Statistical Methods for Machine Learning

Discover how to Transform Data into Knowledge with Python

Jason Brownlee

MACHINE LEARNING MASTERY

Disclaimer

Acknowledgements

Special thanks to my copy editor Sarah Martin and my technical editors Arun Koshy and Andrei Cheremskoy.

Copyright

Edition: v1.3

Contents

Preface

Statistics is Important

Statistics is important to machine learning practitioners.

- Statistics is a prerequisite in most courses and books on applied machine learning.
- Statistical methods are used at each step in an applied machine learning project.
- Statistical learning is the applied statistics equivalent of predictive modeling in machine learning.

A machine learning practitioner cannot be effective without an understanding of basic statistical concepts and statistics methods, and an effective practitioner cannot excel without being aware of and leveraging the terminology and methods used in the sister field of statistical learning.

Practitioners Don't Know Stats

Developers don't know statistics and this is a huge problem. Programmers don't need to know and use statistical methods in order to develop software. Software engineering and computer science courses generally don't include courses on statistics, let alone advanced statistical tests. As such, it is common for machine learning practitioners coming from the computer science or developer tradition to not know and not value statistical methods. This is a problem given the pervasive use of statistical methods and statistical thinking in the preparation of data, evaluation of learned models, and all other steps in a predictive modeling project.

Practitioners Study The Wrong Stats

Eventually, machine learning practitioners realize the need for skills in statistics. This might start with a need to better interpret descriptive statistics or data visualizations and may progress to the need to start using sophisticated hypothesis tests. The problem is, they don't seek out the statistical information they need. Instead, they try to read through a text book on statistics or work through the material for an undergraduate course on statistics. This approach is slow, it's boring, and it covers a breadth and depth of material on statistics that is beyond the needs of the machine learning practitioner.

Practitioners Study Stats The Wrong Way

It's worse than this. Regardless of the medium used to learn statistics, be it books, videos, or course material, machine learning practitioners study statistics the wrong way. Because the material is intended for undergraduate students that need to pass a test, the material is focused on the theory, on proofs, on derivations. This is great for testing students but terrible for practitioners that need results. Practitioners need methods that clearly state when they are appropriate and instruction on how to interpret the result. They need code examples that they can use immediately on their project.

A Better Way

I set out to write a playbook for machine learning practitioners that gives them only those parts of statistics that they need to know in order to work through a predictive modeling project. I set out to present statistical methods in the way that practitioners learn–that is with simple language and working code examples. Statistics is important to machine learning, and I believe that if it is taught at the right level for practitioners, that it can be a fascinating, fun, directly applicable, and immeasurably useful area of study. I hope that you agree.

Jason Brownlee
2018

Part I

Introduction

Welcome

Welcome to *Statistical Methods for Machine Learning*. The field of statistics is hundreds of years old and statistical methods are central to working through predictive modeling problems with machine learning. Statistical methods refer to a range of techniques from simple summary statistics intended to help better understand data, to statistical hypothesis tests and estimation statistics that can be used to interpret the results of experiments and predictions from models. I designed this book to teach you step-by-step the basics of statistical methods with concrete and executable examples in Python.

Who Is This Book For?

Before we get started, let's make sure you are in the right place. This book is for developers that may know some applied machine learning. Maybe you know how to work through a predictive modeling problem end-to-end, or at least most of the main steps, with popular tools. The lessons in this book do assume a few things about you, such as:

- You know your way around basic Python for programming.
- You may know some basic NumPy for array manipulation.
- You want to learn statistical methods to deepen your understanding and application of machine learning.

This guide was written in the top-down and results-first machine learning style that you're used to from Machine Learning Mastery.

About Your Outcomes

This book will teach you the basics of statistical methods that you need to know as a machine learning practitioner. After reading and working through this book, you will know:

- About the field of statistics, how it relates to machine learning, and how to harness statistical methods on a machine learning project.
- How to calculate and interpret common summary statistics and how to present data using standard data visualization techniques.
- Findings from mathematical statistics that underlie much of the field, such as the central limit theorem and the law of large numbers.

- How to evaluate and interpret the relationship between variables and the independence of variables.
- How to calculate and interpret parametric statistical hypothesis tests for comparing two or more data samples.
- How to calculate and interpret interval statistics for distributions, population parameters, and observations.
- How to use statistical resampling to make good economic use of available data in order to evaluate predictive models.
- How to calculate and interpret nonparametric statistical hypothesis tests for comparing two or more data samples that do not conform to the expectations of parametric tests.

This new basic understanding of statistical methods will impact your practice of machine learning in the following ways:

- Use descriptive statistics and data visualizations to quickly and more deeply understand the shape and relationships in data.
- Use inferential statistical tests to quickly and effectively quantify the relationships between samples, such as the results of experiments with different predictive algorithms or differing configurations.
- Use estimation statistics to quickly and effectively quantify the confidence in estimated model skill and model predictions.

This book is not a substitute for an undergraduate course in statistics or a textbook for such a course, although it could complement such materials. For a good list of top courses, textbooks, and other resources on statistics, see the *Further Reading* section at the end of each tutorial.

How to Read This Book

This book was written to be read linearly, from start to finish. That being said, if you know the basics and need help with a specific notation or operation, then you can flip straight to that section and get started. This book was designed for you to read on your workstation, on the screen, not on a tablet or eReader. My hope is that you have the book open right next to your editor and run the examples as you read about them.

This book is not intended to be read passively or be placed in a folder as a reference text. It is a playbook, a workbook, and a guidebook intended for you to learn by doing and then apply your new understanding with working Python examples. To get the most out of the book, I would recommend playing with the examples in each tutorial. Extend them, break them, then fix them. Try some of the extensions presented at the end of each lesson and let me know how you do.

About the Book Structure

This book was designed around major statistical techniques that are directly relevant to applied machine learning. There are a lot of things you could learn about statistics, from theory to abstract concepts to APIs. My goal is to take you straight to developing an intuition for the elements you must understand with laser-focused tutorials. I designed the tutorials to focus on how to get things done with statistics. They give you the tools to both rapidly understand and apply each technique or operation.

Each of the tutorials are designed to take you about one hour to read through and complete, excluding the extensions and further reading. You can choose to work through the lessons one per day, one per week, or at your own pace. I think momentum is critically important, and this book is intended to be read and used, not to sit idle. I would recommend picking a schedule and sticking to it. The tutorials are divided into 6 parts, they are:

- **Part 1: Statistics**. Provides a gentle introduction to the field of statistics, the relationship to machine learning, and the importance that statistical methods have when working through a predictive modeling problem.
- **Part 2: Foundation**. Introduction to descriptive statistics, data visualization, random numbers, and important findings in statistics such as the law of large numbers and the central limit theorem.
- **Part 3: Hypothesis Testing**. Covers statistical hypothesis tests for comparing populations of samples and the interpretation of tests with p-values and critical values.
- **Part 4: Resampling**. Covers methods from statistics used to economically use small samples of data to evaluate predictive models such as k-fold cross-validation and the bootstrap.
- **Part 5: Estimation Statistics**. Covers an alternative to hypothesis testing called estimation statistics, including tolerance intervals, confidence intervals, and prediction intervals.
- **Part 6: Nonparametric Methods**. Covers nonparametric statistical hypothesis testing methods for use when data does not meet the expectations of parametric tests.

Each part targets a specific learning outcome, and so does each tutorial within each part. This acts as a filter to ensure you are only focused on the things you need to know to get to a specific result and do not get bogged down in the math or near-infinite number of digressions. The tutorials were not designed to teach you everything there is to know about each of the theories or techniques of statistics. They were designed to give you an understanding of how they work, how to use them, and how to interpret the results the fastest way I know how: to learn by doing.

About Python Code Examples

The code examples were carefully designed to demonstrate the purpose of a given lesson. Code examples are complete and standalone. The code for each lesson will run as-is with no code

from prior lessons or third-parties required beyond the installation of the required packages. A complete working example is presented with each tutorial for you to inspect and copy-and-paste. All source code is also provided with the book and I would recommend running the provided files whenever possible to avoid any copy-paste issues.

The provided code was developed in a text editor and intended to be run on the command line. No special IDE or notebooks are required. If you are using a more advanced development environment and are having trouble, try running the example from the command line instead. All code examples were tested on a POSIX-compatible machine with Python 3.

About Further Reading

Each lesson includes a list of further reading resources. This may include:

- Books and book chapters.
- API documentation.
- Articles and Webpages.

Wherever possible, I try to list and link to the relevant API documentation for key functions used in each lesson so you can learn more about them. I have tried to link to books on Amazon so that you can learn more about them. I don't know everything, and if you discover a good resource related to a given lesson, please let me know so I can update the book.

About Getting Help

You might need help along the way. Don't worry; you are not alone.

- **Help with a Technique?** If you need help with the technical aspects of a specific operation or technique, see the *Further Reading* section at the end of each lesson.
- **Help with Python APIs?** If you need help with using the NumPy or SciPy libraries, see the list of resources in the *Further Reading* section at the end of each lesson, and also see *Appendix A*.
- **Help with your workstation?** If you need help setting up your environment, I would recommend using Anaconda and following my tutorial in *Appendix B*.
- **Help with the math?** I provided a list of locations where you can search for answers and ask questions about statistics math in *Appendix A*. You can also see *Appendix C* for a crash course on math notation.
- **Help in general?** You can shoot me an email. My details are in *Appendix A*.

Summary

Are you ready? Let's dive in! Next up you will discover a gentle introduction to the field of statistics.

Part II

Statistics

Chapter 1

Introduction to Statistics

Statistics is a collection of tools that you can use to get answers to important questions about data. You can use descriptive statistical methods to transform raw observations into information that you can understand and share. You can use inferential statistical methods to reason from small samples of data to whole domains. In this chapter, you will discover clearly why statistics is important in general and for machine learning and generally the types of methods that are available. After reading this chapter, you will know:

- Statistics is generally considered a prerequisite to the field of applied machine learning.
- We need statistics to help transform observations into information and to answer questions about samples of observations.
- Statistics is a collection of tools developed over hundreds of years for summarizing data and quantifying properties of a domain given a sample of observations.

Let's get started.

1.1 Statistics is Required Prerequisite

Machine learning and statistics are two tightly related fields of study. So much so that statisticians refer to machine learning as *applied statistics* or *statistical learning* rather than the computer-science-centric name. Machine learning is almost universally presented to beginners assuming that the reader has some background in statistics. We can make this concrete with a few cherry picked examples. Take a look at this quote from the beginning of a popular applied machine learning book titled *Applied Predictive Modeling*:

> ... the reader should have some knowledge of basic statistics, including variance, correlation, simple linear regression, and basic hypothesis testing (e.g. p-values and test statistics).
>
> — Page vii, *Applied Predictive Modeling*, 2013.

Here's another example from the popular *Introduction to Statistical Learning* book:

> We expect that the reader will have had at least one elementary course in statistics.

— Page 9, *An Introduction to Statistical Learning with Applications in R*, 2013.

Even when statistics is not a prerequisite, some primitive prior knowledge is required as can be seen in this quote from the widely read *Programming Collective Intelligence*:

> ... this book does not assume you have any prior knowledge of [...] or statistics. [...] but having some knowledge of trigonometry and basic statistics will help you understand the algorithms.

— Page xiii, *Programming Collective Intelligence: Building Smart Web 2.0 Applications*, 2007.

In order to be able to understand machine learning, some basic understanding of statistics is required. To see why this is the case, we must first understand why we need the field of statistics in the first place.

1.2 Why Learn Statistics?

Raw observations alone are data, but they are not information or knowledge. Data raises questions, such as:

- What is the most common or expected observation?
- What are the limits on the observations?
- What does the data look like?

Although they appear simple, these questions must be answered in order to turn raw observations into information that we can use and share. Beyond raw data, we may design experiments in order to collect observations. From these experimental results we may have more sophisticated questions, such as:

- What variables are most relevant?
- What is the difference in an outcome between two experiments?
- Are the differences real or the result of noise in the data?

Questions of this type are important. The results matter to the project, to stakeholders, and to effective decision making. Statistical methods are required to find answers to the questions that we have about data. We can see that in order to both understand the data used to train a machine learning model and to interpret the results of testing different machine learning models, that statistical methods are required. This is just the tip of the iceberg as each step in a predictive modeling project will require the use of a statistical method.

1.3 What is Statistics?

Statistics is a subfield of mathematics. It refers to a collection of methods for working with data and using data to answer questions.

> Statistics is the art of making numerical conjectures about puzzling questions. [...] The methods were developed over several hundred years by people who were looking for answers to their questions.
>
> — Page xiii, *Statistics*, Fourth Edition, 2007.

It is because the field is comprised of a grab bag of methods for working with data that it can seem large and amorphous to beginners. It can be hard to see the line between methods that belong to statistics and methods that belong to other fields of study. Often a technique can be both a classical method from statistics and a modern algorithm used for feature selection or modeling. Although a working knowledge of statistics does not require deep theoretical knowledge, some important and easy-to-digest theorems from the relationship between statistics and probability can provide a valuable foundation.

Two examples include the law of large numbers and the central limit theorem; the first aids in understanding why bigger samples are often better and the second provides a foundation for how we can compare the expected values between samples (e.g mean values). When it comes to the statistical tools that we use in practice, it can be helpful to divide the field of statistics into two large groups of methods: descriptive statistics for summarizing data and inferential statistics for drawing conclusions from samples of data.

> Statistics allow researchers to collect information, or data, from a large number of people and then summarize their typical experience. [...] Statistics are also used to reach conclusions about general differences between groups. [...] Statistics can also be used to see if scores on two variables are related and to make predictions.
>
> — Pages ix-x, *Statistics in Plain English*, Third Edition, 2010.

1.3.1 Descriptive Statistics

Descriptive statistics refer to methods for summarizing raw observations into information that we can understand and share. Commonly, we think of descriptive statistics as the calculation of statistical values on samples of data in order to summarize properties of the sample of data, such as the common expected value (e.g. the mean or median) and the spread of the data (e.g. the variance or standard deviation). Descriptive statistics may also cover graphical methods that can be used to visualize samples of data. Charts and graphics can provide a useful qualitative understanding of both the shape or distribution of observations as well as how variables may relate to each other.

1.3.2 Inferential Statistics

Inferential statistics is a fancy name for methods that aid in quantifying properties of the domain or population from a smaller set of obtained observations called a sample. Commonly, we think

of inferential statistics as the estimation of quantities from the population distribution, such as the expected value or the amount of spread.

More sophisticated statistical inference tools can be used to quantify the likelihood of observing data samples given an assumption. These are often referred to as tools for statistical hypothesis testing, where the base assumption of a test is called the null hypothesis. There are many examples of inferential statistical methods given the range of hypotheses we may assume and the constraints we may impose on the data in order to increase the power or likelihood that the finding of the test is correct.

1.4 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

1.4.1 Books

- Applied Predictive Modeling, 2013.
  https://amzn.to/2InAS0T
- An Introduction to Statistical Learning with Applications in R, 2013.
  https://amzn.to/2Gvhkqz
- Programming Collective Intelligence: Building Smart Web 2.0 Applications, 2007.
  https://amzn.to/2GIN9jc
- Statistics, Fourth Edition, 2007.
  https://amzn.to/2pUA0tU
- All of Statistics: A Concise Course in Statistical Inference, 2004.
  https://amzn.to/2H224Tp
- Statistics in Plain English, Third Edition, 2010.
  https://amzn.to/2Gv0A2V

1.4.2 Articles

- Statistics on Wikipedia.
  https://en.wikipedia.org/wiki/Statistics
- Portal:Statistics on Wikipedia.
  https://en.wikipedia.org/wiki/Portal:Statistics
- List of statistics articles on Wikipedia.
  https://en.wikipedia.org/wiki/List_of_statistics_articles
- Mathematical statistics on Wikipedia.
  https://en.wikipedia.org/wiki/Mathematical_statistics
- History of statistics on Wikipedia.
  https://en.wikipedia.org/wiki/History_of_statistics

- Descriptive Statistics on Wikipedia.
  https://en.wikipedia.org/wiki/Descriptive_statistics
- Statistical Inference on Wikipedia.
  https://en.wikipedia.org/wiki/Statistical_inference

1.5 Summary

In this chapter, you discovered clearly why statistics is important in general and for machine learning, and generally the types of methods that are available. Specifically, you learned:

- Statistics is generally considered a prerequisite to the field of applied machine learning.
- We need statistics to help transform observations into information and to answer questions about samples of observations.
- Statistics is a collection of tools developed over hundreds of years for summarizing data and quantifying properties of a domain given a sample of observations.

1.5.1 Next

In the next section, you will discover the tight relationship and differences between machine learning and statistics.

Chapter 2

Statistics vs Machine Learning

The machine learning practitioner has a tradition of algorithms and a pragmatic focus on results and model skill above other concerns such as model interpretability. Statisticians work on much the same type of modeling problems under the names of applied statistics and statistical learning. Coming from a mathematical background, they have more of a focus on the behavior of models and explainability of predictions.

The very close relationship between the two approaches to the same problem means that both fields have a lot to learn from each other. The statisticians need to consider algorithmic methods was called out in the classic *two cultures* paper. Machine learning practitioners must also take heed, keep an open mind, and learn both the terminology and relevant methods from applied statistics. In this chapter, you will discover that machine learning and statistical learning are two closely related but different perspectives on the same problem. After reading this chapter, you will know:

- *Machine learning* and *predictive modeling* are a computer science perspective on modeling data with a focus on algorithmic methods and model skill.
- *Statistics* and *statistical learning* are a mathematical perspective on modeling data with a focus on data models and on goodness of fit.
- Machine learning practitioners must keep an open mind and leverage methods and understand the terminology from the closely related fields of applied statistics and statistical learning.

Let's get started.

2.1 Machine Learning

Machine learning is a subfield of artificial intelligence and is related to the broader field of computer science. When it comes to developing machine learning models in order to make predictions, there is a heavy focus on algorithms, code, and results. Machine learning is a lot broader than developing models in order to make predictions, as can be seen by the definition in the classic 1997 textbook by Tom Mitchell.

> The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience.

— Page xv, *Machine Learning*, 1997.

Here, we can see that from a research perspective, machine learning is really the study of learning with computer programs. It just so happens that some of these learning programs are useful for predictive modeling problems, and some in fact have been borrowed from other fields, such as statistics. Linear regression is a perfect example. It is a more-than-a-century-old method from the (at the time: nascent) field of statistics that is used for fitting a line or plane to real-valued data. From a machine learning perspective, we look at it as a system for learning weights (coefficients) in response to examples from a domain.

Many methods have been developed in the field of artificial intelligence and machine learning, sometimes by statisticians, that prove very useful for the task of predictive modeling. A good example is classification and regression trees that bears no resemblance to classical methods in statistics.

2.2 Predictive Modeling

The useful part of machine learning for the practitioner may be called predictive modeling. This explicitly ignores distinctions between statistics and machine learning. It also shucks off the broader objectives of statistics (understanding data) and machine learning (understanding learning in software) and only concerns itself, as its name suggests, with developing models that make predictions.

> The term predictive modeling may stir associations such as machine learning, pattern recognition, and data mining. Indeed, these associations are appropriate and the methods implied by these terms are an integral piece of the predictive modeling process. But predictive modeling encompasses much more than the tools and techniques for uncovering patterns within data. The practice of predictive modeling defines the process of developing a model in a way that we can understand and quantify the model's prediction accuracy on future, yet-to-be-seen data.

— Page vii, *Applied Predictive Modeling*, 2013

Predictive modeling provides a laser-focus on developing models with the objective of getting the best possible results with regard to some measure of model skill. This pragmatic approach often means that results in the form of maximum skill or minimum error are sought at the expense of almost everything else. It doesn't really matter what we call the process, machine learning or predictive modeling. In some sense it is marketing and group identification. Getting results and delivering value matters more to the practitioner.

2.3 Statistical Learning

The process of working with a dataset and developing a predictive model is also a task in statistics. A statistician may have traditionally referred to the activity as applied statistics. Statistics is a subfield of mathematics, and this heritage gives a focus of well defined, carefully chosen methods. A need to understand not only why a specific model was chosen, but also how

and why specific predictions are made. From this perspective, often model skill is important, but less important than the interpretability of the model.

Nevertheless, modern statisticians have formulated a new perspective as a subfield of applied statistics called *statistical learning*. It may be the statistics equivalent of *predictive modeling* where model skill is important, but perhaps a stronger emphasis is given to careful selection and introduction of the learning models.

> Statistical learning refers to a set of tools for modeling and understanding complex datasets. It is a recently developed area in statistics and blends with parallel developments in computer science and, in particular, machine learning.
>
> — Page vii, *An Introduction to Statistical Learning with Applications in R*, 2013.

We can see that there is a bleeding of ideas between fields and subfields in statistics. The machine learning practitioner must be aware of both the machine learning and statistical-based approach to the problem. This is especially important given the use of different terminology in both domains. In his course on statistics, Rob Tibshirani, a statistician who also has a foot in machine learning, provides a glossary that maps terms in statistics to terms in machine learning, reproduced below.

Glossary

Machine learning	Statistics
network, graphs	model
weights	parameters
learning	fitting
generalization	test set performance
supervised learning	regression/classification
unsupervised learning	density estimation, clustering
large grant = $1,000,000	large grant= $50,000
nice place to have a meeting: Snowbird, Utah, French Alps	nice place to have a meeting: Las Vegas in August

Figure 2.1: Glossary mapping terms in statistics to terms in machine learning.

This highlights the deeper need for the machine learning practitioner to focus on predictive modeling and stay open to methods, ideas, and terminology, regardless of the field of origin. This may apply to modern fields like bioinformatics and econometrics but applies more so to the tightly related and much older field of statistics.

2.4 Two Cultures

Recently, and perhaps still now, applied statisticians looked down the field of machine learning and the practice of results-at-any-cost predictive modeling. Both fields offer tremendous value, but perhaps on subtly different flavors of the same general problem of predictive modeling. Real and valuable contributions have been made to modeling from the computer science perspective of machine learning such as decision trees mentioned above and artificial neural networks, more recently relabeled deep learning, to name two well known examples. Just as the machine learning practitioner must keep an eye on applied statistics and statistical learning, the statistician must keep an eye on machine learning.

This call was made clearly in the now (perhaps famous) 2001 paper titled *Statistical Modeling: The Two Cultures* by Leo Breiman. In it, he contrasts the *data modeling culture* of statisticians to the *algorithmic modeling culture* of all other fields, to which machine learning belongs. He highlights these cultures as ways of thinking about the same problem of mapping inputs to outputs, where the statistical approach is to focus on goodness of fit tests and the algorithmic approach focuses on predictive accuracy.

He suggests that the field of statistics will suffer both by losing relevance and in the fragility of the methods by ignoring the algorithmic approach. The classical approach he refers to as *data models*, a subtle but important shift in focus where a practitioner chooses and focuses on the behavior of the model (e.g. logistic regression) rather than the data and processes that may have generated it. This might be characterized (perhaps unfairly) as focusing on making the data fit the model rather than choosing or adapting the model to fit the data.

> The statistical community has been committed to the almost exclusive use of data models. This commitment has led to irrelevant theory, questionable conclusions, and has kept statisticians from working on a large range of interesting current problems. [...] If our goal as a field is to use data to solve problems, then we need to move away from exclusive dependence on data models and adopt a more diverse set of tools.
>
> — *Statistical Modeling: The Two Cultures*, Leo Breiman, 2001.

It's an important paper, still relevant and a great read more than 15 years later. The emergence of subfields like *statistical learning* by statisticians suggests that headway is being made.

2.5 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

- Statistical Modeling: The Two Cultures, 2001.
  http://projecteuclid.org/euclid.ss/1009213726
- Statistics vs. Machine Learning, fight!, 2008.
  http://brenocon.com/blog/2008/12/statistics-vs-machine-learning-fight/
- The Two Cultures: statistics vs. machine learning? on Cross Validated.
  https://stats.stackexchange.com/questions/6/the-two-cultures-statistics-vs-machine-

- Glossary: machine learning vs statistics, Modern Applied Statistics: Elements of Statistical Learning.
  http://statweb.stanford.edu/~tibs/stat315a/glossary.pdf

2.6 Summary

In this chapter, you discovered that machine learning and statistical learning are two closely related but different perspectives on the same problem. Specifically, you learned:

- *Machine learning* and *predictive modeling* are a computer science perspective on modeling data with a focus on algorithmic methods and model skill.
- *Statistics* and *statistical learning* are a mathematical perspective on modeling data with a focus on data models and on goodness of fit.
- Machine learning practitioners must keep an open mind and leverage methods and understand the terminology from the closely related fields of applied statistics and statistical learning.

2.6.1 Next

In the next section, you will discover specific examples of where statistical methods are used in applied machine learning.

Chapter 3

Examples of Statistics in Machine Learning

Statistics and machine learning are two very closely related fields. In fact, the line between the two can be very fuzzy at times. Nevertheless, there are methods that clearly belong to the field of statistics that are not only useful, but invaluable when working on a machine learning project. It would be fair to say that statistical methods are required to effectively work through a machine learning predictive modeling project. In this chapter, you will discover specific examples of statistical methods that are useful and required at key steps in a predictive modeling problem. After completing this chapter, you will know:

- Exploratory data analysis, data summarization, and data visualizations can be used to help frame your predictive modeling problem and better understand the data.
- That statistical methods can be used to clean and prepare data ready for modeling.
- That statistical hypothesis tests and estimation statistics can aid in model selection and in presenting the skill and predictions from final models.

Let's get started.

3.1 Overview

In this chapter, we are going to look at 10 examples of where statistical methods are used in an applied machine learning project. This will demonstrate that a working knowledge of statistics is essential for successfully working through a predictive modeling problem.

1. Problem Framing
2. Data Understanding
3. Data Cleaning
4. Data Selection
5. Data Preparation

6. Model Evaluation
7. Model Configuration
8. Model Selection
9. Model Presentation
10. Model Predictions

3.2 Problem Framing

Perhaps the point of biggest leverage in a predictive modeling problem is the framing of the problem. This is the selection of the type of problem, e.g. regression or classification, and perhaps the structure and types of the inputs and outputs for the problem. The framing of the problem is not always obvious. For newcomers to a domain, it may require significant exploration of the observations in the domain.

For domain experts that may be stuck seeing the issues from a conventional perspective, they too may benefit from considering the data from multiple perspectives. Statistical methods that can aid in the exploration of the data during the framing of a problem include:

- **Exploratory Data Analysis**. Summarization and visualization in order to explore ad hoc views of the data.
- **Data Mining**. Automatic discovery of structured relationships and patterns in the data.

3.3 Data Understanding

Data understanding means having an intimate grasp of both the distributions of variables and the relationships between variables. Some of this knowledge may come from domain expertise, or require domain expertise in order to interpret. Nevertheless, both experts and novices to a field of study will benefit from actually handling real observations form the domain. Two large branches of statistical methods are used to aid in understanding data; they are:

- **Summary Statistics**. Methods used to summarize the distribution and relationships between variables using statistical quantities.
- **Data Visualizations**. Methods used to summarize the distribution and relationships between variables using visualizations such as charts, plots, and graphs.

3.4 Data Cleaning

Observations from a domain are often not pristine. Although the data is digital, it may be subjected to processes that can damage the fidelity of the data, and in turn any downstream processes or models that make use of the data. Some examples include:

- Data corruption.

- Data errors.
- Data loss.

The process of identifying and repairing issues with the data is called data cleaning Statistical methods are used for data cleaning; for example:

- **Outlier detection**. Methods for identifying observations that are far from the expected value in a distribution.
- **Imputation**. Methods for repairing or filling in corrupt or missing values in observations.

3.5 Data Selection

Not all observations or all variables may be relevant when modeling. The process of reducing the scope of data to those elements that are most useful for making predictions is called data selection. Two types of statistical methods that are used for data selection include:

- **Data Sample**. Methods to systematically create smaller representative samples from larger datasets.
- **Feature Selection**. Methods to automatically identify those variables that are most relevant to the outcome variable.

3.6 Data Preparation

Data can often not be used directly for modeling. Some transformation is often required in order to change the shape or structure of the data to make it more suitable for the chosen framing of the problem or learning algorithms. Data preparation is performed using statistical methods. Some common examples include:

- **Scaling**. Methods such as standardization and normalization.
- **Encoding**. Methods such as integer encoding and one hot encoding.
- **Transforms**. Methods such as power transforms like the Box-Cox method.

3.7 Model Evaluation

A crucial part of a predictive modeling problem is evaluating a learning method. This often requires the estimation of the skill of the model when making predictions on data not seen during the training of the model. Generally, the planning of this process of training and evaluating a predictive model is called experimental design. This is a whole subfield of statistical methods.

- **Experimental Design**. Methods to design systematic experiments to compare the effect of independent variables on an outcome, such as the choice of a machine learning algorithm on prediction accuracy.

As part of implementing an experimental design, methods are used to resample a dataset in order to make economic use of available data in order to estimate the skill of the model. These methods are another subfield of statistical methods.

- **Resampling Methods**. Methods for systematically splitting a dataset into subsets for the purposes of training and evaluating a predictive model.

3.8 Model Configuration

A given machine learning algorithm often has a suite of hyperparameters that allow the learning method to be tailored to a specific problem. The configuration of the hyperparameters is often empirical in nature, rather than analytical, requiring large suites of experiments in order to evaluate the effect of different hyperparameter values on the skill of the model. The interpretation and comparison of the results between different hyperparameter configurations is made using one of two subfields of statistics, namely:

- **Statistical Hypothesis Tests**. Methods that quantify the likelihood of observing the result given an assumption or expectation about the result (presented using critical values and p-values).
- **Estimation Statistics**. Methods that quantify the uncertainty of a result using confidence intervals.

3.9 Model Selection

One among many machine learning algorithms may be appropriate for a given predictive modeling problem. The process of selecting one method as the solution is called model selection. This may involve a suite of criteria both from stakeholders in the project and the careful interpretation of the estimated skill of the methods evaluated for the problem. As with model configuration, two classes of statistical methods can be used to interpret the estimated skill of different models for the purposes of model selection. They are:

- **Statistical Hypothesis Tests**. Methods that quantify the likelihood of observing the result given an assumption or expectation about the result (presented using critical values and p-values).
- **Estimation Statistics**. Methods that quantify the uncertainty of a result using confidence intervals.

3.10 Model Presentation

Once a final model has been trained, it can be presented to stakeholders prior to being used or deployed to make actual predictions on real data. A part of presenting a final model involves presenting the estimated skill of the model. Methods from the field of estimation statistics can be used to quantify the uncertainty in the estimated skill of the machine learning model through the use of tolerance intervals and confidence intervals.

- **Estimation Statistics**. Methods that quantify the uncertainty in the skill of a model via confidence intervals.

3.11 Model Predictions

Finally, it will come time to start using a final model to make predictions for new data where we do not know the real outcome. As part of making predictions, it is important to quantify the confidence of the prediction. Just like with the process of model presentation, we can use methods from the field of estimation statistics to quantify this uncertainty, such as confidence intervals and prediction intervals.

- **Estimation Statistics**. Methods that quantify the uncertainty for a prediction via prediction intervals.

3.12 Summary

In this tutorial, you discovered the importance of statistical methods throughout the process of working through a predictive modeling project. Specifically, you learned:

- Exploratory data analysis, data summarization, and data visualizations can be used to help frame your predictive modeling problem and better understand the data.
- That statistical methods can be used to clean and prepare data ready for modeling.
- That statistical hypothesis tests and estimation statistics can aid in model selection and in presenting the skill and predictions from final models.

3.12.1 Next

This is the end of part II, in the next part you will discover the foundations you need to know in statistics, starting with how to summarize the Gaussian distribution.

Part III

Foundation

Chapter 4

Gaussian and Summary Stats

A sample of data is a snapshot from a broader population of all possible observations that could be taken of a domain or generated by a process. Interestingly, many observations fit a common pattern or distribution called the normal distribution, or more formally, the Gaussian distribution. A lot is known about the Gaussian distribution, and as such, there are whole sub-fields of statistics and statistical methods that can be used with Gaussian data. In this tutorial, you will discover the Gaussian distribution, how to identify it, and how to calculate key summary statistics of data drawn from this distribution. After completing this tutorial, you will know:

- That the Gaussian distribution describes many observations, including many observations seen during applied machine learning.
- That the central tendency of a distribution is the most likely observation and can be estimated from a sample of data as the mean or median.
- That the variance is the average deviation from the mean in a distribution and can be estimated from a sample of data as the variance and standard deviation.

Let's get started.

4.1 Tutorial Overview

This tutorial is divided into 6 parts; they are:

1. Gaussian Distribution
2. Sample vs Population
3. Test Dataset
4. Central Tendencies
5. Variance
6. Describing a Gaussian

4.2 Gaussian Distribution

A distribution of data refers to the shape it has when you graph it, such as with a histogram. The most commonly seen and therefore well-known distribution of continuous values is the bell curve. It is known as the *normal* distribution, because it the distribution that a lot of data falls into. It is also known as the Gaussian distribution, more formally, named for Carl Friedrich Gauss. As such, you will see references to data being normally distributed or Gaussian, which are interchangeable, both referring to the same thing: that the data looks like the Gaussian distribution. Some examples of observations that have a Gaussian distribution include:

- People's heights.
- IQ scores.
- Body temperature.

Let's look at a normal distribution. Below is some code to generate and plot an idealized Gaussian distribution.

```
# generate and plot an idealized gaussian
from numpy import arange
from matplotlib import pyplot
from scipy.stats import norm
# x-axis for the plot
x_axis = arange(-3, 3, 0.001)
# y-axis as the gaussian
y_axis = norm.pdf(x_axis, 0, 1)
# plot data
pyplot.plot(x_axis, y_axis)
pyplot.show()
```

Listing 4.1: Example density line plot of Gaussian probability density function.

Running the example generates a plot of an idealized Gaussian distribution. The x-axis are the observations and the y-axis is the likelihood of each observation. In this case, observations around 0.0 are the most common and observations around -3.0 and 3.0 are rare or unlikely. Technically, this is called a probability density function, covered in more detail in Chapter 10.

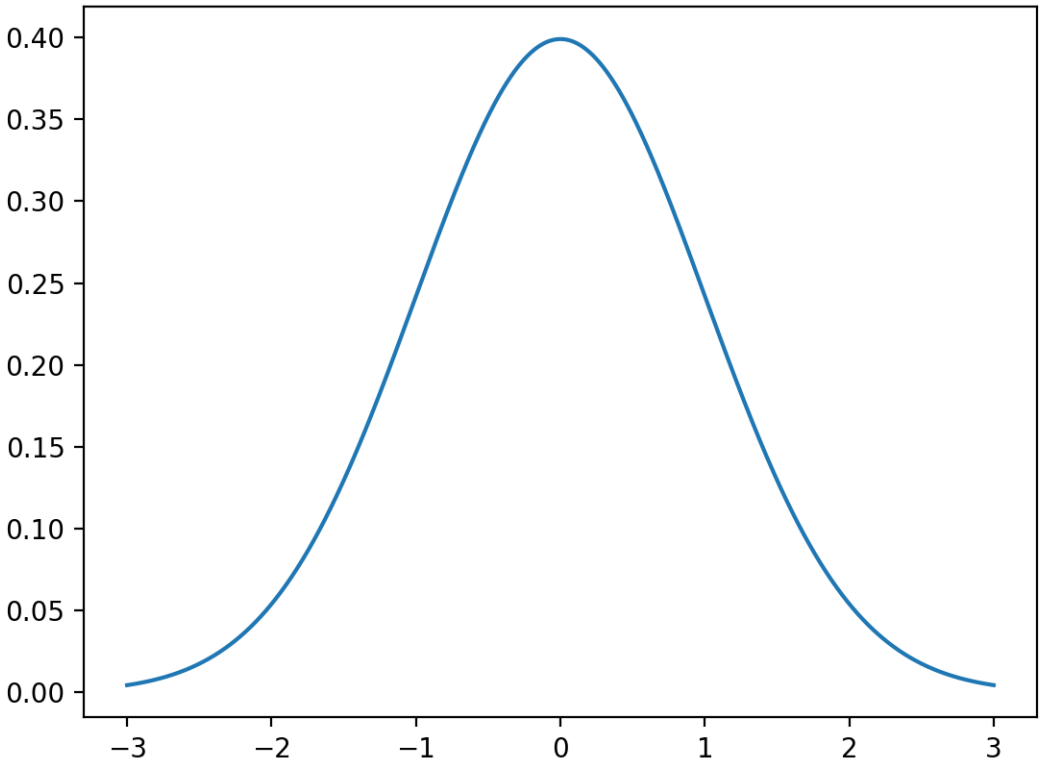

Figure 4.1: Density line plot of the Gaussian probability density function.

