	Floors / stories: <input type="button" value="Select the number of floors / stories"/>	Lot size (sq. ft.): <input type="text"/>
For quarter and half bathrooms, increment by .25 and .5 respectively (e.g., 4.75 for four full bathrooms and three quarter bathrooms).		
Living space size (sq. ft.): <input type="text"/>	Basement size (sq. ft.): <input type="text"/>	Condition: <input type="button" value="Select the condition"/>
The square footage of the house not including the basement. The square footage of the basement only. Enter 0 if no basement. The condition of the house. See the CapitalR Housing Guidebook for descriptions of each condition level.		
Grade: <input type="button" value="Select the grade"/>	View: <input type="button" value="Select the view"/>	Waterfront property? <input checked="" type="radio"/> No <input type="radio"/> Yes
The grade/type of house. See the CapitalR Housing Guidebook for descriptions of each grade level. The quality of the view from the house. See the CapitalR Housing Guidebook for descriptions of each view level.		
<input type="button" value="Submit"/>		

The fields in this form should look familiar. They all pertain to features used to train the model, though presented in a more user-friendly way.

- c) Fill out the form using the following values:

Form Field	Value
ZIP Code	98103
Year built / renovated	2010

Form Field	Value
Bedrooms	3
Bathrooms	2.5
Floors / stories	2
Lot size (sq. ft.)	3890
Living space size (sq. ft.)	1550
Basement size (sq. ft.)	1230
Condition	(3) Fair
Grade	(7) Average
View	(1) Fair
Waterfront property?	No

- d) Select the **Submit** button.
- e) Verify that your browser brings you to the top of the page and presents you with the predicted price from the model.



Fill out the following form for a house on the market.

Predicted home price: \$537,554

ZIP Code™:	Year built / renovated:
98103	2010

5. Examine the API request involved in the prediction form submission.

- a) In Firefox, press F12 to open the developer tools.



- b) In the developer tools pane at the bottom of the screen, select the **Network** tab.
- c) On the page, scroll back down and select **Submit** again.

- d) Verify that the GET request involved in the submission is displayed in the **Network** activity list.

Status	Method	Domain	File	Initiator
200	GET	localhost:5000		predict?inputZIPCode=98103&inputYear=2010&inputBedrooms=3&inputBathrooms=predictor:27 (fetch)

- e) Select this event to open more information about it from a pane on the right.

Status	200 OK
Version	HTTP/1.1
Transferred	206 B (27 B size)
Referrer Policy	strict-origin-when-cross-origin

Response Headers (179 B)	
Connection	close
Content-Length	27
Content-Type	application/json
Date	Mon, 12 Sep 2022 17:53:43 GMT
Server	Werkzeug/2.2.2 Python/3.9.14

By default, the GET request header information is shown.

- f) Examine the GET request itself.
- The request accessed an endpoint on the web app called `/api/predict`.
 - The question mark (?) indicates the start of a query string.
 - The query string after this contains multiple key-value pairs, each one separated by an ampersand (&). These pairs act as arguments that the web app can handle.
 - For example, the first argument sets `inputZIPCode` equal to `98103`—the value you typed into the ZIP Code form field.
 - Each key in the query string refers to a field in the form. In this case, the keys are self-explanatory.

- g) From the pane on the right, select the **Response** tab.



- h) Examine the JSON response.

This is the response that the API sent back to your browser. It was able to obtain a prediction from the model based on your form inputs and send that prediction back to you. As a user, this process is invisible to you.

The JSON in this case is very simple since it's only returning the value of a single variable. Web apps can send as much JSON back to the user as needed.

- i) Select the X to the right of the developer tools pane (not the browser itself) to close it.

6. Examine the `model_operations.py` script.

- Return to the file browser and double-click `model_operations.py` to open it.
- Observe lines 9 through 118.
 - This script doesn't directly accept the form input—`app.py` does, which you'll see shortly. This script does, however, work with the data after it has been accepted by the app.
 - This code creates a class called `ModelOperations`.
 - This code loads the pickled model file that you generated earlier.
 - It sends values that align with the fields in the prediction form to the model for prediction.
 - Lines 73 through 86 encode the year value. In the form, a user enters a year value from 1900 to 2022. However, the model was trained on 20-year-wide *bins* that were label encoded. So, this code converts the year to its appropriate ordinal integer.
 - Lines 89 through 94 encode the ZIP Code. In the form, a user enters a ZIP Code value from 98000 to 98199. However, the model was trained on a *binary encoded* form of the ZIP Codes—seven total columns, each one holding either a 1 or a 0. So, this code loads the pickled encoder file that was created in training, and uses it to apply that same transformation to the raw ZIP Code provided by the user.
 - Lines 116 through 118 get the prediction from the model using the processed data, then return the prediction.
 - Various events are logged throughout the code.
- Close `model_operations.py`.

7. Examine the `app.py` script.

- Double-click `app.py` to open it.
- Observe line 25.
The Flask `app` object is instantiated on this line.
- Observe line 37.
This line instantiates a `ModelOperations` class object from `model_operations.py` that you just looked at.
- Observe lines 56 through 58.
 - Line 56 begins with the `@app.route()` decorator. This tells Flask what URL will trigger the function that follows it.
 - In this case, the top-most URL (identified by a single slash) is what triggers the `homepage()` function that begins on the next line.
 - The `homepage()` function merely returns the HTML template file called `index.html`, which is the main page you saw earlier.

- In Flask, you use this paradigm to write code that performs a task when the user (or browser) initiates a request. You can route a request to a function that does anything a normal Python function can do.
- e) Observe lines 67 through 84.
- The rest of these Flask routes define what happens when the user selects any of the other pages in the app.
 - The `/predictor` and `/trainer` routes just render their respective HTML templates.
 - The `/admin` route calls other functions to fetch log data.
- f) Observe lines 88 through 117.
- These routes handle signing in and signing out of the app.
- g) Observe lines 121 through 155.
- This is the prediction API endpoint that the web app calls when a form is submitted.
 - On line 121, the route is pointing to `/api/predict`—the same endpoint path that was in the form's GET request.
 - Lines 125 through 136 get each form field value that was submitted to the app. These are analogous to the key–value pairs in the GET request query string.
 - Lines 140 through 153 use the `model` class object to call its `predict()` method. Remember, this code was defined in `model_operations.py`. So, the code in that script is being put to use here.
 - Line 155 returns the prediction to the web page.
 - As before, there are various logging statements throughout the code.
- h) Close `app.py`.
- i) Leave the **CapitalR Real Estate—Predictor** web page open in Firefox, and keep the web app running in the container.

8. Aside from real estate agents getting house price predictions for existing houses, how else might they use the prediction form?

Summary

In this lesson, you took the machine learning models you built and put them into production. You also automated various machine learning tasks like data preparation and model training by making them part of an overall pipeline. Lastly, you integrated the models into consumer-facing systems where they can provide intelligent decision-making capabilities to end users.

What applications or other systems do you plan on incorporating your machine learning models into?

Do you think you'll rely on cloud services to host the deployment and pipeline resources for your machine learning models? Why or why not?



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.

12

Maintaining Machine Learning Operations

Lesson Time: 2 hours, 30 minutes

Lesson Introduction

Software development is an ongoing process, and machine learning is no different. Once you push your system to production, you cannot simply walk away and expect it to work flawlessly. Instead, you must put effort into maintaining that production environment, whether it's a simple deployment server or a more advanced data pipeline.

Lesson Objectives

In this lesson, you will:

- Ensure machine learning pipelines are properly secured.
- Ensure machine learning models maintain their value to the business even after they are built and deployed.

TOPIC A

Secure Machine Learning Pipelines

Like any platform that supports software and data, machine learning pipelines must be secured against attack. In this topic, you'll ensure your pipelines follow important cybersecurity practices.

The Importance of Securing ML Pipelines

Any computing product or service that a business offers to customers must follow at least basic cybersecurity practices. Where there is hardware, software, and data, there are malicious users ready and willing to compromise those things. Too many businesses, both large and small, have taken a reactive, hands-off approach to security, and they (and their users) have paid the price. The number of major breaches that have appeared in the news headlines over the past decade are proof of this.

Of course, it helps to be cognizant of what cybersecurity (or a lack thereof) means for machine learning pipelines in particular. One primary concern is the leakage of private or sensitive data. If data is not properly secured as it enters and exits the pipeline, or as it is stored within the pipeline, it will be at risk of unauthorized exposure. This can lead to frustrated users, a tarnishing of your organization's brand, and even legal action against your organization.

Another high-level risk of operating a machine learning pipeline is the unauthorized access of intellectual property, namely the models themselves. Attackers may be able to directly or indirectly obtain the parameters and weights that comprise your model, even without the data or source code. If this happens, your organization may lose its competitive advantage.

Likewise, if the pipeline platform itself is not properly secured, the attacker could insert a bogus model into the pipeline as part of a supply-chain attack. If you don't carefully monitor your pipeline, you may not even notice that the model has been crafted to produce bad outputs. These outputs can lead to failures in business operations as well as harm to users.

Model Poisoning and Evasion

Your models are vulnerable not just from unauthorized access, but also to being "poisoned." If your online model continuously trains on new data provided by users, a malicious user could provide the model contaminated input data that, in sufficient volume, skews the model's results. They might do this to bias the model in favor of some cause, or to bias it against an opposing cause (i.e., in a disinformation campaign), or to simply waste the organization's time and money by degrading the model's skill.

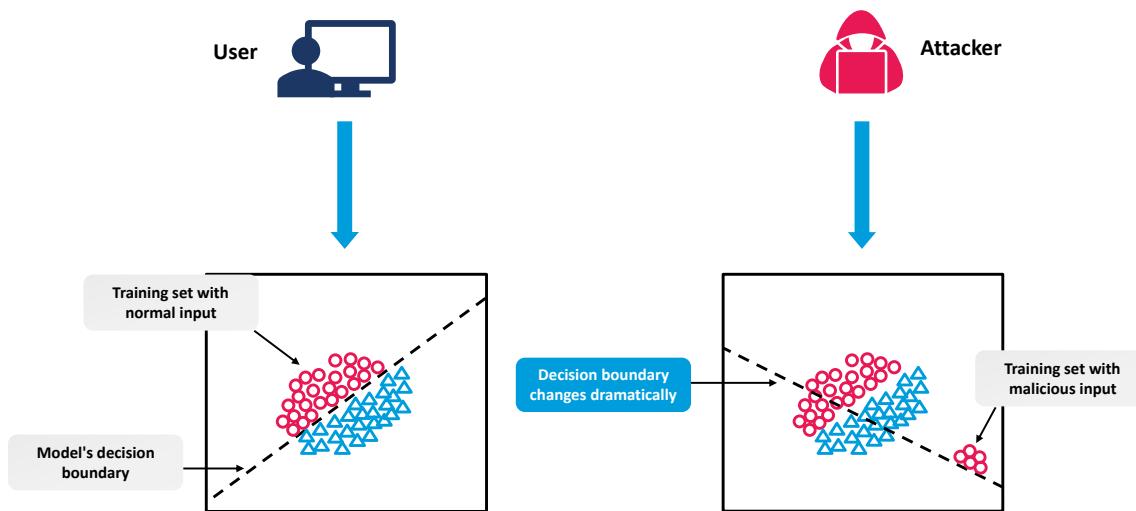


Figure 12-1: A data poisoning attack that has contaminated a model's classification outputs. An attack like this would be fairly easy to detect by monitoring for outliers. More sophisticated attacks can be harder to spot.

Even if malicious users are not directly poisoning your model, they may be trying to find ways to evade models that are designed to identify and stop unwanted behavior. For example, if a spammer learns how a spam detection model works, they could figure out how to trigger false negatives in the model, continuing their spam campaign unabated.



Note: The defense against attacks on machine learning models is called [adversarial machine learning](#).

Adversarial Machine Learning Threat Matrix

In 2020, Microsoft® and the MITRE Corporation teamed up to develop the Adversarial Machine Learning (ML) Threat Matrix to assist engineers and other MLOps stakeholders in identifying and mitigating adversarial attacks. The Adversarial ML Threat Matrix is styled after MITRE's ATT&CK matrix, a popular framework for defending against general cybersecurity attacks.

For more information on the Adversarial ML Threat Matrix, visit <https://github.com/mitre/advmcthreatmatrix>.

Pipeline Platform Security

Securing the pipeline platform itself is a good first step in implementing a robust cybersecurity plan for your project. Before you can do this, however, you need to understand the platform and how it operates. This means identifying the platform's built-in security mechanisms (if it has any), the dependencies it has on other platforms and services, its network entry points and the protocols it uses to establish connections with endpoints, and the hardware and software it uses to construct the virtual environment the pipeline runs on. Determining all of these factors can be a challenge in closed-source ecosystems like cloud services.

Most platforms will offer at least some built-in security mechanisms in the form of data encryption, user access control, intrusion detection, security zoning, and so on. Security zones help keep different parts of the pipeline separate from others in terms of which security policies are enforced. For example, personnel performing data analysis might need to use a GPU server cluster to apply a transformation to the dataset. The data analysis zone might enforce strict data control policies that go above and beyond the policies enforced in the GPU server cluster.

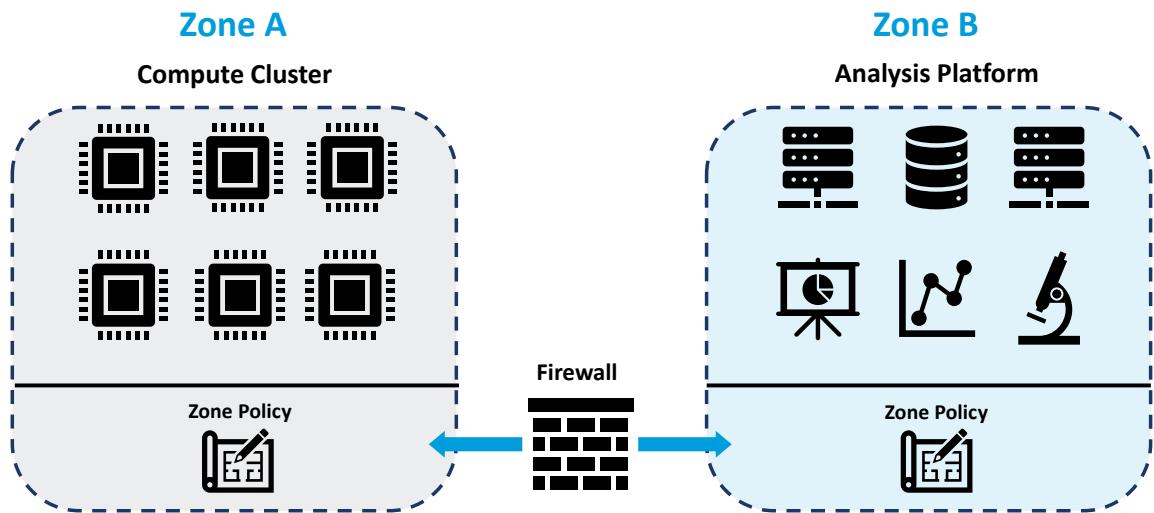


Figure 12-2: Security zones for two distinct elements of a pipeline.

You should not always rely solely on built-in security tools, however. It's always a good idea, whenever feasible, to conduct security tests of your pipeline before it is pushed to production—and even after. For example, a **penetration test**, or pen test, uses active tools to evaluate security by executing what is effectively an authorized "attack" on a system. In the realm of machine learning pipelines, this could mean everything from attempting to obtain sensitive data to conducting poisoning attacks. There are also less direct forms of testing, like vulnerability assessments, in which code, configurations, and running software are scanned for common weak points—without necessarily exploiting them.

Platform Updates

Keeping the pipeline platform up to date is crucial to its security. As software ages, more and more vulnerabilities are discovered—sometimes by security researchers, other times by attackers. Even a single vulnerability can nullify the effects of the security techniques you've applied, or circumvent them altogether.

That's why it's important to have a plan in place for applying periodic updates. The cadence of those updates will at least be dictated by how quickly the software vendor releases them. Many, but not all, vendors choose to release major feature updates every few months, while issuing security fixes and patches as soon as possible. So, you might choose to update on something like a monthly schedule, or you may choose to monitor critical patches and update as soon as they are available.

Keep in mind that, in some cases—particularly with managed cloud services—you won't be able to set your own update schedule. The cloud provider will simply update the service whenever it deems necessary. They might give you an option to update when it comes to major changes to the service, but for security fixes, they'll probably just apply them automatically.

Whatever amount of control you have over platform updates, and whatever cadence you set for those updates, your plan must incorporate testing. Even minor updates can break the platform, so you should have a separate testing environment outside of production that you apply the updates to first. Then you can launch various tests on the updated pipeline—whether something like stress testing or just engaging in normal operations—to determine if you should push the update out to production as is or wait until you can make the necessary changes.



Note: Having a continuous integration/continuous delivery (CI/CD) architecture will make the platform updating process more robust.

Pipeline Job/Task Security

Of course, it's not enough for the overall pipeline to be secure. Each individual component, job, and task within that pipeline must also be secure. Consider that, even if you update the platform regularly, the programming libraries you use to clean data and train models will stay the same unless you also have a process in place to update them. Any vulnerabilities in these libraries become vulnerabilities within your pipeline, so you must be prepared to update them as needed. Keep in mind that updates to programming libraries—especially cutting-edge data science ones—can cause problems in your code or even outright break it. This is yet another case for having a separate testing environment.

Writing secure jobs/tasks in a pipeline involves several best practices, including:

- Reduce any unnecessary complexity.
- Write code to be readable by humans—for example, using meaningful variable names, avoiding double negatives, and following other established conventions for clear coding style.
- Use comments to show and explain how code is intended to be used.
- For common tasks, prefer well-maintained and well-tested existing code, rather than creating new code.
- Review all third-party applications, code, libraries, and APIs to determine whether they are really required, and whether they function safely.
- Do not deserialize any untrusted data source. Use safer text-based loading functions whenever possible.
- Use code signing, such as checksums or *hashes*, to verify the integrity of code, including libraries, executables, interpreted code, resources, and configuration files.
- Run services and applications with the least permissions required for the task they must perform.
- Ensure each endpoint is applying the appropriate level of transmission and storage encryption to the data as it flows from job to job.
- Ensure jobs can handle unexpected conditions and errors gracefully, rather than fail in such a way that could expose a vulnerability.