It is helpful when data is Gaussian or when we assume a Gaussian distribution for calculating statistics. This is because the Gaussian distribution is very well understood. So much so that large parts of the field of statistics are dedicated to methods for this distribution. Thankfully, much of the data we work with in machine learning often fits a Gaussian distribution. Examples include the input data we may use to fit a model and the multiple evaluations of a model on different samples of training data. Not all data is Gaussian, and it is sometimes important to make this discovery either by reviewing histogram plots of the data or using statistical tests to check. Some examples of observations that do not fit a Gaussian distribution and instead may fit an exponential (hockey-stick shape) include:

- People's incomes.
- Population of cities.
- Sales of books.

A uniform distribution is another common distribution, often seen when each item or value has an equal value for being selected. The shape of a graph of the uniform distribution is a flat line.

4.3 Sample vs Population

We can think of data being generated by some unknown process. The data that we collect is called a data sample, whereas all possible data that could be collected is called the population.

- **Data Sample**: A subset of observations from a group.
- **Data Population**: All possible observations from a group.

This is an important distinction because different statistical methods are used on samples vs populations, and in applied machine learning, we are often working with samples of data. If you read or use the word *population* when talking about data in machine learning, it very likely means sample when it comes to statistical methods.

Two examples of data samples that you will encounter in machine learning include:

- The train and test datasets.
- The performance scores for a model.

When using statistical methods, we often want to make claims about the population using only observations in the sample. Two clear examples of this include:

- The training sample must be representative of the population of observations so that we can fit a useful model.
- The test sample must be representative of the population of observations so that we can develop an unbiased evaluation of the model skill.

Because we are working with samples and making claims about a population, it means that there is always some uncertainty, and it is important to understand and report this uncertainty.

4.4 Test Dataset

Before we explore some important summary statistics for data with a Gaussian distribution, let's first generate a sample of data that we can work with. We can use the `randn()` NumPy function to generate a sample of random numbers drawn from a Gaussian distribution. There are two key parameters that define any Gaussian distribution; they are the mean and the standard deviation. We will go more into these parameters later as they are also key statistics to estimate when we have data drawn from an unknown Gaussian distribution.

The `randn()` function will generate a specified number of random numbers (e.g. 10,000) drawn from a Gaussian distribution with a mean of zero and a standard deviation of 1. We can then scale these numbers to a Gaussian of our choosing by rescaling the numbers. This can be made consistent by adding the desired mean (e.g. 50) and multiplying the value by the standard deviation (5). Note, generating random numbers is covered in greater detail in Chapter 6.

```
...
data = 5 * randn(10000) + 50
```

Listing 4.2: Example of creating a sample of random Gaussian numbers.

We can then plot the dataset using a histogram and look for the expected shape of the plotted data. The complete example is listed below.

```
# generate a sample of random gaussians
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# histogram of generated data
pyplot.hist(data)
pyplot.show()
```

Listing 4.3: Example of calculating and plotting the sample of Gaussian random numbers.

Running the example generates the dataset and plots it as a histogram. Simple plotting is covered in Chapter 5. We can almost see the Gaussian shape to the data, but it is blocky. This highlights an important point. Sometimes, the data will not be a perfect Gaussian, but it will have a Gaussian-like distribution. It is almost Gaussian and maybe it would be more Gaussian if it was plotted in a different way, scaled in some way, or if more data was gathered. Often, when working with Gaussian-like data, we can treat it as Gaussian and use all of the same statistical tools and get reliable results.

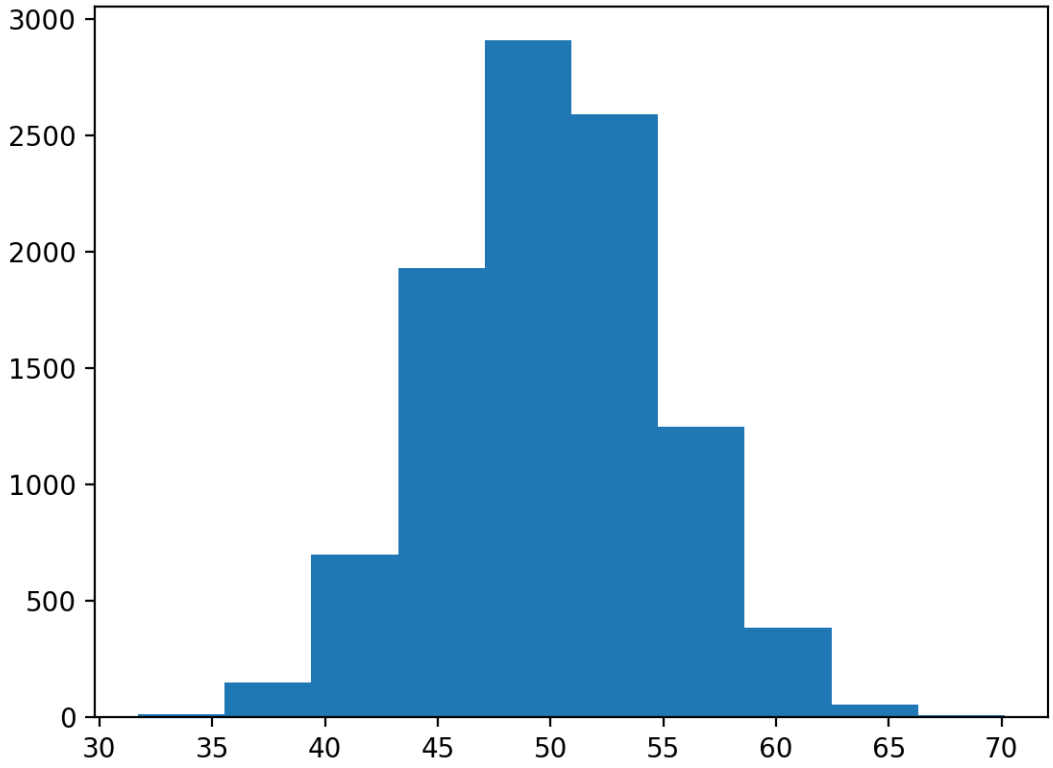

Figure 4.2: Histogram of the sample of random Gaussian numbers.

In the case of this dataset, we do have enough data and the plot is blocky because the plotting function chooses an arbitrary sized bucket for splitting up the data. We can choose a different, more granular way to split up the data and better expose the underlying Gaussian distribution. The updated example with the more refined plot is listed below.

```
# generate a sample of random gaussians
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# histogram of generated data
pyplot.hist(data, bins=100)
pyplot.show()
```

Listing 4.4: Example of calculating and plotting the sample of Gaussian random numbers with more bins.

Running the example, we can see that choosing 100 splits of the data does a much better job of creating a plot that clearly shows the Gaussian distribution of the data. The dataset was generated from a perfect Gaussian, but the numbers were randomly chosen and we only

chose 10,000 observations for our sample. You can see, even with this controlled setup, there is obvious noise in the data sample. This highlights another important point: that we should always expect some noise or limitation in our data sample. The data sample will always contain errors compared to the pure underlying distribution.

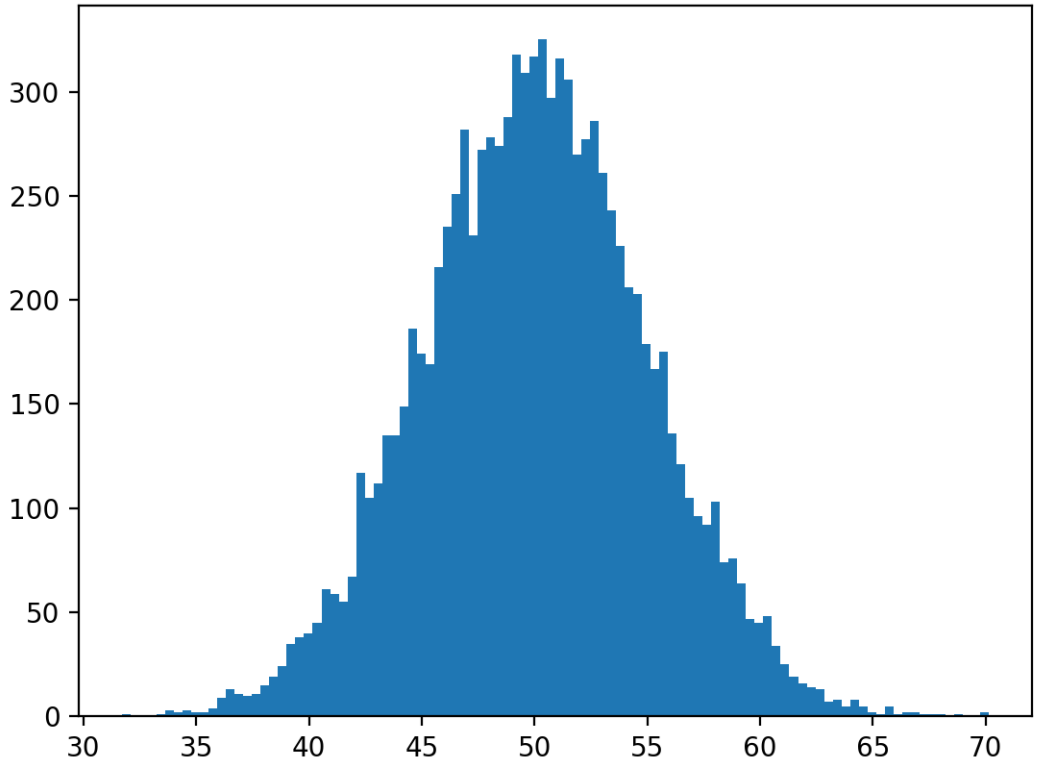

Figure 4.3: Histogram with more bins for the sample of random Gaussian numbers.

4.5 Central Tendency

The central tendency of a distribution refers to the middle or typical value in the distribution. The most common or most likely value. In the Gaussian distribution, the central tendency is called the mean, or more formally, the arithmetic mean, and is one of the two main parameters that defines any Gaussian distribution. The mean of a sample is calculated as the sum of the observations divided by the total number of observations in the sample.

$$mean(x) = \frac{\sum_{i=1}^{n} x_i}{n} \tag{4.1}$$

Where x_i is the i^{th} observation from the dataset and n is the total number of observations.

It is also written in a more compact form as:

$$mean(x) = \frac{1}{n} \times \sum_{i=1}^{n} x_i \tag{4.2}$$

The notation for the population mean is the Greek lower case letter mu (μ). The notation for the sample mean is the variable with a bar above, such as x-bar ($\bar{x}$). We can calculate the mean of a sample by using the `mean()` NumPy function on an array.

```
...
result = mean(data)
```

Listing 4.5: Example of calculating the mean in NumPy.

The example below demonstrates this on the test dataset developed in the previous section.

```
# calculate the mean of a sample
from numpy.random import seed
from numpy.random import randn
from numpy import mean
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# calculate mean
result = mean(data)
print('Mean: %.3f' % result)
```

Listing 4.6: Example of calculating the mean of a data sample.

Running the example calculates and prints the mean of the sample. This calculation of the arithmetic mean of the sample is an estimate of the parameter of the underlying Gaussian distribution of the population from which the sample was drawn. As an estimate, it will contain errors. Because we know the underlying distribution has the true mean of 50, we can see that the estimate from a sample of 10,000 observations is reasonably accurate.

```
Mean: 50.049
```

Listing 4.7: Sample output from calculating the mean of a data sample.

The mean is easily influenced by outlier values, that is, rare values far from the mean. These may be legitimately rare observations on the edge of the distribution or errors. Further, the mean may be misleading. Calculating a mean on another distribution, such as a uniform distribution or power distribution, may not make a lot of sense as although the value can be calculated, it will refer to a seemingly arbitrary expected value rather than the true central tendency of the distribution.

In the case of outliers or a non-Gaussian distribution, an alternate and commonly used central tendency to calculate is the median. The median is calculated by first sorting all data and then locating the middle value in the sample. This is straightforward if there is an odd number of observations. If there is an even number of observations, the median is calculated as the average of the middle two observations. We can calculate the median of a sample of an array by calling the `median()` NumPy function.

```
...
result = median(data)
```

Listing 4.8: Example of calculating the median in NumPy.

The example below demonstrates this on the test dataset.

```
# calculate the median of a sample
from numpy.random import seed
from numpy.random import randn
from numpy import median
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# calculate median
result = median(data)
print('Median: %.3f' % result)
```

Listing 4.9: Example of calculating the median of a data sample.

Running the example, we can see that median is calculated from the sample and printed. The result is not too dissimilar from the mean because the sample has a Gaussian distribution. If the data had a different (non-Gaussian) distribution, the median may be very different from the mean and perhaps a better reflection of the central tendency of the underlying population.

```
Median: 50.042
```

Listing 4.10: Sample output from calculating the median of a data sample.

4.6 Variance

The variance of a distribution refers to how much on average that observations vary or differ from the mean value. It is useful to think of the variance as a measure of the spread of a distribution. A low variance will have values grouped around the mean (e.g. a narrow bell shape), whereas a high variance will have values spread out from the mean (e.g. a wide bell shape.) We can demonstrate this with an example, by plotting idealized Gaussians with low and high variance. The complete example is listed below.

```
# generate and plot gaussians with different variance
from numpy import arange
from matplotlib import pyplot
from scipy.stats import norm
# x-axis for the plot
x_axis = arange(-3, 3, 0.001)
# plot low variance
pyplot.plot(x_axis, norm.pdf(x_axis, 0, 0.5))
# plot high variance
pyplot.plot(x_axis, norm.pdf(x_axis, 0, 1))
pyplot.show()
```

Listing 4.11: Example of generating and plotting samples of random Gaussian numbers with different variances.

Running the example plots two idealized Gaussian distributions: the blue with a low variance grouped around the mean and the orange with a higher variance with more spread.

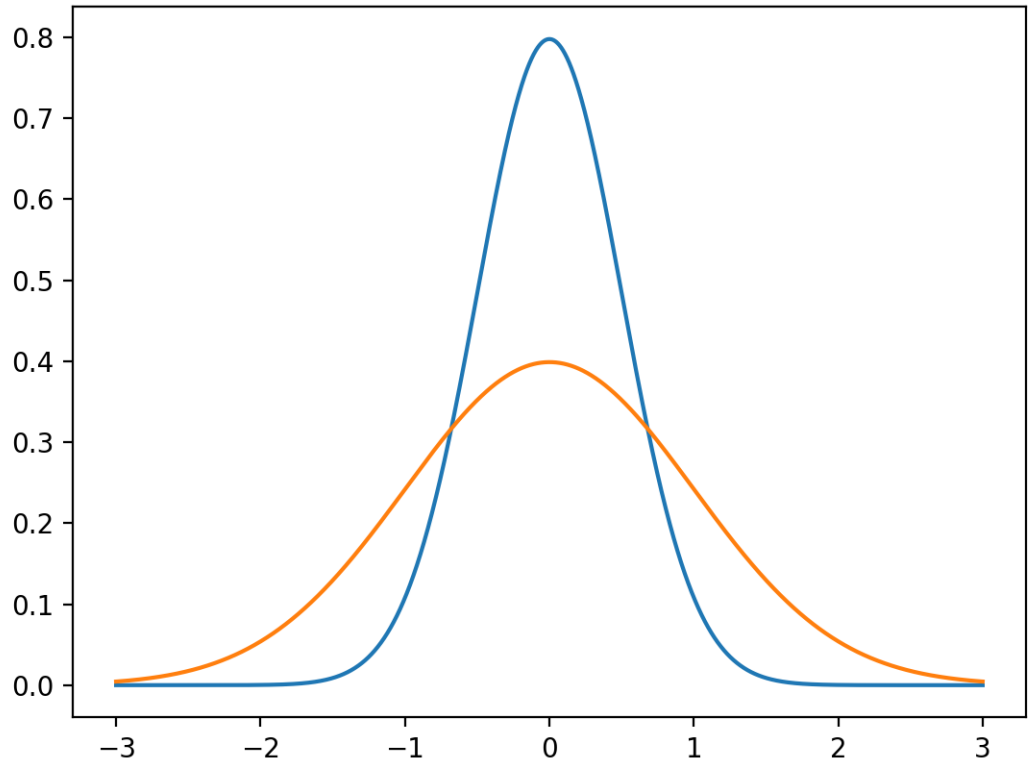

Figure 4.4: Line plots of Gaussian distributions with different variances.

The variance of a data sample drawn from a Gaussian distribution is calculated as the average squared difference of each observation in the sample from the sample mean:

$$variance(x) = \frac{1}{n-1} \times \sum_{i=1}^{n} (x_i - mean(x))^2 \tag{4.3}$$

Where *variance* is often denoted as s^2 or the lowercase Greek letter sigma σ^2, clearly showing the squared units of the measure. You may see the equation without the (-1) from the number of observations, and this is the calculation of the variance for the population, not the sample. We can calculate the variance of a data sample in NumPy using the `var()` function. The example below demonstrates calculating variance on the test problem.

```
# calculate the variance of a sample
from numpy.random import seed
from numpy.random import randn
from numpy import var
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# calculate variance
result = var(data)
```

```
print('Variance: %.3f' % result)
```

Listing 4.12: Example of calculating the variance of a data sample.

Running the example calculates and prints the variance.

```
Variance: 24.939
```

Listing 4.13: Sample output from calculating the variance of a data sample.

It is hard to interpret the variance because the units are the squared units of the observations. We can return the units to the original units of the observations by taking the square root of the result. For example, the square root of 24.939 is about 4.9. Often, when the spread of a Gaussian distribution is summarized, it is described using the square root of the variance. This is called the standard deviation. The standard deviation, along with the mean, are the two key parameters required to specify any Gaussian distribution.

We can see that the value of 4.9 is very close to the value of 5 for the standard deviation specified when the samples were created for the test problem. We can wrap the variance calculation in a square root to calculate the standard deviation directly.

$$stdev(x) = \sqrt{\frac{1}{n-1} \times \sum_{i=1}^{n}(x_i - mean(x))^2} \tag{4.4}$$

Or:

$$stdev(x) = \sqrt{variance(x)} \tag{4.5}$$

Where the standard deviation is often written as s or as the Greek lowercase letter sigma (σ). The standard deviation can be calculated directly in NumPy for an array via the `std()` function. The example below demonstrates the calculation of the standard deviation on the test problem.

```
# calculate the standard deviation of a sample
from numpy.random import seed
from numpy.random import randn
from numpy import std
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# calculate standard deviation
result = std(data)
print('Standard Deviation: %.3f' % result)
```

Listing 4.14: Example of calculating the standard deviation of a data sample.

Running the example calculates and prints the standard deviation of the sample. The value matches the square root of the variance and is very close to 5.0, the value specified in the definition of the problem.

```
Standard Deviation: 4.994
```

Listing 4.15: Sample output from calculating the standard deviation of a data sample.

Measures of variance can be calculated for non-Gaussian distributions, but generally require the distribution to be identified so that a specialized measure of variance specific to that distribution can be calculated.

4.7 Describing a Gaussian

In applied machine learning, you will often need to report the results of an algorithm. That is, report the estimated skill of the model on out-of-sample data. This is often done by reporting the mean performance from a k-fold cross-validation, or some other repeated sampling procedure. When reporting model skill, you are in effect summarizing the distribution of skill scores, and very likely the skill scores will be drawn from a Gaussian distribution.

It is common to only report the mean performance of the model. This would hide two other important details of the distribution of the skill of the model. As a minimum I would recommend reporting the two parameters of the Gaussian distribution of model scores and the size of the sample. Ideally, it would also be a good idea to confirm that indeed the model skill scores are Gaussian or look Gaussian enough to defend reporting the parameters of the Gaussian distribution. This is important because the distribution of skill scores can be reconstructed by readers and potentially compared to the skill of models on the same problem in the future.

4.8 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Develop your own test problem and calculate the central tendency and variance measures.
- Develop a function to calculate a summary report on a given data sample.
- Load and summarize the variables for a standard machine learning dataset.

If you explore any of these extensions, I'd love to know.

4.9 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

4.9.1 APIs

- `scipy.stats.norm` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html
- `numpy.random.seed` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.seed.html
- `numpy.random.randn` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.randn.html

- `matplotlib.pyplot.hist` API.
  https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html
- `numpy.mean` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html
- `numpy.median` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.median.html
- `numpy.var` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.var.html
- `numpy.std` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.std.html

4.9.2 Articles

- Normal distribution on Wikipedia.
  https://en.wikipedia.org/wiki/Normal_distribution
- Central tendency on Wikipedia.
  https://en.wikipedia.org/wiki/Central_tendency
- Arithmetic mean on Wikipedia.
  https://en.wikipedia.org/wiki/Arithmetic_mean
- Median on Wikipedia.
  https://en.wikipedia.org/wiki/Median
- Variance on Wikipedia.
  https://en.wikipedia.org/wiki/Variance
- Standard deviation on Wikipedia.
  https://en.wikipedia.org/wiki/Standard_deviation

4.10 Summary

In this tutorial, you discovered the Gaussian distribution, how to identify it, and how to calculate key summary statistics of data drawn from this distribution. Specifically, you learned:

- That the Gaussian distribution describes many observations, including many observations seen during applied machine learning.
- That the central tendency of a distribution is the most likely observation and can be estimated from a sample of data as the mean or median.
- That the variance is the average deviation from the mean in a distribution and can be estimated from a sample of data as the variance and standard deviation.

4.10.1 Next

In the next section, you will discover how to visualize data using simple charts and graphs.

Chapter 5

Simple Data Visualization

Sometimes data does not make sense until you can look at in a visual form, such as with charts and plots. Being able to quickly visualize your data samples for yourself and others is an important skill both in applied statistics and in applied machine learning. In this tutorial, you will discover the five types of plots that you will need to know when visualizing data in Python and how to use them to better understand your own data. After completing this tutorial, you will know:

- How to chart time series data with line plots and categorical quantities with bar charts.
- How to summarize data distributions with histograms and box plots.
- How to summarize the relationship between variables with scatter plots.

Let's get started.

5.1 Tutorial Overview

This tutorial is divided into 7 parts; they are:

1. Data Visualization
2. Introduction to Matplotlib
3. Line Plot
4. Bar Chart
5. Histogram Plot
6. Box and Whisker Plot
7. Scatter Plot

5.2 Data Visualization

Data visualization is an important skill in applied statistics and machine learning. Statistics does indeed focus on quantitative descriptions and estimations of data. Data visualization provides an important suite of tools for gaining a qualitative understanding. This can be helpful when exploring and getting to know a dataset and can help with identifying patterns, corrupt data, outliers, and much more. With a little domain knowledge, data visualizations can be used to express and demonstrate key relationships in plots and charts that are more visceral to yourself and stakeholders than measures of association or significance.

Data visualization and exploratory data analysis are whole fields themselves and I will recommend a deeper dive into some the books mentioned at the end. In this tutorial, let's look at basic charts and plots you can use to better understand your data. There are five key plots that you need to know well for basic data visualization. They are the Line Plot, Bar Chart, Histogram Plot, Box and Whisker Plot, and the Scatter Plot. With a knowledge of these plots, you can quickly get a qualitative understanding of most data that you come across. For the rest of this tutorial, we will take a closer look at each plot type.

5.3 Introduction to Matplotlib

There are many excellent plotting libraries in Python and I recommend exploring them in order to create presentable graphics. For quick and dirty plots intended for your own use, I recommend using the Matplotlib library. It is the foundation for many other plotting libraries and plotting support in higher-level libraries such as Pandas. The Matplotlib provides a context, one in which one or more plots can be drawn before the image is shown or saved to file. The context can be accessed via functions on `pyplot`. The context can be imported as follows:

```
# import matplotlib context
from matplotlib import pyplot
...
```

Listing 5.1: Example of importing the Matplotlib context.

There is some convention to import this context and name it `plt`; for example:

```
# import matplotlib context
import matplotlib.pyplot as plt
...
```

Listing 5.2: Example of alternate way of importing the Matplotlib context.

We will not use this convention, instead we will stick to the standard Python import convention. Charts and plots are made by making and calling on context; for example:

```
...
# create a plot
pyplot.plot(...)
```

Listing 5.3: Example of creating a plot.

Elements such as axis, labels, legends, and so on can be accessed and configured on this context as separate function calls. The drawings on the context can be shown in a new window by calling the `show()` function:

```
...
# display the plot
pyplot.show()
```

Listing 5.4: Example of displaying a plot.

Alternately, the drawings on the context can be saved to file, such as a PNG formatted image file. The `savefig()` function can be used to save images.

```
...
# save plot to file
pyplot.savefig('my_image.png')
```

Listing 5.5: Example of saving a plot to file.

This is the most basic crash course for using the Matplotlib library. For more detail, see the User Guide and other resources at the end of the tutorial.

5.4 Line Plot

A line plot is generally used to present observations collected at regular intervals. The x-axis represents the regular interval, such as time. The y-axis shows the observations, ordered by the x-axis and connected by a line. A line plot can be created by calling the `plot()` function and passing the x-axis data for the regular interval, and y-axis for the observations.

```
...
# create line plot
pyplot.plot(x, y)
```

Listing 5.6: Example of creating a line plot.

Line plots are useful for presenting time series data as well as any sequence data where there is an ordering between observations. The example below creates a sequence of 100 floating point values as the x-axis and a sine wave as a function of the x-axis as the observations on the y-axis. The results are plotted as a line plot.

```
# example of a line plot
from numpy import sin
from matplotlib import pyplot
# consistent interval for x-axis
x = [x*0.1 for x in range(100)]
# function of x for y-axis
y = sin(x)
# create line plot
pyplot.plot(x, y)
# show line plot
pyplot.show()
```

Listing 5.7: Example creating a line plot from data.

Running the example creates a line plot showing the familiar sine wave pattern on the y-axis across the x-axis with a consistent interval between observations.

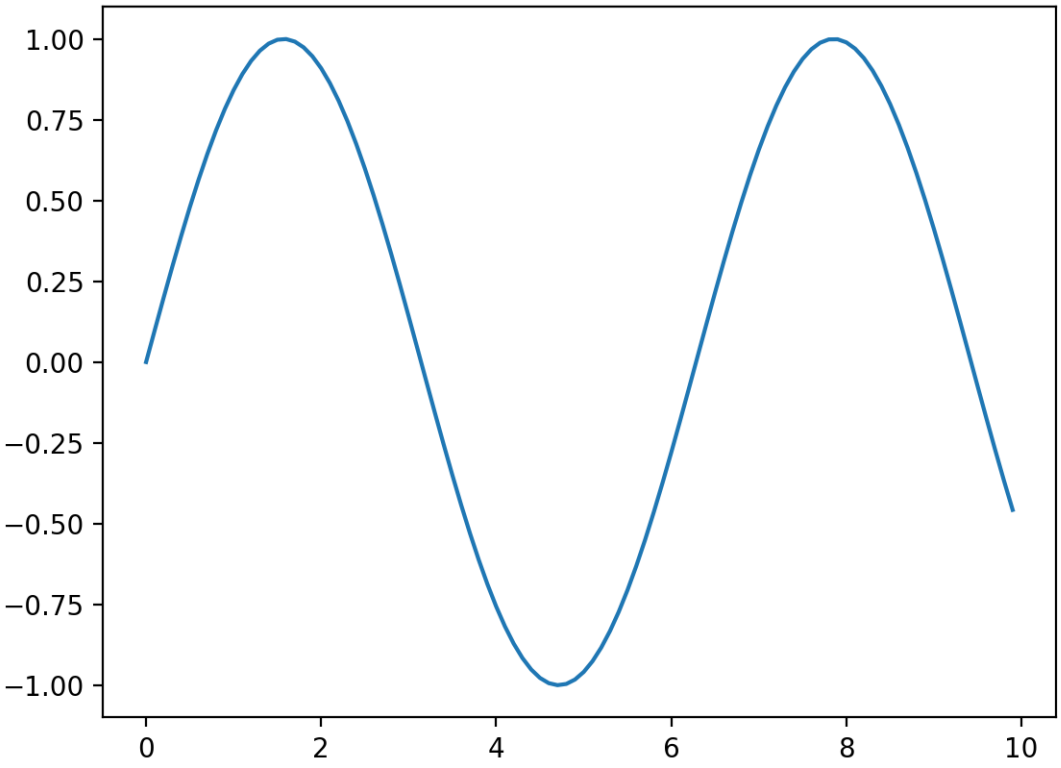

Figure 5.1: Example of a line plot of data.

5.5 Bar Chart

A bar chart is generally used to present relative quantities for multiple categories. The x-axis represents the categories and are spaced evenly. The y-axis represents the quantity for each category and is drawn as a bar from the baseline to the appropriate level on the y-axis. A bar chart can be created by calling the `bar()` function and passing the category names for the x-axis and the quantities for the y-axis.

```
...
# create bar chart
pyplot.bar(x, y)
```

Listing 5.8: Example of creating a bar chart.

Bar charts can be useful for comparing multiple point quantities or estimations. The example below creates a dataset with three categories, each defined with a string label. A single random integer value is drawn for the quantity in each category.

```
# example of a bar chart
from random import seed
from random import randint
from matplotlib import pyplot
```

```
# seed the random number generator
seed(1)
# names for categories
x = ['red', 'green', 'blue']
# quantities for each category
y = [randint(0, 100), randint(0, 100), randint(0, 100)]
# create bar chart
pyplot.bar(x, y)
# show line plot
pyplot.show()
```

Listing 5.9: Example creating a bar chart from data.

Running the example creates the bar chart showing the category labels on the x-axis and the quantities on the y-axis.

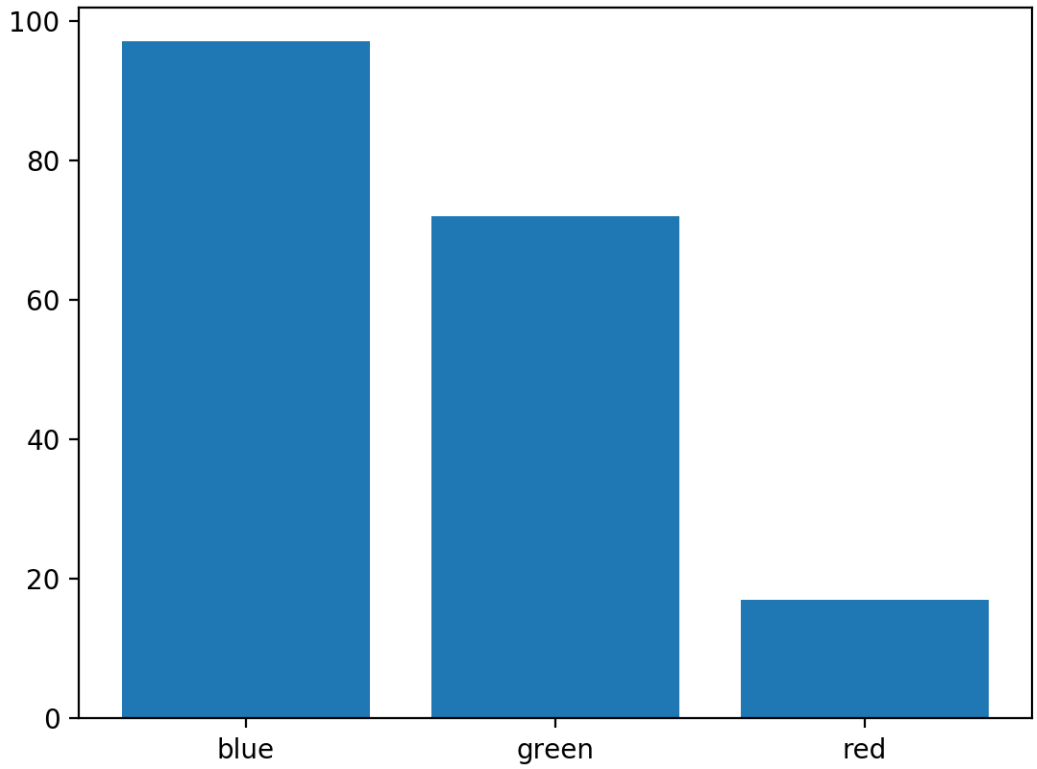

Figure 5.2: Example of a bar chart of data.

5.6 Histogram Plot

A histogram plot is generally used to summarize the distribution of a data sample. The x-axis represents discrete bins or intervals for the observations. For example observations with values between 1 and 10 may be split into five bins, the values [1,2] would be allocated to the first bin,

[3,4] would be allocated to the second bin, and so on. The y-axis represents the frequency or count of the number of observations in the dataset that belong to each bin. Essentially, a data sample is transformed into a bar chart where each category on the x-axis represents an interval of observation values.

> Histograms are density estimates. A density estimate gives a good impression of the distribution of the data.[...] The idea is to locally represent the data density by counting the number of observations in a sequence of consecutive intervals (bins) ...
>
> — Page 11, *Applied Multivariate Statistical Analysis*, 2015.

A histogram plot can be created by calling the `hist()` function and passing in a list or array that represents the data sample.

```
...
# create histogram plot
pyplot.hist(x)
```

Listing 5.10: Example of creating a histogram plot.

Histograms are valuable for summarizing the distribution of data samples. The example below creates a dataset of 1,000 random numbers drawn from a standard Gaussian distribution, then plots the dataset as a histogram.

```
# example of a histogram plot
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# random numbers drawn from a Gaussian distribution
x = randn(1000)
# create histogram plot
pyplot.hist(x)
# show line plot
pyplot.show()
```

Listing 5.11: Example creating a histogram plot from data.

Running the example, we can see that the shape of the bars shows the bell-shaped curve of the Gaussian distribution. We can see that the function automatically chose the number of bins, in this case splitting the values into groups by integer value.

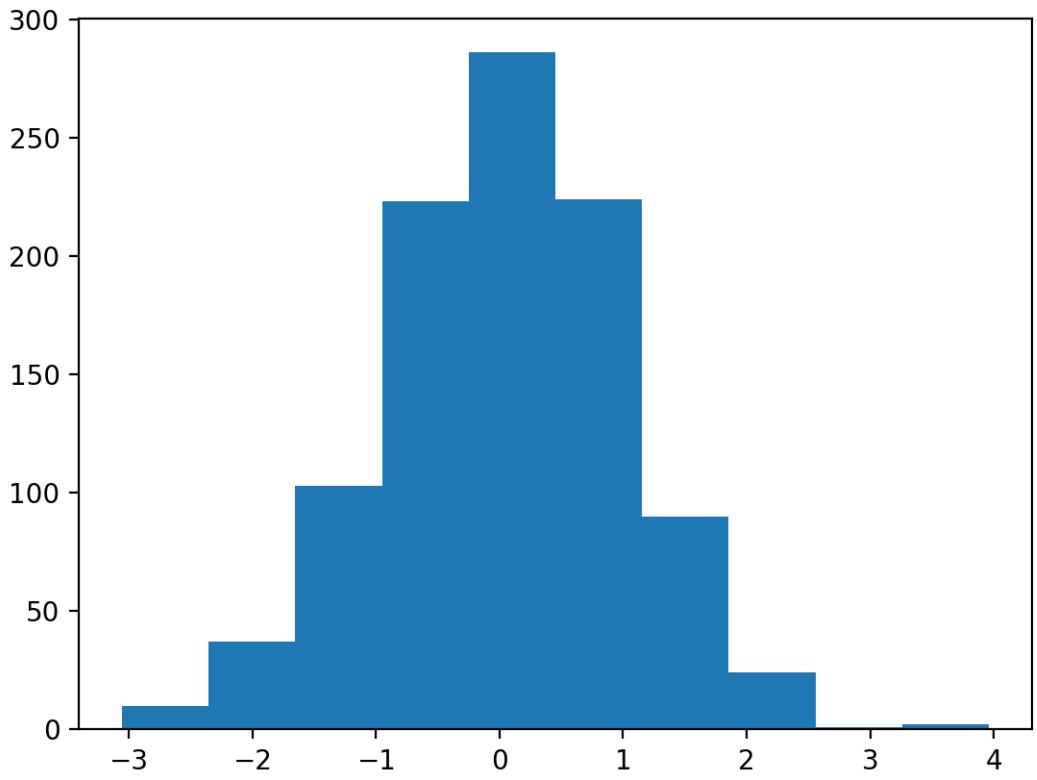

Figure 5.3: Example of a histogram plot of data.

Often, careful choice of the number of bins can help to better expose the shape of the data distribution. The number of bins can be specified by setting the `bins` argument; for example:

```
...
# create histogram plot with specified bins
pyplot.hist(x, bins=100)
```

Listing 5.12: Example of creating a histogram plot with a specific number of bins.

5.7 Box and Whisker Plot

A box and whisker plot, or boxplot for short, is generally used to summarize the distribution of a data sample. The x-axis is used to represent the data sample, where multiple boxplots can be drawn side by side on the x-axis if desired.

The y-axis represents the observation values. A box is drawn to summarize the middle 50% of the dataset starting at the observation at the 25th percentile and ending at the 75th percentile. The median, or 50th percentile, is drawn with a line. A value called the interquartile range, or IQR, is calculated as 1.5 * the difference between the 75th and 25th percentiles. Lines called whiskers are drawn extending from both ends of the box with the length of the IQR to

demonstrate the expected range of sensible values in the distribution. Observations outside the whiskers might be outliers and are drawn with small circles.

> The boxplot is a graphical technique that displays the distribution of variables. It helps us see the location, skewness, spread, tile length and outlying points. [...] The boxplot is a graphical representation of the Five Number Summary.
>
> — Page 5, *Applied Multivariate Statistical Analysis*, 2015.

Boxplots can be drawn by calling the `boxplot()` function passing in the data sample as an array or list.

```
...
# create box and whisker plot
pyplot.boxplot(x)
```

Listing 5.13: Example of creating a box and whisker plot.

Boxplots are useful to summarize the distribution of a data sample as an alternative to the histogram. They can help to quickly get an idea of the range of common and sensible values in the box and in the whisker respectively. Because we are not looking at the shape of the distribution explicitly, this method is often used when the data has an unknown or unusual distribution, such as non-Gaussian.

The example below creates three boxplots in one chart, each summarizing a data sample drawn from a slightly different Gaussian distribution. Each data sample is created as an array and all three data sample arrays are added to a list that is padded to the plotting function.

```
# example of a box and whisker plot
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# random numbers drawn from a Gaussian distribution
x = [randn(1000), 5 * randn(1000), 10 * randn(1000)]
# create box and whisker plot
pyplot.boxplot(x)
# show line plot
pyplot.show()
```

Listing 5.14: Example creating a box and whisker plot from data.

Running the example creates a chart showing the three box and whisker plots. We can see that the same scale is used on the y-axis for each, making the first plot look squashed and the last plot look spread out. In this case, we can see the black box for the middle 50% of the data, the orange line for the median, the lines for the whiskers summarizing the range of sensible data, and finally dots for the possible outliers.

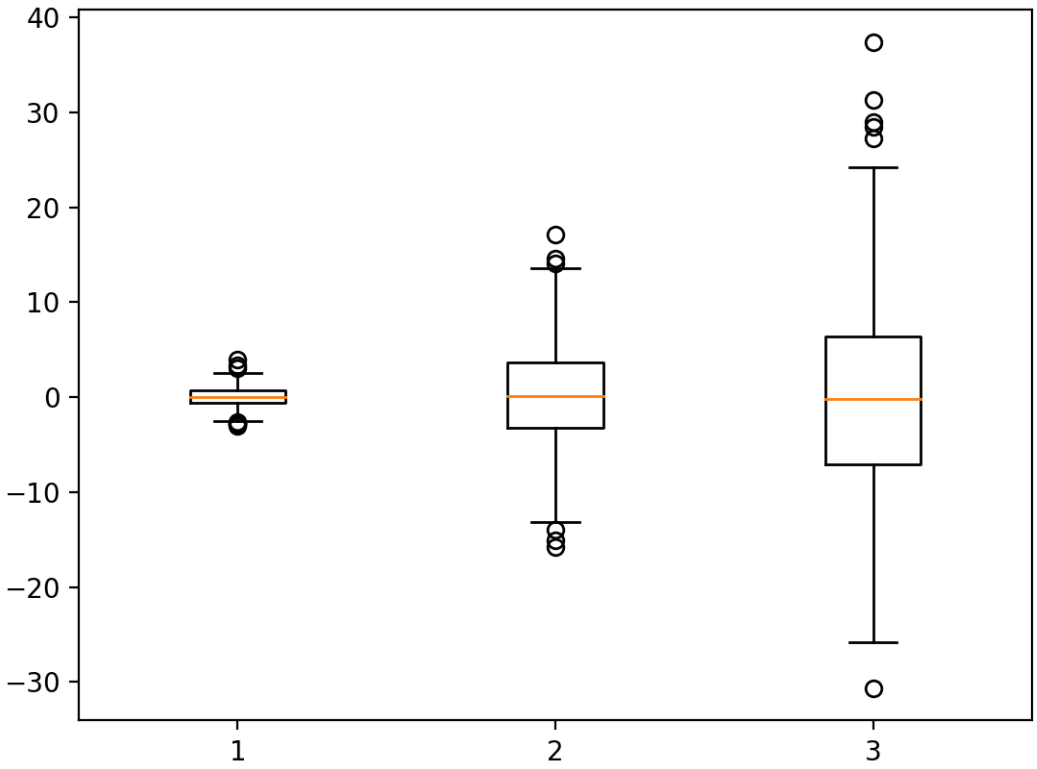

Figure 5.4: Example of a box and whisker plot of data.

5.8 Scatter Plot

A scatter plot, or scatterplot, is generally used to summarize the relationship between two paired data samples. Paired data samples means that two measures were recorded for a given observation, such as the weight and height of a person. The x-axis represents observation values for the first sample, and the y-axis represents the observation values for the second sample. Each point on the plot represents a single observation.

> Scatterplots are bivariate or trivariate plots of variables against each other. They help us understand relationships among the variables of a dataset. A downward-sloping scatter indicates that as we increase the variable on the horizontal axis, the variable on the vertical axis decreases.
>
> — Page 19, *Applied Multivariate Statistical Analysis*, 2015.

Scatter plots can be created by calling the `scatter()` function and passing the two data sample arrays.

```
...
# create scatter plot
```

```
pyplot.scatter(x, y)
```

Listing 5.15: Example of creating a scatter plot.

Scatter plots are useful for showing the association or correlation between two variables. A correlation can be quantified, such as a line of best fit, that too can be drawn as a line plot on the same chart, making the relationship clearer. A dataset may have more than two measures (variables or columns) for a given observation. A scatter plot matrix is a cart containing scatter plots for each pair of variables in a dataset with more than two variables. The example below creates two data samples that are related. The first is a sample of random numbers drawn from a standard Gaussian. The second is dependent upon the first by adding a second random Gaussian value to the value of the first measure.

```
# example of a scatter plot
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# first variable
x = 20 * randn(1000) + 100
# second variable
y = x + (10 * randn(1000) + 50)
# create scatter plot
pyplot.scatter(x, y)
# show line plot
pyplot.show()
```

Listing 5.16: Example creating a scatter plot from data.

Running the example creates the scatter plot, showing the positive relationship between the two variables. We will learn more about describing the relationships between variables in Chapter 12.

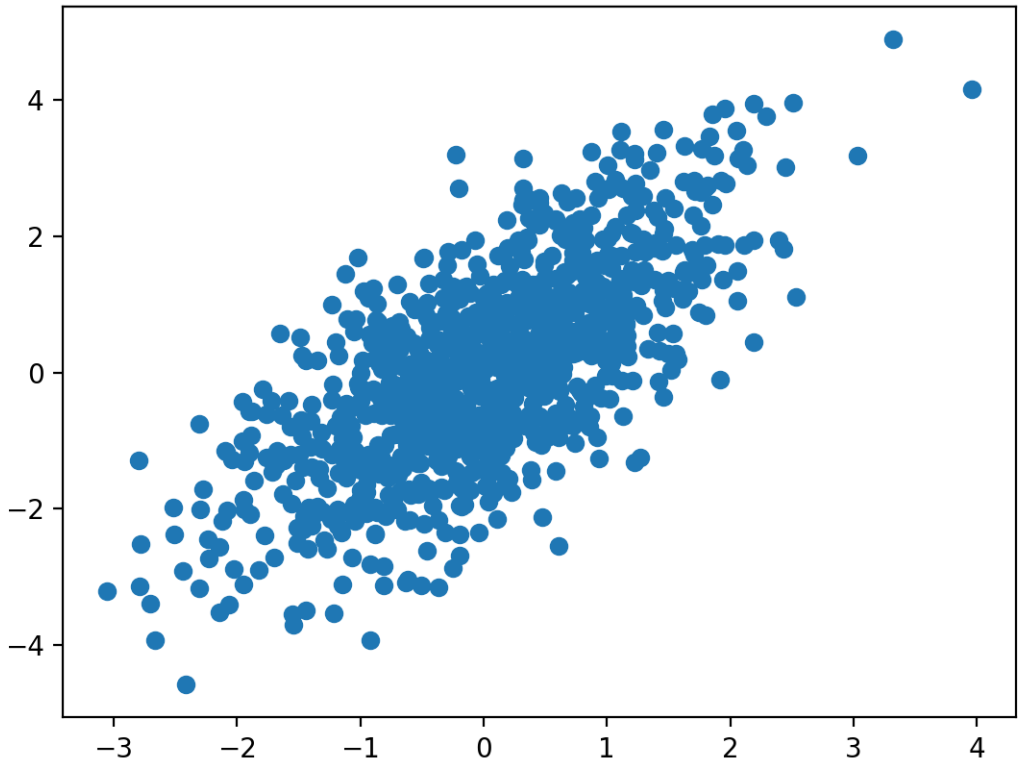

Figure 5.5: Example of a scatter plot of data.

5.9 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Select one example and update it to use your own contrived dataset.
- Load a standard machine learning dataset and plot the variables.
- Write convenience functions to easily create plots for your data, including labels and legends.

If you explore any of these extensions, I'd love to know.

5.10 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

5.10.1 Books

- *The Visual Display of Quantitative Information*, 2001.
  http://amzn.to/2pbC14o
- *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*, 2017.
  http://amzn.to/2Gt8Pgt
- *Applied Multivariate Statistical Analysis*, 2015.
  http://amzn.to/2Dtykv7

5.10.2 API

- Matplotlib library.
  https://matplotlib.org/
- Matplotlib User Guide.
  https://matplotlib.org/users/index.html
- `matplotlib.pyplot` API.
  https://matplotlib.org/api/pyplot_api.html
- `matplotlib.pyplot.show` API.
  https://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html
- `matplotlib.pyplot.savefig` API.
  https://matplotlib.org/api/_as_gen/matplotlib.pyplot.savefig.html
- `matplotlib.pyplot.plot` API.
  https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html
- `matplotlib.pyplot.bar` API.
  https://matplotlib.org/api/_as_gen/matplotlib.pyplot.bar.html
- `matplotlib.pyplot.hist` API.
  https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html
- `matplotlib.pyplot.boxplot` API.
  https://matplotlib.org/api/_as_gen/matplotlib.pyplot.boxplot.html
- `matplotlib.pyplot.scatter` API.
  https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html

5.10.3 Articles

- Data visualization on Wikipedia.
  https://en.wikipedia.org/wiki/Data_visualization
- Bar chart on Wikipedia.
  https://en.wikipedia.org/wiki/Bar_chart

- Histogram on Wikipedia.
  https://en.wikipedia.org/wiki/Histogram
- Box plot on Wikipedia.
  https://en.wikipedia.org/wiki/Box_plot
- Interquartile range on Wikipedia.
  https://en.wikipedia.org/wiki/Interquartile_range
- Scatter plot on Wikipedia.
  https://en.wikipedia.org/wiki/Scatter_plot

5.11 Summary

In this tutorial, you discovered a gentle introduction to visualization data in Python. Specifically, you learned:

- How to chart time series data with line plots and categorical quantities with bar charts.
- How to summarize data distributions with histograms and boxplots.
- How to summarize the relationship between variables with scatter plots.

5.11.1 Next

In the next section, you will discover how to generate samples of random numbers using Python and NumPy.

Chapter 6

Random Numbers

Randomness is a big part of machine learning. Randomness is used as a tool or a feature in preparing data and in learning algorithms that map input data to output data in order to make predictions. In order to understand the need for statistical methods in machine learning, you must understand the source of randomness in machine learning. The source of randomness in machine learning is a mathematical trick called a pseudorandom number generator. In this tutorial, you will discover pseudorandom number generators and when to control and control-for randomness in machine learning. After completing this tutorial, you will know:

- The sources of randomness in applied machine learning with a focus on algorithms.
- What a pseudorandom number generator is and how to use them in Python.
- When to control the sequence of random numbers and when to control-for randomness.

Let's get started.

6.1 Tutorial Overview

This tutorial is divided into 7 parts; they are:

1. Randomness in Machine Learning
2. Pseudorandom Number Generators
3. Random Numbers with Python
4. Random Numbers with NumPy
5. When to Seed the Random Number Generator
6. How to Control for Randomness
7. Common Questions

6.2 Randomness in Machine Learning

There are many sources of randomness in applied machine learning. Randomness is used as a tool to help the learning algorithms be more robust and ultimately result in better predictions and more accurate models. Let's look at a few sources of randomness.

6.2.1 Randomness in Data

There is a random element to the sample of data that we have collected from the domain that we will use to train and evaluate the model. The data may have mistakes or errors. More deeply, the data contains noise that can obscure the crystal-clear relationship between the inputs and the outputs.

6.2.2 Randomness in Evaluation

We do not have access to all the observations from the domain. We work with only a small sample of the data. Therefore, we harness randomness when evaluating a model, such as using k-fold cross-validation to fit and evaluate the model on different subsets of the available dataset. We do this to see how the model works on average rather than on a specific set of data.

6.2.3 Randomness in Algorithms

Machine learning algorithms use randomness when learning from a sample of data. This is a feature, where the randomness allows the algorithm to achieve a better performing mapping of the data than if randomness was not used. Randomness is a feature, which allows an algorithm to attempt to avoid overfitting the small training set and generalize to the broader problem.

Algorithms that use randomness are often called stochastic algorithms rather than random algorithms. This is because although randomness is used, the resulting model is limited to a more narrow range, e.g. like limited randomness. Some clear examples of randomness used in machine learning algorithms include:

- The shuffling of training data prior to each training epoch in stochastic gradient descent.
- The random subset of input features chosen for split points in a random forest algorithm.
- The random initial weights in an artificial neural network.

We can see that there are both sources of randomness that we must control-for, such as noise in the data, and sources of randomness that we have some control over, such as algorithm evaluation and the algorithms themselves. Next, let's look at the source of randomness that we use in our algorithms and programs.

6.3 Pseudorandom Number Generators

The source of randomness that we inject into our programs and algorithms is a mathematical trick called a pseudorandom number generator. A random number generator is a system that generates random numbers from a true source of randomness. Often something physical, such

as a Geiger counter, where the results are turned into random numbers. There are even books of random numbers generated from a physical source that you can purchase, for example: *A Million Random Digits with 100,000 Normal Deviates*[1].

We do not need true randomness in machine learning. Instead we can use pseudorandomness. Pseudorandomness is a sample of numbers that look close to random, but were generated using a deterministic process. Shuffling data and initializing coefficients with random values use pseudorandom number generators. These little programs are often a function that you can call that will return a random number. Called again, they will return a new random number. Wrapper functions are often also available and allow you to get your randomness as an integer, floating point, within a specific distribution, within a specific range, and so on.

The numbers are generated in a sequence. The sequence is deterministic and is seeded with an initial number. If you do not explicitly seed the pseudorandom number generator, then it may use the current system time in seconds or milliseconds as the seed. The value of the seed does not matter. Choose anything you wish. What does matter is that the same seeding of the process will result in the same sequence of random numbers. Let's make this concrete with some examples.

6.4 Random Numbers with Python

The Python standard library provides a module called `random` that offers a suite of functions for generating random numbers. Python uses a popular and robust pseudorandom number generator called the Mersenne Twister. In this section, we will look at a number of use cases for generating and using random numbers and randomness with the standard Python API.

6.4.1 Seed The Random Number Generator

The pseudorandom number generator is a mathematical function that generates a sequence of nearly random numbers. It takes a parameter to start off the sequence, called the seed. The function is deterministic, meaning given the same seed, it will produce the same sequence of numbers every time. The choice of seed does not matter. The `seed()` function will seed the pseudorandom number generator, taking an integer value as an argument, such as 1 or 7. If the `seed()` function is not called prior to using randomness, the default is to use the current system time in milliseconds from epoch (1970). The example below demonstrates seeding the pseudorandom number generator, generates some random numbers, and shows that reseeding the generator will result in the same sequence of numbers being generated.

```
# seed the pseudorandom number generator
from random import seed
from random import random
# seed random number generator
seed(1)
# generate some random numbers
print(random(), random(), random())
# reset the seed
seed(1)
# generate some random numbers
print(random(), random(), random())
```

[1] http://amzn.to/2CM9dDv

Listing 6.1: Example of seeding the Python random number generator.

Running the example seeds the pseudorandom number generator with the value 1, generates 3 random numbers, reseeds the generator, and shows that the same three random numbers are generated.

```
0.13436424411240122 0.8474337369372327 0.763774618976614
0.13436424411240122 0.8474337369372327 0.763774618976614
```

Listing 6.2: Example output from seeding the Python random number generator.

It can be useful to control the randomness by setting the seed to ensure that your code produces the same result each time, such as in a production model. For running experiments where randomization is used to control for confounding variables, a different seed may be used for each experimental run.

6.4.2 Random Floating Point Values

Random floating point values can be generated using the `random()` function. Values will be generated in the range between 0 and 1, specifically in the interval [0,1). Values are drawn from a uniform distribution, meaning each value has an equal chance of being drawn. The example below generates 10 random floating point values.

```
# generate random floating point values
from random import seed
from random import random
# seed random number generator
seed(1)
# generate random numbers between 0-1
for _ in range(10):
  value = random()
  print(value)
```

Listing 6.3: Example of generating random floats with Python.

Running the example generates and prints each random floating point value.

```
0.13436424411240122
0.8474337369372327
0.763774618976614
0.2550690257394217
0.49543508709194095
0.4494910647887381
0.651592972722763
0.7887233511355132
0.0938595867742349
0.02834747652200631
```

Listing 6.4: Example output from generating random floats with Python.

The floating point values could be rescaled to a desired range by multiplying them by the size of the new range and adding the min value, as follows:

$$scaledvalue = min + (value \times (max - min)) \tag{6.1}$$

Where *min* and *max* are the minimum and maximum values of the desired range respectively, and *value* is the randomly generated floating point value in the range between 0 and 1.

6.4.3 Random Integer Values

Random integer values can be generated with the `randint()` function. This function takes two arguments: the start and the end of the range for the generated integer values. Random integers are generated within and including the start and end of range values, specifically in the interval [start, end]. Random values are drawn from a uniform distribution. The example below generates 10 random integer values between 0 and 10.

```
# generate random integer values
from random import seed
from random import randint
# seed random number generator
seed(1)
# generate some integers
for _ in range(10):
  value = randint(0, 10)
  print(value)
```

Listing 6.5: Example of generating random integers with Python.

Running the example generates and prints 10 random integer values.

```
2
9
1
4
1
7
7
7
10
6
```

Listing 6.6: Example output from generating random integers with Python.

6.4.4 Random Gaussian Values

Random floating point values can be drawn from a Gaussian distribution using the `gauss()` function. This function takes two arguments that correspond to the parameters that control the size of the distribution, specifically the mean and the standard deviation. The example below generates 10 random values drawn from a Gaussian distribution with a mean of 0.0 and a standard deviation of 1.0. Note that these parameters are not the bounds on the values and that the spread of the values will be controlled by the bell shape of the distribution, in this case proportionately likely above and below 0.0.

```
# generate random Gaussian values
from random import seed
from random import gauss
# seed random number generator
seed(1)
```

```
# generate some Gaussian values
for _ in range(10):
  value = gauss(0, 1)
  print(value)
```

Listing 6.7: Example of generating random Gaussian values with Python.

Running the example generates and prints 10 Gaussian random values.

```
1.2881847531554629
1.449445608699771
0.06633580893826191
-0.7645436509716318
-1.0921732151041414
0.03133451683171687
-1.022103170010873
-1.4368294451025299
0.19931197648375384
0.13337460465860485
```

Listing 6.8: Example output from generating random Gaussian values with Python.

6.4.5 Randomly Choosing From a List

Random numbers can be used to randomly choose an item from a list. For example, if a list had 10 items with indexes between 0 and 9, then you could generate a random integer between 0 and 9 and use it to randomly select an item from the list. The `choice()` function implements this behavior for you. Selections are made with a uniform likelihood. The example below generates a list of 20 integers and gives five examples of choosing one random item from the list.

```
# choose a random element from a list
from random import seed
from random import choice
# seed random number generator
seed(1)
# prepare a sequence
sequence = [i for i in range(20)]
print(sequence)
# make choices from the sequence
for _ in range(5):
  selection = choice(sequence)
  print(selection)
```

Listing 6.9: Example of generating random choices with Python.

Running the example first prints the list of integer values, followed by five examples of choosing and printing a random value from the list.

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
4
18
2
8
3
```

Listing 6.10: Example output from generating random choices with Python.

6.4.6 Random Subsample From a List

We may be interested in repeating the random selection of items from a list to create a randomly chosen subset. Importantly, once an item is selected from the list and added to the subset, it should not be added again. This is called selection without replacement because once an item from the list is selected for the subset, it is not added back to the original list (i.e. is not made available for re-selection). This behavior is provided in the `sample()` function that selects a random sample from a list without replacement. The function takes both the list and the size of the subset to select as arguments. Note that items are not actually removed from the original list, only selected into a copy of the list. The example below demonstrates selecting a subset of five items from a list of 20 integers.

```
# select a random sample without replacement
from random import seed
from random import sample
# seed random number generator
seed(1)
# prepare a sequence
sequence = [i for i in range(20)]
print(sequence)
# select a subset without replacement
subset = sample(sequence, 5)
print(subset)
```

Listing 6.11: Example of generating random samples with Python.

Running the example first prints the list of integer values, then the random sample is chosen and printed for comparison.

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[4, 18, 2, 8, 3]
```

Listing 6.12: Example output from generating random samples with Python.

6.4.7 Randomly Shuffle a List

Randomness can be used to shuffle a list of items, like shuffling a deck of cards. The `shuffle()` function can be used to shuffle a list. The shuffle is performed in place, meaning that the list provided as an argument to the `shuffle()` function is shuffled rather than a shuffled copy of the list being made and returned. The example below demonstrates randomly shuffling a list of integer values.

```
# randomly shuffle a sequence
from random import seed
from random import shuffle
# seed random number generator
seed(1)
# prepare a sequence
sequence = [i for i in range(20)]
print(sequence)
# randomly shuffle the sequence
shuffle(sequence)
print(sequence)
```

Listing 6.13: Example of shuffling a list with Python.

Running the example first prints the list of integers, then the same list after it has been randomly shuffled.

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[11, 5, 17, 19, 9, 0, 16, 1, 15, 6, 10, 13, 14, 12, 7, 3, 8, 2, 18, 4]
```

Listing 6.14: Example output from shuffling a list with Python.

6.5 Random Numbers with NumPy

In machine learning, you are likely using libraries such as scikit-learn and Keras. These libraries make use of NumPy under the covers, a library that makes working with vectors and matrices of numbers very efficient. NumPy also has its own implementation of a pseudorandom number generator and convenience wrapper functions. NumPy also implements the Mersenne Twister pseudorandom number generator. Let's look at a few examples of generating random numbers and using randomness with NumPy arrays.

6.5.1 Seed The Random Number Generator

The NumPy pseudorandom number generator is different from the Python standard library pseudorandom number generator. Importantly, seeding the Python pseudorandom number generator does not impact the NumPy pseudorandom number generator. It must be seeded and used separately. The `seed()` function can be used to seed the NumPy pseudorandom number generator, taking an integer as the seed value. The example below demonstrates how to seed the generator and how reseeding the generator will result in the same sequence of random numbers being generated.

```
# seed the pseudorandom number generator
from numpy.random import seed
from numpy.random import rand
# seed random number generator
seed(1)
# generate some random numbers
print(rand(3))
# reset the seed
seed(1)
# generate some random numbers
print(rand(3))
```

Listing 6.15: Example of seeding the random number generator in NumPy.

Running the example seeds the pseudorandom number generator, prints a sequence of random numbers, then reseeds the generator showing that the exact same sequence of random numbers is generated.

```
[4.17022005e-01 7.20324493e-01 1.14374817e-04]
[4.17022005e-01 7.20324493e-01 1.14374817e-04]
```

Listing 6.16: Example output from seeding the random number generator in NumPy.

6.5.2 Array of Random Floating Point Values

An array of random floating point values can be generated with the `rand()` NumPy function. If no argument is provided, then a single random value is created, otherwise the size of the array can be specified. The example below creates an array of 10 random floating point values drawn from a uniform distribution.

```
# generate random floating point values
from numpy.random import seed
from numpy.random import rand
# seed random number generator
seed(1)
# generate random numbers between 0-1
values = rand(10)
print(values)
```

Listing 6.17: Example of generating an array of random floats with NumPy.

Running the example generates and prints the NumPy array of random floating point values.

```
[4.17022005e-01 7.20324493e-01 1.14374817e-04 3.02332573e-01
 1.46755891e-01 9.23385948e-02 1.86260211e-01 3.45560727e-01
 3.96767474e-01 5.38816734e-01]
```

Listing 6.18: Example output from generating an array of random floats with NumPy.

6.5.3 Array of Random Integer Values

An array of random integers can be generated using the `randint()` NumPy function. This function takes three arguments, the lower end of the range, the upper end of the range, and the number of integer values to generate or the size of the array. Random integers will be drawn from a uniform distribution including the lower value and excluding the upper value, e.g. in the interval [lower, upper). The example below demonstrates generating an array of random integers.

```
# generate random integer values
from numpy.random import seed
from numpy.random import randint
# seed random number generator
seed(1)
# generate some integers
values = randint(0, 10, 20)
print(values)
```

Listing 6.19: Example of generating an array of random integers with NumPy.

Running the example generates and prints an array of 20 random integer values between 0 and 10.

```
[5 8 9 5 0 0 1 7 6 9 2 4 5 2 4 2 4 7 7 9]
```

Listing 6.20: Example output from generating an array of random integers with NumPy.

6.5.4 Array of Random Gaussian Values

An array of random Gaussian values can be generated using the `randn()` NumPy function. This function takes a single argument to specify the size of the resulting array. The Gaussian values are drawn from a standard Gaussian distribution; this is a distribution that has a mean of 0.0 and a standard deviation of 1.0. The example below shows how to generate an array of random Gaussian values.

```
# generate random Gaussian values
from numpy.random import seed
from numpy.random import randn
# seed random number generator
seed(1)
# generate some Gaussian values
values = randn(10)
print(values)
```

Listing 6.21: Example of generating an array of random Gaussian values with NumPy.

Running the example generates and prints an array of 10 random values from a standard Gaussian distribution.

```
[ 1.62434536 -0.61175641 -0.52817175 -1.07296862 0.86540763 -2.3015387
  1.74481176 -0.7612069 0.3190391 -0.24937038]
```

Listing 6.22: Example output from generating an array of random Gaussian values with NumPy.

Values from a standard Gaussian distribution can be scaled by multiplying the value by the standard deviation and adding the mean from the desired scaled distribution. For example:

$$scaledvalue = mean + value \times stdev \tag{6.2}$$

Where *mean* and *stdev* are the mean and standard deviation for the desired scaled Gaussian distribution and *value* is the randomly generated value from a standard Gaussian distribution.

6.5.5 Shuffle NumPy Array

A NumPy array can be randomly shuffled in-place using the `shuffle()` NumPy function. The example below demonstrates how to shuffle a NumPy array.

```
# randomly shuffle a sequence
from numpy.random import seed
from numpy.random import shuffle
# seed random number generator
seed(1)
# prepare a sequence
sequence = [i for i in range(20)]
print(sequence)
# randomly shuffle the sequence
shuffle(sequence)
print(sequence)
```

Listing 6.23: Example of shuffling an array in NumPy.

Running the example first generates a list of 20 integer values, then shuffles and prints the shuffled array.

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[3, 16, 6, 10, 2, 14, 4, 17, 7, 1, 13, 0, 19, 18, 9, 15, 8, 12, 11, 5]
```

Listing 6.24: Example output from shuffling an array in NumPy.

Now that we know how controlled randomness is generated, let's look at where we can use it effectively.

6.6 When to Seed the Random Number Generator

There are times during a predictive modeling project when you should consider seeding the random number generator. Let's look at two cases:

- **Data Preparation**. Data preparation may use randomness, such as a shuffle of the data or selection of values. Data preparation must be consistent so that the data is always prepared in the same way during fitting, evaluation, and when making predictions with the final model.
- **Data Splits**. The splits of the data such as for a train/test split or k-fold cross-validation must be made consistently. This is to ensure that each algorithm is trained and evaluated in the same way on the same subsamples of data.

You may wish to seed the pseudorandom number generator once before each task or once before performing the batch of tasks. It generally does not matter which. Sometimes you may want an algorithm to behave consistently, perhaps because it is trained on exactly the same data each time. This may happen if the algorithm is used in a production environment. It may also happen if you are demonstrating an algorithm in a tutorial environment. In that case, it may make sense to initialize the seed prior to fitting the algorithm.

6.7 How to Control for Randomness

A stochastic machine learning algorithm will learn slightly differently each time it is run on the same data. This will result in a model with slightly different performance each time it is trained. As mentioned, we can fit the model using the same sequence of random numbers each time. When evaluating a model, this is a bad practice as it hides the inherent uncertainty within the model.

A better approach is to evaluate the algorithm in such a way that the reported performance includes the measured uncertainty in the performance of the algorithm. We can do that by repeating the evaluation of the algorithm multiple times with different sequences of random numbers. The pseudorandom number generator could be seeded once at the beginning of the evaluation or it could be seeded with a different seed at the beginning of each evaluation. There are two aspects of uncertainty to consider here:

- **Data Uncertainty**: Evaluating an algorithm on multiple splits of the data will give insight into how the algorithms performance varies with changes to the train and test data.

- **Algorithm Uncertainty**: Evaluating an algorithm multiple times on the same splits of data will give insight into how the algorithm performance varies alone.

In general, I would recommend reporting on both of these sources of uncertainty combined. This is where the algorithm is fit on different splits of the data each evaluation run and has a new sequence of randomness. The evaluation procedure can seed the random number generator once at the beginning and the process can be repeated perhaps 30 or more times to give a population of performance scores that can be summarized. This will give a fair description of model performance taking into account variance both in the training data and in the learning algorithm itself.

6.8 Common Questions

- **Can I predict random numbers?**
  You cannot predict the sequence of random numbers, even with a deep neural network.
- **Will real random numbers lead to better results?**
  As far as I have read, using real randomness does not help in general, unless you are working with simulations of physical processes.
- **What about the final model?**
  The final model is the chosen algorithm and configuration trained on all available training data that you can use to make predictions. The performance of this model will fall within the variance of the evaluated model.

6.9 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Confirm that seeding the Python pseudorandom number generator does not impact the NumPy pseudorandom number generator.
- Practice generating random numbers between different ranges.
- Locate the equation for and implement a very simple pseudorandom number generator.

If you explore any of these extensions, I'd love to know.

6.10 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

6.10.1 API

- `random` Python API.
  https://docs.python.org/3/library/random.html
- Random Sampling in NumPy.
  https://docs.scipy.org/doc/numpy/reference/routines.random.html

6.10.2 Articles

- Random number generation on Wikipedia.
  https://en.wikipedia.org/wiki/Random_number_generation
- Pseudorandom number generator.
  https://en.wikipedia.org/wiki/Pseudorandom_number_generator
- Mersenne Twister.
  https://en.wikipedia.org/wiki/Mersenne_Twister

6.11 Summary

In this tutorial, you discovered the role of randomness in applied machine learning and how to control and harness it. Specifically, you learned:

- Machine learning has sources of randomness such as in the sample of data and in the algorithms themselves.
- Randomness is injected into programs and algorithms using pseudorandom number generators.
- There are times when the randomness requires careful control, and times when the randomness needs to be controlled-for.

6.11.1 Next

In the next section, you will discover the law of large numbers.

Chapter 7

Law of Large Numbers

We have an intuition that more observations are better. This is the same intuition behind the idea that if we collect more data, our sample of data will be more representative of the problem domain. There is a theorem in statistics and probability that supports this intuition that is a pillar of both of these fields and has important implications in applied machine learning. The name of this theorem is the law of large numbers. In this tutorial, you will discover the law of large numbers and why it is important in applied machine learning. After completing this tutorial, you will know:

- The law of large numbers supports the intuition that the sample becomes more representative of the population as its size is increased.
- How to develop a small example in Python to demonstrate the decrease in error from the increase in sample size.
- The law of large numbers is critical for understanding the selection of training datasets, test datasets, and in the evaluation of model skill in machine learning.

Let's get started.

7.1 Tutorial Overview

This tutorial is divided into 3 parts; they are:

1. Law of Large Numbers
2. Worked Example
3. Implications in Machine Learning

7.2 Law of Large Numbers

The law of large numbers is a theorem from probability and statistics that suggests that the average result from repeating an experiment multiple times will better approximate the true or expected underlying result.

> The law of large numbers explains why casinos always make money in the long run.
>
> — Page 79, *Naked Statistics: Stripping the Dread from the Data*, 2014.

We can think of a trial of an experiment as one observation. The standalone and independent repetition of the experiment will perform multiple trials and lead to multiple observations. All sample observations for an experiment are drawn from an idealized population of observations.

- **Observation**: Result from one trial of an experiment.
- **Sample**: Group of results gathered from separate independent trials.
- **Population**: Space of all possible observations that could be seen from a trial.

Using these terms from statistics, we can say that as the size of the sample increases, the mean value of the sample will better approximate the mean or expected value in the population. As the sample size goes to infinity, the sample mean will converge to the population mean.

> ... a crowning achievement in probability, the law of large numbers. This theorem says that the mean of a large sample is close to the mean of the distribution.
>
> — Page 76, *All of Statistics: A Concise Course in Statistical Inference*, 2004.

This is an important theoretical finding for statistics and probability, as well as for applied machine learning.

7.2.1 Independent and Identically Distributed

It is important to be clear that the observations in the sample must be independent. This means that the trial is run in an identical manner and does not depend on the results of any other trial. This is often reasonable and easy to achieve in computers, although can be difficult elsewhere (e.g. how do you achieve identically random rolls of a dice?). In statistics, this expectation is called *independent and identically distributed*, or IID, iid, or i.i.d. for short. This is to ensure that the samples are indeed drawn from the same underlying population distribution.

7.2.2 Regression to the Mean

The law of large numbers helps us understand why we cannot trust a single observation from an experiment in isolation. We expect that a single result or the mean result from a small sample is likely. That is close to the central tendency, the mean of the population distribution. It may not be; in fact, it may be very strange or unlikely. The law reminds us to repeat the experiment in order to develop a large and representative sample of observations before we start making inferences about what the result means.

As we increase the sample size, the finding or mean of the sample will move back toward the population mean, back toward the true underlying expected value. This is called regression to the mean or sometimes reversion to the mean. It is why we must be skeptical of inferences from small sample sizes, called small n.

7.2.3 Law of Truly Large Numbers

Related to the regression to the mean is the idea of the law of truly large numbers. This is the idea that when we start investigating or working with extremely large samples of observations, we increase the likelihood of seeing something strange. That by having so many samples of the underlying population distribution, the sample will contain some astronomically rare events.

Again, we must be wary not to make inferences from single cases. This is especially important to consider when running queries and investigating big data.

7.3 Worked Example

We can demonstrate the law of large numbers with a small worked example. First, we can design an idealized underlying distribution. We will use a Gaussian distribution with a mean of 50 and a standard deviation of 5. The expected value or mean of this population is therefore 50. Below is some code that generates a plot of this idealized distribution.

```
# idealized population distribution
from numpy import arange
from matplotlib import pyplot
from scipy.stats import norm
# x-axis for the plot
xaxis = arange(30, 70, 1)
# y-axis for the plot
yaxis = norm.pdf(xaxis, 50, 5)
# plot ideal population
pyplot.plot(xaxis, yaxis)
pyplot.show()
```

Listing 7.1: Example of an idealized Gaussian distribution.

Running the code creates a plot of the designed population with the familiar bell shape.