Access Control

In the realm of security, **access control** is the process of allowing only authorized users or systems to observe, modify, or otherwise take possession of the resources of a computer system or physical property. The two major concepts that implement access control are authentication and authorization. **Authentication** is the act of verifying identity—someone or something is who they claim to be. **Authorization** is the act of assigning an authenticated user a set of permissions or rights by which they can interact with a protected system.

A machine learning pipeline is no different than any other computing system in that it must enforce some level of access control. There are several approaches to access control, some of which may be dictated by the platform you're using.

- **Discretionary access control (DAC)**—Each user is given control over their own data. The data owner can grant different access levels to that data. DAC is commonly used in desktop operating systems, where multiple users may own different sets of data. DAC is not common in machine learning pipelines since there's usually just *the* data rather than any one user's data.

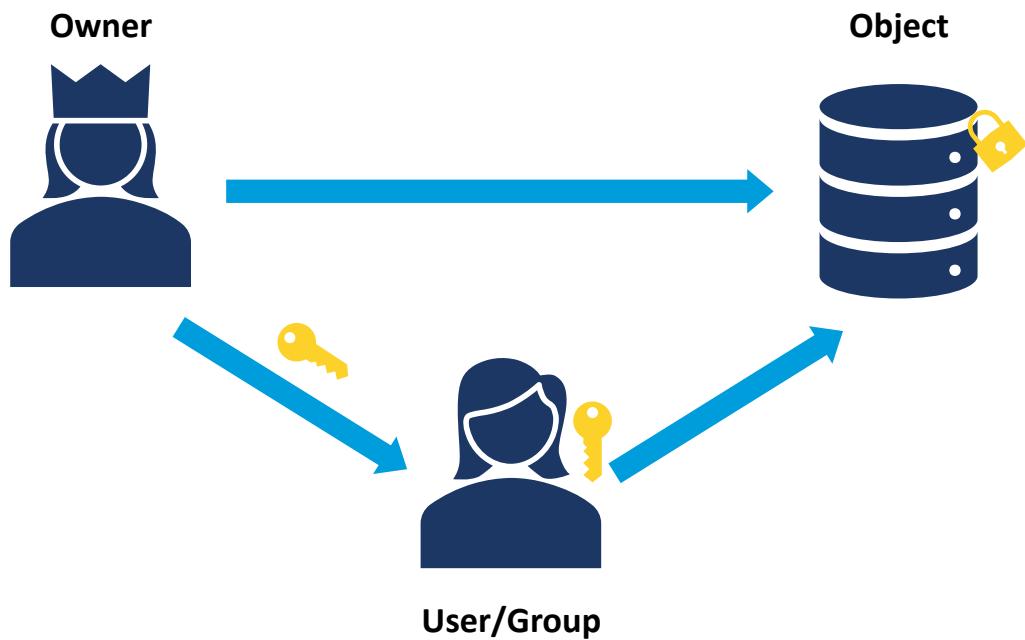
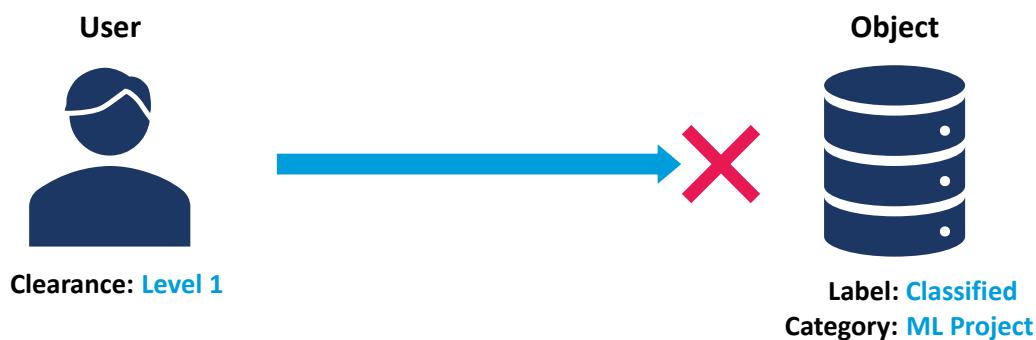


Figure 12-3: A DAC approach in which a data owner grants a user or group access to the data.

- **Mandatory access control (MAC)**—This is the strictest form of access control and is appropriate for systems requiring very high levels of security assurance. Access levels are enforced by the system and cannot be changed by a user or system administrator. Objects (resources) are assigned security labels that indicate the classification (top secret, confidential, unclassified, etc.) of the resource as well as its category (department, project, or management level). Subjects (users) are given security clearance levels. A subject can only access an object if the subject's security clearance level matches the object's classification and category level. MAC is most common in high-security fields like national defense and is not particularly common in machine learning pipelines.



Clearance	Classification for ML Project
Level 4	Top secret, secret, classified, unclassified
Level 3	Secret, classified, unclassified
Level 2	Classified, unclassified
Level 1	Unclassified

Figure 12-4: A MAC approach in which a user is denied access to data because their clearance level is insufficient.

- **Role-based access control (RBAC)**—This is a type of DAC in which users are assigned to roles or personas based on need or job function, and the access level of the role is determined by the system administrator. The roles are typically implemented as groups. If a user belongs to a certain group, that user can access whatever the group can access. RBAC is perhaps the most common form of access control implemented in machine learning pipelines, particularly with cloud services.

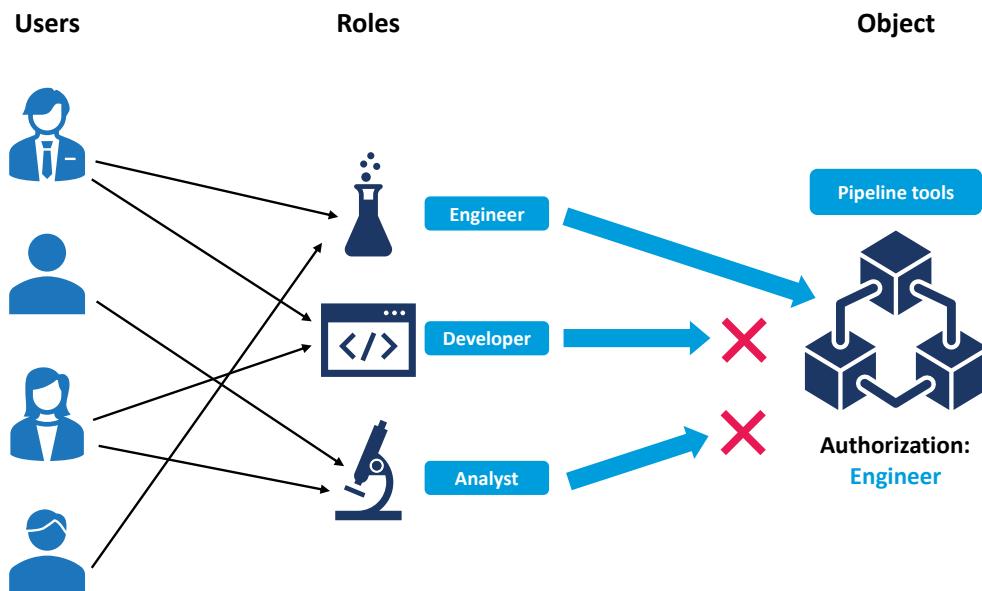


Figure 12-5: An RBAC approach in which only users in the role of Engineer are granted access to pipeline configuration tools.

- **Rule-based access control**—In this type, access is granted based on rules set by the administrator. It doesn't matter who the subject is; access is only granted if the activity matches

the rule. This type of access control is most commonly used on firewalls and routers to control network traffic. However, some pipeline platforms do offer rule-based access control by enabling the administrator to define what type of activity is or is not allowed, such as rejecting user input that contains data well outside of expected distributions.

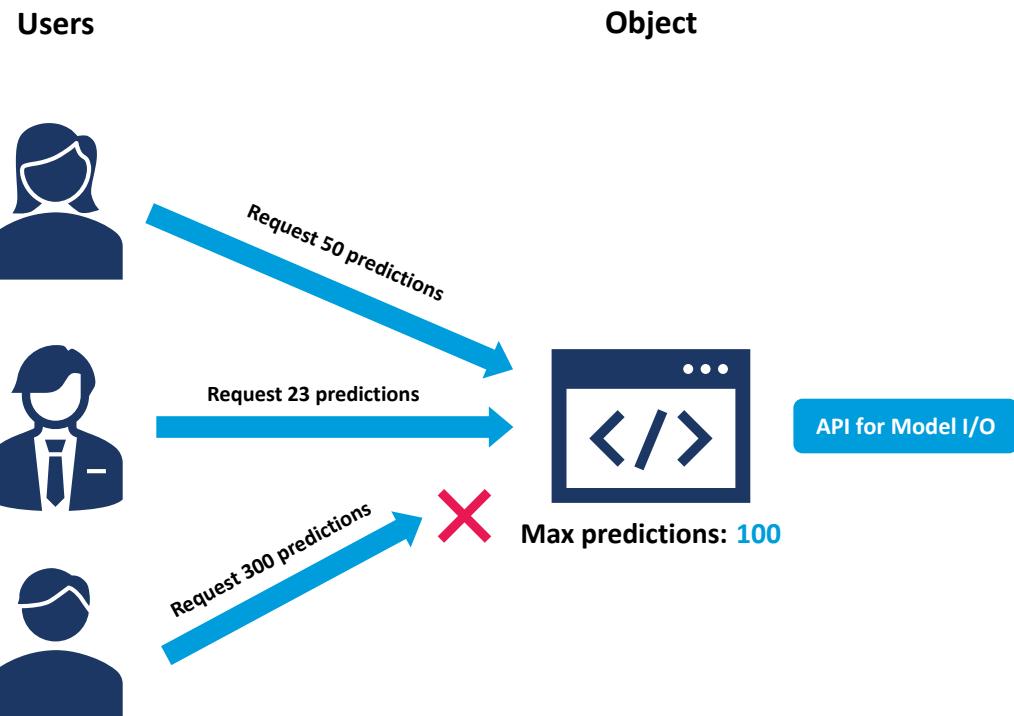


Figure 12-6: A rule-based access control approach in which an API rejects prediction requests that exceed 100.



Note: People occasionally use the acronym RBAC to refer to rule-based access control, though it's more common to assign the acronym to role-based access control. Be careful not to confuse the two.

User Role Management

Assuming you're likely to rely on RBAC as your access control method of choice, the pipeline administrator—whether it's you or a colleague—will need to be able to manage the necessary roles and the users who are assigned to them. This includes defining the roles according to the function of the pipeline and business needs, assigning permissions to those roles, and creating user personas to fill those roles.

When it comes to applying access control of any sort, one of the most important concepts to keep in mind is the **principle of least privilege**. This principle states that users should only be given the rights and permissions they need to do their jobs, and no more than that. That way, if a user persona is compromised, the damage will be limited to whatever permissions that user has.

There is not one correct way to design user roles for every pipeline, but consider the following potential user roles/personas:

- **End users**—These people are directly using the product or service provided by the pipeline. A user in this role may need to send data destined for a model and receive output from that model. Therefore, the end-user role must have permission to send well-formed data to an endpoint that handles model input for estimation. The user must also be able to send data to an endpoint that handles data collection for retraining. In either case, the user needs permission to receive model output, but only if it is tied to their input.

- **End devices/apps**—In many cases, users are not actively participating in the sending and receiving of data for training or estimation. Their devices or apps are doing this for them. So, a separate but similar role should be created for devices and apps that automatically communicate with endpoint APIs.
- **Data scientists**—Users in this role are primarily given permission to work with data in the phases concerning collection, preparation, and analysis. Therefore, someone in this role must have deeper access to one or more pipeline components that an end user or device does not have. This might involve direct access to the code, or access to the data through a layer of abstraction.
- **Machine learning practitioners**—Users in this role are given access to the pipeline components that involve training, evaluating, and testing models. Most likely they are given permission to access the code that involves these tasks. They may also be given access to data in earlier phases of the pipeline, since the performance of the models are obviously tied to that data.
- **Pipeline engineers**—These users are given permissions to construct and modify the pipeline platform and its components. They apply updates, implement security techniques, manage APIs, manage endpoints, and so on.
- **Testers**—Users in this role perform tests on one or more components of the pipeline to ensure they are meeting performance and security expectations. Testers likely have access to test harnesses, which enable them to automate their tests. They may also be given read-only access to pipeline code so that they can customize their test harnesses around code that is specific to the organization's pipeline.
- **Pipeline administrators**—These users are given the authority to delegate users to the appropriate roles, or to remove users from certain roles if need be. They may also have all of the rights and permissions that the other roles have.

Many platforms enable you to define these roles as groups, where you assign permissions to those groups and then assign users to whichever group(s) they need to belong to. Note that a user can belong to multiple groups/roles. This is usually only necessary when you design your roles in a non-hierarchical fashion; for example, the pipeline engineer role is only allowed to make changes to the pipeline and is not also given permission to retrieve model output like an end user would.

User Actions Management

In addition to defining user roles, you need to define the specific actions each role is permitted to take. Some platforms will keep a long list of actions or rules that you can assign as needed. These actions can range from the very broad ("Can send data to prediction endpoint"), to the very granular ("Can send well-formed JSON data through an HTTP POST request to prediction endpoint matching region of origin"). However, not all pipeline platforms offer the same level of support when it comes to user actions, so you may need to define them yourself through code.

When you're thinking about which actions to assign to which users/roles, the principle of least privilege will go a long way. Think about what someone in the role will need to be able to do their job and do it effectively. For example, a machine learning practitioner cannot do their job without being able to write and review code that creates, evaluates, and tests machine learning models. On the other hand, you can think about what someone in a particular role does *not* need to do their job. A machine learning practitioner doesn't need to restructure the pipeline to make it more efficient, even if they believe it will benefit their work. So, they shouldn't be assigned permissions enabling them to modify the entire pipeline. Using this approach will help keep each role distinct from the others and prevent the boundaries between roles from blurring.

You should also consider other factors that go into the actions a user or role can take; not just *what* they can do, but *how*, *when*, and *where*. For example, an end-user app should obviously have permission to send data to the pipeline to use in retraining. But to protect against model poisoning attacks, you may only allow the app to send a certain amount of data within a certain time span. And, you might limit apps from sending data if their IP addresses originate from a known malicious range. There are many such ways you can make access control more robust in the pipeline.

Guidelines for Securing Machine Learning Pipelines



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

Follow these guidelines when securing machine learning pipelines.

Secure Machine Learning Pipelines

When securing machine learning pipelines:

- Build security processes into the beginning of the pipeline development lifecycle and throughout all phases; don't just tack it on at the end.
- Ensure you understand the main risks associated with most pipelines: data leakage and attacks on machine learning models.
- Review your platform for any built-in security mechanisms and leverage them whenever possible.
- Review your platform for potential areas of weakness that could be exploited in an attack.
- Employ common security techniques like encryption, access control, intrusion detection, and security zoning within the platform.
- Conduct tests of the pipeline's security to ensure it is meeting expectations.
- Update the pipeline platform on a regular schedule, if you are able to.
- Test updates on a separate environment before pushing to production.
- Employ secure coding techniques to protect individual pipeline jobs and tasks.
- Employ access control methods like RBAC to ensure users and devices are granted the appropriate level of access to pipeline resources.
- Design user roles that make sense for the pipeline you've built and the purpose it is intended to serve.
- Assign actions to roles that enable users in each role to do their job successfully.
- Configure actions/rules to control access more granularly, as needed.
- Observe the principle of least privilege in all aspects of access control.

ACTIVITY 12-1

Securing a Machine Learning Pipeline

Data Files

All files in /home/CAIP/MLOps

Before You Begin

The Docker container is running, as is the web app. You are on the **CapitalR Real Estate—Predictor** page.

Scenario

Your pipeline is set up, as is the web app frontend it's integrated with. However, you need to confirm that both are adequately secured. Otherwise, attackers could compromise what you've worked so hard to achieve.

There are many security techniques you could apply, but for now, you'll:

- Restrict access to different elements of the pipeline using a role-based access control (RBAC) approach.
- Ensure the integrity of the model through hashing.

1. Attempt to access the admin page.

- a) Select the **CapitalR** logo to return to the home page.
- b) Select the **Admin** icon.
- c) Verify that you are presented with an error indicating that you have been denied access to the admin page.

You are not authorized to access that resource.

Recall that you're logged in as **agent@capitalr.co**. The "agent" persona does not have the rights to access the admin page.

2. Examine the **authentication.py** script.

- a) Return to the file browser and double-click **authentication.py** to open it.
- b) Observe lines 6 through 16.
 - Lines 6 through 9 create a class with three roles based on the three main frontend components: admin, predict, and train.
 - Lines 12 through 16 create three main users. Each user has a user name, a password, and a role:
 - **admin@capitalr.co** has the **ADMIN** role.
 - **agent@capitalr.co** has the **PREDICT** role.
 - **data_sci@capitalr.co** has the **TRAIN** role.
- c) Observe lines 36 through 41.

The `valid_login()` function checks to see if the user is in the list of valid users. If not, the function returns `False`.



Note: In a real-world app, you'd hash the users' passwords and store those hashes in a separate file, rather than hardcode the passwords in plaintext, as in this example.

- d) Observe lines 44 through 64.

The `with_role()` function handles user sessions or other access attempts.

- Lines 48 through 51 handle access attempts using an API key instead of a user session.
- Lines 55 through 60 handle user sessions using the previously defined roles.
- This function comes into play in the `app.py` script.

- e) Close `authentication.py`.

3. Examine how access is controlled from the `app.py` script.

- a) Double-click `app.py` to open it.

You've looked at this file before, but now you can focus on the code that implements access control.

- b) Observe line 68.

- Underneath the route to the `/predictor` page is a decorator that calls the `with_role()` function from `authentication.py`.
- The function is being called using the `ADMIN` and `PREDICT` roles. In other words, users with either of those roles are being granted access to the page that presents the house prices prediction form.

- c) Observe line 80.

This is another access control decorator statement, this time restricting access to the `/admin` page to only those in the `ADMIN` role.

There are many more such statements throughout this script, each one controlling access to some functionality in the web app based on the predefined roles.