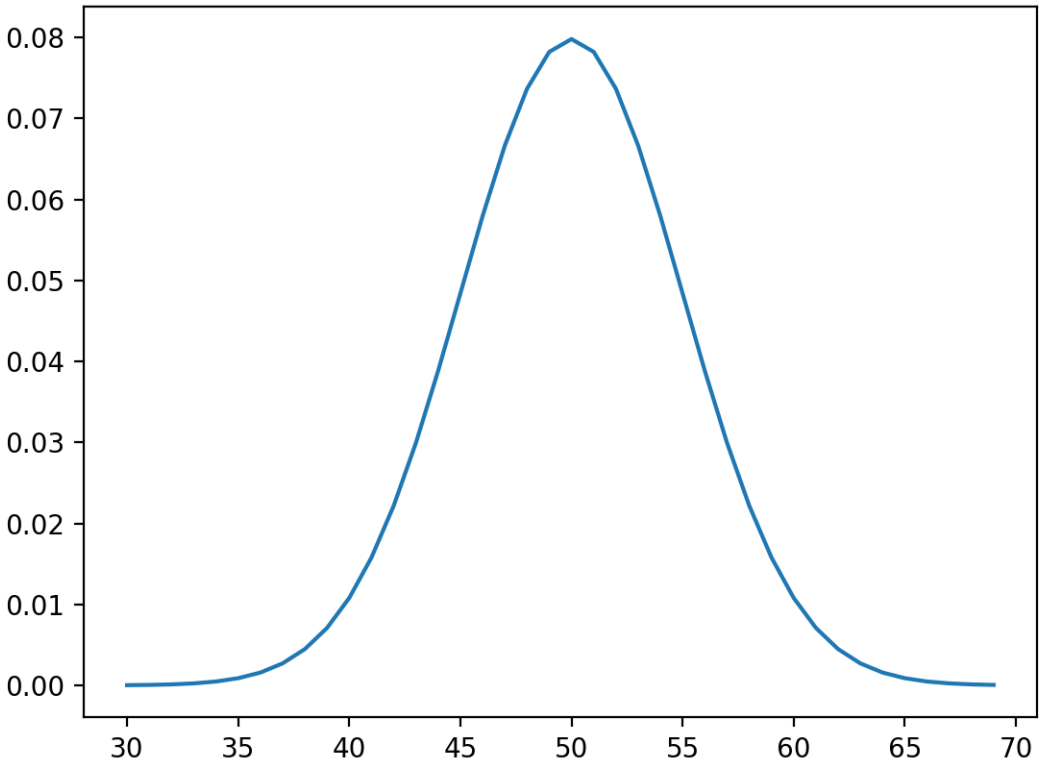

Figure 7.1: Line plot of an idealized Gaussian distribution.

Now, we can pretend to forget everything that we know about the population and make independent random samples from the population. We can create samples of different sizes and calculate the mean. Given our intuition and the law of large numbers, we expect that as the size of the sample is increased, the sample mean will better approximate the population mean. The example below calculates samples of different sizes, estimates the mean and plots the errors in the mean from the expected mean (50).

```
# demonstrate the law of large numbers
from numpy.random import seed
from numpy.random import randn
from numpy import mean
from numpy import array
from matplotlib import pyplot
# seed the random number generator
seed(1)
# sample sizes
sizes = list()
for x in range (10, 20000, 200):
  sizes.append(x)
# generate samples of different sizes and calculate their means
means = [mean(5 * randn(size) + 50) for size in sizes]
# plot sample mean error vs sample size
pyplot.scatter(sizes, array(means)-50)
```

```
pyplot.show()
```

Listing 7.2: Example of estimating the mean from different sized samples.

Running the example creates a plot that compares the size of the sample to the error of the sample mean from the population mean. Generally, we can see that larger sample sizes have less error, and we would expect this trend to continue, on average. We can also see that some sample means overestimate and some underestimate. Do not fall into the trap of assuming that the underestimate will fall on one side or another.

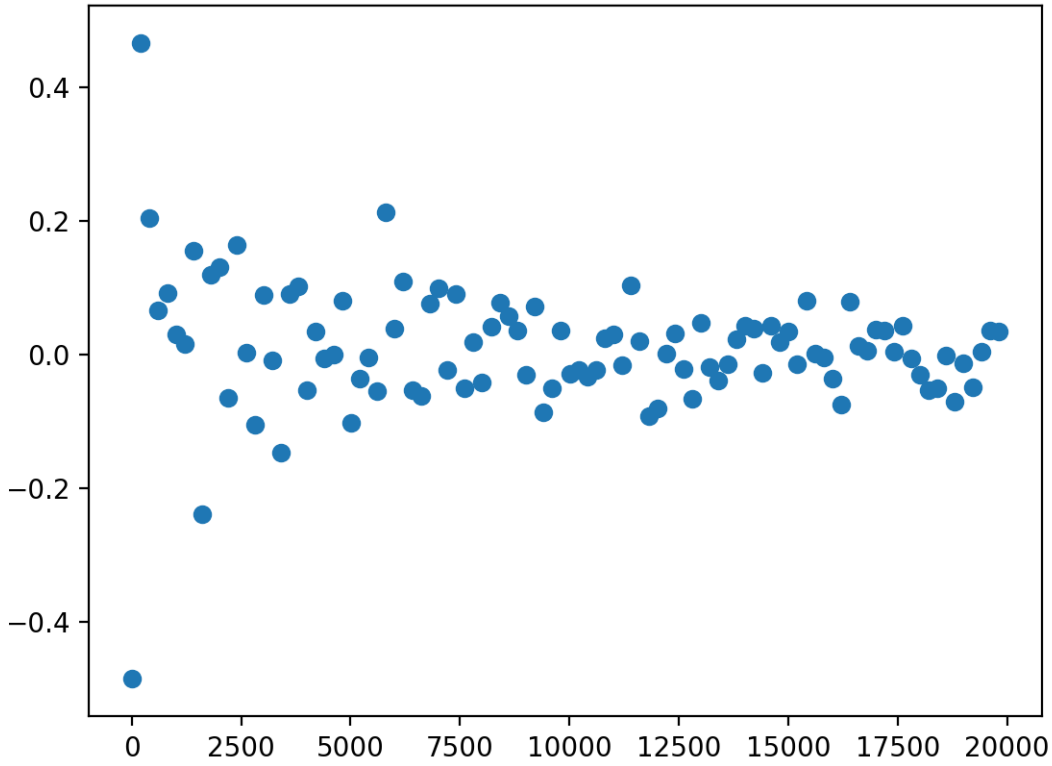

Figure 7.2: Scatter plot of the error in the population mean vs sample size.

7.4 Implications in Machine Learning

The law of large numbers has important implications in applied machine learning. Let's take a moment to highlight a few of these implications.

7.4.1 Training Data

The data used to train the model must be representative of the observations from the domain. This really means that it must contain enough information to generalize to the true unknown and underlying distribution of the population. This is easy to conceptualize with a single input

variable for a model, but is also just as important when you have multiple input variables. There will be unknown relationships or dependencies between the input variables and together, the input data will represent a multivariate distribution from which observations will be drawn to comprise your training sample.

Keep this in mind during data collection, data cleaning, and data preparation. You may choose to exclude sections of the underlying population by setting hard limits on observed values (e.g. for outliers) where you expect data to be too sparse to model effectively.

7.4.2 Test Data

The thoughts given to the training dataset must also be given to the test dataset. This is often neglected with the blind use of 80/20 spits for train/test data or the blind use of 10-fold cross-validation, even on datasets where the size of 1/10th of the available data may not be a suitable representative of observations from the problem domain.

7.4.3 Model Skill Evaluation

Consider the law of large numbers when presenting the estimated skill of a model on unseen data. It provides a defense for not simply reporting or proceeding with a model based on a skill score from a single train/test evaluation. It highlights the need to develop a sample of multiple independent (or close to independent) evaluations of a given model such that the mean reported skill from the sample is an accurate enough estimate of population mean.

7.5 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Brainstorm two additional areas of machine learning where the law of large numbers applies.
- Find five research papers where you are skeptical of the results given the law of large numbers.
- Develop your own ideal distribution and samples and plot the relationship between sample size and sample mean error.

If you explore any of these extensions, I'd love to know.

7.6 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

7.6.1 Books

- *Naked Statistics: Stripping the Dread from the Data*, 2014.
  http://amzn.to/2F5tVnX

- *All of Statistics: A Concise Course in Statistical Inference*, 2004.
  http://amzn.to/2FNFQns

7.6.2 API

- `scipy.stats.norm` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html
- `numpy.random.seed` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.seed.html
- `numpy.random.randn` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.randn.html
- `numpy.mean` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html

7.6.3 Articles

- Law of large numbers on Wikipedia.
  https://en.wikipedia.org/wiki/Law_of_large_numbers
- Independent and identically distributed random variables on Wikipedia.
  https://en.wikipedia.org/wiki/Independent_and_identically_distributed_random_variables
- Law of truly large numbers on Wikipedia.
  https://en.wikipedia.org/wiki/Law_of_truly_large_numbers
- Regression toward the mean.
  https://en.wikipedia.org/wiki/Regression_toward_the_mean

7.7 Summary

In this tutorial, you discovered the law of large numbers and why it is important in applied machine learning. Specifically, you learned:

- The law of large numbers supports the intuition that the sample becomes more representative of the population as its size is increased.
- How to develop a small example in Python to demonstrate the decrease in error from the increase in sample size.
- The law of large numbers is critical for understanding the selection of training datasets, test datasets, and in the evaluation of model skill in machine learning.

7.7.1 Next

In the next section, you will discover the central limit theorem.

Chapter 8

Central Limit Theorem

The central limit theorem is an often quoted, but misunderstood pillar from statistics and machine learning. It is often confused with the law of large numbers. Although the theorem may seem esoteric to beginners, it has important implications about how and why we can make inferences about the skill of machine learning models, such as whether one model is statistically better than another and confidence intervals on models skill. In this tutorial, you will discover the central limit theorem and the implications of this important pillar of statistics and probability on applied machine learning. After completing this tutorial, you will know:

- The central limit theorem describes the shape of the distribution of sample means as a Gaussian, which is a distribution that statistics knows a lot about.
- How to develop an example of simulated dice rolls in Python to demonstrate the central limit theorem.
- How the central limit theorem and knowledge of the Gaussian distribution is used to make inferences about model performance in applied machine learning.

Let's get started.

8.1 Tutorial Overview

This tutorial is divided into 3 parts; they are:

1. Central Limit Theorem
2. Worked Example with Dice
3. Impact on Machine Learning

8.2 Central Limit Theorem

The Central Limit Theorem, or CLT for short, is an important finding and pillar in the fields of statistics and probability. It may seem a little esoteric at first, so hang in there. It turns out that the finding is critically important for making inferences in applied machine learning.

The theorem states that as the size of the sample increases, the distribution of the mean across multiple samples will approximate a Gaussian distribution. Let's break this down.

We can imagine performing a trial and getting a result or an observation. We can repeat the trial again and get a new independent observation. Collected together, multiple observations represents a sample of observations. A sample is a group of observations from a broader population of all possible observations that could be made given trials.

- Observation: Result from one trial of an experiment.
- Sample: Group of results gathered from separate independent trials.
- Population: Space of all possible observations that could be seen from a trial.

If we calculate the mean of a sample, it will be an estimate of the mean of the population distribution. But, like any estimate, it will be wrong and will contain some error. If we draw multiple independent samples, and calculate their means, the distribution of those means will form a Gaussian distribution. It is important that each trail that results in an observation be independent and performed in the same way. This is to ensure that the sample is drawing from the same underlying population distribution. More formally, this expectation is referred to as independent and identically distributed, or iid.

Firstly, the central limit theorem is impressive, especially as this will occur no matter the shape of the population distribution from which we are drawing samples. It demonstrates that the distribution of errors from estimating the population mean fit a distribution that the field of statistics knows a lot about. Secondly, this estimate of the Gaussian distribution will be more accurate as the size of the samples drawn from the population is increased. This means that if we use our knowledge of the Gaussian distribution in general to start making inferences about the means of samples drawn from a population, that these inferences will become more useful as we increase our sample size.

One interesting implication of the central limit theorem mentioned to me one time by a very clever scientist is that you can use it to generate Gaussian random numbers. You can generate uniformly random integers, sum groups of them together, and the results of the sums will be Gaussian. Remember that the mean is just the normalized sum of the sample. It's a slower method for generating random Gaussian variables than other methods (like the Box-Muller method), but a clear (and clever) application of the theorem.

8.2.1 Law of Large Numbers

The central limit theorem is often confused with the law of large numbers by beginners. The law of large numbers is another different theorem from statistics. It is simpler in that it states that as the size of a sample is increased, the more accurate of an estimate the sample mean will be of the population mean. The central limit theorem does not state anything about a single sample mean; instead, it is broader and states something about the shape or the distribution of sample means.

The law of large numbers is intuitive. It is why we think that collecting more data will lead to a more representative sample of observations from the domain. The theorem supports this intuition. The central limit theorem is not intuitive. Instead, it is a finding that we can exploit in order to make claims about sample means.

8.3 Worked Example with Dice

We can make the central limit theorem concrete with a worked example involving the rolling of die. Remember that a die is a cube with a different number on each side from 1-to-6. Each number has a 1-in-6 likelihood to turn up from a roll. The distribution of the numbers that turn up from a dice roll is uniform given the equal likelihood. We can use the `randint()` NumPy function to generate a specific number of random dice rolls (e.g. 50) between 1 and 6.

```
...
# generate a sample of die rolls
rolls = randint(1, 7, 50)
```

Listing 8.1: Example of how to simulate dice rolls.

The complete example is listed below.

```
# generate random dice rolls
from numpy.random import seed
from numpy.random import randint
from numpy import mean
# seed the random number generator
seed(1)
# generate a sample of die rolls
rolls = randint(1, 7, 50)
print(rolls)
print(mean(rolls))
```

Listing 8.2: Example calculating the average of simulated dice rolls.

Running the example generates and prints the sample of 50 die rolls and the mean value of the sample. We know that the mean value of the distribution is 3.5 calculated as $\frac{1+2+3+4+5+6}{6}$ or $\frac{21}{6}$. We can see that the mean of the sample is slightly wrong, which is to be expected because it is an estimate of the population mean.

```
[6 4 5 1 2 4 6 1 1 2 5 6 5 2 3 5 6 3 5 4 5 3 5 6 3 5 2 2 1 6 2 2 6 2 2 1 5
 2 1 1 6 4 3 2 1 4 6 2 2 4]
3.44
```

Listing 8.3: Sample output from simulated dice rolls.

This is the result of rolling the simulated die 50 times. We can then repeat this process multiple times, such as 1,000. This will give us a result of 1,000 sample means. According to the central limit theorem, the distribution of these sample means will be Gaussian. The example below performs this experiment and plots the resulting distribution of sample means.

```
# demonstration of the central limit theorem
from numpy.random import seed
from numpy.random import randint
from numpy import mean
from matplotlib import pyplot
# seed the random number generator
seed(1)
# calculate the mean of 50 dice rolls 1000 times
means = [mean(randint(1, 7, 50)) for _ in range(1000)]
# plot the distribution of sample means
pyplot.hist(means)
```

```
pyplot.show()
```

Listing 8.4: Example of calculating the average of multiple simulations of dice rolls.

Running the example creates a histogram plot of the sample means. We can tell from the shape of the distribution that the distribution is Gaussian. It's interesting to note the amount of error in the sample mean that we can see in 1,000 trials of 50 dice rolls. Further, the central limit theorem also states that as the size of each sample, in this case 50, is increased, then the better the sample means will approximate a Gaussian distribution.

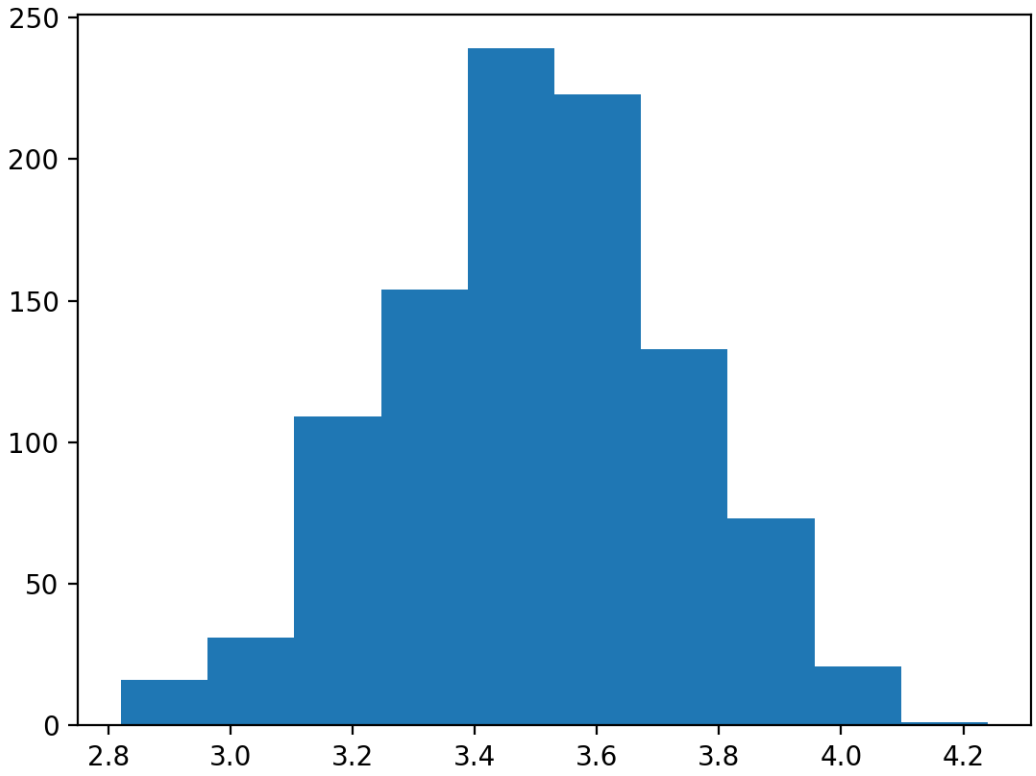

Figure 8.1: Histogram plot of sample means from dice roll simulations.

8.4 Impact on Machine Learning

The central limit theorem has important implications in applied machine learning. The theorem does inform the solution to linear algorithms such as linear regression, but not exotic methods like artificial neural networks that are solved using numerical optimization methods. Instead, we must use experiments to observe and record the behavior of the algorithms and use statistical methods to interpret their results. Let's look at two important examples.

8.4.1 Significance Tests

In order to make inferences about the skill of a model compared to the skill of another model, we must use tools such as statistical significance tests. These tools estimate the likelihood that the two samples of model skill scores were drawn from the same or a different unknown underlying distribution of model skill scores. If it looks like the samples were drawn from the same population, then no difference between the models skill is assumed, and any actual differences are due to statistical noise.

The ability to make inference claims like this is due to the central limit theorem and our knowledge of the Gaussian distribution and how likely the two sample means are to be a part of the same Gaussian distribution of sample means.

8.4.2 Confidence Intervals

Once we have trained a final model, we may wish to make an inference about how skillful the model is expected to be in practice. The presentation of this uncertainty is called a confidence interval. We can develop multiple independent (or close to independent) evaluations of a model accuracy to result in a population of candidate skill estimates. The mean of these skill estimates will be an estimate (with error) of the true underlying estimate of the model skill on the problem.

With knowledge that the sample mean will be a part of a Gaussian distribution from the central limit theorem, we can use knowledge of the Gaussian distribution to estimate the likelihood of the sample mean based on the sample size and calculate an interval of desired confidence around the skill of the model.

8.5 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Suggest two additional areas in an applied machine learning where the central limit theorem may be relevant.
- Implement a function for generating random Gaussian numbers exploiting the central limit theorem and numbers drawn from a uniform distribution.
- Update the demonstration of dice rolls to demonstrate the relationship between sample size and the fidelity of the Gaussian distribution of the sample means.

If you explore any of these extensions, I'd love to know.

8.6 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

8.6.1 API

- `numpy.random.seed` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.seed.html
- `numpy.random.randint` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.randint.html
- `numpy.mean` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html

8.6.2 Articles

- Central limit theorem on Wikipedia.
  https://en.wikipedia.org/wiki/Central_limit_theorem
- Illustration of the central limit theorem on Wikipedia.
  https://en.wikipedia.org/wiki/Illustration_of_the_central_limit_theorem
- Law of large numbers on Wikipedia.
  https://en.wikipedia.org/wiki/Law_of_large_numbers

8.7 Summary

In this tutorial, you discovered the central limit theorem and the implications of this important pillar of statistics and probability on applied machine learning. Specifically, you learned:

- The central limit theorem describes the shape of the distribution of sample means as a Gaussian, which is a distribution that statistics knows a lot about.
- How to develop an example of simulated dice rolls in Python to demonstrate the central limit theorem.
- How the central limit theorem and knowledge of the Gaussian distribution are used to make inferences about model performance in applied machine learning.

8.7.1 Next

This is the end of part III, in the next part you will discover statistical hypothesis testing.

Part IV

Hypothesis Testing

Chapter 9

Statistical Hypothesis Testing

Data must be interpreted in order to discover meaning. We can interpret data by assuming a specific structured outcome and use statistical methods to confirm or reject the assumption. The assumption is called a hypothesis and the statistical tests used for this purpose are called statistical hypothesis tests. Whenever we want to make claims about the distribution of data or whether one set of results are different from another set of results in applied machine learning, we must rely on statistical hypothesis tests.

In this tutorial, you will discover statistical hypothesis testing and how to interpret and carefully state the results from statistical tests. After completing this tutorial, you will know:

- Statistical hypothesis tests are important for quantifying answers to questions about samples of data.
- The interpretation of a statistical hypothesis test requires a correct understanding of p-values and critical values.
- Regardless of the significance level, the finding of hypothesis tests may still contain errors.

Let's get started.

9.1 Tutorial Overview

This tutorial is divided into 4 parts; they are:

1. Statistical Hypothesis Testing
2. Statistical Test Interpretation
3. Errors in Statistical Tests
4. Degrees of Freedom

9.2 Statistical Hypothesis Testing

Data alone is not interesting. It is the interpretation of the data that we are really interested in. In statistics, when we wish to start asking questions about the data and interpret the results, we use statistical methods that provide a confidence or likelihood about the answers. In general, this class of methods is called statistical hypothesis testing, or significance tests.

The term *hypothesis* may make you think about science, where we investigate a hypothesis. This is along the right track. In statistics, a hypothesis test calculates some quantity under a given assumption. The result of the test allows us to interpret whether the assumption holds or whether the assumption has been violated. Two concrete examples that we will use a lot in machine learning are:

- A test that assumes that data has a normal distribution.
- A test that assumes that two samples were drawn from the same underlying population distribution.

The assumption of a statistical test is called the null hypothesis, or hypothesis zero (H0 for short). It is often called the default assumption, or the assumption that nothing has changed. A violation of the test's assumption is often called the first hypothesis, hypothesis one or H1 for short. H1 is really a short hand for *some other hypothesis*, as all we know is that the evidence suggests that the H0 can be rejected.

- **Hypothesis 0 (H0)**: Assumption of the test fails to be rejected.
- **Hypothesis 1 (H1)**: Assumption of the test does not hold and is rejected at some level of significance.

Before we can reject or fail to reject the null hypothesis, we must interpret the result of the test.

9.3 Statistical Test Interpretation

The results of a statistical hypothesis test must be interpreted for us to start making claims. This is a point that may cause a lot of confusion for beginners and experienced practitioners alike. There are two common forms that a result from a statistical hypothesis test may take, and they must be interpreted in different ways. They are the p-value and critical values.

9.3.1 Interpret the p-value

We describe a finding as statistically significant by interpreting the p-value. For example, we may perform a Student's t-test on two data samples and find that it is unlikely that the samples have the same mean. We reject the null hypothesis that the samples have the same mean at a chosen level of statistical significance (or confidence). A statistical hypothesis test may return a value called p or the p-value. This is a quantity that we can use to interpret or quantify the result of the test and either *reject* or *fail to reject* the null hypothesis. This is done by comparing the p-value to a threshold value chosen beforehand called the significance level.

The significance level is often referred to by the Greek lower case letter alpha (α). A common value used for alpha is 5% or 0.05. A smaller alpha value suggests a more robust interpretation of the result, such as 1% or 0.1%. The p-value is compared to the pre-chosen alpha value. A result is statistically significant when the p-value is less than or equal to alpha. This signifies a change was detected: that the default or null hypothesis can be rejected.

- `p-value` $\leq$ `alpha`: significant result, reject null hypothesis (H1).
- `p-value` $>$ `alpha`: not significant result, fail to reject the null hypothesis (H0).

The significance level can be inverted by subtracting it from 1 to give a confidence level of the hypothesis given the observed sample data.

$$\text{confidence level} = 1 - \text{significance level} \tag{9.1}$$

9.3.2 "*Reject*" vs "*Failure to Reject*"

The p-value is probabilistic. This means that when we interpret the result of a statistical test, we do not know what is true or false, only what is likely. Rejecting the null hypothesis means that there is sufficient statistical evidence that the null hypothesis does not look likely. Otherwise, it means that there is not sufficient statistical evidence to reject the null hypothesis.

We may think about the statistical test in terms of the dichotomy of rejecting and accepting the null hypothesis. The danger is that if we say that we *accept* the null hypothesis, the language suggests that the null hypothesis is true. Instead, it is safer to say that we *fail to reject* the null hypothesis, as in, there is insufficient statistical evidence to reject it. When reading *reject* vs *fail to reject* for the first time, it is confusing to beginners. You can think of it as *reject* vs *accept* in your mind, as long as you remind yourself that the result is probabilistic and that even an *accepted* null hypothesis still has a small probability of being wrong.

9.3.3 Common p-value Misinterpretations

This section highlights some common misinterpretations of the p-value in the results of statistical tests.

True or False Null Hypothesis

The interpretation of the p-value does not mean that the null hypothesis is true or false. It does mean that we have chosen to reject or fail to reject the null hypothesis at a specific statistical significance level based on empirical evidence and the chosen statistical test. You are limited to making probabilistic claims, not crisp binary or true/false claims about the result.

p-value as Probability

A common misunderstanding is that the p-value is a probability of the null hypothesis being true or false given the data. In probability, this would be written as follows:

$$Pr(hypothesis|data) \tag{9.2}$$

This is incorrect. Instead, the p-value can be thought of as the probability of the data given the pre-specified assumption embedded in the statistical test.

Again, using probability notation, this would be written as:

$$Pr(data|hypothesis) \tag{9.3}$$

It allows us to reason about whether or not the data fits the hypothesis. Not the other way around. The p-value is a measure of how likely the data sample would be observed if the null hypothesis were true.

Post-Hoc Tuning

It does not mean that you can re-sample your domain or tune your data sample and re-run the statistical test until you achieve a desired result. It also does not mean that you can choose your p-value after you run the test. This is called p-hacking or hill climbing and will mean that the result you present will be fragile and not representative. In science, this is at best unethical, and at worst fraud.

9.3.4 Interpret Critical Values

Some tests do not return a p-value. Instead, they might return a test statistic value from a specific data distribution that can be interpreted in the context of critical values. A critical value is a value from the distribution of the test statistic after which point the result is significant and the null hypothesis can be rejected.

- `Test Statistic` $<$ `Critical Value`: not significant result, fail to reject null hypothesis (H0).
- `Test Statistic` $\geq$ `Critical Value`: significant result, reject null hypothesis (H1).

It requires that you know the distribution of the test statistic and how to sample the distribution to retrieve the critical value. This is demonstrated for standard distributions in Chapter 11. The p-value is calculated from the critical value.

Again, the meaning of the result is similar in that the chosen significance level is a probabilistic decision on the rejection or failure of rejection of the base assumption of the test given the data. Results are presented in the same way as with a p-value, as either significance level or confidence level. For example, if a normality test was calculated and the test statistic was compared to the critical value at the 5% significance level, results could be stated as:

> The test found that the data sample was normal, failing to reject the null hypothesis at a 5% significance level.

Or:

> The test found that the data was normal, failing to reject the null hypothesis at a 95% confidence level.

9.4 Errors in Statistical Tests

The interpretation of a statistical hypothesis test is probabilistic. That means that the evidence of the test may suggest an outcome and be mistaken. For example, if alpha was 5%, it suggests that (at most) 1 time in 20 that the null hypothesis would be mistakenly rejected because of the statistical noise in the data sample. Given a small p-value (reject the null hypothesis) either means that the null hypothesis false (we got it right) or it is true and some rare and unlikely event has been observed (we made a mistake). If this type of error is made, it is called a *false positive*. We falsely believe the rejection of the null hypothesis.

Alternately, given a large p-value (fail to reject the null hypothesis), it may mean that the null hypothesis is true (we got it right) or that the null hypothesis is false and some unlikely event occurred (we made a mistake). If this type of error is made, it is called a *false negative*. We falsely believe the null hypothesis or assumption of the statistical test.

Each of these two types of error has a specific name.

- **Type I Error**: The incorrect rejection of a true null hypothesis, called a *false positive*.
- **Type II Error**: The incorrect failure of rejecting a false null hypothesis, called a *false negative*.

All statistical hypothesis tests have a chance of making either of these types of errors. False findings or false discoveries are more than possible; they are probable. Ideally, we want to choose a significance level that minimizes the likelihood of one of these errors. E.g. a very small significance level. Although significance levels such as 0.05 and 0.01 are common in many fields of science, harder sciences, such as physics, are more aggressive.

It is common to use a significance level of $(3 \times 10)^{-7}$ or 0.0000003, often referred to as 5-sigma. This means that the finding was due to chance with a probability of 1 in 3.5 million independent repeats of the experiments. To use a threshold like this may require a much large data sample. Nevertheless, these types of errors are always present and must be kept in mind when presenting and interpreting the results of statistical tests. It is also a reason why it is important to have findings independently verified.

9.5 Degrees of Freedom in Statistics

When we calculate statistics, we often must include some information about the size of the data sample. The way we do this is via the degrees of freedom. The degrees of freedom is the number of independent pieces of information that are used to estimate a parameter or calculate a statistic from the data sample.

The degrees of freedom may be written as df or dof and in an equation may use the Greek lowercase letter nu (ν). The degrees of freedom will always be equal to or less than the size of the sample, often denoted as n. If the statistic being calculated makes use of another statistic in an intermediate step, then the number of degrees of freedom must be corrected via making a subtraction. For example, when we calculate the sample mean as an estimate of the population mean, we can use the sample size as the degrees of freedom:

$$mean(x) = \frac{1}{n} \times \sum_{i=1}^{n} x_i \tag{9.4}$$

When we calculate the sample variance as an estimate of the population variance, we must correct the degrees of freedom because the sample mean is used in the calculation.

$$variance(x) = \frac{1}{n-1} \times \sum_{i=1}^{n}(x_i - mean(x))^2 \tag{9.5}$$

The correction updates the degrees of freedom for the single intermediate statistic used that itself can introduce some additional uncertainty into the calculation. We will see mention and use of the degrees of freedom as a parameter or correction in a suite of statistical hypothesis tests.

9.6 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Find an example of a research paper that does not present results using p-values.
- Find an example of a research paper that presents results with statistical significance, but makes one of the common misinterpretations of p-values.
- Find an example of a research paper that presents results with statistical significance and correctly interprets and presents the p-value and findings.

If you explore any of these extensions, I'd love to know.

9.7 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

9.7.1 Articles

- Statistical hypothesis testing on Wikipedia.
  https://en.wikipedia.org/wiki/Statistical_hypothesis_testing
- Statistical significance on Wikipedia.
  https://en.wikipedia.org/wiki/Statistical_significance
- p-value on Wikipedia.
  https://en.wikipedia.org/wiki/P-value
- Critical value on Wikipedia.
  https://en.wikipedia.org/wiki/Critical_value
- Type I and type II errors on Wikipedia.
  https://en.wikipedia.org/wiki/Type_I_and_type_II_errors
- Data dredging on Wikipedia.
  https://en.wikipedia.org/wiki/Data_dredging

- Misunderstandings of p-values on Wikipedia.
  https://en.wikipedia.org/wiki/Misunderstandings_of_p-values
- What does the 5 sigma mean?
  http://www.physics.org/article-questions.asp?id=103

9.8 Summary

In this tutorial, you discovered statistical hypothesis testing and how to interpret and carefully state the results from statistical tests. Specifically, you learned:

- Statistical hypothesis tests are important for quantifying answers to questions about samples of data.
- The interpretation of a statistical hypothesis test requires a correct understanding of p-values.
- Regardless of the significance level, the finding of hypothesis tests may still contain errors.

9.8.1 Next

In the next section, you will discover the three key distributions that you need to know well when working with statistical hypothesis tests.

Chapter 10

Statistical Distributions

A sample of data will form a distribution, and by far the most well-known distribution is the Gaussian distribution, often called the Normal distribution. The distribution provides a parameterized mathematical function that can be used to calculate the probability for any individual observation from the sample space. This distribution describes the grouping or the density of the observations, called the probability density function. We can also calculate the likelihood of an observation having a value equal to or lesser than a given value. A summary of these relationships between observations is called a cumulative density function.

In this tutorial, you will discover the Gaussian and related distribution functions and how to calculate probability and cumulative density functions for each. After completing this tutorial, you will know:

- A gentle introduction to standard distributions to summarize the relationship of observations.
- How to calculate and plot probability and density functions for the Gaussian distribution.
- The Student's t and Chi-Squared distributions related to the Gaussian distribution.

Let's get started.

10.1 Tutorial Overview

This tutorial is divided into 4 parts; they are:

1. Distributions
2. Gaussian Distribution
3. Student's t-Distribution
4. Chi-Squared Distribution

10.2 Distributions

From a practical perspective, we can think of a distribution as a function that describes the relationship between observations in a sample space. For example, we may be interested in the age of humans, with individual ages representing observations in the domain, and ages 0 to 125 the extent of the sample space. The distribution is a mathematical function that describes the relationship of observations of different heights.

> A distribution is simply a collection of data, or scores, on a variable. Usually, these scores are arranged in order from smallest to largest and then they can be presented graphically.
>
> — Page 6, *Statistics in Plain English*, Third Edition, 2010.

Many data conform to well-known and well-understood mathematical functions, such as the Gaussian distribution. A function can fit the data with a modification of the parameters of the function, such as the mean and standard deviation in the case of the Gaussian. Once a distribution function is known, it can be used as a shorthand for describing and calculating related quantities, such as likelihoods of observations, and plotting the relationship between observations in the domain.

10.2.1 Density Functions

Distributions are often described in terms of their density or density functions. Density functions are functions that describe how the proportion of data or likelihood of the proportion of observations change over the range of the distribution. Two types of density functions are probability density functions and cumulative density functions.

- **Probability Density function**: calculates the probability of observing a given value.
- **Cumulative Density function**: calculates the probability of an observation equal or less than a value.

A probability density function, or PDF, can be used to calculate the likelihood of a given observation in a distribution. It can also be used to summarize the likelihood of observations across the distribution's sample space. Plots of the PDF show the familiar shape of a distribution, such as the bell-curve for the Gaussian distribution. Distributions are often defined in terms of their probability density functions with their associated parameters.

A cumulative density function, or CDF, is a different way of thinking about the likelihood of observed values. Rather than calculating the likelihood of a given observation as with the PDF, the CDF calculates the cumulative likelihood for the observation and all prior observations in the sample space. It allows you to quickly understand and comment on how much of the distribution lies before and after a given value. A CDF is often plotted as a curve from 0 to 1 for the distribution.

Both PDFs and CDFs are continuous functions. The equivalent of a PDF for a discrete distribution is called a probability mass function, or PMF. Next, let's look at the Gaussian distribution and two other distributions related to the Gaussian that you will encounter when using statistical methods. We will look at each in turn in terms of their parameters, probability, and cumulative density functions.

10.3 Gaussian Distribution

The Gaussian distribution, named for Carl Friedrich Gauss, is the focus of much of the field of statistics. Data from many fields of study surprisingly can be described using a Gaussian distribution, so much so that the distribution is often called the *normal* distribution because it is so common. A Gaussian distribution can be described using two parameters:

- `mean`: Denoted with the Greek lowercase letter mu (μ), is the expected value of the distribution.
- `variance`: Denoted with the Greek lowercase letter sigma (σ^2) raised to the second power (because the units of the variable are squared) , describes the spread of observation from the mean.

It is common to use a normalized calculation of the variance called the standard deviation

- `standard deviation`: Denoted with the Greek lowercase letter sigma (σ), describes the normalized spread of observations from the mean.

We can work with the Gaussian distribution via the `norm` SciPy module. The `norm.pdf()` function can be used to create a Gaussian probability density function with a given sample space, mean, and standard deviation. The example below creates a Gaussian PDF with a sample space from -5 to 5, a mean of 0, and a standard deviation of 1. A Gaussian with these values for the mean and standard deviation is called the Standard Gaussian.

```
# plot the gaussian pdf
from numpy import arange
from matplotlib import pyplot
from scipy.stats import norm
# define the distribution parameters
sample_space = arange(-5, 5, 0.001)
mean = 0.0
stdev = 1.0
# calculate the pdf
pdf = norm.pdf(sample_space, mean, stdev)
# plot
pyplot.plot(sample_space, pdf)
pyplot.show()
```

Listing 10.1: Example density line plot of Gaussian probability density function.

Running the example creates a line plot showing the sample space in the x-axis and the likelihood of each value of the y-axis. The line plot shows the familiar bell-shape for the Gaussian distribution. The top of the bell shows the most likely value from the distribution, called the expected value or the mean, which in this case is zero, as we specified in creating the distribution.

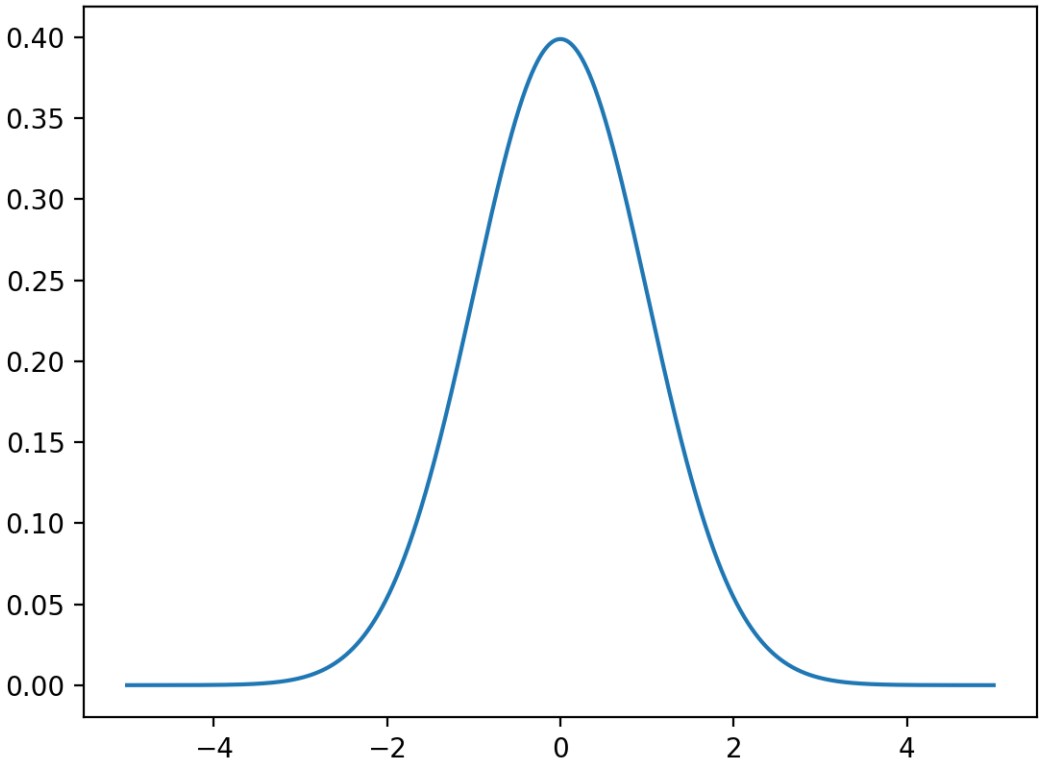

Figure 10.1: Density line plot of the Gaussian probability density function.

The `norm.cdf()` function can be used to create a Gaussian cumulative density function. The example below creates a Gaussian CDF for the same sample space.

```
# plot the gaussian cdf
from numpy import arange
from matplotlib import pyplot
from scipy.stats import norm
# define the distribution parameters
sample_space = arange(-5, 5, 0.001)
# calculate the cdf
cdf = norm.cdf(sample_space)
# plot
pyplot.plot(sample_space, cdf)
pyplot.show()
```

Listing 10.2: Example density line plot of Gaussian cumulative density function.

Running the example creates a plot showing an S-shape with the sample space on the x-axis and the cumulative probability of the y-axis. We can see that a value of 2 covers close to 100% of the observations, with only a very thin tail of the distribution beyond that point. We can also see that the mean value of zero shows 50% of the observations before and after that point.

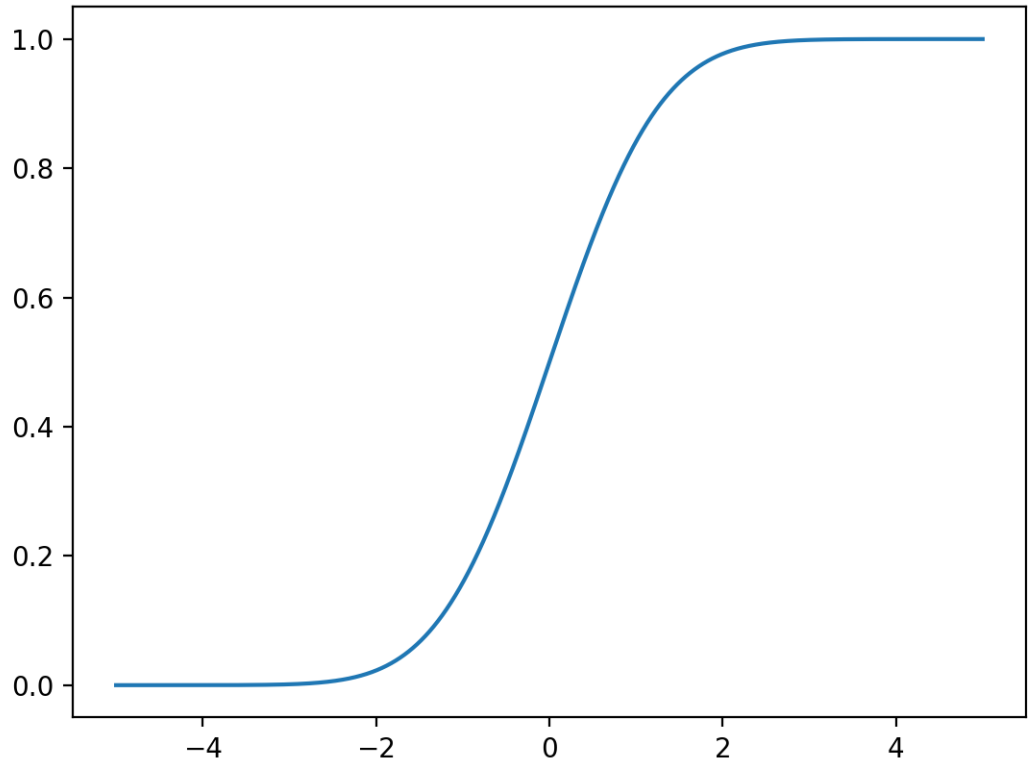

Figure 10.2: Density line plot of the Gaussian cumulative density function.

10.4 Student's t-Distribution

The Student's t-distribution, or just t-distribution for short, is named for the pseudonym *Student* by William Sealy Gosset. It is a distribution that arises when attempting to estimate the mean of a normal distribution with different sized samples. As such, it is a helpful shortcut when describing uncertainty or error related to estimating population statistics for data drawn from Gaussian distributions when the size of the sample must be taken into account.

Although you may not use the Student's t-distribution directly, you may estimate values from the distribution required as parameters in other statistical methods, such as statistical significance tests. The distribution can be described using a single parameter:

- `number of degrees of freedom`: denoted with the lowercase Greek letter nu (ν), denotes the number degrees of freedom.

Key to the use of the t-distribution is knowing the desired number of degrees of freedom. The number of degrees of freedom describes the number of pieces of information used to describe a population quantity. For example, the mean has `n` degrees of freedom as all `n` observations in the sample are used to calculate the estimate of the population mean. A statistical quantity that makes use of another statistical quantity in its calculation must subtract 1 from the degrees

of freedom, such as the use of the mean in the calculation of the sample variance. Observations in a Student's t-distribution are calculated from observations in a normal distribution in order to describe the interval for the populations mean in the normal distribution. Observations are calculated as:

$$data = \frac{x - mean(x)}{\frac{S}{\sqrt{n}}} \tag{10.1}$$

Where x is the observations from the Gaussian distribution, *mean* is the average observation of x, S is the standard deviation and n is the total number of observations. The resulting observations form the t-observation with $(n-1)$ degrees of freedom. In practice, if you require a value from a t-distribution in the calculation of a statistic, then the number of degrees of freedom will likely be $n-1$, where n is the size of your sample drawn from a Gaussian distribution.

> Which specific distribution you use for a given problem depends on the size of your sample.
>
> — Page 93, *Statistics in Plain English*, Third Edition, 2010.

SciPy provides tools for working with the t-distribution in the `stats.t` module. The `t.pdf()` function can be used to create a Student's t-distribution with the specified degrees of freedom. The example below creates a t-distribution using the sample space from -5 to 5 and (10,000 - 1) degrees of freedom.

```
# plot the t-distribution pdf
from numpy import arange
from matplotlib import pyplot
from scipy.stats import t
# define the distribution parameters
sample_space = arange(-5, 5, 0.001)
dof = len(sample_space) - 1
# calculate the pdf
pdf = t.pdf(sample_space, dof)
# plot
pyplot.plot(sample_space, pdf)
pyplot.show()
```

Listing 10.3: Example density line plot of Student's t probability density function.

Running the example creates and plots the t-distribution PDF. We can see the familiar bell-shape to the distribution much like the normal. A key difference is the fatter tails in the distribution (hard to see by eye), highlighting the increased likelihood of observations in the tails compared to that of the Gaussian.

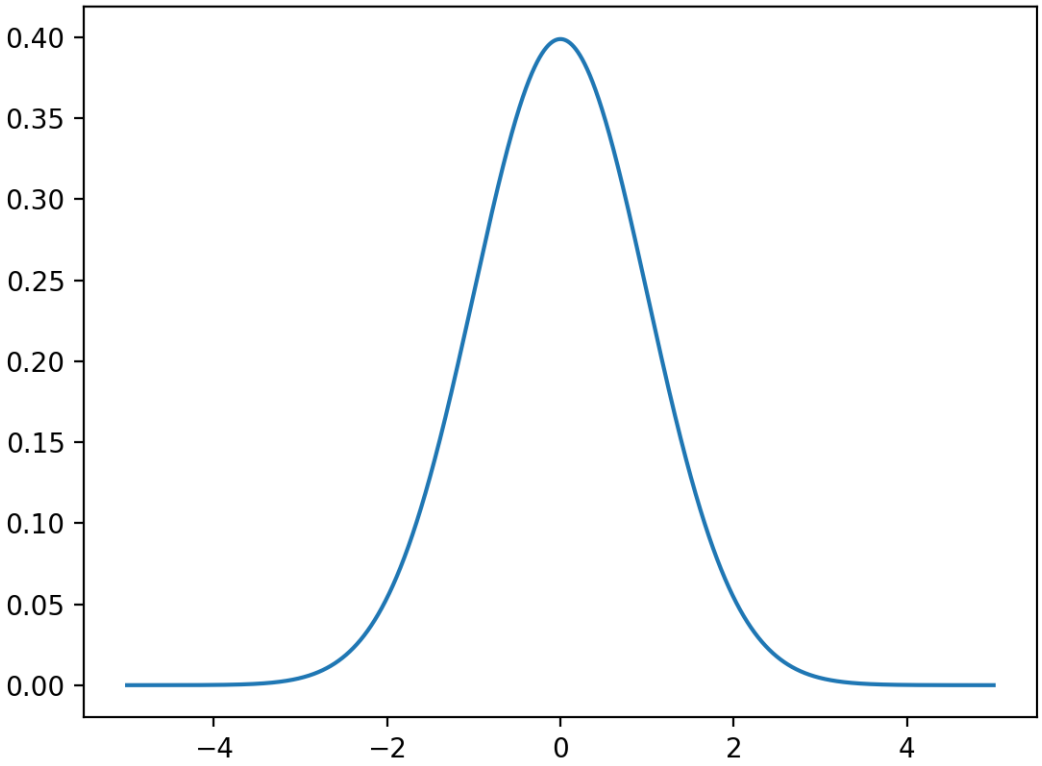

Figure 10.3: Density line plot of the Student's t probability density function.

The `t.cdf()` function can be used to create the cumulative density function for the t-distribution. The example below creates the CDF over the same range as above.

```
# plot the t-distribution cdf
from numpy import arange
from matplotlib import pyplot
from scipy.stats import t
# define the distribution parameters
sample_space = arange(-5, 5, 0.001)
dof = len(sample_space) - 1
# calculate the cdf
cdf = t.cdf(sample_space, dof)
# plot
pyplot.plot(sample_space, cdf)
pyplot.show()
```

Listing 10.4: Example density line plot of Student's t cumulative density function.

Running the example, we see the familiar S-shaped curve as we see with the Gaussian distribution, although with slightly softer transitions from zero-probability to one-probability for the fatter tails.

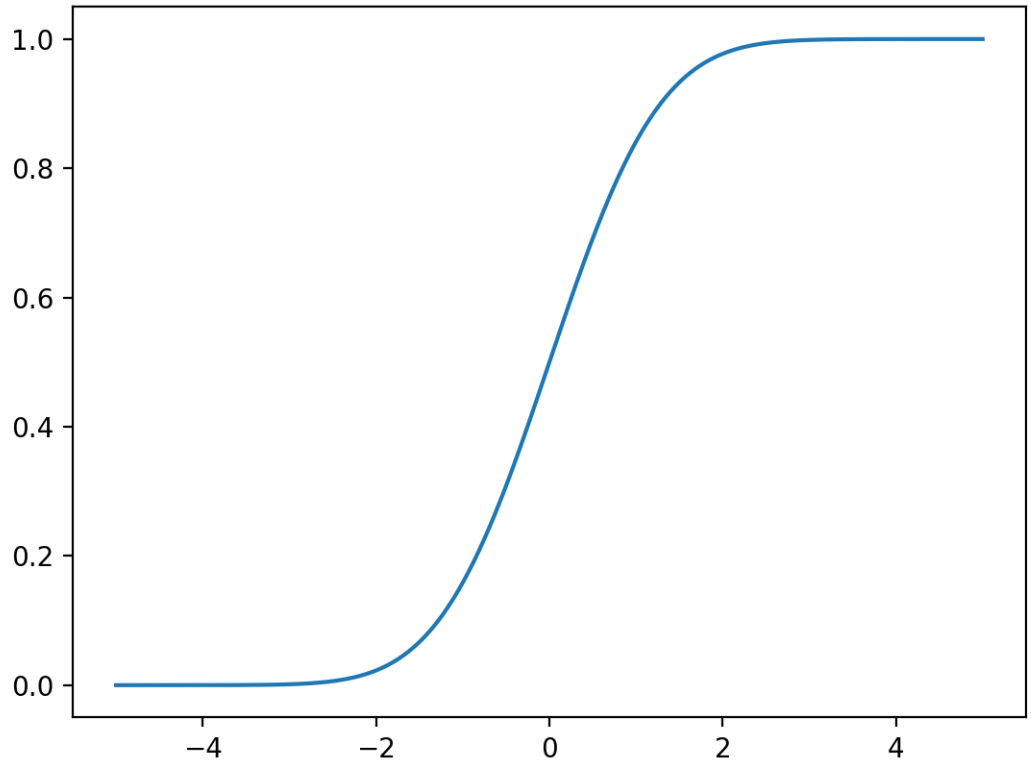

Figure 10.4: Density line plot of the Student's t cumulative density function.

10.5 Chi-Squared Distribution

The Chi-Squared distribution is denoted as the lowercase Greek letter chi (χ) pronounced "ki" as in "kite", raised to the second power (χ^2). It is easier to write *Chi-Squared*. Like the Student's t-distribution, the Chi-Squared distribution is also used in statistical methods on data drawn from a Gaussian distribution to quantify the uncertainty. For example, the Chi-Squared distribution is used in the Chi-Squared statistical tests for independence. In fact, the Chi-Squared distribution is used in the derivation of the Student's t-distribution. The Chi-Squared distribution has one parameter:

- `degrees of freedom`, denoted k.

An observation in a Chi-Squared distribution is calculated as the sum of k squared observations drawn from a Gaussian distribution.

$$chi = \sum_{i=1}^{k} x_i^2 \tag{10.2}$$

Where *chi* is an observation that has a Chi-Squared distribution, x are observation drawn from a Gaussian distribution, and k is the number of x observations which is also the number of degrees

of freedom for the Chi-Squared distribution. Again, as with the Student's t-distribution, data does not fit a Chi-Squared distribution; instead, observations are drawn from this distribution in the calculation of statistical methods for a sample of Gaussian data.

SciPy provides the `stats.chi2` module for calculating statistics for the Chi-Squared distribution. The `chi2.pdf()` function can be used to calculate the Chi-Squared distribution for a sample space between 0 and 50 with 20 degrees of freedom. Recall that the sum squared values must be positive, hence the need for a positive sample space.

```
# plot the chi-squared pdf
from numpy import arange
from matplotlib import pyplot
from scipy.stats import chi2
# define the distribution parameters
sample_space = arange(0, 50, 0.01)
dof = 20
# calculate the pdf
pdf = chi2.pdf(sample_space, dof)
# plot
pyplot.plot(sample_space, pdf)
pyplot.show()
```

Listing 10.5: Example density line plot of Chi-Squared probability density function.

Running the example calculates the Chi-Squared PDF and presents it as a line plot. With 20 degrees of freedom, we can see that the expected value of the distribution is just short of the value 20 on the sample space. This is intuitive if we think most of the density in the Gaussian distribution lies between -1 and 1 and then the sum of the squared random observations from the standard Gaussian would sum to just under the number of degrees of freedom, in this case 20 (approximately 20×1). Although the distribution has a bell-like shape, the distribution is not symmetric.

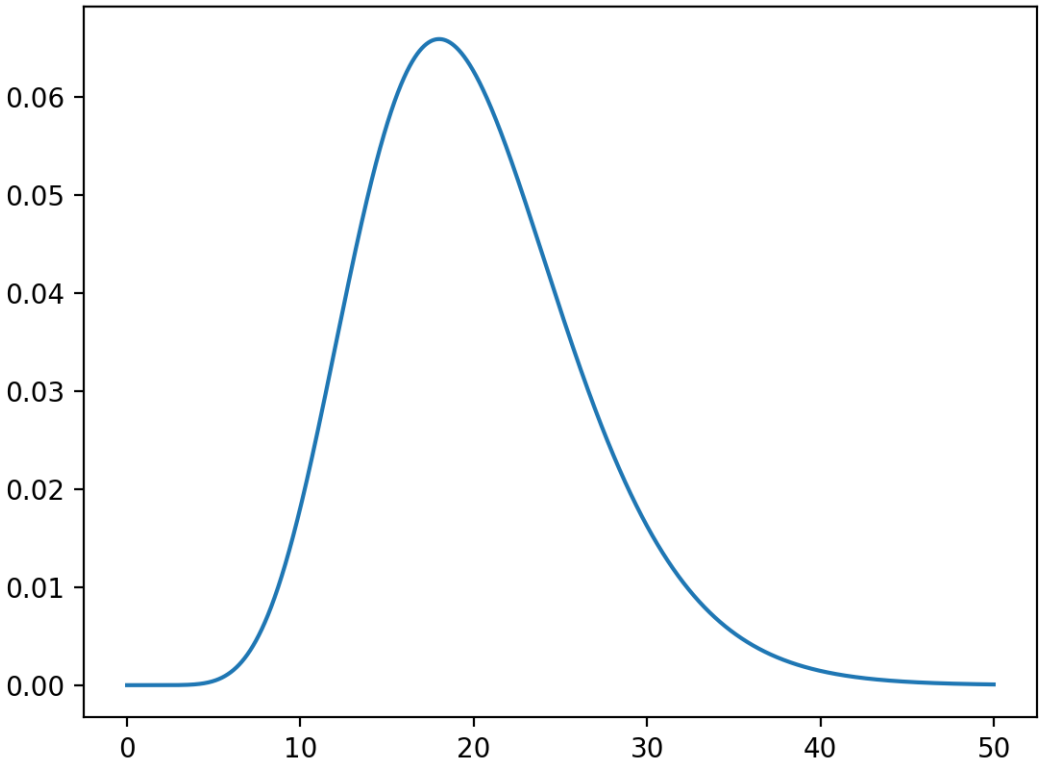

Figure 10.5: Density line plot of the Chi-Squared probability density function.

The `chi2.cdf()` function can be used to calculate the cumulative density function over the same sample space.

```
# plot the chi-squared cdf
from numpy import arange
from matplotlib import pyplot
from scipy.stats import chi2
# define the distribution parameters
sample_space = arange(0, 50, 0.01)
dof = 20
# calculate the cdf
cdf = chi2.cdf(sample_space, dof)
# plot
pyplot.plot(sample_space, cdf)
pyplot.show()
```

Listing 10.6: Example density line plot of Chi-Squared cumulative density function.

Running the example creates a plot of the cumulative density function for the Chi-Squared distribution. The distribution helps to see the likelihood for the Chi-Squared value around 20 with the fat tail to the right of the distribution that would continue on long after the end of the plot.

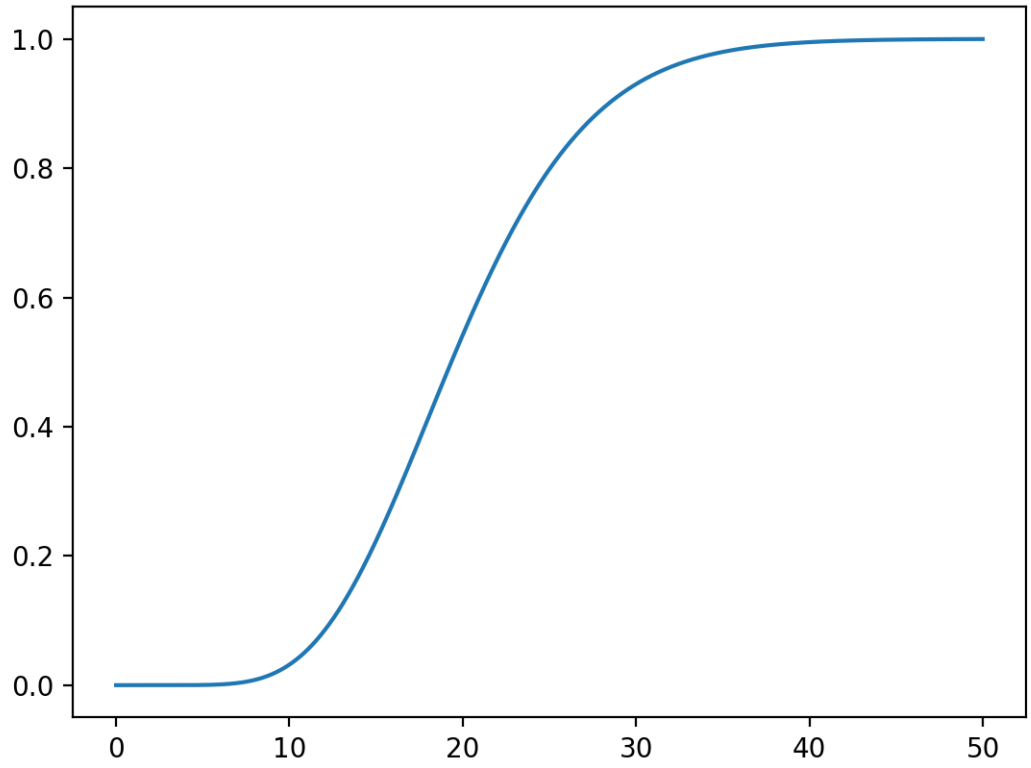

Figure 10.6: Density line plot of the Chi-Squared cumulative density function.

10.6 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Recreate the PDF and CDF plots for one distribution with a new sample space.
- Calculate and plot the PDF and CDF for the Cauchy and Laplace distributions.
- Look up and implement the equations for the PDF and CDF for one distribution from scratch.

If you explore any of these extensions, I'd love to know.

10.7 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

10.7.1 Books

- *Statistics in Plain English*, Third Edition, 2010.
  http://amzn.to/2FTs5TB

10.7.2 API

- Statistics (`scipy.stats`).
  https://docs.scipy.org/doc/scipy/reference/tutorial/stats.html
- `scipy.stats.norm` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html
- `scipy.stats.t` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.t.html
- `scipy.stats.chi2` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chi2.html

10.7.3 Articles

- Probability density function on Wikipedia.
  https://en.wikipedia.org/wiki/Probability_density_function
- Cumulative distribution function on Wikipedia.
  https://en.wikipedia.org/wiki/Cumulative_distribution_function
- Probability mass function on Wikipedia.
  https://en.wikipedia.org/wiki/Probability_mass_function
- Normal distribution on Wikipedia.
  https://en.wikipedia.org/wiki/Normal_distribution
- Student's t-distribution on Wikipedia.
  https://en.wikipedia.org/wiki/Student%27s_t-distribution
- Chi-Squared distribution on Wikipedia.
  https://en.wikipedia.org/wiki/Chi-squared_distribution

10.8 Summary

In this tutorial, you discovered the Gaussian and related distribution functions and how to calculate probability and cumulative density functions for each. Specifically, you learned:

- A gentle introduction to standard distributions to summarize the relationship of observations.
- How to calculate and plot probability and density functions for the Gaussian distribution.
- The Student's t and Chi-Squared distributions related to the Gaussian distribution.

10.8.1 Next

In the next section, you will discover critical values in the key distributions and how to calculate them.

Chapter 11

Critical Values

It is common, if not standard, to interpret the results of statistical hypothesis tests using a p-value. Not all implementations of statistical tests return p-values. In some cases, you must use alternatives, such as critical values. In addition, critical values are used when estimating the expected intervals for observations from a population, such as in tolerance intervals. In this tutorial, you will discover critical values, why they are important, how they are used, and how to calculate them in Python using SciPy. After completing this tutorial, you will know:

- Examples of statistical hypothesis tests and their distributions from which critical values can be calculated and used.
- How exactly critical values are used on one-tail and two-tail statistical hypothesis tests.
- How to calculate critical values for the Gaussian, Student's t, and Chi-Squared distributions.

Let's get started.

11.1 Tutorial Overview

This tutorial is divided into 4 parts; they are:

1. Why Do We Need Critical Values?
2. What Is a Critical Value?
3. How to Use Critical Values
4. How to Calculate Critical Values

11.2 Why Do We Need Critical Values?

Many statistical hypothesis tests return a p-value that is used to interpret the outcome of the test. Some tests do not return a p-value, requiring an alternative method for interpreting the calculated test statistic directly. A statistic calculated by a statistical hypothesis test can be interpreted using critical values from the distribution of the test statistic. Some examples of

statistical hypothesis tests and their distributions from which critical values can be calculated are as follows:

- **Z-Test**: Gaussian distribution.
- **Student's t-Test**: Student's t-distribution.
- **Chi-Squared Test**: Chi-Squared distribution.
- **ANOVA**: F-distribution.

Critical values are also used when defining intervals for expected (or unexpected) observations in distributions. Calculating and using critical values may be appropriate when quantifying the uncertainty of estimated statistics or intervals such as confidence intervals and tolerance intervals. Note, a p-value can be calculated from a test statistic by retrieving the probability from the test statistics cumulative density function (CDF).

11.3 What Is a Critical Value?

A critical value is defined in the context of the population distribution and a probability. An observation from the population with a value equal to or lesser than a critical value with the given probability. We can express this mathematically as follows:

$$Pr[X \leq \text{critical value}] = probability \tag{11.1}$$

Where `Pr` is the calculation of probability, X are observations from the population, `critical value` is the calculated critical value, and probability is the chosen probability. Critical values are calculated using a mathematical function where the probability is provided as an argument. For most common distributions, the value cannot be calculated analytically; instead it must be estimated using numerical methods. Historically it is common for tables of pre-calculated critical values to be provided in the appendices of statistics textbooks for reference purposes. Critical values are used in statistical significance testing. The probability is often expressed as a significance, denoted as the lowercase Greek letter alpha (α), which is the inverted probability.

$$probability = 1 - alpha \tag{11.2}$$

Standard alpha values are used when calculating critical values, chosen for historical reasons and continually used for consistency reasons. These alpha values include:

- 1% (alpha=0.01)
- 5% (alpha=0.05)
- 10% (alpha=0.10)

Critical values provide an alternative and equivalent way to interpret statistical hypothesis tests to the p-value.

11.4 How to Use Critical Values

Calculated critical values are used as a threshold for interpreting the result of a statistical test. The observation values in the population beyond the critical value are often called the *critical region* or the *region of rejection*.

> Critical Value: A value appearing in tables for specified statistical tests indicating at what computed value the null hypothesis can be rejected (the computed statistic falls in the rejection region).

— Page 265, *Handbook of Research Methods: A Guide for Practitioners and Students in the Social Sciences*, 2003.

A statistical test may be one-tailed or two-tailed.

11.4.1 One-Tailed Test

A one-tailed test has a single critical value, such as on the left or the right of the distribution. Often, a one-tailed test has a critical value on the right of the distribution for non-symmetrical distributions (such as the Chi-Squared distribution). The statistic is compared to the calculated critical value. If the statistic is less than or equal to the critical value, the null hypothesis of the statistical test rejected or it is failed to be rejected. We can summarize this interpretation as follows:

- `Test Statistic` $<$ `Critical Value`: not significant result, fail to reject null hypothesis (H0).
- `Test Statistic` $\geq$ `Critical Value`: significant result, reject null hypothesis (H0).

11.4.2 Two-Tailed Test

A two-tailed test has two critical values, one on each side of the distribution, which is often assumed to be symmetrical (e.g. Gaussian and Student-t distributions.). When using a two-tailed test, a significance level (or alpha) used in the calculation of the critical values must be divided by 2. The critical value will then use a portion of this alpha on each side of the distribution. To make this concrete, consider an alpha of 5%. This would be split to give two alpha values of 2.5% on either side of the distribution with an acceptance area in the middle of the distribution of 95%.

We can refer to each critical value as the lower and upper critical values for the left and right of the distribution respectively. Test statistic values more than or equal to the lower critical value and less than or equal to the upper critical value indicate the failure to reject the null hypothesis. Whereas test statistic values less than the lower critical value and more than the upper critical value indicate rejection of the null hypothesis for the test. We can summarize this interpretation as follows:

- `Lower CR` $<$ `Test Statistic` $>$ `Upper CR`: not significant result, fail to reject null hypothesis (H0).

- `Test Statistic` $\leq$ `Lower CR OR Test Statistic` $\geq$ `Upper CR`: significant result, reject null hypothesis (H0).

If the distribution of the test statistic is symmetric around a mean of zero, then we can shortcut the check by comparing the absolute (positive) value of the test statistic to the upper critical value.

- |`Test Statistic`| $<$ `Upper Critical Value`: not significant result, fail to reject null hypothesis (H0), distributions same.
- |`Test Statistic`| $\geq$ `Upper Critical Value`: significant result, reject null hypothesis (H0), distributions differ.

Where |`Test Statistic`| is the absolute value of the calculated test statistic.

11.5 How to Calculate Critical Values

Density functions return the probability of an observation in the distribution. Recall the definitions of the PDF and CDF as follows:

- **Probability Density Function (PDF)**: Returns the probability for an observation having a specific value from the distribution.
- **Cumulative Density Function (CDF)**: Returns the probability for an observation equal to or lesser than a specific value from the distribution.

In order to calculate a critical value, we require a function that, given a probability (or significance), will return the observation value from the distribution. Specifically, we require the inverse of the cumulative density function, where given a probability, we are given the observation value that is less than or equal to the probability. This is called the percent point function (PPF), or more generally the quantile function.

- **Percent Point Function (PPF)**: Returns the observation value for the provided probability that is less than or equal to the provided probability from the distribution.

Specifically, a value from the distribution will equal or be less than the value returned from the PPF with the specified probability. Let's make this concrete with three distributions from which it is commonly required to calculate critical values. Namely, the Gaussian distribution, Student's t-distribution, and the Chi-Squared distribution. We can calculate the percent point function in SciPy using the `ppf()` function on a given distribution. It should also be noted that you can also calculate the `ppf()` using the inverse survival function called `isf()` in SciPy. This is mentioned as you may see use of this alternate approach in third party code.

11.5.1 Gaussian Critical Values

The example below calculates the percent point function for 95% on the standard Gaussian distribution.

```
# gaussian percent point function
from scipy.stats import norm
# define probability
p = 0.95
# retrieve value <= probability
value = norm.ppf(p)
print(value)
# confirm with cdf
p = norm.cdf(value)
print(p)
```

Listing 11.1: Example of calculating critical values for the Gaussian distribution.

Running the example first prints the value that marks 95% or less of the observations from the distribution of about 1.65. This value is then confirmed by retrieving the probability of the observation from the CDF, which returns 95%, as expected. We can see that the value 1.65 aligns with our expectation with regard to the number of standard deviations from the mean that cover 95% of the distribution in the 68-95-99.7 rule (linked in the Further Reading section).

```
1.6448536269514722
0.95
```

Listing 11.2: Sample output from calculating critical values for the Gaussian distribution.

11.5.2 Student's t Critical Values

The example below calculates the percentage point function for 95% on the standard Student's t-distribution with 10 degrees of freedom.

```
# student t-distribution percent point function
from scipy.stats import t
# define probability
p = 0.95
df = 10
# retrieve value <= probability
value = t.ppf(p, df)
print(value)
# confirm with cdf
p = t.cdf(value, df)
print(p)
```

Listing 11.3: Example of calculating critical values for the t distribution.

Running the example returns the value of about 1.812 or less that covers 95% of the observations from the chosen distribution. The probability of the value is then confirmed (with minor rounding error) via the CDF.

```
1.8124611228107335
0.949999999999923
```

Listing 11.4: Sample output from calculating critical values for the t distribution.

11.5.3 Chi-Squared Critical Values

The example below calculates the percentage point function for 95% on the standard Chi-Squared distribution with 10 degrees of freedom.

```
# chi-squared percent point function
from scipy.stats import chi2
# define probability
p = 0.95
df = 10
# retrieve value <= probability
value = chi2.ppf(p, df)
print(value)
# confirm with cdf
p = chi2.cdf(value, df)
print(p)
```

Listing 11.5: Example of calculating critical values for the Chi-Squared distribution.

Running the example first calculates the value of 18.3 or less that covers 95% of the observations from the distribution. The probability of this observation is confirmed by using it as input to the CDF.

```
18.307038053275143
0.95
```

Listing 11.6: Sample output from calculating critical values for the Chi-Squared distribution.

11.6 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Calculate critical values for 90%, 95% and 99% for each of the Gaussian, Student's t, and Chi-Squared distributions.
- Develop an example to calculate critical values from another distribution, such as the F distribution.
- Write code to calculate a p-value from a critical value for each of the Gaussian, Student's t, and Chi-Squared distributions.

If you explore any of these extensions, I'd love to know.

11.7 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

11.7.1 Books

- *Handbook of Research Methods: A Guide for Practitioners and Students in the Social Sciences*, 2003.
  http://amzn.to/2G4vG4k

11.7.2 API

- `scipy.stats.norm` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html
- `scipy.stats.t` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.t.html
- `scipy.stats.chi2` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chi2.html

11.7.3 Articles

- Critical value on Wikipedia.
  https://en.wikipedia.org/wiki/Critical_value
- P-value on Wikipedia.
  https://en.wikipedia.org/wiki/P-value
- One- and two-tailed tests on Wikipedia.
  https://en.wikipedia.org/wiki/One-_and_two-tailed_tests
- Quantile function on Wikipedia.
  https://en.wikipedia.org/wiki/Quantile_function
- 68-95-99.7 rule on Wikipedia.
  https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule

11.8 Summary

In this tutorial, you discovered critical values, why they are important, how they are used, and how to calculate them in Python using SciPy. Specifically, you learned:

- Examples of statistical hypothesis tests and their distributions from which critical values can be calculated and used.
- How exactly critical values are used on one-tail and two-tail statistical hypothesis tests.
- How to calculate critical values for the Gaussian, Student's t, and Chi-Squared distributions.

11.8.1 Next

In the next section, you will discover how to quantify the relationship between samples of data.

Chapter 12

Covariance and Correlation

There may be complex and unknown relationships between the variables in your dataset. It is important to discover and quantify the degree to which variables in your dataset are dependent upon each other. This knowledge can help you better prepare your data to meet the expectations of machine learning algorithms, such as linear regression, whose performance will degrade with the presence of these interdependencies. In this tutorial, you will discover that correlation is the statistical summary of the relationship between variables and how to calculate it for different types variables and relationships. After completing this tutorial, you will know:

- How to calculate a covariance matrix to summarize the linear relationship between two or more variables.
- How to calculate the covariance to summarize the linear relationship between two variables.
- How to calculate the Pearson's correlation coefficient to summarize the linear relationship between two variables.

Let's get started.

12.1 Tutorial Overview

This tutorial is divided into 4 parts; they are:

- What is Correlation?
- Test Dataset
- Covariance
- Pearson's Correlation

12.2 What is Correlation?

Variables within a dataset can be related for lots of reasons. For example:

- One variable could cause or depend on the values of another variable.

- One variable could be lightly associated with another variable.
- Two variables could depend on a third unknown variable.

It can be useful in data analysis and modeling to better understand the relationships between variables. The statistical relationship between two variables is referred to as their correlation. A correlation could be positive, meaning both variables move in the same direction, or negative, meaning that when one variable's value increases, the other variables' values decrease. Correlation can also be neural or zero, meaning that the variables are unrelated.

- **Positive Correlation**: Both variables change in the same direction.
- **Neutral Correlation**: No relationship in the change of the variables.
- **Negative Correlation**: Variables change in opposite directions.