- d) Examine lines 88 through 117.

These functions implement authentication through the web app's sign-in/sign-out form.

- Lines 91 through 103 initiate a POST request so that the authentication service can validate the credentials that the user input in the sign-in form.
- If the credentials match the list of valid users, the user is signed in. If not, the user is given a warning message.
- Lines 109 through 116 sign the user out, assuming they're already signed in.

- e) Close `app.py`.

4. What is the advantage of using an RBAC approach instead of just giving individual users access to the pipeline elements they need?

5. Access the admin page using the proper credentials.

- a) Return to the web app in Firefox.
- b) Select the **Sign out** button.
- c) Verify that you have been signed out.

You have been signed out.

- d) Select **Sign in**, then sign in as **admin@capitalr.co** with a password of **secret**
- e) Select the **Admin** icon.
- f) Verify that you are able to access the **CapitalR Real Estate—Admin** page.



Note: You'll take a closer look at this page in the next activity.

- g) Select the **CapitalR** logo to return to the home page.

6. Corrupt the model file to simulate an integrity violation.

- a) Return to the terminal running the web app.
- b) Select **File→Open Terminal** to open a new terminal.
- c) Verify that the current directory is `~/CAIP/MLOps/docker`.

```
(base) student@debian:~/CAIP/MLOps/docker$
```

- d) Enter `cd model` to change to the `model` directory.
 - e) Enter `cp kc_rf_regressor.pickle kc_rf_regressor_BACKUP.pickle && echo 'corrupt' >> kc_rf_regressor.pickle`
- There are two commands in one statement:
- The first command creates a backup of the original model file.
 - The second command adds garbage data to the end of the model file.

7. Confirm that the web app detects the integrity violation.

- a) Return to the terminal running the web app.
- b) Press **Ctrl+C** to stop the web app from running.
- c) Press the **Up Arrow** once to retrieve the command that starts the web app.



Note: If it doesn't appear, the command is `FLASK_ENV=development flask run --host=0.0.0.0 --debugger`

- d) Press **Enter**.
- e) Verify that the web app fails to execute and identifies a hash mismatch as the reason.

```
AssertionError: Hash mismatch on model:  
[Saved] 960e5cda18a2e8428c7c4c179f6a42a4ed0f5320ec676280fa505a32936ef17f  
[Model] ed5e5fb16510a234e1a1c43451b1551cd91f61014da1af2d4450b095ec5b0c4d  
caip@2df25bf70594:~/web_app$
```

Hashing is a cryptographic technique in which an algorithm transforms plaintext input into an indecipherable fixed-length output. It has several uses, one of which is to verify the integrity of data. Since hashing is deterministic, the same exact data will always produce the same exact hash. If that data is altered at all—even by just changing a single bit—the hash will be completely different.

- When the model was trained, the hash of that model was saved to a file.
- When the model is loaded into the web app, the app hashes the model and checks to see if it matches the saved hash.
- Since the two hashes are different, it confirms that the model was altered since it was last trained.

8. Examine the code that implements the hash check.

- a) Open the `docker/scripts/kc_train.py` script in Mousepad.
- b) Examine lines 80 and 81.

This is the hashing function you saw earlier. It uses the SHA-256 algorithm to perform the hashing of the input data.



Note: Different algorithms produce different hashes. If you're using hashing for integrity verification, you must be sure to perform all checks using the same algorithm.

- c) Examine lines 375 through 378.

These lines actually perform the hashing of the model, then save the hash output to a file.

- d) Close `kc_train.py`.
e) Open `docker/web_app/model_operations.py`.
f) Examine lines 25 through 28.
- Part of the `load_model()` function is to hash the loaded model and then compare the output to the hash that was saved during training.
 - Line 28 raises an `AssertionError` if the hashes do not match.
- g) Close `model_operations.py`.

9. Restore the backup model file and restart the web app.

- a) Return to the terminal where you created the backup and corrupted the model file.
The prompt should show `~/CAIP/MLOps/docker/model` as the current directory.
b) Enter `mv kc_rf_regressor_BACKUP.pickle kc_rf_regressor.pickle`
c) Return to the terminal with the Docker container prompt.
The terminal should show `~/web_app` as the current directory.

- d) Restart the web app using the same command as before.



Note: Remember, you can press the **Up Arrow** to retrieve the last command you entered.

- e) Verify that the web app started without any errors.

```
caip@2df25bf70594:~/web_app$ FLASK_ENV=development flask run --host=0.0.0.0 --de
bugger
  * Environment: development
  * Debug mode: on
  * Debugger is active!
  * Debugger PIN: 213-329-331
```

- f) Leave the web app running in the terminal.

10. Currently, the pipeline can only detect a model mismatch when the web app is started. If the web app is running and the model is corrupted, nothing happens.

How might you ensure that the web app handles model integrity violations dynamically?

TOPIC B

Maintain Models in Production

You've successfully operationalized your models, but you're not done yet. You need to ensure those models stay effective and continue to bring value to the business over time. Maintaining the model after it's been pushed to production is therefore crucial.

Pipeline Monitoring

One of the major benefits of a production pipeline is that every intermediary task, as well as its output, is contained within a manageable environment. This should make it easier for anyone operating or administrating the pipeline to monitor what exactly is going on. You don't want to be in a situation where you set up the pipeline, pull the data through, and end up with a result you weren't expecting. Your model might outright fail at some point due to a misconfiguration, or it could experience more subtle effects like skill degradation. Another possibility is that the process has unwanted side effects, like private user data leaking outside of the pipeline. Whatever the issue may be, if you catch it early enough, you can avoid wasting a lot of time and effort. This is why it's important to be able to monitor your environment.

What's actually involved in monitoring will depend on the platform you're using and the needs of the business. For example, you might log each epoch of a model in training to ensure it is progressing along an acceptable schedule. If the model's training speed appears to decrease beyond some threshold, you can be notified and then take action. Or, perhaps you have a dashboard that continually updates the evaluation metrics for a model as it undergoes hyperparameter tuning. You can quickly verify whether your model's skill is improving or degrading over time. Essentially, you can monitor your models at any level of granularity you desire.

Logging: Events to Log

As part of your pipeline monitoring efforts, you'll likely need to produce logs of some type. Logs can record many different types of events, including:

- **Access attempts to endpoints.** For example, whenever a user or device sends input data to your model estimation API.
- **Changes to the pipeline configuration.** For example, if a user sets up a new endpoint.
- **Changes to roles or permissions.** For example, an administrator moves a user into a role/group with more privileges.
- **New data collected.** For example, if a new set of training data enters the pipeline where it will eventually be used to update a model.
- **Errors, issues, or other unusual events in data preparation.** For example, if a large amount of training data is missing and cannot be easily imputed.
- **Outcomes of new training cycles.** For example, whenever a model is retrained on new data, including the version of the model, the time it took to train, and its scores on a test set.
- **Errors between pipeline components.** For example, if the pipeline is unable to take a model from the training environment and deploy it to production.
- **Failed connections and other end-user issues.** For example, the pipeline cannot properly complete a user request due to latency issues.
- **Intrusion attempts or other signs of compromise.** For example, data is being transmitted outside of the pipeline at an anomalous rate.

These are just a few examples—there are many more events you could log in your environment to help you identify security or performance issues. It might seem like a good idea to log everything, but keep in mind that the more you log, the harder it will be to separate the signal from the noise.

Too much logging can also become a burden to both processing power and storage capacity, so make sure you're only logging the events you've identified as the most critical to your operations.

Logging: Format

In some cases, you may have control over how the events in your environment are logged. Even if you don't, you should at least be familiar with the format of the logs your environment produces. Otherwise, you'll have trouble reading them and gleaning the relevant information.

Most logs record the *when*, *where*, *who*, and *what* of an event.

- **When**—The date and timestamp of an event. In some cases, when the logging service is not instantaneous, there might be two separate timestamps: one for when the event took place, and one for when the event was logged.
- **Where**—The "location" of an event, i.e., where it took place virtually. This can be everything from the name of the server that processed the event, to the network address of the server, to the specific code module that handled the event, and so on. In situations where an event crosses multiple virtual locations, the relevant data for each location may be logged.
- **Who**—The user, device, or service that initiated the event. This can be everything from the user's real name, to their account name, to their IP address, and so on. Or, if a device or service, its identity within an account management database.
- **What**—The type of event, or the actual content of the event. The event might be categorized, such as a security violation, a network connection issue, a code execution error, an HTTP request status, and so on. The severity of the event might also be ranked, such as 0 for an event needing the most urgent attention, and 7 as a simple debug message that doesn't necessarily indicate a problem. Most logs will also include some concise description of the event.

Logging: Best Practices

When you implement logging in your production environment, you should ensure that your logs are:

- **Auditable**—All events that are tracked must be formally documented by the system, and those logs must be kept for the designated time period.
- **Traceable**—Logging must be able to show the path an event took before it was logged so that you can more easily determine the root cause.
- **Visible**—Although someone on the team might manually review logs, in most cases, a critical event should immediately trigger an alert that will notify someone with the proper authority to address the issue.
- **Upholding integrity**—You must be able to ensure that logs have not been overwritten or tampered with by unauthorized users.
- **Confidential**—Logs must be stored securely behind layers of access control and encryption, not just to prevent tampering, but to prevent unauthorized users from reading the logs and gleaning potentially sensitive information about the environment.
- **Judicious**—Certain events or aspects of an event should not be logged, like sensitive user data, user credentials, application source code, etc., to minimize risk if the logs were compromised.
- **Legally compliant**—Laws and regulations may dictate what you can and cannot log, particularly when it comes to private user data.

Continuous Testing

A major part of maintaining a model in production is to identify problems with that model (or in other pipeline components). However, not every problem will be revealed in a log. That's why you need to implement a process of continuous testing. Testing is not just done prior to production, or even once or twice after production, but repeatedly over an indefinite amount of time.



Note: Continuous testing can be considered one part of continuous integration (CI). As you might recall, CI is necessary for MLOps level 1 maturity.

There are many types of tests you can perform: security assessments, penetration tests, "health" checks on pipeline resources, model performance tests, code reviews and walkthroughs, unit tests on modules, system integration tests, regression tests to ensure that software still works as expected after a change, and so on.

Tests should be conducted in a virtual location that is separate from the production environment so as not to disrupt live operations. The testing environment can be a 1:1 copy of the production environment, or you might configure it in multiple different ways to identify the effect that changes have on the environment.

Model Drift

Over time, machine learning models may become less effective in making estimations on new data due to **concept drift**, also called **model drift**. Model drift occurs when the domain being studied has variables that naturally change over time. This accounts for a great deal of domains, but some change much more rapidly than others, so model drift may be more of a concern in certain domains than others.

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. Websites or apps that used to generate a lot of traffic may decline in popularity, so advertisements on those sites don't produce the same results they used to. The effectiveness of the model may decay as the concept it is trying to estimate 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 when they are integrated into long-term software solutions. Online models, for instance, can help stem the tide of model drift. Since the model's knowledge of some domain is constantly kept up to date, the assumptions it expresses about that domain can also be considered up to date. In the example of online shoppers' behavior changing over time, you may deploy the solution so that this behavior is continuously tracked and then fed into a pipeline where it can produce a new model. Assuming the model maintains a level of skill, it can help you make advertising decisions that are based on current trends rather than obsolete data.

However, keep in mind that online models are usually more difficult to implement because they require a constant ingest of new and useful data, which may not always be available. It also takes a lot of time and resources to constantly train, tune, test, and deploy updated models.

Concept Drift vs. Data Drift

Some practitioners distinguish concept drift from data drift, and may consider both as subsets of model drift. In such cases, concept drift refers to a change in the fundamental relationship between model inputs and outputs, whereas data drift refers to a change in the input data (particularly the distributions of its variables).

For example, a gradual rise in house prices over time leads to data drift for a model that was trained on labeled house price data. On the other hand, a model that classifies bank activity as fraudulent or not fraudulent will experience concept drift if the definition of "fraudulent activity" changes over time.

In practice, these terms are often used interchangeably.

Model Retraining: Detecting Drift

To combat model drift, you must prepare to retrain your models over time. The first step in retraining is to actually detect the presence of model drift. Even if you plan to retrain on the model

at regular intervals, it's a good idea to see how performance has degraded in your existing model, if at all.

If you have access to new test data (i.e., labeled data that was collected recently and reflects the current state of the knowledge domain), then you can evaluate your existing model's performance on that new data. If it performs worse, then your model *might* be drifting. Recall that the stochastic nature of many machine learning algorithms makes it difficult to get consistent results every time, so even if the model performs slightly worse, it may not indicate a larger pattern of drift. Therefore, you might want to evaluate on multiple new test sets, or collect new test sets over several time intervals. If the model continues to perform worse, then drift is more likely. You can also employ hypothesis tests on the model to determine if there is statistical significance in the hypothesis that the model is underperforming due to drift rather than due to random chance.

Another, related strategy for detecting drift is to train a new version of the model on new data every so often. So, instead of just evaluating one model on new test data, you train a separate model on new data you split into training and testing sets. If the retrained model scores significantly better on the testing set than the existing model, you could be seeing the effects of model drift.

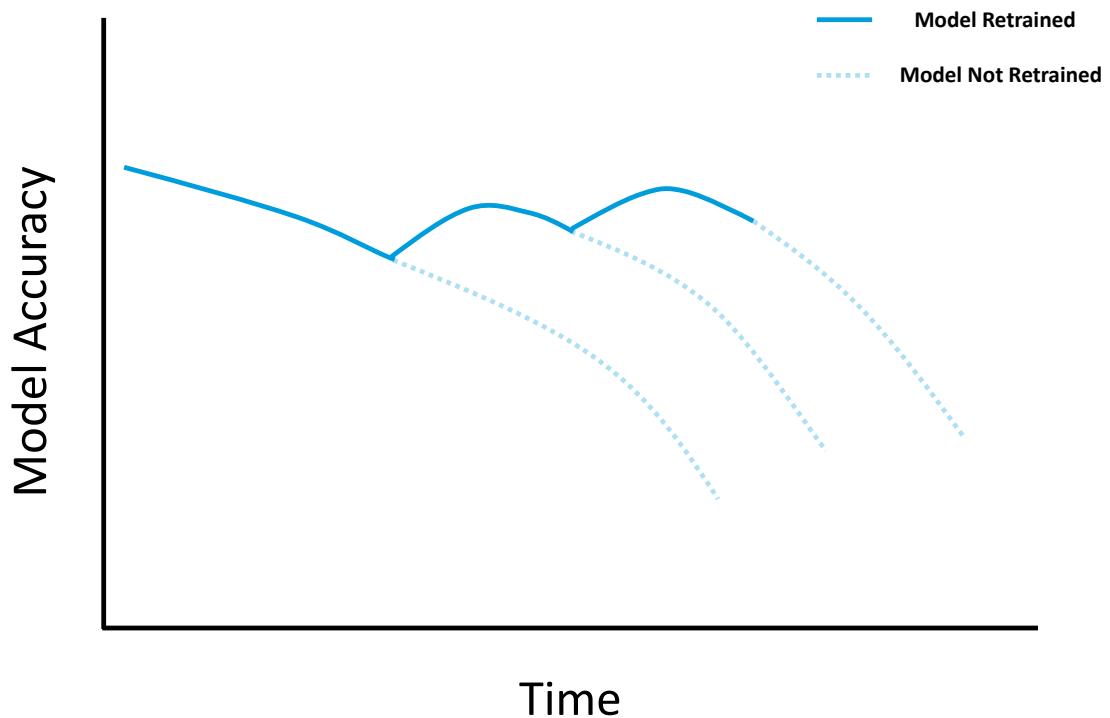


Figure 12-7: Demonstrating model drift over time by comparing a retrained model to a model that is not retrained.

Alternative Approaches to Detecting Model Drift

If you don't have enough new labeled data to test your model on, then you'll need to use an alternative method for detecting drift. These methods only really work if you have *some* data—e.g., you have the features but not the labels, or you have the labels but only some of the features, etc.

One method is to compare the feature distributions of the new set to the distributions of your existing set. If the distributions appear to change over time, then you can infer that the model will experience some drift.

A related method is to compare the distributions of the target variable (labels) across the datasets, assuming you have the labels.

You can also compare how each feature is correlated within each dataset. Assuming the feature correlations should remain static, then any change in those correlations could indicate drift.

Retiring a Model

In some situations, you may not be able to continuously acquire new labeled data from which to retrain a model. Model drift may, therefore, be unavoidable. So, after the model starts drifting, you may decide that the model is no longer bringing value to business and should be retired. The difficulty is determining when the drift starts to occur. If you don't have data to train on, you may not have enough data to test on either. So, you may need to employ one of the previously mentioned alternative detection methods.

Model Retraining: Changing Data Only

Keep in mind that "model retraining" can mean multiple things, or at least it can have multiple implications. For example, you could:

- Use a different dataset for training.
- Engineer different features from the same dataset.
- Train the model with different hyperparameters.
- Train a model using a different algorithm.
- Employ some combination of these approaches.

However, in the strictest sense, retraining the model means gathering new data for training, and nothing else. All of the features should be engineered in the same way; the same hyperparameters used; the same algorithm used; and so on. The only variable is the data itself. This is because *concept* drift implies that something in the knowledge domain the model applies to has changed, not that your way of building the model is outdated.

Still, it's perfectly valid (and even encouraged) to alter other aspects of the training process to produce a better model. But, you should consider that a *different* model, requiring a different process for testing and evaluation before you push it to production.

Model Retraining: When to Do It

Recall that there are several triggers for collecting new data in the pipeline: on demand, on a schedule, on availability of new data, and on the degradation of the model. Which trigger you choose is highly dependent on the model itself and your pipeline infrastructure. You can even use multiple triggers instead of relying on just one every time.

For example, to predict customer churn (i.e., whether they will return to make more purchases), you may collect data quarterly. You could therefore retrain your model each quarter with a new set of data. That follows a scheduled approach. On the other hand, if some major product rollout happens, you may want to collect data immediately after that rollout rather than wait for the quarter to end. That's an on-demand approach. In another scenario, your team may not be able to collect sufficient data for the next quarter, so you'll have to wait for the data to be available in order to retrain your models. Or, perhaps you don't expect your customers' attitudes to change all that frequently, and you opt to monitor your model for signs of drift, then retrain only if you've confirmed the drift.

Ultimately, you need to decide what is best for the business and what you can actually support with the environment you have. Although retraining on a schedule is often ideal, if you don't have the data or the processing power to keep up with this schedule, then it won't work for you.

Nevertheless, if you know your model is going to drift at some point, then choosing any retraining approach is better than choosing none at all.

Automatic Retraining

Depending on the tools available with your pipeline platform, you may be able to automate model retraining. Some monitoring tools can detect signs of drift within the model and then trigger the retraining process without waiting for an engineer to do so manually. The engineer should still

receive alerts from the monitoring system so that they are aware of any retraining, and what prompted it.

Checkpoints and Rollbacks

Another advantage of having an automated virtual environment on which to build your machine learning models is the ability to quickly and easily revert to prior versions of a model should something go wrong. Perhaps the model was retrained on corrupted or inaccurately recorded data (whether accidental or intentional). Maybe the model is starting to degrade—not due to drift, but due to some configuration error that causes later versions to perform worse than earlier ones. Or, the model may have interoperability issues due to some change elsewhere in the pipeline. Whatever the reason, being able to go back to a known, good version of your model is a good option to have.

Virtualized environments, regardless of whether they host machine learning projects, usually have some sort of checkpoint system built in. Checkpoints capture the state of the environment at a certain point in time. This includes all of the data stored on the environment, as well as any data in memory. If you make changes to the environment, then create another checkpoint, that second checkpoint will record the difference between the current state of the environment and the state it was in from the previous checkpoint. This enables you to roll back to prior versions whenever you need to.

Each platform will handle checkpoints differently. Unmanaged services will likely expose this checkpointing functionality to you directly. Managed cloud services, even if they don't, might have some sort of versioning functionality for the models you create. In any case, you should investigate how to enable versioning for your models. If none is available, you should seriously consider writing code to do it yourself. Or, integrate a third-party versioning library that can do it for you.

Guidelines for Maintaining Models in Production

Follow these guidelines when maintaining models in production.

Maintain Models in Production

When maintaining models in production:

- Establish a process for monitoring the security and performance of models while in production.
- Ensure logging is enabled and that the appropriate events are being logged to make tracking pipeline issues much easier.
- Recognize that the more you log, the harder it will be to identify actual issues among a sea of benign events.
- Consider the strain that logging will place on your environment, both in terms of storage space and processing power.
- Identify the format that logged events will take to make it easier to read them.
- Follow other best practices for logging, such as ensuring logs are auditable, traceable, visible, etc.
- Employ a process of continuous testing so that you can identify issues that are not always apparent in logs.
- Conduct tests in an environment that is separate from production.
- Use one or more approaches to detecting model drift.
- To retrain a model, provide new data to the model without changing any other part of the process.
- Consider changes to the process as producing *new* models that require new tests.
- Assess how often you'll expect to obtain new data, and how often your operational infrastructure can handle retraining models, to determine when you should perform retraining.
- Leverage checkpointing and versioning functionality for your environment so you can quickly roll back changes that have undesired effects.

ACTIVITY 12-2

Maintaining a Model in Production

Data Files

All files in /home/CAIP/MLOps

Before You Begin

The Docker container is running, as is the web app. You are on the home page and are signed in as `admin@capitalr.co`.

Scenario

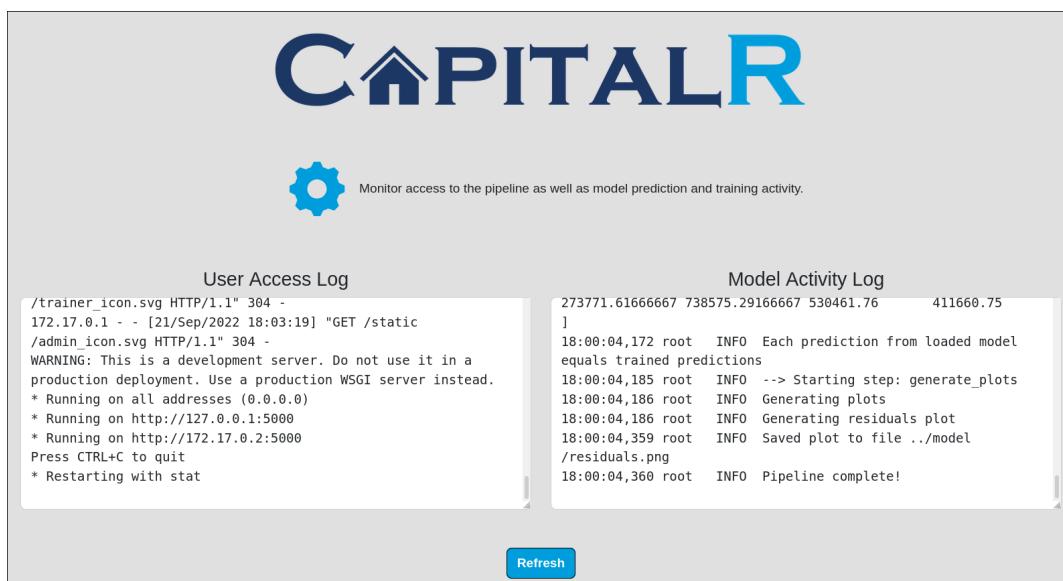
Your machine learning system has been put into production and has been a hit with CapitalR's real estate agents. However, this doesn't mean your work is done. You still need to maintain the system so that it continues to provide value to the business.

First, you'll ensure the system has an adequate logging capability so that you can track user access and model training activity over the system's lifespan.

You're also concerned that the model may be outdated. After all, it was trained on house purchases from 2014–2015, and the real estate market has changed significantly since then. Thankfully, you have access to more up-to-date housing data that you can use to retrain the model and hopefully avoid the problems associated with concept drift.

1. Examine the access and activity logs that have been generated so far.

- On the home page, select the **Admin** icon.
- Verify that you are shown two logs: **User Access Log** and **Model Activity Log**.



The screenshot shows the CapitalR dashboard with the following interface elements:

- CapitalR Logo:** Large blue logo at the top center.
- Monitoring Icon:** A blue gear icon with a house symbol inside, labeled "Monitor access to the pipeline as well as model prediction and training activity."
- User Access Log:** A log table with the following entries:

/trainer_icon.svg HTTP/1.1" 304 -	172.17.0.1 - - [21/Sep/2022 18:03:19] "GET /static
/admin_icon.svg HTTP/1.1" 304 -	
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.	
* Running on all addresses (0.0.0.0)	
* Running on http://127.0.0.1:5000	
* Running on http://172.17.0.2:5000	
Press CTRL+C to quit	
* Restarting with stat	
- Model Activity Log:** A log table with the following entries:

273771.61666667 738575.29166667 530461.76 411660.75]
18:00:04,172 root INFO Each prediction from loaded model	
equals trained predictions	
18:00:04,185 root INFO --> Starting step: generate_plots	
18:00:04,186 root INFO Generating plots	
18:00:04,186 root INFO Generating residuals plot	
18:00:04,359 root INFO Saved plot to file ../model	
/residuals.png	
18:00:04,360 root INFO Pipeline complete!	
- Refresh Button:** A blue button at the bottom center labeled "Refresh".

This page acts as a simple dashboard for reviewing pipeline activity. Each log has recorded events generated by the pipeline's Python scripts at key points. The logs are formatted so that the most recent entries are at the bottom.

- Observe the **User Access Log**, scrolling up as needed.

This log has various entries, including:

- RESTful API requests to endpoints and other resources associated with the web app, like GET requests to the `/predictor` endpoint.
- The content of the prediction request—i.e., the housing form data—along with the prediction returned from the model.
- Messages indicating the app is starting and stopping, along with its network location.
- Messages indicating authentication attempts (i.e., users signing in). Whether or not the attempt succeeded is also recorded.
- Miscellaneous warnings.

The API requests are formatted to include the IP address that originated the request, the time and date of the request, the content of the request, and the status code of the response.

d) Observe the **Model Activity Log**.

The entries in this log are primarily concerned with each step in the automated training process—everything from loading the data to training and saving the model. There are entries for most of the functions you saw in `kc_train.py`.

The format of these entries is the date and time of the event, the user that initiated the event, the severity of the event (mostly `INFO` and `DEBUG`), and the content of the event.

2. Why is it important to maintain and monitor logs such as these?

- Evaluate the current model's performance on a single sample of new, labeled data.
 - Return to the home page, then open the **Predictor** page.
 - Fill out the form with the following information:

<i>Form Field</i>	<i>Value</i>
ZIP Code	98122
Year built / renovated	2013
Bedrooms	3
Bathrooms	2.75
Floors / stories	1
Lot size (sq. ft.)	5563
Living space size (sq. ft.)	2400
Basement size (sq. ft.)	1200
Condition	(3) Fair
Grade	(8) Good
View	(0) None
Waterfront property?	No

This is an actual house sale that took place in late August of 2022. The label (i.e., the sale price) was **\$1,460,000**.

You'll use this as a single point of test data to see how well (or poorly) the current model performs when applied to a more recent real estate market.

- c) Submit the form and verify the resulting prediction.

The prediction is **\$657,935**—an error of **\$802,065**. That's quite a large difference in predicted price vs. actual price. This could just be an outlier, but given the age of the original data, it's safe to assume that the model is experiencing drift.

4. Briefly examine the new training and testing data.

- If necessary, open a file browser to `~/CAIP/MLOps/docker/housing_data`.
- Open `kc_house_data_new_train.csv` in Mousepad.
- Examine the first few lines of the dataset.

```

1 "id","date","price","bedrooms","bathrooms","sqft_living","sqft_lot","floors","waterfront","view","condition"
2 "4278900003","20220829T000000",1460000.0,3.0,2.75,2400.0,5563.0,1,0,0,0,"Fair",8,1200.0,1200.0,2013
3 "4385701065","20220826T000000",776000.0,1.0,1.0,840.0,3000.0,1,0,0,0,"Fair",6,840.0,0,0,1916.0,"981
4 "5316100395","20220825T000000",2050000.0,4.0,3.75,3190.0,4304.0,1,0,0,0,"Good",8,2190.0,1290.0,1928
5 "5700000790","20220824T000000",1750000.0,4.0,3.25,4790.0,5000.0,3,0,0,0,"Fair",9,3590.0,1520.0,2016
6 "5700001540","20220823T000000",1535000.0,4.0,2.5,2860.0,5000.0,1,0,0,0,"Good",7,1990.0,870.0,1921.6
7 "9829200675","20220823T000000",1490000.0,2.0,2.0,1240.0,7500.0,2,0,0,0,"Fair",8,1240.0,620.0,1976.0
8 "220000005","20220817T000000",1700000.0,2,0,1,5,2200.0,6200.0,2,0,0,0,"Good",0,2200.0,1100.0,1006

```

- These house sales were collected from the period between June 2021 and June 2022.
- There are ~27,000 records, which is a few thousand more than the original 2014–2015 dataset.
- The structure and format of the data is similar to the original dataset as well.

- Close the file.
- Verify that the folder also contains `kc_house_data_new_test.csv`.

This data is similar, except that it ranges from June 2022 to August 2022. You'll use this data to evaluate the fit of the current model as well as the retrained model.

5. Examine the code that implements model testing and retraining.

- Open `docker/MLOps/web_app/model_operations.py` in Mousepad.
- Observe lines 120 through 146.

The `test()` function takes a labeled dataset file as input and uses that to evaluate the current model.

- Lines 122 through 126 run a testing script using the specified dataset. This script, `kc_test.py`, essentially calls the functions of the `kc_train.py` pipeline that prepare the data. It then implements its own custom testing logic using the current model.
- Lines 129 through 133 check for errors in reactivating the training script.
- Lines 137 through 141 display a residuals plot to the user based on the model's test performance.
- Line 144 reloads the model.
- Line 146 returns attributes of the tested model.

- Observe lines 148 through 174.

The `retrain()` function initiates retraining based on user input.

- Lines 150 through 154 run the training script using the specified dataset as a new source of training data.
- The rest of the code in this function is similar to code in the `test()` function.

- Close the file.
- Open `docker/MLOps/web_app/app.py` in Mousepad.
- Observe lines 180 through 225.

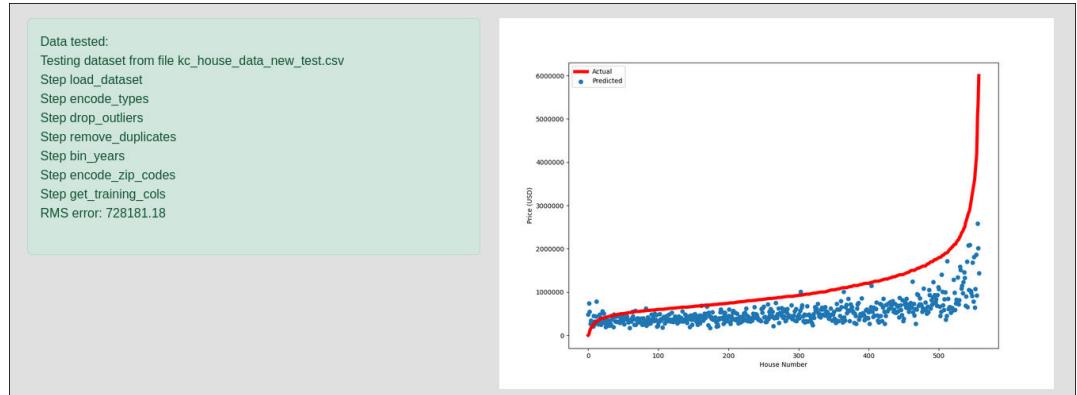
This is the endpoint used for testing or retraining the model.

- Lines 182 through 194 handle the file upload and copying the file to the proper data directory.
- Lines 197 through 225 use conditional logic to call either `retrain()` or `test()`, depending on what the user selects on the page.
- The code in each conditional branch outputs results to the user.

- Close the file and return to the web app home page.

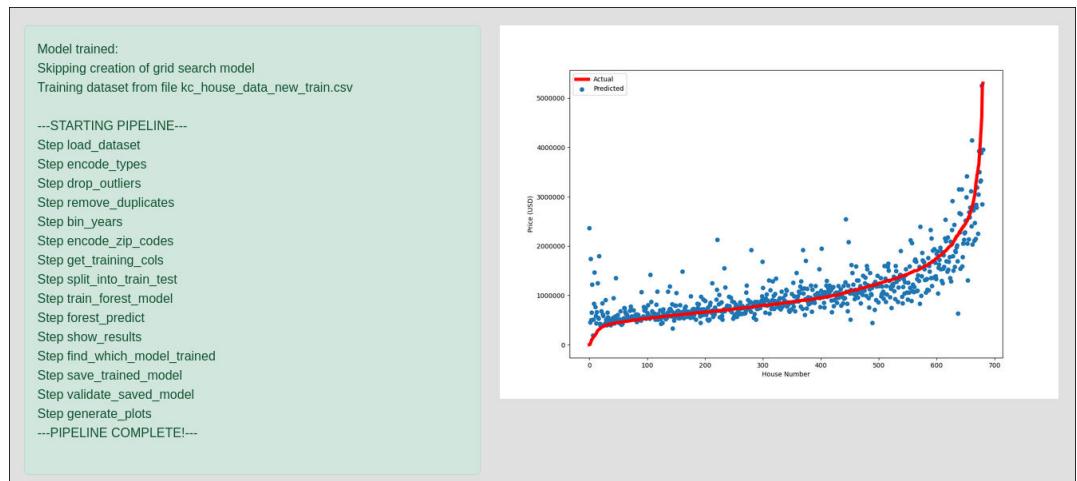
6. Evaluate the current model's performance on the entire test dataset.

- Select the **Trainer** page.
- Ensure the **Evaluate** radio button is selected.
- Select **Browse**, then navigate to `student/CAIP/MLOps/docker/housing_data`.
- Double-click `kc_house_data_new_test.csv`.
- Select **Submit**.
- After the page finishes loading, verify that the model's error value and its residual plot are shown.



- The current model was trained on years-old real estate data. The test data you just input is much more recent.
- The RMSE of the model on this new data is 728,181.18. This is fairly high.
- The residuals plot shows that almost all of the error comes from underestimating the price of homes. That makes sense—home prices have gone up considerably since the original data was collected.
- This is likely evidence of drift, so you'll retrain the model on the newer data.

- Retrain the model using the dataset of King County house sales from 2021–2022.
 - Select the **Retrain** radio button.
 - Select **Browse**, then double-click `kc_house_data_new_train.csv`.
 - Select **Submit**.
 - After the page finishes loading, verify that the model was successfully retrained, and that you are shown the residuals plot of this retrained model.



- Evaluate the retrained model's performance on the entire test dataset.

- Select the **Evaluate** radio button.
- Submit `kc_house_data_new_test.csv`.

	Caution: If the page appears to stall, check the terminal running the web app. If it shows a command prompt for the <code>caip</code> user, then the app has crashed. Restart the app, reload the web page, and try submitting the file again.
---	---

- Verify that the error score and residuals of the retrained model.
 - The RMSE has almost been reduced by half—it's now 465,630.06. Even though this may still be high overall, it's a significant improvement over the original model.
 - The residuals plot confirms that the new model is no longer underestimating house prices, at least not in the aggregate.

9. Evaluate the retrained model's performance on the same sample of new data.

- Return to the **Predictor** page and input the same form values as in Step 3.
- Submit the form and observe the prediction from the retrained model.

The prediction is **\$1,133,608**—an error of **\$326,392**. The retrained model isn't perfect, but it's much closer than the initial model was.

10. This model was retrained based on the availability of new data.

What other retraining strategies might you use for this model, or at least investigate to see if they're feasible?

11. Shut down the web app and Docker container.

- Close Firefox.
- In the running web app terminal, press **Ctrl+C** to stop the web app.
- Press **Ctrl+D** to stop the container.

Summary

In this lesson, you treated machine learning operations as an ongoing project requiring routine maintenance. You secured your pipeline from attack and monitored your models in production. By maintaining your operations, you can be sure that your machine learning product continues to bring value to the business over its lifetime.

What kind of user roles do you think would be most appropriate for your machine learning pipeline?

How often do you expect to retrain your models? 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.