The performance of some algorithms can deteriorate if two or more variables are tightly related, called multicollinearity. An example is linear regression, where one of the offending correlated variables should be removed in order to improve the skill of the model. We may also be interested in the correlation between input variables with the output variable in order provide insight into which variables may or may not be relevant as input for developing a model.

The structure of the relationship may be known, e.g. it may be linear, or we may have no idea whether a relationship exists between variables or what structure it may take. Depending what is known about the relationship and the distribution of the variables, different correlation scores can be calculated. In this tutorial, we will look at one score for variables that have a Gaussian distribution and a linear relationship and another that does not assume a distribution and will report on any monotonic (increasing or decreasing) relationship.

12.3 Test Dataset

Before we look at correlation methods, let's define a dataset we can use to test the methods. We will generate 1,000 samples of two two variables with a strong positive correlation. The first variable will be random numbers drawn from a Gaussian distribution with a mean of 100 and a standard deviation of 20. The second variable will be values from the first variable with Gaussian noise added with a mean of a 50 and a standard deviation of 10.

We will use the `randn()` function to generate random Gaussian values with a mean of 0 and a standard deviation of 1, then multiply the results by our own standard deviation and add the mean to shift the values into the preferred range. The pseudorandom number generator is seeded to ensure that we get the same sample of numbers each time the code is run.

```
# generate related variables
from numpy import mean
from numpy import std
from numpy.random import randn
from numpy.random import seed
from matplotlib import pyplot
# seed random number generator
seed(1)
# prepare data
```

```
data1 = 20 * randn(1000) + 100
data2 = data1 + (10 * randn(1000) + 50)
# summarize
print('data1: mean=%.3f stdv=%.3f' % (mean(data1), std(data1)))
print('data2: mean=%.3f stdv=%.3f' % (mean(data2), std(data2)))
# plot
pyplot.scatter(data1, data2)
pyplot.show()
```

Listing 12.1: Example of preparing a test dataset with correlated variables.

Running the example first prints the mean and standard deviation for each variable.

```
data1: mean=100.776 stdv=19.620
data2: mean=151.050 stdv=22.358
```

Listing 12.2: Sample output from preparing a test dataset with correlated variables.

A scatter plot of the two variables is created. Because we contrived the dataset, we know there is a relationship between the two variables. This is clear when we review the generated scatter plot where we can see an increasing trend.

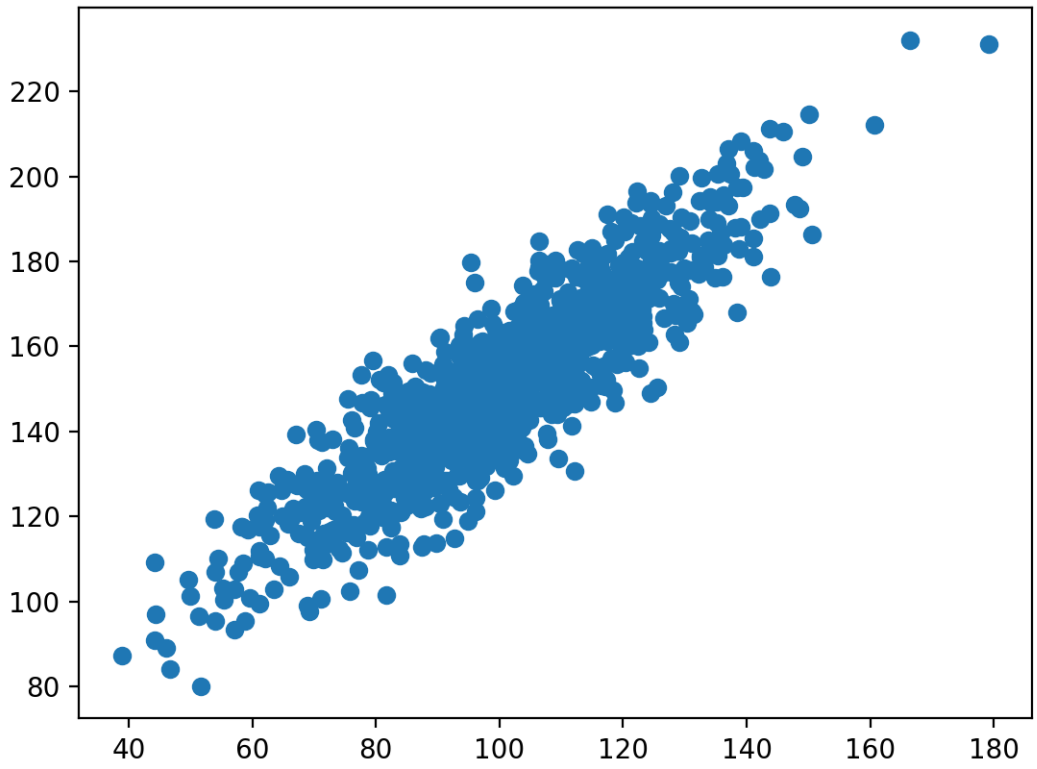

Figure 12.1: Scatter plot of the test correlation dataset.

Before we look at calculating some correlation scores, we must first look at an important statistical building block, called covariance.

12.4 Covariance

Variables can be related by a linear relationship. This is a relationship that is consistently additive across the two data samples. This relationship can be summarized between two variables, called the covariance. It is calculated as the average of the product between the values from each sample, where the values haven been centered (had their mean subtracted). The calculation of the sample covariance is as follows:

$$cov(x, y) = \frac{1}{n} \times \sum_{i=1}^{n}(x_i - mean(x)) \times (y_i - mean(y)) \tag{12.1}$$

The use of the mean in the calculation suggests the need for each data sample to have a Gaussian or Gaussian-like distribution. The sign of the covariance can be interpreted as whether the two variables change in the same direction (positive) or change in different directions (negative). The magnitude of the covariance is not easily interpreted. A covariance value of zero indicates that both variables are completely independent. The `cov()` NumPy function can be used to calculate a covariance matrix between two or more variables.

```
...
# calculate the covariance between two samples
covariance = cov(data1, data2)
```

Listing 12.3: Example of calculating covariance between two samples.

The main diagonal of the matrix contains the covariance between each variable and itself. The other values in the matrix represent the covariance between the two variables; in this case, the remaining two values are the same given that we are calculating the covariance for only two variables. We can calculate the covariance matrix for the two variables in our test problem. The complete example is listed below.

```
# calculate the covariance between two variables
from numpy.random import randn
from numpy.random import seed
from numpy import cov
# seed random number generator
seed(1)
# prepare data
data1 = 20 * randn(1000) + 100
data2 = data1 + (10 * randn(1000) + 50)
# calculate covariance matrix
covariance = cov(data1, data2)
print(covariance)
```

Listing 12.4: Example of calculating covariance on the test dataset.

The covariance and covariance matrix are used widely within statistics and multivariate analysis to characterize the relationships between two or more variables. Running the example calculates and prints the covariance matrix. Because the dataset was contrived with each variable drawn from a Gaussian distribution and the variables linearly correlated, covariance is a reasonable method for describing the relationship. The covariance between the two variables is 389.75. We can see that it is positive, suggesting the variables change in the same direction as we expect.

```
[[385.33297729 389.7545618 ]
 [389.7545618 500.38006058]]
```

Listing 12.5: Sample output from calculating covariance on the test dataset.

A problem with covariance as a statistical tool alone is that it is challenging to interpret. This leads us to the Pearson's correlation coefficient next.

12.5 Pearson's Correlation

The Pearson's correlation coefficient (named for Karl Pearson) can be used to summarize the strength of the linear relationship between two data samples. The Pearson's correlation coefficient is calculated as the covariance of the two variables divided by the product of the standard deviation of each data sample. It is the normalization of the covariance between the two variables to give an interpretable score.

$$\text{Pearson's correlation coefficient} = \frac{cov(x, y)}{stdev(x) \times stdev(y)} \tag{12.2}$$

The use of mean and standard deviation in the calculation suggests the need for the two data samples to have a Gaussian or Gaussian-like distribution. The result of the calculation, the correlation coefficient can be interpreted to understand the relationship. The coefficient returns a value between -1 and 1 that represents the limits of correlation from a full negative correlation to a full positive correlation. A value of 0 means no correlation. The value must be interpreted, where often a value below -0.5 or above 0.5 indicates a notable correlation, and values below those values suggests a less notable correlation. See the table below to help with interpretation the correlation coefficient.

TABLE 7.1

Correlation Coefficient for a Direct Relationship	Correlation Coefficient for an Indirect Relationship	Relationship Strength of the Variables
0.0	0.0	None/trivial
0.1	−0.1	Weak/small
0.3	−0.3	Moderate/medium
0.5	−0.5	Strong/large
1.0	−1.0	Perfect

Figure 12.2: Table of correlation coefficient values and their interpretation. Taken from *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach.*

The Pearson's correlation is a statistical hypothesis test that does assume that there is no relationship between the samples (null hypothesis). The p-value can be interpreted as follows:

- `p-value` $\leq$ `alpha`: significant result, reject null hypothesis, some relationship (H1).
- `p-value` $>$ `alpha`: not significant result, fail to reject null hypothesis, no relationship (H0).

The `pearsonr()` SciPy function can be used to calculate the Pearson's correlation coefficient between two data samples with the same length. We can calculate the correlation between the two variables in our test problem. The complete example is listed below.

```
# calculate the pearson's correlation between two variables
from numpy.random import randn
from numpy.random import seed
from scipy.stats import pearsonr
# seed random number generator
seed(1)
# prepare data
data1 = 20 * randn(1000) + 100
data2 = data1 + (10 * randn(1000) + 50)
# calculate Pearson's correlation
corr, p = pearsonr(data1, data2)
# display the correlation
print('Pearsons correlation: %.3f' % corr)
# interpret the significance
alpha = 0.05
if p > alpha:
  print('No correlation (fail to reject H0)')
else:
  print('Some correlation (reject H0)')
```

Listing 12.6: Example of calculating correlation on the test dataset.

Running the example calculates and prints the Pearson's correlation coefficient and interprets the p-value. We can see that the two variables are positively correlated and that the correlation is 0.888. This suggests a high level of correlation (as we expected).

```
Pearsons correlation: 0.888
Some correlation (reject H0)
```

Listing 12.7: Sample output from calculating correlation on the test dataset.

The Pearson's correlation coefficient can be used to evaluate the relationship between more than two variables. This can be done by calculating a matrix of the relationships between each pair of variables in the dataset. The result is a symmetric matrix called a correlation matrix with a value of 1.0 along the main diagonal as each column always perfectly correlates with itself.

12.6 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Generate your own datasets with positive and negative relationships and calculate both correlation coefficients.
- Write functions to calculate Pearson's correlation coefficient for a provided dataset.
- Load a standard machine learning dataset and calculate correlation coefficients between all pairs of real-valued variables.

If you explore any of these extensions, I'd love to know.

12.7 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

12.7.1 API

- `numpy.cov` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.cov.html
- `scipy.stats.pearsonr` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html

12.7.2 Articles

- Correlation and dependence on Wikipedia.
  https://en.wikipedia.org/wiki/Correlation_and_dependence
- Covariance on Wikipedia.
  https://en.wikipedia.org/wiki/Covariance
- Pearson's correlation coefficient on Wikipedia.
  https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

12.8 Summary

In this tutorial, you discovered that correlation is the statistical summary of the relationship between variables and how to calculate it for different types variables and relationships. Specifically, you learned:

- How to calculate a covariance matrix to summarize the linear relationship between two or more variables.
- How to calculate the covariance to summarize the linear relationship between two variables.
- How to calculate the Pearson's correlation coefficient to summarize the linear relationship between two variables.

12.8.1 Next

In the next section, you will discover how to calculate statistical hypothesis tests in Python.

Chapter 13

Significance Tests

Parametric statistical methods often mean those methods that assume the data samples have a Gaussian distribution. In applied machine learning, we need to compare data samples, specifically the mean of the samples. Perhaps to see if one technique performs better than another on one or more datasets. To quantify this question and interpret the results, we can use parametric hypothesis testing methods such as the Student's t-test and ANOVA. In this tutorial, you will discover parametric statistical significance tests that quantify the difference between the means of two or more samples of data. After completing this tutorial, you will know:

- The Student's t-test for quantifying the difference between the mean of two independent data samples.
- The paired Student's t-test for quantifying the difference between the mean of two dependent data samples.
- The ANOVA and repeated measures ANOVA for checking the similarity or difference between the means of 2 or more data samples.

Let's get started.

13.1 Tutorial Overview

This tutorial is divided into 6 parts, they are:

1. Parametric Statistical Significance Tests
2. Test Data
3. Student's t-Test
4. Paired Student's t-Test
5. Analysis of Variance Test
6. Repeated Measures ANOVA Test

13.2 Parametric Statistical Significance Tests

Parametric statistical tests assume that a data sample was drawn from a specific population distribution. They often refer to statistical tests that assume the Gaussian distribution. Because it is so common for data to fit this distribution, parametric statistical methods are more commonly used. A typical question we may have about two or more samples of data is whether they have the same distribution. Parametric statistical significance tests are those statistical methods that assume data comes from the same Gaussian distribution, that is a data distribution with the same mean and standard deviation: the parameters of the distribution.

In general, each test calculates a test statistic that must be interpreted with some background in statistics and a deeper knowledge of the statistical test itself. Tests also return a p-value that can be used to interpret the result of the test. The p-value can be thought of as the probability of observing the two data samples given the base assumption (null hypothesis) that the two samples were drawn from a population with the same distribution. The p-value can be interpreted in the context of a chosen significance level called alpha. A common value for alpha is 5%, or 0.05. If the p-value is below the significance level, then the test says there is enough evidence to reject the null hypothesis and that the samples were likely drawn from populations with differing distributions.

- `p-value` $\leq$ `alpha`: significant result, reject null hypothesis (H0), distributions differ.
- `p-value` $>$ `alpha`: not significant result, fail to reject null hypothesis (H0), distributions same.

13.3 Test Data

Before we look at specific parametric significance tests, let's first define a test dataset that we can use to demonstrate each test. We will generate two samples drawn from different distributions. Each sample will be drawn from a Gaussian distribution. We will use the `randn()` NumPy function to generate a sample of 100 Gaussian random numbers in each sample with a mean of 0 and a standard deviation of 1. Observations in the first sample are scaled to have a mean of 50 and a standard deviation of 5. Observations in the second sample are scaled to have a mean of 51 and a standard deviation of 5.

We expect the statistical tests to discover that the samples were drawn from differing distributions, although the small sample size of 100 observations per sample will add some noise to this decision. The complete code example is listed below.

```
# generate gaussian data samples
from numpy.random import seed
from numpy.random import randn
from numpy import mean
from numpy import std
# seed the random number generator
seed(1)
# generate two sets of univariate observations
data1 = 5 * randn(100) + 50
data2 = 5 * randn(100) + 51
# summarize
print('data1: mean=%.3f stdv=%.3f' % (mean(data1), std(data1)))
```

```
print('data2: mean=%.3f stdv=%.3f' % (mean(data2), std(data2)))
```

Listing 13.1: Example of test datasets.

Running the example generates the data samples, then calculates and prints the mean and standard deviation for each sample, confirming their different distribution.

```
data1: mean=50.303 stdv=4.426
data2: mean=51.764 stdv=4.660
```

Listing 13.2: Sample output from preparing a test dataset.

13.4 Student's t-Test

The Student's t-test is a statistical hypothesis test that two independent data samples known to have a Gaussian distribution, have the same Gaussian distribution, named for William Gosset, who used the pseudonym *Student*.

> One of the most commonly used t-tests is the independent samples t-test. You use this test when you want to compare the means of two independent samples on a given variable.
>
> — Page 93, *Statistics in Plain English*, Third Edition, 2010.

The assumption or null hypothesis of the test is that the means of two populations are equal. A rejection of this hypothesis indicates that there is sufficient evidence that the means of the populations are different, and in turn that the distributions are not equal.

- **Fail to Reject H0**: No difference between the sample means.
- **Reject H0**: Some difference between the sample means.

The Student's t-test is available in Python via the `ttest_ind()` SciPy function. The function takes two data samples as arguments and returns the calculated statistic and p-value. The test assumes that both samples have the same variance, if this is not the case, a corrected version of the test can be used by setting the `equal_var` to `False`. We can demonstrate the Student's t-test on the test problem with the expectation that the test discovers the difference in distribution between the two independent samples. The complete code example is listed below.

```
# student's t-test
from numpy.random import seed
from numpy.random import randn
from scipy.stats import ttest_ind
# seed the random number generator
seed(1)
# generate two independent samples
data1 = 5 * randn(100) + 50
data2 = 5 * randn(100) + 51
# compare samples
stat, p = ttest_ind(data1, data2)
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
```

```
alpha = 0.05
if p > alpha:
  print('Same distributions (fail to reject H0)')
else:
  print('Different distributions (reject H0)')
```

Listing 13.3: Example of Student's t-test on the test dataset.

Running the example calculates the Student's t-test on the generated data samples and prints the statistic and p-value. The interpretation of the statistic finds that the sample means are different, with a significance of at least 5%.

```
Statistics=-2.262, p=0.025
Different distributions (reject H0)
```

Listing 13.4: Sample output from the Student's t-test on the test dataset.

13.5 Paired Student's t-Test

We may wish to compare the means between two data samples that are related in some way. For example, the data samples may represent two independent measures or evaluations of the same object. These data samples are repeated or dependent and are referred to as paired samples or repeated measures. Because the samples are not independent, we cannot use the Student's t-test. Instead, we must use a modified version of the test that corrects for the fact that the data samples are dependent, called the paired Student's t-test.

> A dependent samples t-test is also used to compare two means on a single dependent variable. Unlike the independent samples test, however, a dependent samples t-test is used to compare the means of a single sample or of two matched or paired samples.
>
> — Page 94, *Statistics in Plain English*, Third Edition, 2010.

The test is simplified because it no longer assumes that there is variation between the observations, that observations were made in pairs, before and after a treatment on the same subject or subjects. The default assumption, or null hypothesis of the test, is that there is no difference in the means between the samples. The rejection of the null hypothesis indicates that there is enough evidence that the sample means are different.

- **Fail to Reject H0**: Paired sample distributions are equal.
- **Reject H0**: Paired sample distributions are not equal.

The paired Student's t-test can be implemented in Python using the `ttest_rel()` SciPy function. As with the unpaired version, the function takes two data samples as arguments and returns the calculated statistic and p-value. We can demonstrate the paired Student's t-test on the test dataset. Although the samples are independent, not paired, we can pretend for the sake of the demonstration that the observations are paired and calculate the statistic. The complete example is listed below.

```
# paired student's t-test
from numpy.random import seed
from numpy.random import randn
from scipy.stats import ttest_rel
# seed the random number generator
seed(1)
# generate two independent samples
data1 = 5 * randn(100) + 50
data2 = 5 * randn(100) + 51
# compare samples
stat, p = ttest_rel(data1, data2)
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
  print('Same distributions (fail to reject H0)')
else:
  print('Different distributions (reject H0)')
```

Listing 13.5: Example of a paired Student's t-test on the test dataset.

Running the example calculates and prints the test statistic and p-value. The interpretation of the result suggests that the samples have different means and therefore different distributions.

```
Statistics=-2.372, p=0.020
Different distributions (reject H0)
```

Listing 13.6: Sample output from the paired Student's t-test on the test dataset.

13.6 Analysis of Variance Test

There are sometimes situations where we may have multiple independent data samples. We can perform the Student's t-test pairwise on each combination of the data samples to get an idea of which samples have different means. This can be onerous if we are only interested in whether all samples have the same distribution or not. To answer this question, we can use the analysis of variance test, or ANOVA for short. ANOVA is a statistical test that assumes that the mean across 2 or more groups are equal. If the evidence suggests that this is not the case, the null hypothesis is rejected and at least one data sample has a different distribution.

- **Fail to Reject H0**: All sample distributions are equal.
- **Reject H0**: One or more sample distributions are not equal.

Importantly, the test can only comment on whether all samples are the same or not; it cannot quantify which samples differ or by how much.

> The purpose of a one-way analysis of variance (one-way ANOVA) is to compare the means of two or more groups (the independent variable) on one dependent variable to see if the group means are significantly different from each other.
>
> — Page 105, *Statistics in Plain English*, Third Edition, 2010.

The test requires that the data samples are a Gaussian distribution, that the samples are independent, and that all data samples have the same standard deviation. The ANOVA test can be performed in Python using the `f_oneway()` SciPy function. The function takes two or more data samples as arguments and returns the test statistic and f-value. We can modify our test problem to have two samples with the same mean and a third sample with a slightly different mean. We would then expect the test to discover that at least one sample has a different distribution. The complete example is listed below.

```
# analysis of variance test
from numpy.random import seed
from numpy.random import randn
from scipy.stats import f_oneway
# seed the random number generator
seed(1)
# generate three independent samples
data1 = 5 * randn(100) + 50
data2 = 5 * randn(100) + 50
data3 = 5 * randn(100) + 52
# compare samples
stat, p = f_oneway(data1, data2, data3)
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
  print('Same distributions (fail to reject H0)')
else:
  print('Different distributions (reject H0)')
```

Listing 13.7: Example of an ANOVA test on the test dataset.

Running the example calculates and prints the test statistic and the p-value. The interpretation of the p-value correctly rejects the null hypothesis indicating that one or more sample means differ.

```
Statistics=3.655, p=0.027
Different distributions (reject H0)
```

Listing 13.8: Sample output from the ANOVA test on the test dataset.

13.7 Repeated Measures ANOVA Test

We may have multiple data samples that are related or dependent in some way. For example, we may repeat the same measurements on a subject at different time periods. In this case, the samples will no longer be independent; instead we will have multiple paired samples. We could repeat the pairwise Student's t-test multiple times. Alternately, we can use a single test to check if all of the samples have the same mean. A variation of the ANOVA test can be used, modified to test across more than 2 samples. This test is called the repeated measures ANOVA test.

The default assumption or null hypothesis is that all paired samples have the same mean, and therefore the same distribution. If the samples suggest that this is not the case, then the null hypothesis is rejected and one or more of the paired samples have a different mean.

- **Fail to Reject H0**: All paired sample distributions are equal.

- **Reject H0**: One or more paired sample distributions are not equal.

> Repeated-measures ANOVA has a number of advantages over paired t-tests, however. First, with repeated-measures ANOVA, we can examine differences on a dependent variable that has been measured at more than two time points, whereas with an independent t-test we can only compare scores on a dependent variable from two time points.
>
> — Page 131, *Statistics in Plain English*, Third Edition, 2010.

Unfortunately, at the time of writing, there is no version of the repeated measures ANOVA test available in SciPy. Hopefully this test will be added soon. I mention this test for completeness in case you require it on your project and are able to seek and find an alternate implementation.

13.8 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Update all examples to operate on data samples that have the same distribution.
- Create a flowchart for choosing each of the three statistical significance tests given the requirements and behavior of each test.
- Consider 3 cases of comparing data samples in a machine learning project, assume a non-Gaussian distribution for the samples, and suggest the type of test that could be used in each case.

If you explore any of these extensions, I'd love to know.

13.9 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

13.9.1 Books

- *Statistics in Plain English*, Third Edition, 2010.

13.9.2 API

- `scipy.stats.ttest_ind` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html
- `scipy.stats.ttest_rel` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_rel.html
- `scipy.stats.f_oneway` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.f_oneway.html

13.9.3 Articles

- Statistical significance on Wikipedia.
  https://en.wikipedia.org/wiki/Statistical_significance
- Student's t-test on Wikipedia.
  https://en.wikipedia.org/wiki/Student%27s_t-test
- Paired difference test on Wikipedia.
  https://en.wikipedia.org/wiki/Paired_difference_test
- Analysis of variance on Wikipedia.
  https://en.wikipedia.org/wiki/Analysis_of_variance
- Repeated measures design on Wikipedia.
  https://en.wikipedia.org/wiki/Repeated_measures_design

13.10 Summary

In this tutorial, you discovered parametric statistical significance tests that quantify the difference between the means of two or more samples of data. Specifically, you learned:

- The Student's t-test for quantifying the difference between the mean of two independent data samples.
- The paired Student's t-test for quantifying the difference between the mean of two dependent data samples.
- The ANOVA and repeated measures ANOVA for checking the similarity or difference between the means of two or more data samples.

13.10.1 Next

In the next section, you will discover how to calculate effect size from an experiment.

Chapter 14

Effect Size

Statistical hypothesis tests report on the likelihood of the observed results given an assumption, such as no association between variables or no difference between groups. Hypothesis tests do not comment on the size of the effect if the association or difference is statistically significant. This highlights the need for standard ways of calculating and reporting a result. Effect size methods refer to a suite of statistical tools for quantifying the size of an effect in the results of experiments that can be used to complement the results from statistical hypothesis tests. In this tutorial, you will discover effect size and effect size measures for quantifying the magnitude of a result. After completing this tutorial, you will know:

- The importance of calculating and reporting effect size in the results of experiments.
- Effect size measures for quantifying the association between variables, such as Pearson's correlation coefficient.
- Effect size measures for quantifying the difference between groups, such as Cohen's d measure.

Let's get started.

14.1 Tutorial Overview

This tutorial is divided into three parts; they are:

1. The Need to Report Effect Size
2. What Is Effect Size?
3. How to Calculate Effect Size

14.2 The Need to Report Effect Size

Once practitioners become versed in statistical methods, it is common to become focused on quantifying the likelihood of a result. This is often seen with the calculation and presentation of the results from statistical hypothesis tests in terms of p-value and the significance level. One

aspect that is often neglected in the presentation of results is to actually quantify the difference or relationship, called the effect. It can be easy to forget that the intention of an experiment is to quantify an effect.

> The primary product of a research inquiry is one or more measures of effect size, not P values.
>
> — *Things I have learned (so far)*, 1990.

The statistical test can only comment on the likelihood that there is an effect. It does not comment on the size of the effect. The results of an experiment could be significant, but the effect so small that it has little consequence.

> It is possible, and unfortunately quite common, for a result to be statistically significant and trivial. It is also possible for a result to be statistically nonsignificant and important.
>
> — Page 4, *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, 2010.

The problem with neglecting the presentation of the effect is that it may be calculated using ad hoc measures or even ignored completely and left to the reader to interpret. This is a big problem as quantifying the size of the effect is essential to interpreting results.

14.3 What Is Effect Size?

An effect size refers to the size or magnitude of an effect or result as it would be expected to occur in a population. The effect size is estimated from samples of data. Effect size methods refers to a collection of statistical tools used to calculate the effect size. Often the field of effect size measures is referred to as simply *effect size*, to note the general concern of the field. It is common to organize effect size statistical methods into groups, based on the type of effect that is to be quantified. Two main groups of methods for calculating effect size are:

- **Association**. Statistical methods for quantifying an association between variables (e.g. correlation).
- **Difference**. Statistical methods for quantifying the difference between variables (e.g. difference between means).

> An effect can be the result of a treatment revealed in a comparison between groups (e.g. treated and untreated groups) or it can describe the degree of association between two related variables (e.g. treatment dosage and health).
>
> — Page 5, *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, 2010.

The result of an effect size calculation must be interpreted, and it depends on the specific statistical method used. A measure must be chosen based on the goals of the interpretation. Three types of calculated result include:

- **Standardized Result**. The effect size has a standard scale allowing it to be interpreted generally regardless of application (e.g. Cohen's d calculation).
- **Original Units Result**. The effect size may use the original units of the variable, which can aid in the interpretation within the domain (e.g. difference between two sample means).
- **Unit Free Result**. The effect size may not have units such as a count or proportion (e.g. a correlation coefficient).

> Thus, effect size can refer to the raw difference between group means, or absolute effect size, as well as standardized measures of effect, which are calculated to transform the effect to an easily understood scale. Absolute effect size is useful when the variables under study have intrinsic meaning (eg, number of hours of sleep).
>
> — *Using Effect Size – or Why the P Value Is Not Enough*, 2012.

It may be a good idea to report an effect size using multiple measures to aide the different types of readers of your findings.

> Sometimes a result is best reported both in original units, for ease of understanding by readers, and in some standardized measure for ease of inclusion in future meta-analyses.
>
> — Page 41, *Understanding The New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis*, 2011.

The effect size does not replace the results of a statistical hypothesis test. Instead, the effect size complements the test. Ideally, the results of both the hypothesis test and the effect size calculation would be presented side-by-side.

- **Hypothesis Test**: Quantify the likelihood of observing the data given an assumption (null hypothesis).
- **Effect Size**: Quantify the size of the effect assuming that the effect is present.

14.4 How to Calculate Effect Size

The calculation of an effect size could be the calculation of a mean of a sample or the absolute difference between two means. It could also be a more elaborate statistical calculation. In this section, we will look at some common effect size calculations for both associations and differences. The examples of methods is not complete; there may be 100s of methods that can be used to calculate an effect size.

14.4.1 Calculate Association Effect Size

The association between variables is often referred to as the *r family* of effect size methods. This name comes from perhaps the most common method for calculating the effect size called Pearson's correlation coefficient, also called Pearson's r. The Pearson's correlation coefficient measures the degree of linear association between two real-valued variables. It is a unit-free effect size measure, that can be interpreted in a standard way, as follows:

- -1.0: Perfect negative relationship.
- -0.7: Strong negative relationship
- -0.5: Moderate negative relationship
- -0.3: Weak negative relationship
- 0.0: No relationship.
- 0.3: Weak positive relationship
- 0.5: Moderate positive relationship
- 0.7: Strong positive relationship
- 1.0: Perfect positive relationship.

The Pearson's correlation coefficient can be calculated in Python using the `pearsonr()` SciPy function. Correlation was covered in detail in Chapter 12. The example below demonstrates the calculation of the Pearson's correlation coefficient to quantify the size of the association between two samples of random Gaussian numbers where one sample has a strong relationship with the second.

```
# calculate the pearson's correlation between two variables
from numpy.random import randn
from numpy.random import seed
from scipy.stats import pearsonr
# seed random number generator
seed(1)
# prepare data
data1 = 10 * randn(10000) + 50
data2 = data1 + (10 * randn(10000) + 50)
# calculate pearson's correlation
corr, _ = pearsonr(data1, data2)
print('Pearsons correlation: %.3f' % corr)
```

Listing 14.1: Example of calculating an association effect size.

Running the example calculates and prints the Pearson's correlation between the two data samples. We can see that the effect shows a strong positive relationship between the samples.

```
Pearson's correlation: 0.712
```

Listing 14.2: Sample output from calculating an association effect size.

Another very popular method for calculating the association effect size is the r-squared measure, or r^2, also called the coefficient of determination. It summarizes the proportion of variance in one variable explained by the other.

14.4.2 Calculate Difference Effect Size

The difference between groups is often referred to as the *d family* of effect size methods. This name comes from perhaps the most common method for calculating the difference between the mean value of groups, called Cohen's d. Cohen's d measures the difference between the mean from two Gaussian-distributed variables. It is a standard score that summarizes the difference in terms of the number of standard deviations. Because the score is standardized, there is a table for the interpretation of the result, summarized as:

- Small Effect Size: d=0.20
- Medium Effect Size: d=0.50
- Large Effect Size: d=0.80

The Cohen's d calculation is not provided in Python; we can calculate it manually. The calculation of the difference between the mean of two samples is as follows:

$$d = \frac{\mu_1 - \mu_2}{s} \tag{14.1}$$

Where d is the Cohen's d, μ_1 is the mean of the first sample, μ_2 is the mean of the second sample, and s is the pooled standard deviation of both samples. The pooled standard deviation for two independent samples can be calculated as follows:

$$s = \sqrt{\frac{(n_1 - 1) \times s_1^2 + (n_2 - 1) \times s_2^2}{n_1 + n_2 - 2}} \tag{14.2}$$

Where s is the pooled standard deviation, n_1 and n_2 are the size of the first sample and second samples and s_1^2 and s_2^2 is the variance for the first and second samples. The subtractions are the adjustments for the number of degrees of freedom. The function below will calculate the Cohen's d measure for two samples of real-valued variables. The NumPy functions `mean()` and `var()` are used to calculate the sample mean and variance respectively.

```
...
# function to calculate Cohen's d for independent samples
def cohend(d1, d2):
  # calculate the size of samples
  n1, n2 = len(d1), len(d2)
  # calculate the variance of the samples
  s1, s2 = var(d1, ddof=1), var(d2, ddof=1)
  # calculate the pooled standard deviation
  s = sqrt(((n1 - 1) * s1 + (n2 - 1) * s2) / (n1 + n2 - 2))
  # calculate the means of the samples
  u1, u2 = mean(d1), mean(d2)
  # calculate the effect size
  return (u1 - u2) / s
```

Listing 14.3: Function for calculating Cohen's d.

The example below calculates the Cohen's d measure for two samples of random Gaussian variables with differing means. The example is contrived such that the means are different by one half standard deviation and both samples have the same standard deviation.

```
# calculate the cohen's d between two samples
from numpy.random import randn
from numpy.random import seed
from numpy import mean
from numpy import var
from math import sqrt

# function to calculate cohen's d for independent samples
def cohend(d1, d2):
  # calculate the size of samples
  n1, n2 = len(d1), len(d2)
  # calculate the variance of the samples
  s1, s2 = var(d1, ddof=1), var(d2, ddof=1)
  # calculate the pooled standard deviation
  s = sqrt(((n1 - 1) * s1 + (n2 - 1) * s2) / (n1 + n2 - 2))
  # calculate the means of the samples
  u1, u2 = mean(d1), mean(d2)
  # calculate the effect size
  return (u1 - u2) / s

# seed random number generator
seed(1)
# prepare data
data1 = 10 * randn(10000) + 60
data2 = 10 * randn(10000) + 55
# calculate cohen's d
d = cohend(data1, data2)
print('Cohens d: %.3f' % d)
```

Listing 14.4: Example of calculating a difference effect size.

Running the example calculates and prints the Cohen's d effect size. We can see that as expected, the difference between the means is one half of one standard deviation interpreted as a medium effect size.

```
Cohen's d: 0.500
```

Listing 14.5: Sample output from calculating a difference effect size.

Two other popular methods for quantifying the difference effect size are:

- **Odds Ratio**. Measures the odds of an outcome occurring from one treatment compared to another.
- **Relative Risk Ratio**. Measures the probabilities of an outcome occurring from one treatment compared to another.

14.5 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Find an example where effect size is reported along with the results of statistical significance in a research paper.

- Implement a function to calculate the Cohen's d for paired samples and demonstrate it on a test dataset.
- Implement and demonstrate another difference effect measure, such as the odds or risk ratios.

If you explore any of these extensions, I'd love to know.

14.6 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

14.6.1 Papers

- Using Effect Size—or Why the P Value Is Not Enough, 2012.
  https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3444174/
- Things I have learned (so far), 1990.
  https://tech.me.holycross.edu/files/2015/03/Cohen_1990.pdf

14.6.2 Books

- *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, 2010.
  https://amzn.to/2JDcwSe
- *Understanding The New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis*, 2011.
  https://amzn.to/2v0wKSI
- *Statistical Power Analysis for the Behavioral Sciences*, 1988.
  https://amzn.to/2GNcmtu

14.6.3 API

- `scipy.stats.pearsonr` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html
- `numpy.var` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.var.html
- `numpy.mean` API.
  https://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html

14.6.4 Articles

- Effect size on Wikipedia.
  https://en.wikipedia.org/wiki/Effect_size

14.7 Summary

In this tutorial, you discovered effect size and effect size measures for quantifying the magnitude of a result. Specifically, you learned:

- The importance of calculating and reporting effect size in the results of experiments.
- Effect size measures for quantifying the association between variables, such as Pearson's correlation coefficient.
- Effect size measures for quantifying the difference between groups, such as Cohen's d measure.

14.7.1 Next

In the next section, you will discover statistical power and how to use it to estimate sample sizes.

Chapter 15

Statistical Power

The statistical power of a hypothesis test is the probability of detecting an effect, if there is a true effect present to detect. Power can be calculated and reported for a completed experiment to comment on the confidence one might have in the conclusions drawn from the results of the study. It can also be used as a tool to estimate the number of observations or sample size required in order to detect an effect in an experiment. In this tutorial, you will discover the importance of the statistical power of a hypothesis test and now to calculate power analyses and power curves as part of experimental design. After completing this tutorial, you will know:

- Statistical power is the probability of a hypothesis test of finding an effect if there is an effect to be found.
- A power analysis can be used to estimate the minimum sample size required for an experiment, given a desired significance level, effect size, and statistical power.
- How to calculate and plot power analysis for the Student's t-test in Python in order to effectively design an experiment.

Let's get started.

15.1 Tutorial Overview

This tutorial is divided into four parts; they are:

1. Statistical Hypothesis Testing
2. What Is Statistical Power?
3. Power Analysis
4. Student's t-Test Power Analysis

15.2 Statistical Hypothesis Testing

A statistical hypothesis test makes an assumption about the outcome, called the null hypothesis. For example, the null hypothesis for the Pearson's correlation test is that there is no relationship between two variables. The null hypothesis for the Student's t-test is that there is no difference between the means of two populations. The test is often interpreted using a p-value, which is the probability of observing the result given that the null hypothesis is true, not the reverse, as is often the case with misinterpretations.