Course Follow-Up

Congratulations! You have completed the *Certified Artificial Intelligence (AI) Practitioner (Exam AIP-210)* 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 put them in production. 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-210) 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 CertNexus® course *Certified Ethical Emerging Technologist™ (CEET): Exam CET-110*. 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 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.

A | Mapping Course Content to CertNexus® Certified Artificial Intelligence (AI) Practitioner (Exam AIP-210)

Obtaining the Certified Artificial Intelligence (AI) Practitioner certification requires candidates to pass CertNexus® Certified Artificial Intelligence (AI) Practitioner Exam AIP-210.

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-210)* 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!

B

Datasets Used in This Course

This course uses several third-party datasets to demonstrate machine learning concepts. The first academic journal article or other publication in which a dataset appeared is cited, if applicable. Additional information about how the dataset was obtained is also provided. The datasets are listed in order of appearance.

House Sales in King County, USA

Public domain. Retrieved from <https://www.kaggle.com/harlfoxem/housesalesprediction>.

Note:

- The dataset was modified so that it could be used to demonstrate various data cleaning procedures.

Large Movie Review Dataset

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011, June). Learning Word Vectors for Sentiment Analysis. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 142–150.

Notes:

- A small subset of the raw data was obtained from <https://ai.stanford.edu/~amaas/data/sentiment/>.
- A full, preprocessed version of the dataset was also obtained by loading it through the Keras library.

Clothing Dataset

Public domain. Retrieved from <https://github.com/alexeygrigorev/clothing-dataset-small>.

Note:

- A small subset of the data was obtained from this link, which is itself a subset of the larger pool of data available at <https://github.com/alexeygrigorev/clothing-dataset>.

Combined Cycle Power Plant Dataset

Tüfekci, P. (2014, September). Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, 60, 126–140. doi: 10.1016/j.ijepes.2014.02.027.

Kaya, H., Tüfekci, P., & Gürgen, S. F. (2012, March). Local and Global Learning Methods for Predicting Power of a Combined Gas & Steam Turbine. *Proceedings of the International Conference on Emerging Trends in Computer and Electronics Engineering (ICETCEE)*, 13–18.

Note:

- The dataset was obtained from <https://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant>.

California Housing

Pace, R. K., & Barry, R. (1997, May). Sparse spatial autoregressions. *Statistics & Probability Letters*, 33, 291–297. doi: 10.1016/S0167-7152(96)00140-X.

Note:

- This dataset was obtained from https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html.

Electricity Net Generation from Solar, All Sectors in Million Kilowatt-hours (SOETPUS)

Public domain. Retrieved from <https://api.eia.gov/v2/total-energy/data>.

Notes:

- Access to the data from the link requires registration for a free API key.

Wage Growth and Inflation (Table D.6)

Mehra, Y. P. (1994). Wage Growth and the Inflation Process: An Empirical Approach. In B.B. Rao (Ed.), *Cointegration for the Applied Economist*, 147–159. Palgrave Macmillan, London. doi: 10.1007/978-1-349-23529-2_5.

Note:

- This dataset was obtained from <https://github.com/selva86/datasets/blob/master/Raotbl6.csv>.

Titanic: Machine Learning from Disaster

Public domain. Retrieved from <https://www.kaggle.com/c/titanic>.

Wine Dataset

Forina, M., Leardi, R., Armanino, C., & Lanteri, S. (1990, March). PARVUS: An extendable package of programs for data exploration, classification, and correlation. *Journal of Chemometrics*, 4(2), 191–193. doi: 10.1002/cem.1180040210.

Note:

- This dataset was loaded from the scikit-learn library.

Iris Plants Dataset

Fisher, R. A. (1936, September). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2), 179–188. doi: 10.1111/j.1469-1809.1936.tb02137.x.

Note:

- This dataset was loaded from the scikit-learn library.

Occupancy Detection

Candanedo, L. M., & Feldheim, V. (2016, January). Accurate occupancy detection of an office room from light, temperature, humidity, and CO₂ measurements using statistical learning models. *Energy and Buildings*, 112, 28–39. doi: 10.1016/j.enbuild.2015.11.071.

Note:

- This dataset was obtained from <https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+>.

Fashion-MNIST

Xiao, H., Rasul, K., & Vollgraf, R. (2017, August). Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. doi: 10.48550/arXiv.1708.07747.

Note:

- This dataset was loaded from the Keras library.

Diabetes Dataset

Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004, June). Least Angle Regression. *Annals of Statistics*, 32(2), 407–451. doi: 10.48550/arXiv.math/0406456.

Note:

- This dataset was loaded from the scikit-learn library.

Airline Passengers Dataset

Box, E. P., & Jenkins, G. M. (1970). *Time Series Analysis: Forecasting and Control* (1st ed.). Holden-Day.

Note:

- This dataset was obtained from <https://github.com/jbrownlee/Datasets/blob/master/airline-passengers.csv>.

Wisconsin Breast Cancer Dataset

Street, W. N., Wolberg, W. H., & Mangasarian, O. L. (1993, July). Nuclear feature extraction for breast tumor diagnosis. *Proceedings of the IS&T/SPIE International Symposium on Electronic Imaging: Science and Technology*, 1905, 861–870. doi: 10.1117/12.148698.

Note:

- This dataset was loaded from the scikit-learn library.

Wholesale Customers Dataset

Abreu, N. (2011). *Analise do perfil do cliente Recheio e desenvolvimento de um sistema promocional* (Master's dissertation). Retrieved from <http://hdl.handle.net/10071/4097>.

Note:

- This data was obtained from <https://archive.ics.uci.edu/ml/datasets/Wholesale+customers>.

Auto MPG Dataset

Quinlan, J. R. (1993, June). Combining Instance-Based and Model-Based Learning. *Machine Learning: Proceedings of the Tenth International Conference*, 236–243. doi: 10.1016/B978-1-55860-307-3.50037-X.

Notes:

- This data was obtained from <https://archive.ics.uci.edu/ml/datasets/Auto+MPG>.
- The dataset was modified to be in a more common format.

Penguins Dataset

Gorman, K. B., Williams, T. D., & Fraser, W. R. (2014, March). Ecological Sexual Dimorphism and Environmental Variability Within a Community of Antarctic Penguins (Genus *Pygoscelis*). *PLoS ONE*, 9(3). doi: 10.1371/journal.pone.0090081.

Note:

- This dataset was obtained from <https://github.com/mwaskom/seaborn-data/blob/master/penguins.csv>.

MNIST

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998, November). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324. doi: 10.1109/5.726791.

Note:

- This dataset was loaded from the Keras library.

Mastery Builders

Mastery Builders are provided for certain lessons as additional learning resources for this course. Mastery Builders are developed for selected lessons within a course in cases when they seem most instructionally useful as well as technically feasible. In general, Mastery Builders are supplemental, optional unguided practice and may or may not be performed as part of the classroom activities. Your instructor will consider setup requirements, classroom timing, and instructional needs to determine which Mastery Builders are appropriate for you to perform, and at what point during the class. If you do not perform the Mastery Builders in class, your instructor can tell you if you can perform them independently as self-study, and if there are any special setup requirements.

Mastery Builder 4-1

Building a Linear Regression Model to Predict Diabetes Progression

Activity Time: 60 minutes

Data File

/home/student/CAIP/Linear Regression/Linear Regression - Diabetes.ipynb

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 features of the dataset:

Feature	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: There is no "correct" answer for every line of code, as there are many ways to write code to do the same thing. You can write the code however you feel most comfortable, as long as it accomplishes the tasks set out for you. If you are stuck and need help, you can "borrow" code from existing notebooks, and/or look up documentation on how to use the various Python libraries.



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 notebook.

- From Jupyter Notebook, select **CAIP/Linear Regression/Linear Regression - Diabetes.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 block 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 data frame, 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. The non-standardization version of the dataset can be found at: <https://www4.stat.ncsu.edu/~boos/var.select/diabetes.tab.txt>.

- Run the code and verify that information about the training data is shown.

4. Examine the distributions of various features.

- In the code block under **Examine the distributions of various features**, write code to plot distribution histograms for all features.
- Run the code and verify that the distributions are shown.

5. Examine descriptive statistics.

- In the code block under **Examine descriptive statistics**, write code to view descriptive statistics (mean, standard deviation, min, max, etc.) for each feature.
- Run the code and verify that the descriptive 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. 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.

8. Split the dataset.

- In the code block under **Split 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.

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 10 predictions to actual values.

- In the code block under **Compare the first 10 predictions to actual values**, write code to make predictions on the test set. Then, view the first 10 records with two columns: the actual label value for that record (disease progression), and the value that your model predicted for that record.
- Run the code and observe the predictions.

11. Calculate the error between predicted and actual values.

- 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.
- Run the code and observe the error value.

12. Plot lines of best fit for four features.

- 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.
- Run the code and examine the plots.

13. In Jupyter Notebook, open **CAIP/Linear Regression/Solutions/Linear Regression - Diabetes (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.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **Linear Regression - Diabetes** tab in Firefox.
- Repeat this process to shut down and close the **Linear Regression - Diabetes (Solution)** kernel.
- Return to **CAIP** in the file hierarchy.

Mastery Builder 5-1

Building a Univariate Forecasting Model to Predict Airline Passengers

Activity Time: 60 minutes

Data Files

/home/student/CAIP/Forecasting/Forecasting - Airline Passengers.ipynb

/home/student/CAIP/Forecasting/passenger_data/airline-passengers.csv

Scenario

An airline wants to improve its ability to anticipate demand, particularly as that demand fluctuates over a calendar year. The airline has enlisted your help in predicting how many passengers it can expect to serve in the future.

You've been given a historical dataset that records the number of passengers that use the airline every month. You'll use this dataset to develop a univariate time series model that can forecast passenger numbers two years in advance.

	<p>Note: There is no "correct" answer for every line of code, as there are many ways to write code to do the same thing. You can write the code however you feel most comfortable, as long as it accomplishes the tasks set out for you. If you are stuck and need help, you can "borrow" code from existing notebooks, and/or look up documentation on how to use the various Python libraries.</p>
	<p>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.</p>

1. Open the notebook.

- From Jupyter Notebook, select **CAIP/Forecasting/Forecasting - Airline Passengers.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 block under **Import software libraries and load the dataset**.
- Run the code and examine the results.

The dataset is loaded for you. There are 144 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 information about the training data is shown.



Note: The passengers are in the thousands.

4. Examine the distribution of the Passengers feature.

- In the code block under **Examine the distribution of the Passengers feature**, write code to plot a distribution histogram for the feature.
- Run the code and verify that the distribution is shown.

5. Examine descriptive statistics.

- In the code block under **Examine descriptive statistics**, write code to view descriptive statistics (mean, standard deviation, min, max, etc.) for the Passengers feature.
- Run the code and verify that the descriptive statistics are printed.

6. Convert Month to a period index.

- In the code block under **Convert Month to a period index**, write code to turn the Month column into a datetime index, then use `pd.to_period()` to convert the datetime index into a period index. Also ensure that the data frame only has the period index and the Passengers column.
Converting the date to a period index makes it easier for the ARIMA class in statsmodels to work with.
- Run the code and verify the formatting of the data frame.

7. Plot the change in number of airline passengers over time.

- In the code block under **Plot the change in number of airline passengers over time**, write code to plot the airline passengers data as a line plot.
- Run the code and observe the trend in the data.

8. Confirm the seasonality of the dataset.

- In the code block under **Confirm the seasonality of the dataset**, write code to seasonally decompose the passengers data, then plot the seasonal decomposition.
 - Use the model you think makes the most sense, given what you saw in the previous plot. A 'multiplicative' model's seasonality increases or decreases over time, whereas an 'additive' model's seasonality is constant.
 - Use the period you think makes the most sense, given what you saw in the previous plot (i.e., the likely seasonal periodicity of airline passengers). Remember that the data time series interval is in months.
- Verify that the seasonal decomposition plot exhibits a pattern similar to the seasonal fluctuations in the raw data.



Note: If it doesn't, try adjusting the period of the decomposition.

9. Split the dataset.

- In the code block under **Split the dataset**, write code to split the data into train and test sets using a split of 95% train and 5% test.



Caution: Make sure not to randomly shuffle the split—it must preserve the sequence of the time series.

- Run the code and observe the shape of the split datasets.

10. Train a seasonal ARIMA model.

- In the code block under **Train a seasonal ARIMA model**, write code to construct an ARIMA model, fit the training data to it, then print a summary of the model.

- If desired, experiment with multiple `order` values. If you're unsure where to start, assign 1 to `p`, `d`, and `q`.
 - Remember to incorporate a `seasonal_order` that has the relevant periodicity value you determined earlier.
- b) Run the code and observe the model summary.

11. Compare the predictions to actual values.

- a) In the code block under **Compare the predictions to actual values**, write code to generate passenger predictions for the test set, using the first month in the test set as the `start` and the last month in the test set as the `end`. Then, compare the predictions to the actual test values.
- b) Run the code and observe the predictions.

12. 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 root mean squared error (RMSE) for the model's predictions on the test set.
- b) Run the code and observe the error value.

13. Forecast the number of airline passengers over the next two years.

- a) In the code block under **Forecast the number of airline passengers over the next two years**, write code to predict the next two years' worth of airline passengers, starting with the month after the last month in the overall dataset.
- b) Run the code and examine the forecasted passenger values.

14. Visualize the forecasting trend.

- a) In the code block under **Visualize the forecasting trend**, write code to plot the historical airline passenger data as a line, and plot the forecasted passengers as another, visually distinct line.
- b) Run the code and compare the trend of the forecasted passenger values to the historical passenger values.

15. In Jupyter Notebook, open **CAIP/Forecasting/Solutions/Forecasting - Airline Passengers (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.

16. 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 **Forecasting - Airline Passengers** tab in Firefox.
- d) Repeat this process to shut down and close the **Forecasting - Airline Passengers (Solution)** kernel.
- e) Return to **CAIP** in the file hierarchy.

Mastery Builder 6-1

Creating a Logistic Regression Model to Predict Breast Cancer Recurrence

Activity Time: 60 minutes

Data File

/home/student/CAIP/Logistic Regression and k-NN/Logistic Regression - Breast Cancer.ipynb

Scenario

Breast cancer affects millions of women across the globe each year. Given the extensive data available regarding breast cancer patients, machine learning can provide a valuable tool for helping to classify which patients are likely to have breast cancer. 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. This dataset is based on images of cell nuclei in breast masses, where various measurements from the images have been recorded. Each observation is of a single image. The dataset has already been cleaned, so it shouldn't require any additional preparation to use it for a simple logistic regression model.

The following are the features of the dataset.

Feature	Description
radius	The mean distance from the center of the nucleus to various points on the perimeter.
texture	The standard deviation of grayscale values in the cell nucleus image.
perimeter	The perimeter of the nucleus.
area	The area of the nucleus.
smoothness	The local variation in radius lengths of the nucleus.
compactness	Defined as $\text{perimeter}^2 / \text{area} - 1$.
concavity	The severity of concave portions of the nucleus contour.
concave points	The number of concave portions of the nucleus contour.
symmetry	The degree to which the nucleus is symmetrical.
fractal dimension	An approximation of the "coastline" of the nucleus minus 1. Cell nuclei, like landmass coastlines, have a fractal dimension that does not have a well-defined length. In fractal geometry, there are methods for approximating the length of a fractal dimension.
target	Whether the mass is malignant (0) or benign (1). The former indicates a positive cancer diagnosis.

Each of these measurements actually has three different aspects: mean, standard error, and "worst." For example, `mean radius` is the mean of `radius` values for all nuclei in the image; `radius error` is the standard error of all `radius` values in the image; and `worst radius` is the mean of the three largest values in `radius`.



Note: There is no "correct" answer for every line of code, as there are many ways to write code to do the same thing. You can write the code however you feel most comfortable, as long as it accomplishes the tasks set out for you. If you are stuck and need help, you can "borrow" code from existing notebooks, and/or look up documentation on how to use the various Python libraries.



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 notebook.

- From Jupyter Notebook, select **CAIP/Logistic Regression/Logistic Regression - Breast Cancer.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 block under **Import software libraries and load the dataset**.
- Run the code and examine the results.

The dataset is loaded for you. There are 569 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 information about the training data is shown.

4. Examine descriptive statistics.

- In the code block under **Examine descriptive statistics**, write code to view descriptive statistics (mean, standard deviation, min, max, etc.) for each feature.
- Run the code and verify that the descriptive statistics are printed.

5. Examine the distributions of various features.

- In the code block under **Examine the distributions of various features**, write code to plot distribution histograms for all features.
- Run the code and verify that the distributions are shown.

6. Split the data into training and validation sets and labels.

- In the code block under **Split the data into training and validation sets and labels**, 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.

7. Create a logistic regression model.

- a) In the code block under **Create a logistic regression model**, write code to construct a logistic regression class object, then fit the training data to that object.

When constructing the logistic regression class object, set 10,000 as the maximum number of iterations.

- b) Run the code.

8. Compare the first 10 predictions to actual values.

- a) In the code block under **Compare the first 10 predictions to actual values**, write code to make predictions on the test set. Then, view the first 10 records with two columns: the actual classification for that record (breast cancer diagnosis), and the classification that your model predicted for that record.
- b) Run the code and observe the predictions.

9. Evaluate the performance of the model on the test set.

- a) In the code block under **Evaluate the performance of the model on the test set**, write code to print the accuracy, precision, recall, and F_1 score for the model's predictions compared to the actual test set labels.
- Consider creating a function to do this. You'll use these metrics again in this notebook.
- b) Run the code and observe the scores on the classification metrics.

10. Optimize the logistic regression model using grid search and cross-validation.

- a) In the code block under **Optimize the logistic regression model using grid search and cross-validation**, write code to use the provided grid in a grid search. Also write code to fit the best model on the training data, and then print the hyperparameters from the best model.

The grid search should:

- Use a `LogisticRegression()` class object.
- Attempt to optimize recall.
- Use stratified 5-fold cross-validation.

- b) Run the code and observe the hyperparameters of the best model identified in the search.

11. Determine if the grid search model improved the scores on the test set.

- a) In the code block under **Determine if the grid search model improved the scores on the test set**, write code to obtain predictions on the test using the best grid search model, then print the scores using the same four metrics as before.
- b) Run the code and observe the scores on the optimized model, determining whether or not the scores (recall in particular) improved meaningfully.

12. In Jupyter Notebook, open CAIP/Logistic Regression/Solutions/ Logistic Regression - Breast Cancer (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 **Logistic Regression - Breast Cancer** tab in Firefox.
 - d) Repeat this process to shut down and close the **Logistic Regression - Breast Cancer (Solution)** kernel.
 - e) Return to **CAIP** in the file hierarchy.
-

Mastery Builder 7-1

Building a Clustering Model for Customer Segmentation

Activity Time: 60 minutes

Data Files

/home/student/CAIP/Clustering/Clustering - Wholesale Customers.ipynb

/home/student/CAIP/Clustering/wholesale_customers_data/
wholesale_customers_data.csv

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 features of the dataset.

Feature	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: There is no "correct" answer for every line of code, as there are many ways to write code to do the same thing. You can write the code however you feel most comfortable, as long as it accomplishes the tasks set out for you. If you are stuck and need help, you can "borrow" code from existing notebooks, and/or look up documentation on how to use the various Python libraries.



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 notebook.

- From Jupyter Notebook, select **CAIP/Clustering/Clustering - Wholesale Customers.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 block 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 information about the training data is shown.

4. Examine the distributions of various features.

- In the code block under **Examine the distributions of various features**, write code to plot distribution histograms for all features.
- Run the code and verify that the distributions are shown.

5. Examine descriptive statistics.

- In the code block under **Examine descriptive statistics**, write code to view descriptive statistics (mean, standard deviation, min, max, etc.) for each feature.
- Run the code and verify that the descriptive statistics are printed.

6. Use a *k*-means model to cluster every row in the dataset.

- In the code block under **Use a *k*-means model to cluster 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, generate the cluster assignments using this truncated dataset.
- Run the code.

7. Attach cluster assignments to the original dataset.

- In the code block under **Attach cluster assignments to the original dataset**, write code to append the cluster assignments to the original dataset, then preview the first five rows.
- Run the code and observe the cluster assignments.

8. Show clusters of customers based on fresh products and milk products sales.

- In the code block under **Show clusters of customers based on fresh products and milk products sales**, 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.



Note: You can use Yellowbrick's `KELbowVisualizer` module to make plotting the elbow easier.

- b) 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.

- a) 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.
- b) Write code to plot the silhouette for each model.



Note: You can use Yellowbrick's `SilhouetteVisualizer` module to make plotting the silhouettes easier.

- c) Write code to print the number of clusters that received the highest silhouette score.
- d) Run the code and examine the suggested number of clusters based on the silhouette analysis.

11. Generate and preview cluster assignments using the full dataset.

- a) In the code block under **Generate and preview cluster assignments 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 generate the cluster assignments, then append the assignments 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/Clustering/Solutions/Clustering - Wholesale Customers (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 - Wholesale Customers** tab in Firefox.
- d) Repeat this process to shut down and close the **Clustering - Wholesale Customers (Solution)** kernel.
- e) Return to **CAIP** in the file hierarchy.

Mastery Builder 8-1

Building a Random Forest Model to Estimate Car Fuel Economy

Activity Time: 60 minutes

Data Files

/home/student/CAIP/Decision Trees and Random Forests/Random Forest - Auto MPG.ipynb

/home/student/CAIP/Decision Trees and Random Forests/car_data/auto-mpg.csv

Scenario

You're consulting with a company in the automotive industry that primarily manufacturers cars. The company wants to be able to estimate the fuel economy of new car models currently undergoing research and development. Their goal is to hone in on the factors that contribute the most to fuel economy so that they can use that information to design more fuel-efficient vehicles.

You've been given a historical dataset of car specifications, which includes a label that describes each car's fuel economy in terms of miles per gallon. Because this is a numeric label, you'll build a regression model, and because the organization is interested in discovering the key factors behind fuel economy, you'll use a random forest technique.

The following are the features of the dataset:

Feature	Description
car name	The make and model of the car.
cylinders	The number of cylinders in the car's engine.
displacement	The engine displacement, a measurement of the volume that is displaced by the pistons as they move in the cylinders, in cubic inches.
horsepower	The horsepower of the car's engine, or how much physical work it is capable of performing.
weight	The weight of the car, in pounds.
acceleration	The time it takes, in seconds, for the car to go from 0 miles per hour to 60 miles per hour.
model year	The year (in the 1900s) the model of car was released.
origin	A label encoding that indicates where the car was manufactured: 1 for America, 2 for Europe, and 3 for Japan.
mpg	The car's fuel economy, measured as miles per gallon.



Note: There is no "correct" answer for every line of code, as there are many ways to write code to do the same thing. You can write the code however you feel most comfortable, as long as it accomplishes the tasks set out for you. If you are stuck and need help, you can "borrow" code from existing notebooks, and/or look up documentation on how to use the various Python libraries.



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 notebook.

- From Jupyter Notebook, select **CAIP/Decision Trees and Random Forests/Random Forest - Auto MPG.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 block under **Import software libraries and load the dataset**.
- Run the code and examine the results.

The dataset is loaded for you. There are 398 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 information about the training data is shown.

4. Drop rows with missing values.

- In the code block under **Drop rows with missing values**, write code to drop all rows that contain missing values. Also write code to verify there are no more missing values.
- Run the code and verify that the rows with missing values have been removed.

5. Examine the distributions of various features.

- In the code block under **Examine the distributions of various features**, write code to plot distribution histograms for all features.
- Run the code and verify that the distributions are shown.

6. Examine descriptive statistics.

- In the code block under **Examine descriptive statistics**, write code to view descriptive statistics (mean, standard deviation, min, max, etc.) for each feature.
- Run the code and verify that the descriptive statistics are printed.

7. Look for columns that correlate with mpg (miles per gallon).

- In the code block under **Look for columns that correlate with mpg (miles per gallon)**, write code to view the correlation values for each feature compared to the label.
- Run the code and examine the feature correlations.

8. One-hot encode the origin feature.

- In the code block under **One-hot encode the origin feature**, write code to one-hot encode the feature that indicates where the car was manufactured. Then, print the data frame to observe the new encoded columns.



Note: Although the feature is already label encoded, this implies a rank, even though the feature is purely categorical.

- Run the code and verify the new one-hot encoded columns were added, and that the original column was removed.

9. Split the dataset.

- In the code block under **Split 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.

10. Create a random forest model.

- In the code block under **Create a random forest model**, write code to construct a random forest regression object, then fit the training data to that object.
Make sure the random forest builds at least 100 trees and will obtain the out-of-bag (OOB) score.
- Run the code.

11. Compare the first 10 predictions to actual values.

- In the code block under **Compare the first 10 predictions to actual values**, write code to make predictions on the test set. Then, view the first 10 records with two columns: the actual label value for that record (miles per gallon), and the value that your model predicted for that record.
- Run the code and observe the predictions.

12. Calculate the error between predicted and actual values.

- In the code block under **Calculate the error between predicted and actual values**, write code to print the root mean squared error (MSE) for the model's predictions on the test set, as well as code to print the OOB error.
- Run the code and observe the error values.

13. Identify feature importances.

- In the code block under **Identify feature importances**, write code to obtain the forest's feature importances and then plot them on a bar chart in descending order of importance.
- Run the code and observe which features are the most important to the model.

14. Visualize one of the trees in the forest.

- In the code block under **Visualize one of the trees in the forest**, write code to plot any one of the decision trees in the forest.
Make sure to limit the depth of the plotted tree so that it's easier to interpret.
- Run the code and examine the tree's decision-making logic.

15. In Jupyter Notebook, open **CAIP/Decision Trees and Random Forests/Solutions/Random Forest - Auto MPG (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.

16. Shut down the Jupyter Notebook kernels.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **Random Forest - Auto MPG** tab in Firefox.
- Repeat this process to shut down and close the **Random Forest - Auto MPG (Solution)** kernel.

e) Return to **CAIP** in the file hierarchy.

Mastery Builder 9-1

Creating an SVM Model to Classify Penguin Species

Activity Time: 60 minutes

Data Files

/home/student/CAIP/SVMs/SVMs - Penguins.ipynb
 /home/student/CAIP/SVMs/penguin_data/penguins.csv

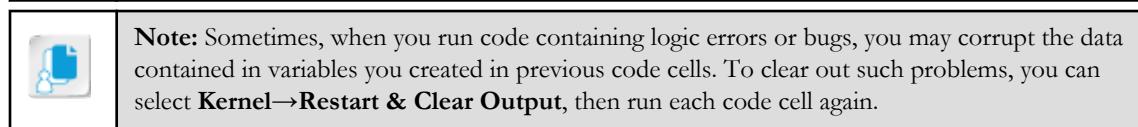
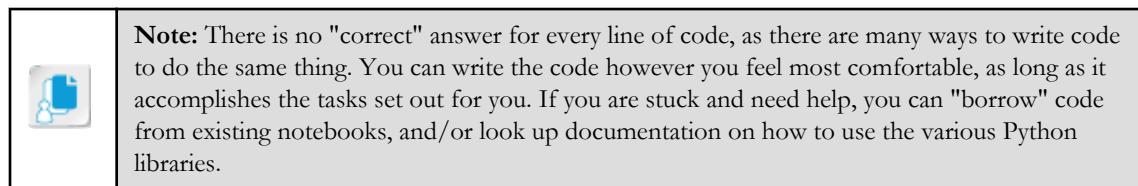
Scenario

You work for a zoo that's setting up a new penguin exhibit. The zoologists want to be able to install educational signage around the exhibit that provides zoo goers with interesting facts about the different penguins, including the names they've been given and their species. However, it can be challenging for the zoologists to identify a penguin's species, so they've asked for your help.

The zoologists have provided you with a dataset of penguins that have been housed at the zoo since its inception. The dataset lists physical measurements of each penguin, as well as the island it came from and its sex. Each penguin's species is also labeled. You'll create a model using SVMs to help the zoologists classify new penguins based on their characteristics.

The following are the features of the dataset.

Feature	Description
species	The species of the penguin—Adélie, gentoo, or chinstrap.
island	The island the penguin is from—the Biscoe Islands, Dream Island, or Torgersen Island. All three are off the coast of the Antarctic Peninsula.
bill_length_mm	The length of the penguin's bill, in millimeters.
bill_depth_mm	The depth of the penguin's bill, in millimeters.
flipper_length_mm	The length of the penguin's flippers, in millimeters.
body_mass_g	The penguin's weight, in grams.
sex	The sex of the penguin.



1. Open the notebook.
 - a) From Jupyter Notebook, select **CAIP/SVMs/SVMs - Penguins.ipynb** to open it.



Note: Be careful *not* to open the solution file.

- b) 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 block under **Import software libraries and load the dataset**.
- Run the code and examine the results.

The dataset is loaded for you. There are 344 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 information about the training data is shown.

4. Drop rows with missing values.

- In the code block under **Drop rows with missing values**, write code to drop all rows that contain missing values. Also write code to verify there are no more missing values.
- Run the code and verify that the rows with missing values have been removed.

5. Examine the distributions of various features.

- In the code block under **Examine the distributions of various features**, write code to plot distribution histograms for all numeric features.
- Run the code and verify that the distributions are shown.

6. Examine descriptive statistics.

- In the code block under **Examine descriptive statistics**, write code to view descriptive statistics (mean, standard deviation, min, max, etc.) for each numeric feature.
- Run the code and verify that the descriptive statistics are printed.

7. Standardize the numeric features.

- In the code block under **Standardize the numeric features**, write code to apply standardization to the numeric features so that they have a mean value of 0 and a standard deviation of 1.
- Run the code and verify that the numeric features have been standardized.

8. Examine the value counts of the label and the categorical features.

- In the code block under **Examine the value counts of the label and the categorical features**, write code to plot bar charts showing the counts for each category of island, sex, and species.
- Run the code and verify the proportion of both categorical features and the label.

9. One-hot encode the island and sex features.

- In the code block under **One-hot encode the island and sex features**, write code to one-hot encode the features that indicate where the penguin is from and whether it's male or female. Then, print the data frame to observe the new encoded columns.



Note: Retain category names for the new columns to make them easier to interpret.

- b) Run the code and verify the new one-hot encoded columns were added, and that the original columns were removed.

10. Split the dataset.

- a) In the code block under **Split the dataset**, write code to split the data into train and test sets, as well as split the label from the features.
- b) Run the code and observe the shape of the split datasets.

11. Create an SVM model.

- a) In the code block under **Create an SVM model**, write code to construct an SVM classifier object, then fit the training data to that object.
Make sure the SVM classifier uses a linear kernel and has a regularization penalty of 1,000.
- b) Run the code.

12. Compare the first 10 predictions to actual values.

- a) In the code block under **Compare the first 10 predictions to actual values**, write code to make predictions on the test set. Then, view the first 10 records with two columns: the actual classification for that penguin (its species), and the classification that your model predicted for that penguin.
- b) Run the code and observe the predictions.

13. Evaluate the performance of the model on the test set.

- a) In the code block under **Evaluate the performance of the model on the test set**, write code to print the accuracy score for the model's predictions compared to the actual test set labels.
- b) Run the code and observe the accuracy score.

14. Change the kernel and soften the margins of the model.

- a) In the code block under **Change the kernel and soften the margins of the model**, write code to create another SVM classifier, this time using a radial basis function (RBF) kernel and a regularization penalty of 1.
- b) Run the code.

15. Evaluate the performance of the tuned model.

- a) In the code block under **Evaluate the performance of the tuned model**, write code to make predictions on the test set using the tuned model, then print the accuracy score for the tuned model's predictions compared to the actual test set labels.
- b) Run the code and verify whether or not the accuracy of the model improved.

16. In Jupyter Notebook, open **CAIP/SVMs/Solutions/SVMs - Penguins (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.

17. 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 **SVMs - Penguins** tab in Firefox.
- d) Repeat this process to shut down and close the **SVMs - Penguins (Solution)** kernel.
- e) Return to **CAIP** in the file hierarchy.

Mastery Builder 10-1

Building a CNN to Classify Handwritten Characters

Activity Time: 60 minutes

Data File

/home/student/CAIP/Neural Networks/Neural Networks - Handwritten Characters.ipynb

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: There is no "correct" answer for every line of code, as there are many ways to write code to do the same thing. You can write the code however you feel most comfortable, as long as it accomplishes the tasks set out for you. If you are stuck and need help, you can "borrow" code from existing notebooks, and/or look up documentation on how to use the various Python libraries.



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 notebook.

- From Jupyter Notebook, select **CAIP/Neural Networks/Neural Networks - Handwritten Characters.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 block 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.



Note: You can ignore the hardware-based TensorFlow warnings in this notebook.

3. Get acquainted with the dataset.

- In the code block under **Get acquainted with the dataset**, write statements to show the 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 20 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 the data 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 current training set into training and validation subsets 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 create the model and add model layers. 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.

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 one epoch using the training data and labels, as well as the validation data and labels.

- b) Run the code cell and wait for the model to be trained.

11. Evaluate the model on the test data.

- a) 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.
- b) Run the code cell and observe the loss and accuracy values.

12. Make predictions on the test data.

- a) In the code block under **Make predictions on the test data**, write statements to make predictions on the test data, showing the first 20 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 20 examples.

- a) In the code block under **Visualize the predictions for 20 examples**, write statements to show the first 20 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/Neural Networks/Solutions/Neural Networks - Handwritten Characters (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 **Neural Networks - Handwritten Characters** tab in Firefox.
 - d) Repeat this process to shut down and close the **Neural Networks - Handwritten Characters (Solution)** kernel.
 - e) Return to **CAIP** in the file hierarchy.
-

Solutions

ACTIVITY 1-1: Identifying AI and ML Solutions for Business Problems

1. How might AI/ML be used to solve this problem?

A: There are several potential solutions, including building automated systems that can: identify the interests of specific customers or segments of customers to be able to better market the clothing products to their unique interests; make useful recommendations to customers; determine when it's best to send targeted deals and other promotions to customers; improve the overall shopping experience and customize it for each customer; and so on.

2. What stakeholders might be involved in the decision to use AI/ML to solve this customer retention problem?

A: Answers may vary, but two obvious stakeholders are the practitioners who will build and implement the AI/ML system and the project managers who will oversee the effort. Other stakeholders might include the online customers themselves, who can provide valuable insight into why they may not return to the store for another purchase, or what would entice them to return. Likewise, the company may want to consult with other members of the industry who have designed or worked with similar solutions. There may also be stakeholders who sponsor the project by providing finances and other resources.

3. How might AI/ML be used to solve this problem?

A: There are several potential solutions, including building automated systems that can: identify network traffic that does not fit any known patterns, but is still malicious; assess user behavior in a computing environment and predict if that behavior is likely to be a prelude to an attack; adjust existing access policies to account for changes in the threat landscape; evaluate organizational data at a high level to determine if there is a risk of private information leakage; and so on.

4. What kind of ethical risks might there be in the power plant implementing AI/ML?

A: Answers may vary, but a powerful system that makes decisions for the organization's security also has the potential to make bad decisions, making the plant even more susceptible to an attack. If something does go wrong, there may be a lack of accountability, since the AI/ML system made the decision and not a human. There are also potential concerns regarding user privacy, especially if the power plant maintains personal data about the customers who receive the power it generates (e.g., where their homes are located).

ACTIVITY 1–2: Formulating a Machine Learning Problem

1. How might you formulate this statement as a problem to be solved by AI/ML?

A: You can start by framing the problem in terms of the task the AI/ML system will accomplish, the experience needed to accomplish the task (i.e., the data), and the performance expected from the system. You can then move on to identifying the rationale for solving the problem with AI/ML and the potential benefits. Documenting a list of assumptions about the project, as well as other, similar problems that have been solved in the past, is also worthwhile. Lastly, you can use what information you've gathered up to this point to make an informed decision as to whether or not AI/ML is appropriate for the problem at hand.

2. What type of outcome—regression, classification, or clustering—would you look for in this example?

A: Classification algorithms can be used to classify whether parts are acceptable or defective. Clustering algorithms might also be effective if the input data is not already classified.

3. What type of outcome would you look 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.

4. Why is this situation acceptable for a machine learning project?

A: Because machine learning is based on probability, the huge amount of data makes the problems in the data less of a problem. With an adequately sized dataset, there are ways to acceptably deal with missing data.