- **p-value (p)**: Probability of obtaining a result equal to or more extreme than was observed in the data.

In interpreting the p-value of a significance test, you must specify a significance level, often referred to as the Greek lower case letter alpha (α). A common value for the significance level is 5% written as 0.05. The p-value is interested in the context of the chosen significance level. A result of a significance test is claimed to be *statistically significant* if the p-value is less than the significance level. This means that the null hypothesis (that there is no result) is rejected.

- `p-value` $\leq$ `alpha`: significant result, reject null hypothesis, distributions differ (H1).
- `p-value` $>$ `alpha`: not significant result, fail to reject null hypothesis, distributions same (H0).

Where:

- **Significance level (alpha)**: Boundary for specifying a statistically significant finding when interpreting the p-value.

We can see that the p-value is just a probability and that in actuality the result may be different. The test could be wrong. Given the p-value, we could make an error in our interpretation. There are two types of errors; they are:

- **Type I Error**. Reject the null hypothesis when there is in fact no significant effect (false positive). The p-value is optimistically small.
- **Type II Error**. Not reject the null hypothesis when there is a significant effect (false negative). The p-value is pessimistically large.

In this context, we can think of the significance level as the probability of rejecting the null hypothesis if it were true. That is the probability of making a Type I Error or a false positive.

15.3 What Is Statistical Power?

Statistical power, or the power of a hypothesis test is the probability that the test correctly rejects the null hypothesis. That is, the probability of a true positive result. It is only useful when the null hypothesis is rejected.

> ... statistical power is the probability that a test will correctly reject a false null hypothesis. Statistical power has relevance only when the null is false.

— Page 60, *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, 2010.

The higher the statistical power for a given experiment, the lower the probability of making a Type II (false negative) error. That is the higher the probability of detecting an effect when there is an effect. In fact, the power is precisely the inverse of the probability of a Type II error.

$$\text{Power} = 1 - \text{Type II Error} \tag{15.1}$$

Or

$$Pr(\text{True Positive}) = 1 - Pr(\text{False Negative}) \tag{15.2}$$

More intuitively, the statistical power can be thought of as the probability of accepting an alternative hypothesis, when the alternative hypothesis is true. When interpreting statistical power, we seek experiential setups that have high statistical power.

- **Low Statistical Power**: Large risk of committing Type II errors, e.g. a false negative.
- **High Statistical Power**: Small risk of committing Type II errors.

Experimental results with too low statistical power will lead to invalid conclusions about the meaning of the results. Therefore a minimum level of statistical power must be sought. It is common to design experiments with a statistical power of 80% or better, e.g. 0.80. This means a 20% probability of encountering a Type II area. This different to the 5% likelihood of encountering a Type I error for the standard value for the significance level.

15.4 Power Analysis

Statistical power is one piece in a puzzle that has four related parts; they are:

- **Effect Size**. The quantified magnitude of a result present in the population. Effect size is calculated using a specific statistical measure, such as Pearson's correlation coefficient for the relationship between variables or Cohen's d for the difference between groups.
- **Sample Size**. The number of observations in the sample.
- **Significance**. The significance level used in the statistical test, e.g. alpha. Often set to 5% or 0.05.
- **Statistical Power**. The probability of accepting the alternative hypothesis if it is true.

All four variables are related. For example, a larger sample size can make an effect easier to detect, and the statistical power can be increased in a test by increasing the significance level. A power analysis involves estimating one of these four parameters given values for three other parameters. This is a powerful tool in both the design and in the analysis of experiments that we wish to interpret using statistical hypothesis tests. For example, the statistical power can be estimated given an effect size, sample size and significance level. Alternately, the sample size can be estimated given different desired levels of significance.

> Power analysis answers questions like "how much statistical power does my study have?" and "how big a sample size do I need?".
>
> — Page 56, *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, 2010.

Perhaps the most common use of a power analysis is in the estimation of the minimum sample size required for an experiment.

> Power analyses are normally run before a study is conducted. A prospective or a priori power analysis can be used to estimate any one of the four power parameters but is most often used to estimate required sample sizes.
>
> — Page 57, *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, 2010.

As a practitioner, we can start with sensible defaults for some parameters, such as a significance level of 0.05 and a power level of 0.80. We can then estimate a desirable minimum effect size, specific to the experiment being performed. A power analysis can then be used to estimate the minimum sample size required. In addition, multiple power analyses can be performed to provide a curve of one parameter against another, such as the change in the size of an effect in an experiment given changes to the sample size. More elaborate plots can be created varying three of the parameters. This is a useful tool for experimental design.

15.5 Student's t-Test Power Analysis

We can make the idea of statistical power and power analysis concrete with a worked example. In this section, we will look at the Student's t-test, which is a statistical hypothesis test for comparing the means from two samples of Gaussian variables. The assumption, or null hypothesis, of the test is that the sample populations have the same mean, e.g. that there is no difference between the samples or that the samples are drawn from the same underlying population.

The test will calculate a p-value that can be interpreted as to whether the samples are the same (fail to reject the null hypothesis), or there is a statistically significant difference between the samples (reject the null hypothesis). A common significance level for interpreting the p-value is 5% or 0.05.

- **Significance level (alpha)**: 5% or 0.05.

The size of the effect of comparing two groups can be quantified with an effect size measure. A common measure for comparing the difference in the mean from two groups is the Cohen's d measure. It calculates a standard score that describes the difference in terms of the number of standard deviations that the means are different. A large effect size for Cohen's d is 0.80 or higher, as is commonly accepted when using the measure.

- **Effect Size**: Cohen's d of at least 0.80.

We can use the default and assume a minimum statistical power of 80% or 0.8.

- **Statistical Power**: 80% or 0.80.

For a given experiment with these defaults, we may be interested in estimating a suitable sample size. That is, how many observations are required from each sample in order to at least detect an effect of 0.80 with an 80% chance of detecting the effect if it is true (20% of a Type II error) and a 5% chance of detecting an effect if there is no such effect (Type I error). We can solve this using a power analysis.

The Statsmodels library provides the `TTestIndPower` class for calculating a power analysis for the Student's t-test with independent samples. Of note is the `TTestPower` class that can perform the same analysis for the paired Student's t-test. The function `solve_power()` can be used to calculate one of the four parameters in a power analysis. In our case, we are interested in calculating the sample size. We can use the function by providing the three pieces of information we know (`alpha`, `effect`, and `power`) and setting the size of argument we wish to calculate the answer of (`nobs1`) to `None`. This tells the function what to calculate.

A note on sample size: the function has an argument called `ratio` that is the ratio of the number of samples in one sample to the other. If both samples are expected to have the same number of observations, then the ratio is 1.0. If, for example, the second sample is expected to have half as many observations, then the ratio would be 0.5. The `TTestIndPower` instance must be created, then we can call the `solve_power()` with our arguments to estimate the sample size for the experiment.

```
...
# perform power analysis
analysis = TTestIndPower()
result = analysis.solve_power(effect, power=power, nobs1=None, ratio=1.0, alpha=alpha)
```

Listing 15.1: Function for calculating statistical power.

The complete example is listed below.

```
# estimate sample size via power analysis
from statsmodels.stats.power import TTestIndPower
# parameters for power analysis
effect = 0.8
alpha = 0.05
power = 0.8
# perform power analysis
analysis = TTestIndPower()
result = analysis.solve_power(effect, power=power, nobs1=None, ratio=1.0, alpha=alpha)
print('Sample Size: %.3f' % result)
```

Listing 15.2: Example of calculating sample size.

Running the example calculates and prints the estimated number of samples for the experiment as 25. This would be a suggested minimum number of samples required to see an effect of the desired size.

```
Sample Size: 25.525
```

Listing 15.3: Sample output from calculating sample size.

We can go one step further and calculate power curves. Power curves are line plots that show how the change in variables, such as effect size and sample size, impact the power of the statistical test.

The `plot_power()` function can be used to create power curves. The dependent variable (x-axis) must be specified by name in the `dep_var` argument. Arrays of values can then be specified for the sample size (`nobs`), effect size (`effect_size`), and significance (`alpha`) parameters. One or multiple curves will then be plotted showing the impact on statistical power. For example, we can assume a significance of 0.05 (the default for the function) and explore the change in sample size between 5 and 100 with low, medium, and high effect sizes.

```
...
# calculate power curves from multiple power analyses
analysis = TTestIndPower()
analysis.plot_power(dep_var='nobs', nobs=arange(5, 100), effect_size=array([0.2, 0.5, 0.8]))
```

Listing 15.4: Function for calculating a power analysis.

The complete example is listed below.

```
# calculate power curves for varying sample and effect size
from numpy import array
from matplotlib import pyplot
from statsmodels.stats.power import TTestIndPower
# parameters for power analysis
effect_sizes = array([0.2, 0.5, 0.8])
sample_sizes = array(range(5, 100))
# calculate power curves from multiple power analyses
analysis = TTestIndPower()
analysis.plot_power(dep_var='nobs', nobs=sample_sizes, effect_size=effect_sizes)
pyplot.show()
```

Listing 15.5: Example of calculating a power analysis.

Running the example creates the plot showing the impact on statistical power (y-axis) for three different effect sizes (es) as the sample size (x-axis) is increased. We can see that if we are interested in a large effect that a point of diminishing returns in terms of statistical power occurs at around 40-to-50 observations.

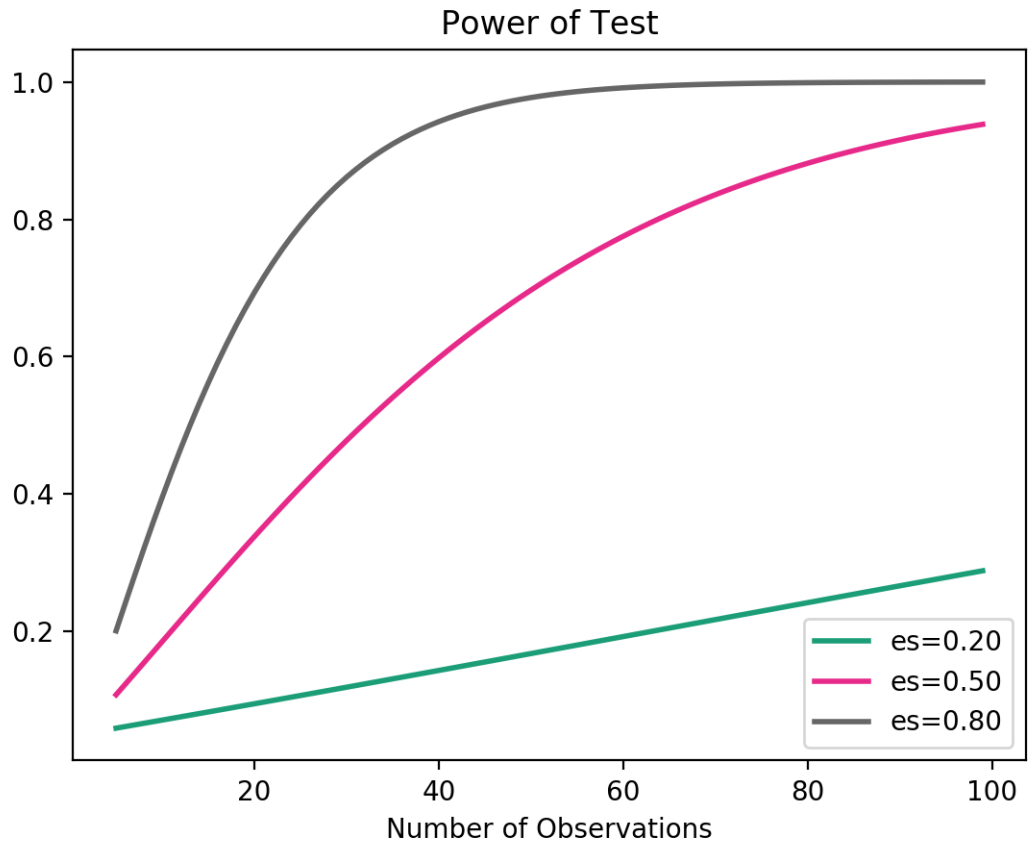

Figure 15.1: Power curves for Student's t-test.

Usefully, Statsmodels has classes to perform a power analysis with other statistical tests, such as the F-test, Z-test, and the Chi-Squared test.

15.6 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Plot the power curves of different standard significance levels against the sample size.
- Find an example of a study that reports the statistical power of the experiment.
- Prepare examples of a performance analysis for other statistical tests provided by Statsmodels.

If you explore any of these extensions, I'd love to know.

15.7 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

15.7.1 Papers

- Using Effect Size—or Why the P Value Is Not Enough, 2012.
  https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3444174/

15.7.2 Books

- *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, 2010.
  https://amzn.to/2JDcwSe
- *Understanding The New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis*, 2011.
  https://amzn.to/2v0wKSI
- *Statistical Power Analysis for the Behavioral Sciences*, 1988.
  https://amzn.to/2GNcmtu
- *Applied Power Analysis for the Behavioral Sciences*, 2010.
  https://amzn.to/2GPS3vI

15.7.3 API

- Statsmodels Power and Sample Size Calculations.
  http://www.statsmodels.org/dev/stats.html#power-and-sample-size-calculations
- `statsmodels.stats.power.TTestPower` API.
  http://www.statsmodels.org/dev/generated/statsmodels.stats.power.TTestPower.html
- `statsmodels.stats.power.TTestIndPower` API.
  http://www.statsmodels.org/dev/generated/statsmodels.stats.power.TTestIndPower.html
- `statsmodels.stats.power.TTestIndPower.solve_power` API.
  http://www.statsmodels.org/dev/generated/statsmodels.stats.power.TTestIndPower.solve_power.html
- `statsmodels.stats.power.TTestIndPower.plot_power` API.
  http://www.statsmodels.org/dev/generated/statsmodels.stats.power.TTestIndPower.plot_power.html
- Statistical Power in Statsmodels, 2013.
  http://jpktd.blogspot.com.au/2013/03/statistical-power-in-statsmodels.html
- Power Plots in Statsmodels, 2013.
  http://jpktd.blogspot.com.au/2013/05/power-plots-in-statsmodels.html

15.7.4 Articles

- Statistical power on Wikipedia.
  https://en.wikipedia.org/wiki/Statistical_power
- Statistical hypothesis testing on Wikipedia.
  https://en.wikipedia.org/wiki/Statistical_hypothesis_testing
- Statistical significance on Wikipedia.
  https://en.wikipedia.org/wiki/Statistical_significance
- Sample size determination on Wikipedia.
  https://en.wikipedia.org/wiki/Sample_size_determination
- Effect size on Wikipedia.
  https://en.wikipedia.org/wiki/Effect_size
- Type I and type II errors on Wikipedia.
  https://en.wikipedia.org/wiki/Type_I_and_type_II_errors

15.8 Summary

In this tutorial, you discovered the statistical power of a hypothesis test and how to calculate power analyses and power curves as part of experimental design. Specifically, you learned:

- Statistical power is the probability of a hypothesis test of finding an effect if there is an effect to be found.
- A power analysis can be used to estimate the minimum sample size required for an experiment, given a desired significance level, effect size, and statistical power.
- How to calculate and plot power analysis for the Student's t-test in Python in order to effectively design an experiment.

15.8.1 Next

This is the end of part IV, in the next part you will discover resampling methods.

Part V

Resampling Methods

Chapter 16

Introduction to Resampling

Data is the currency of applied machine learning. Therefore, it is important that it is both collected and used effectively. Data sampling refers to statistical methods for selecting observations from the domain with the objective of estimating a population parameter. Whereas data resampling refers to methods for economically using a collected dataset to improve the estimate of the population parameter and help to quantify the uncertainty of the estimate. Both data sampling and data resampling are methods that are required in a predictive modeling problem. In this tutorial, you will discover statistical sampling and statistical resampling methods for gathering and making best use of data. After completing this tutorial, you will know:

- Sampling is an active process of gathering observations with the intent of estimating a population variable.
- Resampling is a methodology of economically using a data sample to improve the accuracy and quantify the uncertainty of a population parameter.
- Resampling methods, in fact, make use of a nested resampling method.

Let's get started.

16.1 Tutorial Overview

This tutorial is divided into 2 parts; they are:

1. Statistical Sampling
2. Statistical Resampling

16.2 Statistical Sampling

Each row of data represents an observation about something in the world. When working with data, we often do not have access to all possible observations. This could be for many reasons; for example:

- It may difficult or expensive to make more observations.

- It may be challenging to gather all observations together.
- More observations are expected to be made in the future.

Observations made in a domain represent samples of some broader idealized and unknown population of all possible observations that could be made in the domain. This is a useful conceptualization as we can see the separation and relationship between observations and the idealized population. We can also see that, even if we intend to use big data infrastructure on all available data, that the data still represents a sample of observations from an idealized population. Nevertheless, we may wish to estimate properties of the population. We do this by using samples of observations.

> Sampling consists of selecting some part of the population to observe so that one may estimate something about the whole population.
>
> — Page 1, *Sampling*, Third Edition, 2012.

16.2.1 How to Sample

Statistical sampling is the process of selecting subsets of examples from a population with the objective of estimating properties of the population. Sampling is an active process. There is a goal of estimating population properties and control over how the sampling is to occur. This control falls short of influencing the process that generates each observation, such as performing an experiment. As such, sampling as a field sits neatly between pure uncontrolled observation and controlled experimentation.

> Sampling is usually distinguished from the closely related field of experimental design, in that in experiments one deliberately perturbs some part of the population in order to see what the effect of that action is. [...] Sampling is also usually distinguished from observational studies, in which one has little or no control over how the observations on the population were obtained.
>
> — Pages 1-2, *Sampling*, Third Edition, 2012.

There are many benefits to sampling compared to working with fuller or complete datasets, including reduced cost and greater speed. In order to perform sampling, it requires that you carefully define your population and the method by which you will select (and possibly reject) observations to be a part of your data sample. This may very well be defined by the population parameters that you wish to estimate using the sample. Some aspects to consider prior to collecting a data sample include:

- **Sample Goal**. The population property that you wish to estimate using the sample.
- **Population**. The scope or domain from which observations could theoretically be made.
- **Selection Criteria**. The methodology that will be used to accept or reject observations in your sample.
- **Sample Size**. The number of observations that will constitute the sample.

> Some obvious questions [...] are how best to obtain the sample and make the observations and, once the sample data are in hand, how best to use them to estimate the characteristics of the whole population. Obtaining the observations involves questions of sample size, how to select the sample, what observational methods to use, and what measurements to record.
>
> — Page 1, *Sampling*, Third Edition, 2012.

Statistical sampling is a large field of study, but in applied machine learning, there may be three types of sampling that you are likely to use: simple random sampling, systematic sampling, and stratified sampling.

- **Simple Random Sampling**: Samples are drawn with a uniform probability from the domain.
- **Systematic Sampling**: Samples are drawn using a pre-specified pattern, such as at intervals.
- **Stratified Sampling**: Samples are drawn within pre-specified categories (i.e. strata).

Although these are the more common types of sampling that you may encounter, there are other techniques.

16.2.2 Sampling Errors

Sampling requires that we make a statistical inference about the population from a small set of observations. We can generalize properties from the sample to the population. This process of estimation and generalization is much faster than working with all possible observations, but will contain errors. In many cases, we can quantify the uncertainty of our estimates and add errors bars, such as confidence intervals. There are many ways to introduce errors into your data sample. Two main types of errors include selection bias and sampling error.

- **Selection Bias**. Caused when the method of drawing observations skews the sample in some way.
- **Sampling Error**. Caused due to the random nature of drawing observations skewing the sample in some way.

Other types of errors may be present, such as systematic errors in the way observations or measurements are made. In these cases and more, the statistical properties of the sample may be different from what would be expected in the idealized population, which in turn may impact the properties of the population that are being estimated. Simple methods, such as reviewing raw observations, summary statistics, and visualizations can help expose simple errors, such as measurement corruption and the over- or underrepresentation of a class of observations. Nevertheless, care must be taken both when sampling and when drawing conclusions about the population while sampling.

16.3 Statistical Resampling

Once we have a data sample, it can be used to estimate the population parameter. The problem is that we only have a single estimate of the population parameter, with little idea of the variability or uncertainty in the estimate. One way to address this is by estimating the population parameter multiple times from our data sample. This is called resampling.

Statistical resampling methods are procedures that describe how to economically use available data to estimate a population parameter. The result can be both a more accurate estimate of the parameter (such as taking the mean of the estimates) and a quantification of the uncertainty of the estimate (such as adding a confidence interval). Resampling methods are very easy to use, requiring little mathematical knowledge. They are methods that are easy to understand and implement compared to specialized statistical methods that may require deep technical skill in order to select and interpret.

> The resampling methods [...] are easy to learn and easy to apply. They require no mathematics beyond introductory high-school algebra, but are applicable in an exceptionally broad range of subject areas.
>
> — Page xiii, *Resampling Methods: A Practical Guide to Data Analysis*, 2005.

A downside of the methods is that they can be computationally very expensive, requiring tens, hundreds, or even thousands of resamples in order to develop a robust estimate of the population parameter.

> The key idea is to resample form the original data — either directly or via a fitted model — to create replicate datasets, from which the variability of the quantiles of interest can be assessed without long-winded and error-prone analytical calculation. Because this approach involves repeating the original data analysis procedure with many replicate sets of data, these are sometimes called computer-intensive methods.
>
> — Page 3, *Bootstrap Methods and their Application*, 1997.

Each new subsample from the original data sample is used to estimate the population parameter. The sample of estimated population parameters can then be considered with statistical tools in order to quantify the expected value and variance, providing measures of the uncertainty of the estimate. Statistical sampling methods can be used in the selection of a subsample from the original sample.

A key difference is that process must be repeated multiple times. The problem with this is that there will be some relationship between the samples as observations that will be shared across multiple subsamples. This means that the subsamples and the estimated population parameters are not strictly identical and independently distributed. This has implications for statistical tests performed on the sample of estimated population parameters downstream, i.e. paired statistical tests may be required. Two commonly used resampling methods that you may encounter are k-fold cross-validation and the bootstrap.

- **Bootstrap**. Samples are drawn from the dataset with replacement (allowing the same sample to appear more than once in the sample), where those instances not drawn into the data sample may be used for the test set.

- **k-fold Cross-Validation**. A dataset is partitioned into k groups, where each group is given the opportunity of being used as a held out test set leaving the remaining groups as the training set.

The k-fold cross-validation method specifically lends itself to use in the evaluation of predictive models that are repeatedly trained on one subset of the data and evaluated on a second held-out subset of the data.

> Generally, resampling techniques for estimating model performance operate similarly: a subset of samples are used to fit a model and the remaining samples are used to estimate the efficacy of the model. This process is repeated multiple times and the results are aggregated and summarized. The differences in techniques usually center around the method in which subsamples are chosen.
>
> — Page 69, *Applied Predictive Modeling*, 2013.

The bootstrap method can be used for the same purpose, but is a more general and simpler method intended for estimating a population parameter.

16.4 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- List two examples where statistical sampling is required in a machine learning project.
- List two examples when statistical resampling is required in a machine learning project.
- Find a paper that uses a resampling method that in turn uses a nested statistical sampling method (hint: k-fold cross-validation and stratified sampling).

If you explore any of these extensions, I'd love to know.

16.5 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

16.5.1 Books

- *Sampling*, Third Edition, 2012.
  http://amzn.to/2HNgJAQ
- *Sampling Techniques*, 3rd Edition, 1977.
  http://amzn.to/2FMh8XF
- *Resampling Methods: A Practical Guide to Data Analysis*, 2005.
  http://amzn.to/2G6gMKP

- *An Introduction to the Bootstrap*, 1994.
  http://amzn.to/2praHye
- *Bootstrap Methods and their Application*, 1997.
  http://amzn.to/2FVsmVY
- *Applied Predictive Modeling*, 2013.
  http://amzn.to/2Fmrbib

16.5.2 Articles

- Sample (statistics) on Wikipedia.
  https://en.wikipedia.org/wiki/Sample_(statistics)
- Simple random sample on Wikipedia.
  https://en.wikipedia.org/wiki/Simple_random_sample
- Systematic sampling on Wikipedia.
  https://en.wikipedia.org/wiki/Systematic_sampling
- Stratified sampling on Wikipedia.
  https://en.wikipedia.org/wiki/Stratified_sampling
- Resampling (statistics) on Wikipedia.
  https://en.wikipedia.org/wiki/Resampling_(statistics)
- Bootstrapping (statistics) on Wikipedia.
  https://en.wikipedia.org/wiki/Bootstrapping_(statistics)
- Cross-validation (statistics) on Wikipedia.
  https://en.wikipedia.org/wiki/Cross-validation_(statistics)

16.6 Summary

In this tutorial, you discovered statistical sampling and statistical resampling methods for gathering and making best use of data. Specifically, you learned:

- Sampling is an active process of gathering observations intent on estimating a population variable.
- Resampling is a methodology of economically using a data sample to improve the accuracy and quantify the uncertainty of a population parameter.
- Resampling methods, in fact, make use of a nested resampling method.

16.6.1 Next

In the next section, you will discover the bootstrap method for estimating population parameters from samples.

Chapter 17

Estimation with Bootstrap

The bootstrap method is a resampling technique used to estimate statistics on a population by sampling a dataset with replacement. It can be used to estimate summary statistics such as the mean or standard deviation. It is used in applied machine learning to estimate the skill of machine learning models when making predictions on data not included in the training data.

A desirable property of the results from estimating machine learning model skill is that the estimated skill can be presented with confidence intervals, a feature not readily available with other methods such as cross-validation. In this tutorial, you will discover the bootstrap resampling method for estimating the skill of machine learning models on unseen data. After completing this tutorial, you will know:

- The bootstrap method involves iteratively resampling a dataset with replacement.
- That when using the bootstrap you must choose the size of the sample and the number of repeats.
- The scikit-learn provides a function that you can use to resample a dataset for the bootstrap method.

Let's get started.

17.1 Tutorial Overview

This tutorial is divided into 4 parts; they are:

1. Bootstrap Method
2. Configuration of the Bootstrap
3. Worked Example
4. Bootstrap in Python

17.2 Bootstrap Method

The bootstrap method is a statistical technique for estimating quantities about a population by averaging estimates from multiple small data samples. Importantly, samples are constructed by drawing observations from a large data sample one at a time and returning them to the data sample after they have been chosen. This allows a given observation to be included in a given small sample more than once. This approach to sampling is called sampling with replacement. The process for building one sample can be summarized as follows:

```
1. Choose the size of the sample.
2. While the size of the sample is less than the chosen size
  2.1 Randomly select an observation from the dataset
  2.2 Add it to the sample
```

Listing 17.1: Pseudocode for sampling.

The bootstrap method can be used to estimate a quantity of a population. This is done by repeatedly taking small samples, calculating the statistic, and taking the average of the calculated statistics. We can summarize this procedure as follows:

```
1. Choose a number of bootstrap samples to perform
2. Choose a sample size
3. For each bootstrap sample
  3.1 Draw a sample with replacement with the chosen size
  3.2 Calculate the statistic on the sample
4. Calculate the mean of the calculated sample statistics.
```

Listing 17.2: Pseudocode for the bootstrap for estimating a statistic.

The procedure can also be used to estimate the skill of a machine learning model.

> The bootstrap is a widely applicable and extremely powerful statistical tool that can be used to quantify the uncertainty associated with a given estimator or statistical learning method.
>
> — Page 187, *An Introduction to Statistical Learning*, 2013.

This is done by training the model on the sample and evaluating the skill of the model on those samples not included in the sample. These samples not included in a given sample are called the out-of-bag samples, or OOB for short. This procedure of using the bootstrap method to estimate the skill of the model can be summarized as follows:

```
1. Choose a number of bootstrap samples to perform
2. Choose a sample size
3. For each bootstrap sample
  3.1 Draw a sample with replacement with the chosen size
  3.2 Fit a model on the data sample
  3.3 Estimate the skill of the model on the out-of-bag sample.
4. Calculate the mean of the sample of model skill estimates.
```

Listing 17.3: Pseudocode for the bootstrap for estimating model skill.

> The samples not selected are usually referred to as the "out-of-bag" samples. For a given iteration of bootstrap resampling, a model is built on the selected samples and is used to predict the out-of-bag samples.

— Page 72, *Applied Predictive Modeling*, 2013.

Importantly, any data preparation prior to fitting the model or tuning of the hyperparameter of the model must occur within the for-loop on the data sample. This is to avoid data leakage where knowledge of the test dataset is used to improve the model. This, in turn, can result in an optimistic estimate of the model skill. A useful feature of the bootstrap method is that the resulting sample of estimations often forms a Gaussian distribution. In additional to summarizing this distribution with a central tendency, measures of variance can be given, such as standard deviation and standard error. Further, a confidence interval can be calculated and used to bound the presented estimate. This is useful when presenting the estimated skill of a machine learning model.

17.3 Configuration of the Bootstrap

There are two parameters that must be chosen when performing the bootstrap: the size of the sample and the number of repetitions of the procedure to perform.

17.3.1 Sample Size

In machine learning, it is common to use a sample size that is the same as the original dataset.

> The bootstrap sample is the same size as the original dataset. As a result, some samples will be represented multiple times in the bootstrap sample while others will not be selected at all.

— Page 72, *Applied Predictive Modeling*, 2013.

If the dataset is enormous and computational efficiency is an issue, smaller samples can be used, such as 50% or 80% of the size of the dataset.

17.3.2 Repetitions

The number of repetitions must be large enough to ensure that meaningful statistics, such as the mean, standard deviation, and standard error can be calculated on the sample. A minimum might be 20 or 30 repetitions. A smaller number of repeats can be used, which will further add variance to the estimated statistic. Ideally, the sample of estimates would be as large as possible given the time resources, with hundreds or thousands of repeats.

17.4 Worked Example

We can make the bootstrap procedure concrete with a small worked example. We will work through one iteration of the procedure. Imagine we have a dataset with 6 observations:

```
[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
```

Listing 17.4: Example of a dataset.

The first step is to choose the size of the sample. Here, we will use 4. Next, we must randomly choose the first observation from the dataset. Let's choose 0.2.

```
sample = [0.2]
```

Listing 17.5: Example of sampling one value from the dataset.

This observation is returned to the dataset and we repeat this step 3 more times.

```
sample = [0.2, 0.1, 0.2, 0.6]
```

Listing 17.6: Example of a subsample with replacement from the dataset.

We now have our data sample. The example purposefully demonstrates that the same value can appear zero, one or more times in the sample. Here the observation 0.2 appears twice. An estimate can then be calculated on the drawn sample.

```
statistic = calculation([0.2, 0.1, 0.2, 0.6])
```

Listing 17.7: Example of calculating a statistic on the subsample.

Those observations not chosen for the sample may be used as out of sample observations.

```
oob = [0.3, 0.4, 0.5]
```

Listing 17.8: Example of the out of bag (oob) subsample.

In the case of evaluating a machine learning model, the model is fit on the drawn sample and evaluated on the out-of-bag sample.

```
train = [0.2, 0.1, 0.2, 0.6]
test = [0.3, 0.4, 0.5]
model = fit(train)
statistic = evaluate(model, test)
```

Listing 17.9: Example of training and evaluating a model.

That concludes one repeat of the procedure. It can be repeated 30 or more times to give a sample of calculated statistics.

```
statistics = [...]
```

Listing 17.10: Example of a list of sample statistics.

This sample of statistics can then be summarized by calculating a mean, standard deviation, or other summary values to give a final usable estimate of the statistic.

```
estimate = mean([...])
```

Listing 17.11: Example of estimating the population statistic from the sample statistics.

17.5 Bootstrap in Python

We do not have to implement the bootstrap method manually. The scikit-learn library provides an implementation that will create a single bootstrap sample of a dataset. The `resample()` scikit-learn function can be used. It takes as arguments the data array, whether or not to sample with replacement, the size of the sample, and the seed for the pseudorandom number generator used prior to the sampling. For example, we can create a bootstrap that creates a sample with replacement with 4 observations and uses a value of 1 for the pseudorandom number generator.

```
...
# resample with replacement
boot = resample(data, replace=True, n_samples=4, random_state=1)
```

Listing 17.12: Example of resampling with replacement.

Unfortunately, the API does not include any mechanism to easily gather the out-of-bag observations that could be used as a test set to evaluate a fit model. At least in the univariate case we can gather the out-of-bag observations using a simple Python list comprehension.

```
...
# out of bag observations
oob = [x for x in data if x not in boot]
```

Listing 17.13: Example of collecting an out of bag subsample.

We can tie all of this together with our small dataset used in the worked example of the prior section.

```
# scikit-learn bootstrap
from sklearn.utils import resample
# data sample
data = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
# prepare bootstrap sample
boot = resample(data, replace=True, n_samples=4, random_state=1)
print('Bootstrap Sample: %s' % boot)
# out of bag observations
oob = [x for x in data if x not in boot]
print('OOB Sample: %s' % oob)
```

Listing 17.14: Example of estimating a population statistic with the bootstrap.

Running the example prints the observations in the bootstrap sample and those observations in the out-of-bag sample.

```
Bootstrap Sample: [0.6, 0.4, 0.5, 0.1]
OOB Sample: [0.2, 0.3]
```

Listing 17.15: Example output from estimating a population statistic with the bootstrap.

17.6 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- List 3 summary statistics that you could estimate using the bootstrap method.
- Find 3 research papers that use the bootstrap method to evaluate the performance of machine learning models.
- Implement your own function to create a sample and an out-of-bag sample with the bootstrap method.

If you explore any of these extensions, I'd love to know.

17.7 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

17.7.1 Books

- *Applied Predictive Modeling*, 2013.
  http://amzn.to/2Fmrbib
- *An Introduction to Statistical Learning*, 2013.
  http://amzn.to/2FkHqvW
- *An Introduction to the Bootstrap*, 1994.
  http://amzn.to/2G0Yatr

17.7.2 API

- `sklearn.utils.resample` API.
  http://scikit-learn.org/stable/modules/generated/sklearn.utils.resample.html
- `sklearn.model_selection` API.
  http://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection

17.7.3 Articles

- Resampling (statistics) on Wikipedia.
  https://en.wikipedia.org/wiki/Resampling_(statistics)
- Bootstrapping (statistics) on Wikipedia.
  https://en.wikipedia.org/wiki/Bootstrapping_(statistics)
- Rule of thumb for number of bootstrap samples, CrossValiated.
  https://stats.stackexchange.com/questions/86040/rule-of-thumb-for-number-of-bootstr

17.8 Summary

In this tutorial, you discovered the bootstrap resampling method for estimating the skill of machine learning models on unseen data. Specifically, you learned:

- The bootstrap method involves iteratively resampling a dataset with replacement.
- That when using the bootstrap you must choose the size of the sample and the number of repeats.
- The scikit-learn provides a function that you can use to resample a dataset for the bootstrap method.

17.8.1 Next

In the next section, you will discover the k-fold cross-validation for estimating the skill of a learning model when making predictions on unseen data.

Chapter 18

Estimation with Cross-Validation

Cross-validation is a statistical method used to estimate the skill of machine learning models. It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods. In this tutorial, you will discover a gentle introduction to the k-fold cross-validation procedure for estimating the skill of machine learning models. After completing this tutorial, you will know:

- That k-fold cross-validation is a procedure used to estimate the skill of the model on new data.
- There are common tactics that you can use to select the value of k for your dataset.
- There are commonly used variations on cross-validation such as stratified and repeated that are available in scikit-learn.

Let's get started.

18.1 Tutorial Overview

This tutorial is divided into 5 parts; they are:

1. k-Fold Cross-Validation
2. Configuration of k
3. Worked Example
4. Cross-Validation in Python
5. Variations on Cross-Validation

18.2 k-Fold Cross-Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model. It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

```
1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
  3.1 Take the group as a hold out or test data set
  3.2 Take the remaining groups as a training data set
  3.3 Fit a model on the training set and evaluate it on the test set
  3.4 Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores
```

Listing 18.1: Pseudocode for k-fold cross-validation for estimating the skill of a model.

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model $k - 1$ times.

> This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining $k - 1$ folds.
>
> — Page 181, *An Introduction to Statistical Learning*, 2013.

It is also important that any preparation of the data prior to fitting the model occur on the cross-validation-assigned training dataset within the loop rather than on the broader data set. This also applies to any tuning of hyperparameters. A failure to perform these operations within the loop may result in data leakage and an optimistic estimate of the model skill.

> Despite the best efforts of statistical methodologists, users frequently invalidate their results by inadvertently peeking at the test data.
>
> — Page 708, *Artificial Intelligence: A Modern Approach*, 3rd Edition, 2009.