ACTIVITY 1–3: Selecting Approaches to Machine Learning

1. Which machine learning approach(es) and mode(s) are best suited to solving this problem, and why?

A: Answers may vary, but the company should probably look into robotics—specifically, robotic arms that can manipulate and weld the parts as they come down the assembly line. The robots will likely be faster than any human worker, and their presence will mitigate the safety risks. Reinforcement learning is a common mode to use in the fields of robotics and industrial automation.

2. Which machine learning approach(es) and mode(s) are best suited to solving this problem, and why?

A: Answers may vary, but natural language processing (NLP) is a good starting point. Specifically, the company might want to consider implementing an automated phone system that can recognize human speech and parse its meaning. The system can therefore summarize the content of the call to a customer service team member, saving them valuable time, or the system itself might be able to determine solutions for the customer. For customers who prefer text-based communication, the company might consider employing an automated chatbot that can essentially do the same thing: understand what a customer is looking for, and provide them with solutions. Supervised and semi-supervised learning are both applicable to this scenario, depending on how much of the language data the organization has access to is labeled.

3. Which machine learning approach(es) and mode(s) are best suited to solving this problem and why?

A: Answers may vary, but the company is almost certainly looking for some type of system that can make predictions. In particular, the system should be able to forecast future values based on changes in past values over time. This will take the guesswork out of determining the performance of each product category's sales from month to month. Supervised learning is the most applicable mode in this situation, as the organization has access to past values for units sold and revenue.

ACTIVITY 2–2: Exploring the Dataset

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

A: Some attributes might seem important from a common-sense 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 2–6: Working with Image Data

14. How else might you "augment" the data before you use as it as input to a neural network?

A: In the absence of more source images to add as input, you could apply different transformation to the existing source images and use the results as additional data to feed into a neural network. That way, the network can learn from more data that is subtly different, even if multiple preprocessed images are derived from the same source image.

ACTIVITY 3–1: Training a Machine Learning Model

7. 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 4–2: Building a Regularized Linear Regression Model

11. 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 didn't produce an overall 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 4–3: Building an Iterative Linear Regression Model

3. 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.

6. 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.

8. 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 5–1: Building a Univariate Time Series Model

15. How might you try to improve the skill of this forecasting model?

A: There's not much more in the way of data preparation that can be done, and collecting more data is not feasible. However, tuning the AR, I, and MA hyperparameters is a good idea. There may be better values than the simple (1, 1, 1) chosen in the activity. You can even automate the process of selecting good hyperparameters by training an ARIMA model on multiple combinations of the hyperparameters, and seeing which combination results in the lowest error.

ACTIVITY 5–2: Building a Multivariate Time Series Model

9. What is the difference between an exogenous variable and an endogenous variable, and why is this important?

A: An exogenous variable is not explained by other variables in the model. An endogenous variable *is*. VAR models might require you to specify which is which, as this impacts how the model calculates relationships between the inputs.

ACTIVITY 6–1: Training a Binary Classification Model Using Logistic Regression

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

7. What does each quadrant indicate in terms of predicting survivors of the *Titanic*?

A: The top-left quadrant indicates that there were 133 instances where the model predicted a passenger would perish, and was correct in its prediction. The top-right quadrant indicates that there were 14 instances where the model predicted a passenger would survive, but was incorrect—those passengers died. The bottom-left quadrant indicates that there were 23 instances where the model predicted a passenger would perish, but was incorrect—those passengers survived. The bottom-right quadrant indicates that there were 53 instances where the model predicted a passenger would survive, and was correct in its prediction.

9. 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.

10. 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.

12. 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 three fewer true negatives; three more false positives; six fewer false negatives; and six 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

9. 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 southwest; cluster 1 representing the northwest; cluster 2 representing the northeast; and cluster 3 representing the southeast.

11. How did the clusters form with regard to price per square foot?

A: Cluster 3 includes the lowest-priced homes; cluster 0 includes the second lowest-priced homes; cluster 2 includes the second highest-priced homes; and cluster 1 includes the highest-priced homes. Other than the heatmap, this is also exhibited in the fact that the size of each data point corresponds to its total price.

16. 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.

17. 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 7-2: Building a Hierarchical Clustering Model

12. 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 fewer 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

- 15. 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 precision. If you're focused on recall, then you may prefer the logistic regression model. F_1 and average precision show moderate differences in favor of the logistic regression model. When it comes to accuracy and ROC AUC, the results are pretty much the same. Likewise, you may be unsatisfied in the sense that you'd like to do more optimization to potentially get even better results from either model. After all, a grid or randomized search may find better hyperparameters than the ones you used. 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

- 1. 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.

- 4. 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 ticket fare at the root decision node. Then, on the left, it splits based on whether the lowest-fare passengers are male or female. At the same time, on the right, it splits passengers that paid more than the lowest fare based on whether or not they have "Mr." as an honorific. The second tree (index 1) starts by splitting based on whether or not the passengers are male or female. Then, on the left, it splits based on whether male passengers are less than the oldest age bin (bin 4, or 81–100 years old). At the same time, on the right, it splits female passengers into whether or not they embarked from Cherbourg.

- 6. 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.

13. 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_code` since it had the least importance of the features that were kept. This might be successful if `Fare_code` 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, `Title_Spec`, since it had the highest importance of the two features that were dropped. This might improve the model if including `Title_Spec` 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

6. 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.

12. 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

16. How does backpropagation generate the weights between the neurons of different layers in an MLP neural network?

A: An estimation is made for an example, and then the error between the estimation 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.

17.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

13.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 the last two incorrect cases, the predicted class of clothing seemed to be visually similar to the actual class. For example, the image in row 4, column 3 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. However, the first incorrect prediction (row 3, column 3) is pretty far off. The model classified this image as a bag, but it is very clearly some kind of shoe (a sneaker, to be precise).

14.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 10-3: Building an RNN

8. 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.

9. 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

16. 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, "Hollywood had a long *love* affair ..." The reviewer then goes on to speak somewhat positively about other movies in the same genre, particularly that some were "memorable." After that, the reviewer states that the movie under review was nominated for its "*imaginative* special effects." The reviewer also praises one "*outstanding* positive feature" of the movie, namely its "*beautiful* color and clarity." The network may have picked up on these words and phrases without understanding how context may have changed their meaning.

17. 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: Deploying a Machine Learning Model

5. What infrastructure concerns might you have regarding deployment, and how might you address them?

A: Answers will vary. The way it's currently set up, the prediction script will need to be run over and over to get a high volume of outputs from the model. This could negatively impact the response rate of the model as it struggles to keep up with the requests. A deployment endpoint could help alleviate this somewhat by installing a layer of abstraction between the consumer of the model and the model itself. The requests will therefore be easier to manage in a way that is suitable for the environment. And, the environment itself could be reinforced by adding more containers that each have access to the model or some version of it. This will help make the model more resilient to performance issues.

ACTIVITY 11-2: Automating the Machine Learning Process with MLOps

7. Compared to how Jupyter Notebook has been used throughout the course, how is this training script more appropriate for automating production tasks?

A: Answers may vary. The Jupyter Notebooks were written to apply machine learning tasks in an ad hoc approach. They followed a rigid series of steps meant more for demonstration or proof of concept than for actual deployment. On the other hand, the script in this activity is more modular and meant to run each step of the process without direct human intervention. This makes it more suitable for automation, as the process is repeatable and extensible.

8. How might you make these automated tasks more robust and resilient to failure?

A: Answers may vary. The code itself could be augmented with more conditional logic and/or exception handling so that it's better at addressing undesirable or unforeseen effects when run again on new data. Another option would be to further modularize each step of the pipeline by splitting functionality across several containers. That way, each component of the pipeline can be isolated and managed separately.

ACTIVITY 11–3: Integrating a Model into a Machine Learning System

8. Aside from real estate agents getting house price predictions for existing houses, how else might they use the prediction form?

A: Answers may vary, but one alternative use for the prediction form is to help a client assess the value of their house after making some change to it. For example, a seller might want to add another bathroom to their house in hopes of increasing the house's value. The agent can input the house's details, but with the speculative number of bathrooms instead of the current number, and see if the predicted increase in house price is worth the cost of the home improvement project.

ACTIVITY 12–1: Securing a Machine Learning Pipeline

4. What is the advantage of using an RBAC approach instead of just giving individual users access to the pipeline elements they need?

A: By assigning users to roles, you can more easily manage who should have access to what based on the nature of the element being accessed and the nature of the job that the user is meant to perform. Assigning permission to individual users is not only tedious, but it creates unwanted complexities.

10. How might you ensure that the web app handles model integrity violations dynamically?

A: Answers will vary. You could have the web app run a hash check on the currently saved model every so often. Or, you might have some functionality trigger the check. For example, when a user submits a form for prediction, the app might run the check first. In any case, you'll need to determine what to do if there is a violation. Taking down the entire web app is a possibility, although this obviously impacts availability and makes for a poor user experience. You could instead give the user a warning and say that the service is temporarily unavailable, or perhaps rely on a known older model for the prediction.

ACTIVITY 12–2: Maintaining a Model in Production

2. Why is it important to maintain and monitor logs such as these?

A: Answers may vary, but being able to monitor key events in the pipeline's operation will make it easier to spot issues that need correcting, such as potentially malicious access attempts, prediction requests that aren't handled properly, data preparation tasks that aren't being applied, models that aren't providing the expected outputs, and so on.

10.What other retraining strategies might you use for this model, or at least investigate to see if they're feasible?

A: Answers may vary. It might be feasible to collect the housing data from the source databases on a schedule. For example, every six months you might collect the previous six months' house sales. Then, you could evaluate the model's scores on this data and observe how the model's performance drifts over time. You might decide to retrain if the performance goes below a certain threshold. Alternatively, you might just assume that drift is inevitable in a field like real estate, and initiate the retraining process automatically, regardless of performance metrics. Or, rather than retrain on a schedule, you might trigger retraining as a response to major changes in external market factors or economic conditions (e.g., the Federal Reserve changing interest rates). These strategies assume you can reliably and consistently collect new data, however. You may not be so fortunate—in which case, you'd need to wait for new data to become available, assuming it ever does.

Glossary

access control

The process of allowing only authorized users or systems to observe, modify, or otherwise take possession of the resources of a computer system or physical property.

accuracy

A measure of how frequently each classification is correctly deemed positive or negative.

activation function

A function that computes the output of an artificial neuron to solve non-linear tasks.

adversarial machine learning

The defense against attacks on machine learning models.

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.

amplitude

The magnitude of an audio signal.

ANN

(artificial neural network) A machine approximation of biological neural networks. Used in deep learning.

ARIMA

(autoregressive integrated moving average)
A common algorithm for performing univariate time series forecasting.

aspect ratio

The measurement of an image's width relative to the measurement of its height, expressed as a reduced fraction.

attribute

See *feature*.

AUC

(area under curve) The total space that is under a learning model's ROC curve.

audio sampling

The process of converting a continuous audio signal into a discrete audio signal by recording the value of the continuous signal at certain points in time, at a certain rate.

augmentation

The process of creating multiple transformations of data (i.e., perturbing an image in different ways) to increase the amount of data input to a model.

authentication

The act of verifying the identity of someone or something.

authorization

The act of assigning permissions and rights by which an authenticated user is able to interact with a protected system.

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.

BGD

(batch gradient descent) An approach to gradient descent that uses the entire training dataset to calculate the step-wise gradients.

bias

In machine learning, a type of error that occurs when a model's estimations are different than the ground truth.

black box

The property by which a machine learning model's decisions are difficult to understand or explain.

Box-Cox

A transformation function that obtains a normal distribution of data using log and power transformations.

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

(changing anything changes everything) The principle by which making any change to the data or to the hyperparameters of a model can considerably alter the model itself.

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.

CD

(continuous delivery) The process by which software components are automatically implemented in an environment over short, repeated cycles.

centroid

In a clustering model, the mean (average) of all of the data points that the cluster contains, across all features.

CI

(continuous integration) The process by which software engineers write, test, and merge code to a centralized repository on a frequent basis.

classification

A type of machine learning task in which a data example is placed into one or more categories.

clustering

A type of machine learning task that places data examples into groups based on their similarities.

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.

co-training

An approach to semi-supervised learning in which two models train on two different views of data, then train each other based on pseudo-labels generated from unlabeled data.

coefficient of determination

A statistical measure that indicates how much of a dependent variable's variance is explainable by the independent variables in a statistical model.

collaborative filtering

An approach in which a system makes recommendations based on users who have similar interests.

collinearity

See [multicollinearity](#).

computer vision

A set of techniques by which computers process images and other visual data.

concept drift

See [model drift](#).

confusion matrix

A method of visualizing the truth results of a classification problem.

content filtering

An approach in which a system makes recommendations based on the profile of a user compared to the content being considered for recommendation.

continuous variable

A quantitative 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 estimated 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 partitioning data so that a model is able to generalize to new test data.

curse of dimensionality

The phenomenon by which adding more dimensionality to a dataset reduces a model's ability to learn patterns from the data.

DAI

(distributed artificial intelligence) An advanced deployment approach in which learning "agents" run parallel operations from geographically dispersed locations.

data binning

The process of discretizing a continuous variable by placing its values within specific intervals.

data cleaning

The process of locating and addressing errors and inconsistencies in data.

data encoding

The process of converting data of a certain type into a coded value of a different type.

data munging

See [data wrangling](#).

data preparation

The process of altering data so that it more effectively supports tasks like data analysis and modeling.

data preprocessing

The task of applying various transformation and encoding techniques to data so that it can be interpreted and analyzed by a machine learning algorithm.

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.

deduplication

The process of identifying and removing duplicate entries from a dataset.

deep learning

A type of machine learning that uses artificial neural networks with multiple layers to make complex decisions.

dendrogram

A diagram that represents a tree-like hierarchy, commonly used to visualize hierarchical clustering tasks.

dependent variable

In an experiment, the variable that is being studied and that is affected by one or more independent variables.

deployment

The process by which a machine learning model is put into a production environment, where it can be fed input and produce output.

diagnosis

The task of determining the cause of a problem in some environment.

dimension

In machine learning, the number of features in a dataset or a model that is trained on that dataset.

dimensionality reduction

A task that minimizes irrelevant or unnecessary elements from a dataset in order to improve the machine learning process.

discrete variable

A quantitative 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.

discriminator

One half of a generative adversarial network (GAN) that estimates a label given a set of features. It tries to determine whether the image created by the generator is real or fake.

Docker

An open-source platform for building and maintaining virtual containers.

DOE

(design of experiments) An approach to identifying, analyzing, and controlling variables used in an experiment. Also referred to as experimental design or DOX.

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

The process of condensing a language vocabulary into vectors of relatively small dimensions.

endogenous

The property by which a variable is explained by other variables in a model.

endpoint

An intermediary that consumers and systems interface with for the purpose of sending and receiving input and output over a network.

ensemble learning

An application of machine learning in which the estimations of multiple models are considered in combination.

entropy

In the context of decision trees, the property that multiplies each feature's class probability by a binary logarithm and multiplies that product by -1 .

epoch

In a neural network, a single training pass (both forward and backward) through the entire input dataset.

error

Incorrect or missing values in data.

ETL

(extract, transform, and load) The process of combining data from multiple sources, preparing the data, and loading the resulting data into a destination format.

evaluation metric

A method of assessing the skill, performance, and characteristics of a model based on a specific measurement.

exogenous

The property by which a variable is not explained by other variables in a model.

experimental design

See [DOE](#).

F₁ score

The weighted average (harmonic mean) of both precision and recall.

feature

In machine learning, a measurable property of an example in a training dataset.

feature engineering

The technique of generating and extracting features from data in order to improve the ability for a machine learning model to make estimations.

feature extraction

A type of dimensionality reduction in which new features are derived 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 scaling

The task of transforming the values of multiple features so that those values are all on a similar scale.

feature selection

A type of dimensionality reduction in which a subset of the original features is selected.

filter

The portion of the receptive field that a convolutional layer neuron uses to scan the image at prior layers.

fitting

See [training](#).

FNN

(feedforward neural network) A type of artificial neural network (ANN) in which information flows to and from artificial neurons in a single direction.

forecasting

A task that involves making predictions about future events based on the analysis of relevant past events.

Fourier transformation

The process of decomposing an audio signal into its constituent frequencies.

FPR

(false positive rate) A measure of how frequently the learning model incorrectly classified positive values.

frame rate

The number of frames (images) that are displayed every second in a video.

frequency

In audio processing, the number of sound waves repeated per second.

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 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 estimates 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.

Goodhart's law

A principle that states: "When a measure becomes a target, it ceases to be a good measure." Used as a reminder not to rely too heavily on one or a small number of metrics when evaluating machine learning model performance.

GPU

(graphics processing unit) The computer chip typically used as the core component in a graphics card.

gradient boosting

An iterative ensemble learning method that builds multiple decision trees in succession, where each tree attempts to reduce the errors of the previous tree.

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.

hashing

A process or function that transforms plaintext input into an indecipherable fixed-length output and ensures that this process cannot be feasibly reversed.

HDC

(hierarchical divisive 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.

hidden layer

A layer of neurons in a neural network that is not directly exposed to the input and requires additional analysis.

holdout

A data sampling 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.

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.

image resolution

The total number of pixels in an image, usually expressed as the number of pixels in width by the number of pixels in height.

imputation

The process of filling in missing data values that consists of using statistical calculations to determine what the missing values should be.

independent variable

In an experiment, a variable that can have an effect on the dependent variable.

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.

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.

iteration

In a neural network, a single training pass (both forward and backward) through just one batch of an overall input dataset.

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

A type of clustering algorithm that iteratively updates cluster centroids based on the mean value of each data example in the centroid's cluster.

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 in higher-dimensional space.

label

In supervised machine learning, the ground truth variable that a model is meant to estimate 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.

latent representation

The simplified and feature-reduced form of an image modeled by a neural network.

learning curve

A method of visually comparing the change in a model's score or error to the number of data examples used as input.

learning mode

The way in which a machine learning model learns from data—supervised, unsupervised, semi-supervised, or reinforcement.

learning rate

In gradient descent, the size of each "step" down the slope.

lemmatization

The process of using language morphology to determine the base dictionary form of an inflected word.

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.

logistic regression

A type of regression analysis in which the output is a classification probability 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

An AI discipline in which a machine is able to gradually improve its estimative capabilities without being given explicit instructions.

machine learning model

A specific implementation of an algorithm that is used to generate predictions and other decision-making outcomes based on some training data.

MAE

(mean absolute error) A cost function that calculates the average difference between estimated and actual values without considering the sign of those values.

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.

MLOps

The discipline that involves applying a set of practices for automating the development, testing, deployment, and maintenance of machine learning models in an operational environment.

MLP

(multi-layer perceptron) A neural network algorithm that has multiple distinct layers of threshold logic units (TLUs).

model 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.

model parameter

A parameter that is internal to a machine learning model (i.e., derived from the model as it undergoes the training process).

MSE

(mean squared error) A cost function that squares the error between estimated 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.

multicollinearity

The property that describes multiple variables as exhibiting a linear relationship.

multinomial logistic regression

An algorithm commonly used to solve multi-class classification problems.

multivariate

The property of a dataset having multiple variables that are being studied.

multivariate regression

A type of regression analysis that involves multiple independent variables.

NLP

(natural language processing) A set of techniques by which computers work with and analyze human languages.

noise

Irrelevant or irregular data values, examples, or features that make it difficult to "hear" patterns revealed by other data that is actually relevant.

non-parametric

A description of a machine learning algorithm that indicates the algorithm can generate a potentially infinite number of model parameters.

normal equation

A closed-form solution to linear regression problems.

normalization

A technique in which features are scaled so that the lowest value is 0 and the highest value is 1.

offline model

A machine learning model that is deployed in such a way that it trains on new data every so often, rather than continuously.

online model

A machine learning model that is deployed in such a way that it trains on new data continuously.

ordinal data

Data that can be placed in an order.

outlier

A value outside the main distribution, deviating significantly from the rest of the values in the dataset.

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 estimations fit well to the training data but fail to generalize well to other data. An overfit model exhibits high variance and low bias.

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

The technique of dividing up processing tasks among multiple processors to scale up the performance of a software environment.

parametric

A description of a machine learning algorithm that indicates the algorithm generates a fixed number of model parameters.

penetration test

A test that uses active tools and security utilities to evaluate security by executing an authorized attack on a system.

perceptron

An algorithm used in ANNs to solve binary classification problems.

periodicity

In a seasonal forecasting model, the number of observations that comprise a single season.

perturbation

A set of methods for distorting the pixels in an image without compromising the overall information contained within that image.

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 tasks that automate the machine learning process by feeding the output of one task into the input of the next task.

POC

(proof of concept) Evidence that supports the feasibility of the product or service that a project is meant to create.

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.

prediction

The machine learning task of estimating the state of something in the future based on past or current data.

principle of least privilege

The security principle by which a user is only given the permissions they need to do their job, and no more than that.

privacy by design

An approach to software development that takes privacy into account throughout every phase of development.

problem formulation

The process of identifying an issue that should be addressed and putting that issue in terms that are understandable and actionable.

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.

pseudo-label

In semi-supervised learning, an estimation made from a model that is used as the label in training another model.

qualitative data

Data that holds categorical values.

quantitative data

Data that holds numerical values that represent magnitude.

quasi-identifier

A data value that does not directly contain PII but may be used in combination with other data values to identify an individual.

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.

recommendation system

A system that suggests items, services, and other things of interest to users.

regression

A type of machine learning task that measures the relationship between variables and outputs an estimation for a numeric variable.

regularization

The technique of simplifying a machine learning model by constraining the model parameters, which helps the model avoid overfitting to the training data.

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.

ridge regression

A regularization technique that uses an ℓ_2 norm to constrain features used to train a model.

RMSE

(root mean squared error) The square root of the [MSE](#).

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 estimated 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.

seasonality

The property by which a forecasting model considers time series data to follow a recurring pattern.

self-training

An approach to semi-supervised learning in which a model is trained on a small portion of a labeled dataset so that it can generate pseudo-

labels from the unlabeled portion, which can then be used to train an improved model.

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.

sensitivity

See *recall*.

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 cluster as compared to its neighboring clusters.

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.

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.

specificity

A measure of how frequently a machine learning model correctly identifies all actual negative instances.

spectrogram

A type of plot in which time, frequency, and amplitude of an audio signal are depicted.

stakeholder

A person who has a vested interest in the outcome of a project or who is actively involved in its work.

standardization

A technique in which features are scaled so that the mean value is 0 and the standard deviation is 1.

stationarity

The property of a forecasting model by which the statistical attributes of a variable, like mean, variance, and covariance, are kept constant rather than varying over time.

stemming

The process of removing the affix of a word in order to retrieve the word stem.

stochastic

The property by which a randomly determined process cannot perfectly estimate individual events or data points but can demonstrate a general pattern common to the entire set of data.

stop word

A word in a text document that is so common it is typically removed when the text is processed.

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 estimate 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.

tanh function

(hyperbolic tangent function) An activation function whose output values are constrained between -1 and 1 .

target function

A representation of the mapping between input variables and output variables that best approximates some desired outcome from a machine learning model.

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.

time series

A representation of data in which observations are ordered according to a sequential change in time.

TLU

(threshold logic unit) An output neuron that calculates the weighted sum of input neurons and then implements a step function.

TNR

(true negative rate) See [specificity](#).

tokenization

The process of partitioning text into smaller units.

TPR

(true positive rate) See [recall](#).

training

In machine learning, the process by which a model learns from input data.

underfitting

A problem in machine learning in which a model cannot make effective estimations due to an inability to identify the underlying patterns in the data. An underfit model exhibits low variance and high bias.

univariate

The property of a dataset having only a single variable that is being studied.

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 estimating.

VAR

(vector autoregression) A common algorithm for performing multivariate time series forecasting.

variance

A measurement of the spread between numbers in a dataset, or the variation of a model's estimations across datasets.

waveform

In audio processing, a visual representation of signal amplitude over time.

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.

Index

A

access control
 in ML 511
accuracy 285
activation functions 416
Adversarial Machine Learning Threat Matrix 509
AI 2
AI/ML solutions
 commercial problems 4
 governmental problems 5
 in general 4
 public interest problems 5
 research problems 6
algorithms
 non-parametric 138
 parametric 138
ANN
 layers 414
 overview 410
 shortcomings 432
area under curve, *See* AUC
ARIMA
 example 216
 overview 215
 seasonality 216
artificial intelligence, *See* AI
artificial neural network, *See* ANN
attributes 35
AUC 293
audio data
 defined 91
 Fourier transformation 103
 MFCCs 105
 preprocessing 106

processing 102
sampling 102
spectrogram 104
waveforms 103
automation
 of data collection 483
 of data preparation 484
 of model training 484
autoregressive integrated moving average,
See ARIMA

B

backpropagation 415
backpropagation through time, *See* BPTT
bagging
 defined 366
 out-of-bag error 366
bag of words 93
batch gradient descent, *See* BGD
Bayesian optimization 303
BCSS 316
between-cluster sum of squares, *See* BCSS
BGD 205
black box 134
bootstrap aggregating 366
 See also bagging
Box-Cox 72
BPTT 453

C

C4.5
 vs. CART 353
CACE principle 494
CART

- classification tree example 349
 hyperparameters 350
 overview 347
 regression tree example 350
 vs. C4.5 353
- categorical data 53
 CD 486
 central processing unit, *See* CPU
 CI 485
 classification
 accuracy 285
 confusion matrix 284
 considerations 284
 defined 12
 F1 score 289
 model performance 284
 precision 287
 precision-recall tradeoff 288
 recall 287
 specificity 289
- classification and regression tree, *See* CART
- cloud services
 in model deployment 473–475
- clustering
 defined 13
- CNN
 architecture 439
 filters 432, 434
 overview 432
 padding 436
 pooling layers 438
 stride 437
- coefficient of determination 176
 See also R²
- collaborative filtering 19
- collinearity 188
 See also multicollinearity
- computer vision 20
- concept drift 523
 See also model drift
- confusion matrix 284
- content filtering 19
- continuous delivery, *See* CD
- continuous integration, *See* CI
- continuous variables 73, 75
- convolutional layer 432
- convolutional neural network, *See* CNN
- cost function
 convex 201
 defined 175
- CPU 151
 cross-entropy 275
 cross-validation
 defined 149
 k determination 269
 k-fold 149
 LOOCV 149
 LPOCV 149
 stratified k-fold 149
- D**
- DAI 471
- data
 corrupted 54
 imputation 56, 57
 irregularities 54
 portions of data 35
 quality issues 36
 quantity issues 37
 statistical measures 53
 structure of data 34
 types of 53
- data binning 75
- data cleaning 52, 55
- data collection
 ethical considerations 39
- data encoding
 common methods 72
 overview 72
- data hierarchy
 DIK 3
- data munging 52
- data preparation 52
- data preprocessing 69
- datasets
 in machine learning 34
- data sources 37
- data wrangling 52
- date conversion 55
- Davies–Bouldin index 318, 333
- decision boundary 252
- decision trees
 and continuous variables 353
 C4.5 352
 CART 347
 CART vs. C4.5 353
 comparison to other algorithms 354
 ID3 353
 overview 346
 pruning 351
 deduplication 56

deep learning 2, 133, 471
dendograms

 hierarchical clustering 334

dependent variables 13

deployment

 batch 469

 cloud services 473–475

DAI 471

 deep learning models 471

Docker 475

 endpoints 473

 infrastructure requirements 471

 in ML 468

 model output 469

 offline vs. online models 468

 real-time serving 469

 stakeholder requirements 472

 streaming 470

design of experiments, *See* DOE

diagnostic systems 19

dimensionality

 curse of dimensionality 76

dimensionality reduction

 algorithms 77

 defined 76

dimensions 35

discrete variables 74

discretization 75

discriminator 440

distributed artificial intelligence, *See* DAI

Docker

 for deployment 475

DOE 13

Dunn index 317, 333

E

elastic net regression 188

embedding

 defined 91

 tools 92

endogenous variable 232

ensemble learning 365

entropy

 in decision trees 352

epoch

 in a neural network 415

errors 37

ethics

 in AI/ML 8

 in data collection 39

 in feature engineering 79

in model operationalization 496

in training process 152

ETL 38

evaluation metrics 146

evolutionary optimization 304

See also genetic algorithms

exogenous variable 232

experimental design 13

extract, transform, and load, *See* ETL

F

false positive rate, *See* FPR

feature engineering 69, 79

feature extraction 76

feature map 432

features 35

feature scaling 70, 71

feature selection 76

feature splitting 75

feedforward neural network, *See* FNN

filters 432, 434

FNN 451

forecasting

 stationarity 232

 time series 214

FPR 290

F1 score 289

G

GAN

 architecture 440

 overview 440

gated recurrent unit cell, *See* GRU cell

Gaussian radial basis function kernel, *See*

 Gaussian RBF kernel

 Gaussian RBF kernel 390

generative adversarial network, *See* GAN

generator 440

genetic algorithms 304

Gini index

 and CART 347

Goodhart's Law 147

GPU 151

gradient boosting 368

gradient descent

 and cross-entropy 275

 learning rate 202

 overview 200

 techniques 204

 vs. normal equation 201

graphics processing unit, *See* GPU
 grid search 302
 GRU cell 455

H

HAC 331
 hard-margin classification 384
 HDC 332
 hidden layers 414
 hierarchical agglomerative clustering, *See* HAC
 hierarchical clustering
 dendrograms 334
 spherical dataset 333
 spiral dataset 332
 two approaches 331
 when to stop 333
 hierarchical divisive cluster, *See* HDC
 hyperbolic tangent function, *See* tanh function
 hyperparameter optimization 302
 hyperparameters 138
 hyperplanes
 in SVMs 382

I

ID3 353
 identity matrix 170
 image data
 aspect ratio 97
 augmentation 100
 defined 91
 grayscale conversion 95
 latent representation 94
 normalization 100
 perturbation 99
 reshaping 98
 resolution 97
 scaling 97
 standardization 100
 imputation 56, 57
 independent variables 13
 information gain
 in decision trees 352
 input layers 414
 iteration
 vs. epoch 415
 iterative models 200

K

kernel trick

example 388
 methods 389
 k-means clustering
 centroid 314
 cluster sum of squares 316
 Davies–Bouldin index 318
 Dunn index 317
 elbow point 316
 global vs. local optimization 315
 overview 314
 shortcomings 330
 silhouette analysis 317
 vs. k-nearest neighbor 315
 k-nearest neighbor, *See* k-NN
 k-NN
 and multi-class classification 276
 k determination 268
 overview 267
 vs. k-means clustering 315
 vs. logistic regression 269

L

labels 21
 lasso regression 188
 learning curve 147
 leave-one-out cross-validation, *See* LOOCV
 leave-p-out cross-validation, *See* LPOCV
 lemmatization 94
 linear equation
 data example 165, 166
 overview 165
 shortcomings 168
 linear kernel 389
 linear model
 with high-order fits 172
 with multiple parameters 174
 linear regression
 identity matrix 170
 in machine learning 168, 169
 matrices in 169
 multivariate 174
 overview 164
 shortcomings 250
 logistic function 251, 252
 logistic regression
 cost function 253
 overview 251
 vs. k-NN 269
 long short-term memory cell, *See* LSTM cell
 LOOCV 149
 LPOCV 149

LSTM cell

architecture 454
overview 454
process 455

M

machine learning

computer vision 20
diagnostic systems 19
learning modes 21
model types 12
NLP systems 19
overview 2
pipelines 482, 485
prediction systems 18
recommendation systems 18
robotics 20
stochastic modeling 13
systems and techniques 18

machine learning algorithms

overview 132
selection 133

machine learning models

bias 134
bias-variance tradeoff 135
checkpointing and versioning 526
cross-validation 149
defined 132
evaluation metrics 146
generalization 134
holdout method 136
irreducible error 135
iterative tuning 146
model drift 523
model optimization 148
model retraining 523, 525
models in combination 151
model structure 150
model training time 151
overfitting 134
parameters 137
skillful 146
target function 134
training 132
underfitting 134
variance 134

machine learning systems

design pattern issues 493
documentation 492
integrating models 491
model APIs 491, 492

MAE 175

MBGD 205
mean absolute error, *See* MAE
mean squared error, *See* MSE
memory cells 452

mini-batch gradient descent, *See* MBGD

MLOps

CI/CD 485, 486
overview 482

MLP

backpropagation 415
layers 414
overview 414
model drift 523, 525
model parameters 137
MSE 175
multi-class classification 274, 276
multicollinearity 188
multi-label classification
 overview 274
 perceptrons 411
multi-layer perceptron, *See* MLP
multinomial logistic regression 274

N

natural language processing, *See* NLP
NLP 19
noise 37
normal equation 170, 177, 201
normalization 70
numerical data 53

O

offline models 468
online models 468
ordinal data 53
outliers 37
output layers 414

P

padding 436
parallelization 151
penetration test
 in ML pipelines 510
perceptrons
 multi-label classification 411
 overview 410
 shortcomings 413
 training 412

periodicity 217
 personally identifiable information, *See* PII
 PII 39
 pipelines
 automation using CI/CD 486
 continuous testing 522
 machine learning process 482
 monitoring 521, 522, 526
 securing 508
 triggers 485
 POC 14
 polynomial kernel 390
 pooling layer 438
 PRC 294
 precision 287, 288
 precision–recall curve, *See* PRC
 prediction systems 18
 problem formulation 11
 proof of concept, *See* POC
 pseudo-labels 25

Q

qualitative data 53
See also categorical data
 quantitative data 53
See also numerical data
 quasi-identifiers 80

R

RAE 176
 random forest
 bagging 366
 feature selection benefits 367
 hyperparameters 367
 overview 365
 vs. gradient boosting 368
 randomized search 303
 recall 287, 288
 receiver operating characteristic, *See* ROC
 recommendation systems
 defined 18
 types of 19
 rectified linear unit function, *See* ReLU
 function
 recurrent neural network, *See* RNN
 regression
 defined 12
 regularization
 overview 149
 techniques 187

reinforcement learning
 examples 28
 overview 27
 relative absolute error, *See* RAE
 relative squared error, *See* RSE
 ReLU function 417
 ridge regression 187
 RMSE 175
 RNN
 BPTT 453
 memory cells 452
 overview 451
 shortcomings 453
 text data processing 455
 robotics 20
 ROC 290
 root mean squared error, *See* RMSE
 RSE 176
 R^2 176

S

SAG 205
 security
 access control 511
 continuous testing 522
 hashing 511
 logging 521, 522
 ML pipelines 508, 521
 model poisoning and evasion 508
 penetration test 510
 pipeline jobs/tasks 511
 pipeline platform 509
 pipeline platform updates 510
 principle of least privilege 514, 515
 user actions management 515
 user role management 514
 semi-structured data 34
 semi-supervised learning
 co-training 26
 examples 26
 overview 25
 self-training 25
 sensitivity 292
See also recall
 SGD 205
 sigmoid function 416
 sigmoid kernel 390
 silhouette analysis 317, 333
 soft-margin classification 385
 softmax function 416
 specificity 289

stakeholders

- communication strategy 8
- project role 7

standardization 71

standard score 71

See also z-score

stationarity 232

stemming 93

stochastic average gradient, *See* SAG

stochastic gradient descent, *See* SGD

stochastic modeling 13

stop words 93

stride 437

structured data 34

supervised learning

- examples 22

- overview 21

support-vector machines, *See* SVMs

SVMs

- hard-margin classification 384

- hyperplanes 382

- kernel trick 387–389

- linear classification 382

- non-linear classification 386

- overview 382

- regression 400

- soft-margin classification 385

T

tanh function 416

textual data

- defined 91

- transformation techniques 92, 455

threshold logic unit, *See* TLU

thresholds 291

time series forecasting

- in general 214

- multivariate 230

- stationarity 232

- univariate 214, 215

TLU 410, 414

TNR 289

See also specificity

tokenization 93

TPR 290

true negative rate, *See* TNR

true positive rate, *See* TPR

U

unstructured data

defined 34

examples 91

unsupervised learning

- examples 24

- overview 23

V

VAR

- endogenous and exogenous variables 232

- example 233

- overview 230

vector autoregression, *See* VAR

video data

- defined 91

- preprocessing 101, 102

W

WCSS 316

within-cluster sum of squares, *See* WCSS

X

XGBoost

- and gradient boosting 368

Z

z-score 71

See also standard score