The results of a k-fold cross-validation run are often summarized with the mean of the model skill scores. It is also good practice to include a measure of the variance of the skill scores, such as the standard deviation or standard error.

18.3 Configuration of k

The k value must be chosen carefully for your data sample. A poorly chosen value for k may result in a mis-representative idea of the skill of the model, such as a score with a high variance (that may change a lot based on the data used to fit the model), or a high bias, (such as an overestimate of the skill of the model).

Three common tactics for choosing a value for k are as follows:

- **Representative**: The value for k is chosen such that each train/test group of data samples is large enough to be statistically representative of the broader dataset.
- **k=10**: The value for k is fixed to 10, a value that has been found through experimentation to generally result in a model skill estimate with low bias a modest variance.
- **k=n**: The value for k is fixed to `n`, where `n` is the size of the dataset to give each test sample an opportunity to be used in the hold out dataset. This approach is called leave-one-out cross-validation.

> The choice of k is usually 5 or 10, but there is no formal rule. As k gets larger, the difference in size between the training set and the resampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller
>
> — Page 70, *Applied Predictive Modeling*, 2013.

A value of `k=10` is very common in the field of applied machine learning, and is recommend if you are struggling to choose a value for your dataset.

> To summarize, there is a bias-variance trade-off associated with the choice of k in k-fold cross-validation. Typically, given these considerations, one performs k-fold cross-validation using $k = 5$ or $k = 10$, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance.
>
> — Page 184, *An Introduction to Statistical Learning*, 2013.

If a value for k is chosen that does not evenly split the data sample, then one group will contain a remainder of the examples. It is preferable to split the data sample into k groups with the same number of samples, such that the sample of model skill scores are all equivalent.

18.4 Worked Example

To make the cross-validation procedure concrete, let's look at a worked example. Imagine we have a data sample with 6 observations:

```
[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
```

Listing 18.2: Example of a dataset.

The first step is to pick a value for k in order to determine the number of folds used to split the data. Here, we will use a value of $k = 3$. That means we will shuffle the data and then split the data into 3 groups. Because we have 6 observations, each group will have an equal number of 2 observations.

```
Fold1: [0.5, 0.2]
Fold2: [0.1, 0.3]
Fold3: [0.4, 0.6]
```

Listing 18.3: Example of splitting the dataset into 3-folds.

We can then make use of the sample, such as to evaluate the skill of a machine learning algorithm. Three models are trained and evaluated with each fold given a chance to be the held out test set. For example:

```
Model1: Trained on Fold1 + Fold2, Tested on Fold3
Model2: Trained on Fold2 + Fold3, Tested on Fold1
Model3: Trained on Fold1 + Fold3, Tested on Fold2
```

Listing 18.4: Example of training and testing a model on 3-folds.

The models are then discarded after they are evaluated as they have served their purpose. The skill scores are collected for each model and summarized for use.

18.5 Cross-Validation in Python

We do not have to implement k-fold cross-validation manually. The scikit-learn library provides an implementation that will split a given data sample up. The `KFold()` scikit-learn class can be used. It takes as arguments the number of splits, whether or not to shuffle the sample, and the seed for the pseudorandom number generator used prior to the shuffle. For example, we can create an instance that splits a dataset into 3 folds, shuffles prior to the split, and uses a value of 1 for the pseudorandom number generator.

```
...
# define cross-validation folds
kfold = KFold(3, True, 1)
```

Listing 18.5: Example of creating 3-folds.

The `split()` function can then be called on the class where the data sample is provided as an argument. Called repeatedly, the split will return each group of train and test sets. Specifically, arrays are returned containing the indexes into the original data sample of observations to use for train and test sets on each iteration. For example, we can enumerate the splits of the indices for a data sample using the created KFold instance as follows:

```
...
# enumerate splits
for train, test in kfold.split(data):
  print('train: %s, test: %s' % (train, test))
```

Listing 18.6: Example of enumerating train and test folds.

We can tie all of this together with our small dataset used in the worked example of the prior section.

```
# scikit-learn k-fold cross-validation
from numpy import array
from sklearn.model_selection import KFold
# data sample
data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
# prepare cross validation
kfold = KFold(3, True, 1)
# enumerate splits
for train, test in kfold.split(data):
  print('train: %s, test: %s' % (data[train], data[test]))
```

Listing 18.7: Example of enumerating k-folds.

Running the example prints the specific observations chosen for each train and test set. The indices are used directly on the original data array to retrieve the observation values.

```
train: [0.1 0.4 0.5 0.6], test: [0.2 0.3]
train: [0.2 0.3 0.4 0.6], test: [0.1 0.5]
train: [0.1 0.2 0.3 0.5], test: [0.4 0.6]
```

Listing 18.8: Sample output from enumerating k-folds.

Usefully, the k-fold cross-validation implementation in scikit-learn is provided as a component operation within broader methods, such as grid-searching model hyperparameters and scoring a model on a dataset. Nevertheless, the KFold class can be used directly in order to split up a dataset prior to modeling such that all models will use the same data splits. This is especially helpful if you are working with very large data samples. The use of the same splits across algorithms can have benefits for statistical tests that you may wish to perform on the data later.

18.6 Variations on Cross-Validation

There are a number of variations on the k-fold cross-validation procedure. Three commonly used variations are as follows:

- **Train/Test Split**: Taken to one extreme, k may be set to 1 such that a single train/test split is created to evaluate the model.
- **LOOCV**: Taken to another extreme, k may be set to the total number of observations in the dataset such that each observation is given a chance to be the held out of the dataset. This is called leave-one-out cross-validation, or LOOCV for short.
- **Stratified**: The splitting of data into folds may be governed by criteria such as ensuring that each fold has the same proportion of observations with a given categorical value, such as the class outcome value. This is called stratified cross-validation.
- **Repeated**: This is where the k-fold cross-validation procedure is repeated n times, where importantly, the data sample is shuffled prior to each repetition, which results in a different split of the sample.

The scikit-learn library provides a suite of cross-validation implementation. You can see the full list in the Model Selection API.

18.7 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Find 3 machine learning research papers that use a value of 10 for k-fold cross-validation.
- Write your own function to split a data sample using k-fold cross-validation.
- Develop examples to demonstrate each of the main types of cross-validation supported by scikit-learn.

If you explore any of these extensions, I'd love to know.

18.8 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

18.8.1 Books

- *Applied Predictive Modeling*, 2013.
  http://amzn.to/2Fmrbib
- *An Introduction to Statistical Learning*, 2013.
  http://amzn.to/2FkHqvW
- *Artificial Intelligence: A Modern Approach*, 3rd Edition, 2009.
  http://amzn.to/2thrWHq

18.8.2 API

- `sklearn.model_selection.KFold` API.
  http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
- `sklearn.model_selection` API.
  http://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection

18.8.3 Articles

- Resampling (statistics) on Wikipedia.
  https://en.wikipedia.org/wiki/Resampling_(statistics)
- Cross-validation (statistics) on Wikipedia.
  https://en.wikipedia.org/wiki/Cross-validation_(statistics)

18.9 Summary

In this tutorial, you discovered a gentle introduction to the k-fold cross-validation procedure for estimating the skill of machine learning models. Specifically, you learned:

- That k-fold cross-validation is a procedure used to estimate the skill of the model on new data.
- There are common tactics that you can use to select the value of k for your dataset.
- There are commonly used variations on cross-validation, such as stratified and repeated, that are available in scikit-learn.

18.9.1 Next

This is the end of part V, in the next part you will discover the field of estimation statistics such as tolerance, confidence, and prediction intervals.

Part VI

Estimation Statistics

Chapter 19

Introduction to Estimation Statistics

Statistical hypothesis tests can be used to indicate whether the difference between two samples is due to random chance, but cannot comment on the size of the difference. A group of methods referred to as *new statistics* are seeing increased use instead of or in addition to p-values in order to quantify the magnitude of effects and the amount of uncertainty for estimated values. This group of statistical methods is referred to as *estimation statistics*. In this tutorial, you will discover a gentle introduction to estimation statistics as an alternate or complement to statistical hypothesis testing. After completing this tutorial, you will know:

- Effect size methods involve quantifying the association or difference between samples.
- Interval estimate methods involve quantifying the uncertainty around point estimations.
- Meta-analyses involve quantifying the magnitude of an effect across multiple similar independent studies.

Let's get started.

19.1 Tutorial Overview

This tutorial is divided into 5 parts; they are:

1. Problems with Hypothesis Testing
2. Estimation Statistics
3. Effect Size
4. Interval Estimation
5. Meta-Analysis

19.2 Problems with Hypothesis Testing

Statistical hypothesis testing and the calculation of p-values is a popular way to present and interpret results. Tests like the Student's t-test can be used to describe whether or not two samples have the same distribution. They can help interpret whether the difference between two sample means is likely real or due to random chance. Although they are widely used, they have some problems. For example:

- Calculated p-values are easily misused and misunderstood.
- There's always some significant difference between samples, even if the difference is tiny.

Interestingly, in the last few decades there has been a push back against the use of p-values in the presentation of research. For example, in the 1990s, the Journal of Epidemiology banned the use of p-values. Many related areas in medicine and psychology have followed suit. Although p-values may still be used, there is a push toward the presentation of results using estimation statistics.

19.3 Estimation Statistics

Estimation statistics refers to methods that attempt to quantify a finding. This might include quantifying the size of an effect or the amount of uncertainty for a specific outcome or result.

> ... 'estimation statistics', a term describing the methods that focus on the estimation of effect sizes (point estimates) and their confidence intervals (precision estimates).
>
> — *Estimation statistics should replace significance testing*, 2016.

Estimation statistics is a term to describe three main classes of methods. The three main classes of methods include:

- **Effect Size**. Methods for quantifying the size of an effect given a treatment or intervention.
- **Interval Estimation**. Methods for quantifying the amount of uncertainty in a value.
- **Meta-Analysis**. Methods for quantifying the findings across multiple similar studies.

We will look at each of these groups of methods in more detail in the following sections. Although they are not new, they are being called the *new statistics* given their increased use in research literature over statistical hypothesis testing.

> The new statistics refer to estimation, meta-analysis, and other techniques that help researchers shift emphasis from [null hypothesis statistical tests]. The techniques are not new and are routinely used on some disciplines, but for the [null hypothesis statistical tests] disciplines, their use would be new and beneficial.
>
> — *Understanding The New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis*, 2012.

The main reason that the shift from statistical hypothesis methods to estimation systems is occurring is the results are easier to analyze and interpret in the context of the domain or research question. The quantified size of the effect and uncertainty allows claims to be made that are easier to understand and use. The results are more meaningful.

> Knowing and thinking about the magnitude and precision of an effect is more useful to quantitative science than contemplating the probability of observing data of at least that extremity, assuming absolutely no effect.
>
> — *Estimation statistics should replace significance testing*, 2016.

Where statistical hypothesis tests talk about whether the samples come from the same distribution or not, estimation statistics can describe the size and confidence of the difference. This allows you to comment on how different one method is from another.

> Estimating thinking focuses on how big an effect is; knowing this is usually more valuable than knowing whether or not the effect is zero, which is the fours of dichotomous thinking. Estimation thinking prompts us to plan an experiment to address the question, "How much...?" or "To what extent...?" rater than only the dichotomous null hypothesis statistical tests] question, "Is there an effect?"
>
> — *Understanding The New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis*, 2012.

19.4 Effect Size

The effect size describes the magnitude of a treatment or difference between two samples. Note that effect size was covered in detail in Chapter 14. A hypothesis test can comment on whether the difference between samples is the result of chance or is real, whereas an effect size puts a number on how much the samples differ. Measuring the size of an effect is a big part of applied machine learning, and in fact, research in general.

> I am sometimes asked, what do researchers do? The short answer is that we estimate the size of effects. No matter what phenomenon we have chosen to study we essentially spend our careers thinking up new and better ways to estimate effect magnitudes.
>
> — Page 3, *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, 2010.

There are two main classes of techniques used to quantify the magnitude of effects; they are:

- **Association**. The degree to which two samples change together.
- **Difference**. The degree to which two samples are different.

For example, association effect sizes include calculations of correlation, such as the Pearson's correlation coefficient, and the r^2 coefficient of determination. They may quantify the linear or monotonic way that observations in two samples change together. Difference effect size may include methods such as Cohen's d statistic that provide a standardized measure for how the means of two populations differ. They seek a quantification for the magnitude of difference between observations in two samples.

> An effect can be the result of a treatment revealed in a comparison between groups (e.g., treated and untreated groups) or it can describe the degree of association between two related variables (e.g., treatment dosage and health).

— Page 4, *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, 2010.

19.5 Interval Estimation

Interval estimation refers to statistical methods for quantifying the uncertainty for an observation. Intervals transform a point estimate into a range that provides more information about the estimate, such as its precision, making them easier to compare and interpret.

> The point estimates are the dots, and the intervals indicate the uncertainty of those point estimates.

— Page 9, *Understanding The New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis*, 2012.

There are three main types of intervals that are commonly calculated. They are:

- **Tolerance Interval**: The bounds or coverage of a proportion of a distribution with a specific level of confidence.
- **Confidence Interval**: The bounds on the estimate of a population parameter.
- **Prediction Interval**: The bounds on a single observation.

A tolerance interval may be used to set expectations on observations in a population or help to identify outliers. A confidence interval can be used to interpret the range for a mean of a data sample that can become more precise as the sample size is increased. A prediction interval can be used to provide a range for a prediction or forecast from a model. For example, when presenting the mean estimated skill of a model, a confidence interval can be used to provide bounds on the precision of the estimate. This could also be combined with p-values if models are being compared.

> The confidence interval thus provides a range of possibilities for the population value, rather than an arbitrary dichotomy based solely on statistical significance. It conveys more useful information at the expense of precision of the P value. However, the actual P value is helpful in addition to the confidence interval, and preferably both should be presented. If one has to be excluded, however, it should be the P value.

— *Confidence intervals rather than P values: estimation rather than hypothesis testing*, 1986.

19.6 Meta-Analysis

A meta-analysis refers to the use of a weighting of multiple similar studies in order to quantify a broader cross-study effect. Meta studies are useful when many small and similar studies have been performed with noisy and conflicting findings. Instead of taking the study conclusions at face value, statistical methods are used to combine multiple findings into a stronger finding than any single study.

> ... better known as meta-analysis, completely ignores the conclusions that others have drawn and looks instead at the effects that have been observed. The aim is to combine these independent observations into an average effect size and draw an overall conclusion regarding the direction and magnitude of real-world effects.

— Page 90, *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, 2010.

Although not often used in applied machine learning, it is useful to note meta-analyses as they form part of this trust of new statistical methods.

19.7 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Describe three examples of how estimation statistics can be used in a machine learning project.
- Locate and summarize three criticisms against the use of statistical hypothesis testing.
- Search and locate three research papers that make use of interval estimates.

If you explore any of these extensions, I'd love to know.

19.8 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

19.8.1 Books

- *Understanding The New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis*, 2012.
  http://amzn.to/2HxDjgC
- *Introduction to the New Statistics: Estimation, Open Science, and Beyond*, 2016.
  http://amzn.to/2FEudOI
- *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, 2010.
  http://amzn.to/2p8Ckfs

19.8.2 Papers

- Estimation statistics should replace significance testing, 2016.
  https://www.nature.com/articles/nmeth.3729
- Confidence intervals rather than P values: estimation rather than hypothesis testing, 1986.
  https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1339793/

19.8.3 Articles

- Estimation statistics on Wikipedia.
  https://en.wikipedia.org/wiki/Estimation_statistics
- Effect size on Wikipedia.
  https://en.wikipedia.org/wiki/Effect_size
- Interval estimation on Wikipedia.
  https://en.wikipedia.org/wiki/Interval_estimation
- Meta-analysis on Wikipedia.
  https://en.wikipedia.org/wiki/Meta-analysis

19.9 Summary

In this tutorial, you discovered a gentle introduction to estimation statistics as an alternate or complement to statistical hypothesis testing. Specifically, you learned:

- Effect size methods involve quantifying the association or difference between samples.
- Interval estimate methods involve quantifying the uncertainty around point estimations.
- Meta-analyses involve quantifying the magnitude of an effect across multiple similar independent studies.

19.9.1 Next

In the next section, you will discover tolerance intervals for quantifying likely values in a distribution.

Chapter 20

Tolerance Intervals

It can be useful to have an upper and lower limit on data. These bounds can be used to help identify anomalies and set expectations for what to expect. A bound on observations from a population is called a tolerance interval. A tolerance interval is different from a prediction interval that quantifies the uncertainty for a single predicted value. It is also different from a confidence interval that quantifies the uncertainty of a population parameter such as a mean. Instead, a tolerance interval covers a proportion of the population distribution. In this tutorial, you will discover statistical tolerance intervals and how to calculate a tolerance interval for Gaussian data.

After completing this tutorial, you will know:

- That statistical tolerance intervals provide a bounds on observations from a population.
- That a tolerance interval requires that both a coverage proportion and confidence be specified.
- That the tolerance interval for a data sample with a Gaussian distribution can be easily calculated.

Let's get started.

20.1 Tutorial Overview

This tutorial is divided into 4 parts; they are:

1. Bounds on Data
2. What Are Statistical Tolerance Intervals?
3. How to Calculate Tolerance Intervals
4. Tolerance Interval for Gaussian Distribution

20.2 Bounds on Data

It is useful to put bounds on data. For example, if you have a sample of data from a domain, knowing the upper and lower bound for normal values can be helpful for identifying anomalies or outliers in the data. For a process or model that is making predictions, it can be helpful to know the expected range that sensible predictions may take. Knowing the common range of values can help with setting expectations and detecting anomalies. The range of common values for data is called a tolerance interval.

20.3 What Are Statistical Tolerance Intervals?

The tolerance interval is a bound on an estimate of the proportion of data in a population.

> A statistical tolerance interval [contains] a specified proportion of the units from the sampled population or process.
>
> — Page 3, *Statistical Intervals: A Guide for Practitioners and Researchers*, 2017.

The interval is limited by the sampling error and by the variance of the population distribution. Given the law of large numbers, as the sample size is increased, the probabilities will better match the underlying population distribution. Below is an example of a stated tolerance interval:

> The range from x to y covers 95% of the data with a confidence of 99%.

If the data is Gaussian, the interval can be expressed in the context of the mean value; for example:

> x +/- y covers 95% of the data with a confidence of 99%.

We refer to these intervals as statistical tolerance intervals, to differentiate them from tolerance intervals in engineering that describe limits of acceptability, such as for a design or of a material. Generally, we will describe them as simply *tolerance intervals* for convenience. A tolerance interval is defined in terms of two quantities:

- **Coverage**: The proportion of the population covered by the interval.
- **Confidence**: The probabilistic confidence that the interval covers the proportion of the population.

> The tolerance interval is constructed from data using two coefficients, the coverage and the tolerance coefficient. The coverage is the proportion of the population (p) that the interval is supposed to contain. The tolerance coefficient is the degree of confidence with which the interval reaches the specified coverage. A tolerance interval with coverage of 95% and a tolerance coefficient of 90% will contain 95% of the population distribution with a confidence of 90%.
>
> — Page 175, *Statistics for Environmental Engineers*, Second Edition, 2002.

20.4 How to Calculate Tolerance Intervals

The size of a tolerance interval is proportional to the size of the data sample from the population and the variance of the population. There are two main methods for calculating tolerance intervals depending on the distribution of data: parametric and nonparametric methods.

- **Parametric Tolerance Interval**: Use knowledge of the population distribution in specifying both the coverage and confidence. Often used to refer to a Gaussian distribution.
- **Nonparametric Tolerance Interval**: Use rank statistics to estimate the coverage and confidence, often resulting less precision (wider intervals) given the lack of information about the distribution.

Tolerance intervals are relatively straightforward to calculate for a sample of independent observations drawn from a Gaussian distribution. We will demonstrate this calculation in the next section.

20.5 Tolerance Interval for Gaussian Distribution

In this section, we will work through an example of calculating the tolerance intervals on a data sample. Note, the equations are based on those specified in the Engineering Statistics Handbook[1]. First, let's define our data sample. We will create a sample of 100 observations drawn from a Gaussian distribution with a mean of 50 and a standard deviation of 5.

```
...
# generate dataset
data = 5 * randn(100) + 50
```

Listing 20.1: Example of generating Gaussian random numbers.

During the example, we will assume that we are unaware of the true population mean and standard deviation, and that these values must be estimated. Because the population parameters have to be estimated, there is additional uncertainty. For example, for a 95% coverage, we could use 1.96 (or 2) standard deviations from the estimated mean as the tolerance interval. We must estimate the mean and standard deviation from the sample and take this uncertainty into account, therefore the calculation of the interval is slightly more complex.

Next, we must specify the number of degrees of freedom. This will be used in the calculation of critical values and in the calculation of the interval. Specifically, it is used in the calculation of the standard deviation. Remember that the degrees of freedom are the number of values in the calculation that can vary. Here, we have 100 observations, therefore 100 degrees of freedom. We do not know the standard deviation, therefore it must be estimated using the mean. This means our degrees of freedom will be (N - 1) or 99.

```
...
# specify degrees of freedom
n = len(data)
dof = n - 1
```

Listing 20.2: Example of specifying degrees of freedom.

[1] http://www.itl.nist.gov/div898/handbook/prc/section2/prc263.htm

Next, we must specify the proportional coverage of the data. In this example, we are interested in the middle 95% of the data. The proportion is 95. We must shift this proportion so that it covers the middle 95%, that is from 2.5th percentile to the 97.5th percentile. We know that the critical value for 95% is 1.96 given that we use it so often; nevertheless, we can calculate it directly in Python using the percentage point function, covered in Chapter 11.

```
...
# specify data coverage
prop = 0.95
prop_inv = (1.0 - prop) / 2.0
gauss_critical = norm.ppf(prop_inv)
```

Listing 20.3: Example of calculating a critical value of the Gaussian distribution.

Next, we need to calculate the confidence of the coverage. We can do this by retrieving the critical value from the Chi-Squared distribution for the given number of degrees of freedom and desired probability. We can use the `chi2.ppf()` SciPy function.

```
...
# specify confidence
prob = 0.99
prop_inv = 1.0 - prob
chi_critical = chi2.ppf(prop_inv, dof)
```

Listing 20.4: Example of calculating a critical value of the Chi-Squared distribution.

We now have all of the pieces to calculate the Gaussian tolerance interval. The calculation is as follows:

$$interval = \sqrt{\frac{dof \times (1 + \frac{1}{n})) \times \text{gauss\_critical}^2}{\text{chi\_critical}}} \tag{20.1}$$

Where dof is the number of degrees of freedom, n is the size of the data sample, `gauss_critical` is the critical value from the Gaussian distribution, such as 1.96 for 95% coverage of the population, and `chi_critical` is the critical value from the Chi-Squared distribution for the desired confidence and degrees of freedom.

```
...
# calculate tolerance interval
interval = sqrt((dof * (1 + (1/n)) * gauss_critical**2) / chi_critical)
```

Listing 20.5: Example of calculating a tolerance interval for the Gaussian distribution.

We can tie all of this together and calculate the Gaussian tolerance interval for our data sample. The complete example is listed below.

```
# parametric tolerance interval
from numpy.random import seed
from numpy.random import randn
from numpy import mean
from numpy import sqrt
from scipy.stats import chi2
from scipy.stats import norm
# seed the random number generator
seed(1)
# generate dataset
```

```
data = 5 * randn(100) + 50
# specify degrees of freedom
n = len(data)
dof = n - 1
# specify data coverage
prop = 0.95
prop_inv = (1.0 - prop) / 2.0
gauss_critical = norm.ppf(prop_inv)
print('Gaussian critical value: %.3f (coverage=%d%%)' % (gauss_critical, prop*100))
# specify confidence
prob = 0.99
prop_inv = 1.0 - prob
chi_critical = chi2.ppf(prop_inv, dof)
print('Chi-Squared critical value: %.3f (prob=%d%%, dof=%d)' % (chi_critical, prob*100,
    dof))
# tolerance
interval = sqrt((dof * (1 + (1/n)) * gauss_critical**2) / chi_critical)
print('Tolerance Interval: %.3f' % interval)
# summarize
data_mean = mean(data)
lower, upper = data_mean-interval, data_mean+interval
print('%.2f to %.2f covers %d%% of data with a confidence of %d%%' % (lower, upper,
    prop*100, prob*100))
```

Listing 20.6: Example of calculating Gaussian tolerance interval.

Running the example first calculates and prints the relevant critical values for the Gaussian and Chi-Squared distributions. The tolerance is printed, then presented correctly.

```
Gaussian critical value: -1.960 (coverage=95%)
Chi-Squared critical value: 69.230 (prob=99%, dof=99)
Tolerance Interval: 2.355
47.95 to 52.66 covers 95% of data with a confidence of 99%
```

Listing 20.7: Sample output from calculating Gaussian tolerance interval.

It can also be helpful to demonstrate how the tolerance interval will decrease (become more precise) as the size of the sample is increased. The example below demonstrates this by calculating the tolerance interval for different sample sizes for the same small contrived problem.

```
# plot tolerance interval vs sample size
from numpy.random import seed
from numpy.random import randn
from numpy import sqrt
from scipy.stats import chi2
from scipy.stats import norm
from matplotlib import pyplot
# seed the random number generator
seed(1)
# sample sizes
sizes = range(5,15)
for n in sizes:
  # generate dataset
  data = 5 * randn(n) + 50
  # calculate degrees of freedom
  dof = n - 1
  # specify data coverage
```

```
  prop = 0.95
  prop_inv = (1.0 - prop) / 2.0
  gauss_critical = norm.ppf(prop_inv)
  # specify confidence
  prob = 0.99
  prop_inv = 1.0 - prob
  chi_critical = chi2.ppf(prop_inv, dof)
  # tolerance
  tol = sqrt((dof * (1 + (1/n)) * gauss_critical**2) / chi_critical)
  # plot
  pyplot.errorbar(n, 50, yerr=tol, color='blue', fmt='o')
# plot results
pyplot.show()
```

Listing 20.8: Example of comparing tolerance interval to sample size.

Running the example creates a plot showing the tolerance interval around the true population mean. We can see that the interval becomes smaller (more precise) as the sample size is increased from 5 to 15 examples.

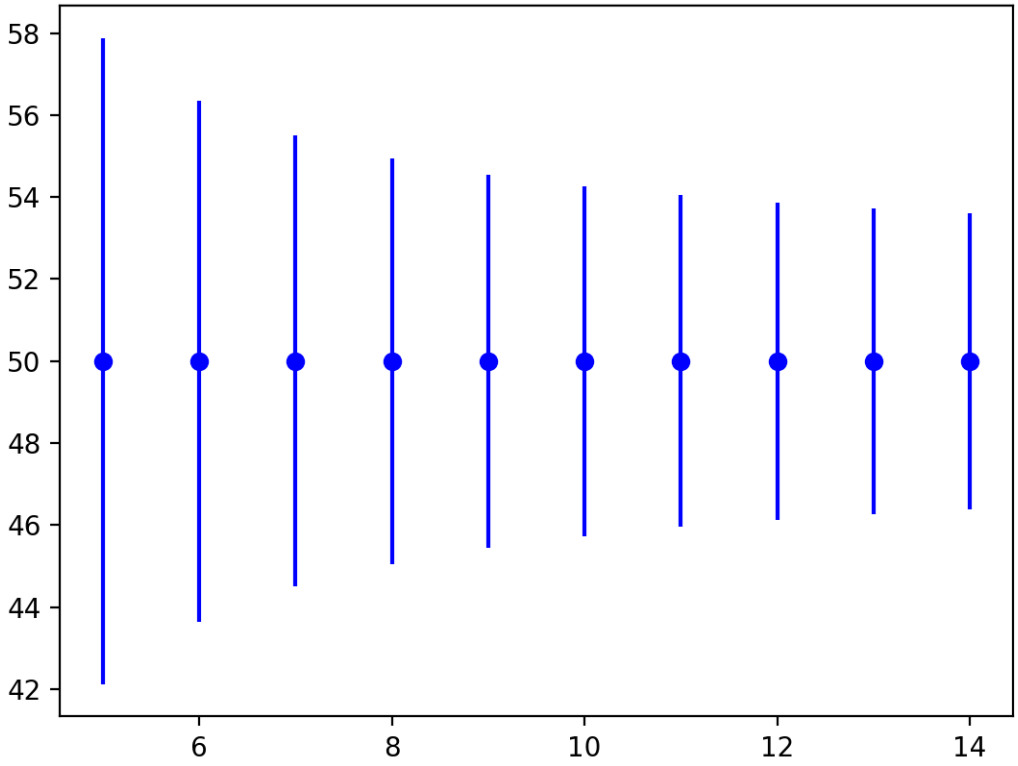

Figure 20.1: Error bar plot of tolerance interval vs sample size.

20.6 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- List 3 cases where a tolerance interval could be used in a machine learning project.
- Locate a dataset with a Gaussian variable and calculate tolerance intervals for it.
- Research and describe one method for calculating a nonparametric tolerance interval.

If you explore any of these extensions, I'd love to know.

20.7 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

20.7.1 Books

- *Understanding The New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis*, 2017.
  http://amzn.to/2oQW6No
- *Statistical Intervals: A Guide for Practitioners and Researchers*, 2017.
  http://amzn.to/2G8w3IL

20.7.2 API

- `scipy.stats.norm` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html
- `scipy.stats.chi2` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chi2.html
- `matplotlib.pyplot.errorbar` API.
  https://matplotlib.org/2.1.0/api/_as_gen/matplotlib.pyplot.errorbar.html

20.7.3 Articles

- Tolerance interval on Wikipedia.
  https://en.wikipedia.org/wiki/Tolerance_interval
- 68-95-99.7 rule on Wikipedia.
  https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule
- Percentile on Wikipedia.
  https://en.wikipedia.org/wiki/Percentile
- Tolerance intervals for a normal distribution.
  http://www.itl.nist.gov/div898/handbook/prc/section2/prc263.htm

20.8 Summary

In this tutorial, you discovered statistical tolerance intervals and how to calculate a tolerance interval for Gaussian data. Specifically, you learned:

- That statistical tolerance intervals provide a bounds on observations from a population.
- That a tolerance interval requires that both a coverage proportion and confidence be specified.
- That the tolerance interval for a data sample with a Gaussian distribution can be easily calculated.

20.8.1 Next

In the next section, you will discover confidence intervals for quantifying likely values for estimated population parameters.

Chapter 21

Confidence Intervals

Much of machine learning involves estimating the performance of a machine learning algorithm on unseen data. Confidence intervals are a way of quantifying the uncertainty of an estimate. They can be used to add a bounds or likelihood on a population parameter, such as a mean, estimated from a sample of independent observations from the population. In this tutorial, you will discover confidence intervals and how to calculate confidence intervals in practice. After completing this tutorial, you will know:

- That a confidence interval is a bounds on an estimate of a population parameter.
- That the confidence interval for the estimated skill of a classification method can be calculated directly.
- That the confidence interval for any arbitrary population statistic can be estimated in a distribution-free way using the bootstrap.

Let's get started.

21.1 Tutorial Overview

This tutorial is divided into 3 parts; they are:

1. What is a Confidence Interval?
2. Interval for Classification Accuracy
3. Nonparametric Confidence Interval

21.2 What is a Confidence Interval?

A confidence interval is a bounds on the estimate of a population variable. It is an interval statistic used to quantify the uncertainty on an estimate.

> A confidence interval is used to contain an unknown characteristic of the population or process. The quantity of interest might be a population property or "parameter", such as the mean or standard deviation of the population or process.

— Page 3, *Statistical Intervals: A Guide for Practitioners and Researchers*, 2017.

A confidence interval is different from a tolerance interval that describes the bounds of data sampled from the distribution. It is also different from a prediction interval that describes the bounds on a single observation. Instead, the confidence interval provides bounds on a population parameter, such as a mean, standard deviation, or similar. In applied machine learning, we may wish to use confidence intervals in the presentation of the skill of a predictive model.

For example, a confidence interval could be used in presenting the skill of a classification model, which could be stated as:

> Given the sample, there is a 95% likelihood that the range x to y covers the true model accuracy.

or

> The accuracy of the model was x +/- y at the 95% confidence level.

Confidence intervals can also be used in the presentation of the error of a regression predictive model; for example:

> There is a 95% likelihood that the range x to y covers the true error of the model.

or

> The error of the model was x +/- y at the 95% confidence level.

The choice of 95% confidence is very common in presenting confidence intervals, although other less common values are used, such as 90% and 99.7%. In practice, you can use any value you prefer.

> The 95% confidence interval (CI) is a range of values calculated from our data, that most likely, includes the true value of what we're estimating about the population.

— Page 4, *Introduction to the New Statistics: Estimation, Open Science, and Beyond*, 2016.

The value of a confidence interval is its ability to quantify the uncertainty of the estimate. It provides both a lower and upper bound and a likelihood. Taken as a radius measure alone, the confidence interval is often referred to as the margin of error and may be used to graphically depict the uncertainty of an estimate on graphs through the use of error bars. Often, the larger the sample from which the estimate was drawn, the more precise the estimate and the smaller (better) the confidence interval.

- **Smaller Confidence Interval**: A more precise estimate.
- **Larger Confidence Interval**: A less precise estimate.

> We can also say that the CI tells us how precise our estimate is likely to be, and the margin of error is our measure of precision. A short CI means a small margin of error and that we have a relatively precise estimate [...] A long CI means a large margin of error and that we have a low precision

— Page 4, *Introduction to the New Statistics: Estimation, Open Science, and Beyond*, 2016.

Confidence intervals belong to a field of statistics called estimation statistics that can be used to present and interpret experimental results instead of, or in addition to, statistical significance tests.

> Estimation gives a more informative way to analyze and interpret results. [...] Knowing and thinking about the magnitude and precision of an effect is more useful to quantitative science than contemplating the probability of observing data of at least that extremity, assuming absolutely no effect.
>
> — *Estimation statistics should replace significance testing*, 2016.

Confidence intervals may be preferred in practice over the use of statistical significance tests. The reason is that they are easier for practitioners and stakeholders to relate directly to the domain. They can also be interpreted and used to compare machine learning models.

> These estimates of uncertainty help in two ways. First, the intervals give the consumers of the model an understanding about how good or bad the model may be. [...] In this way, the confidence interval helps gauge the weight of evidence available when comparing models. The second benefit of the confidence intervals is to facilitate trade-offs between models. If the confidence intervals for two models significantly overlap, this is an indication of (statistical) equivalence between the two and might provide a reason to favor the less complex or more interpretable model.
>
> — Page 416, *Applied Predictive Modeling*, 2013.

Now that we know what a confidence interval is, let's look at a few ways that we can calculate them for predictive models.

21.3 Interval for Classification Accuracy

Classification problems are those where a label or class outcome variable is predicted given some input data. It is common to use classification accuracy or classification error (the inverse of accuracy) to describe the skill of a classification predictive model. For example, a model that makes correct predictions of the class outcome variable 75% of the time has a classification accuracy of 75%, calculated as:

$$accuracy = \frac{\text{total correct predictions}}{\text{total predictions made}} \times 100 \tag{21.1}$$

This accuracy can be calculated based on a hold-out dataset not seen by the model during training, such as a validation or test dataset. Classification accuracy or classification error is a proportion or a ratio. It describes the proportion of correct or incorrect predictions made by the model. Each prediction is a binary decision that could be correct or incorrect. Technically, this is called a Bernoulli trial, named for Jacob Bernoulli. The proportions in a Bernoulli trial have a specific distribution called a binomial distribution. Thankfully, with large sample sizes (e.g. more than 30), we can approximate the distribution with a Gaussian.

> In statistics, a succession of independent events that either succeed or fail is called a Bernoulli process. [...] For large N, the distribution of this random variable approaches the normal distribution.

— Page 148, *Data Mining: Practical Machine Learning Tools and Techniques*, Second Edition, 2005.

We can use the assumption of a Gaussian distribution of the proportion (i.e. the classification accuracy or error) to easily calculate the confidence interval. In the case of classification error, the radius of the interval can be calculated as:

$$interval = z \times \sqrt{\frac{error \times (1 - error)}{n}} \tag{21.2}$$

In the case of classification accuracy, the radius of the interval can be calculated as:

$$interval = z \times \sqrt{\frac{accuracy \times (1 - accuracy)}{n}} \tag{21.3}$$

Where *interval* is the radius of the confidence interval, error and accuracy are classification error and classification accuracy respectively, n is the size of the sample, and z is a critical value from the Gaussian distribution. Technically, this is called the Binomial proportion confidence interval. Commonly used critical values from the Gaussian distribution and their corresponding significance level are as follows:

- 1.64 (90%)
- 1.96 (95%)
- 2.33 (98%)
- 2.58 (99%)

Consider a model with an error of 20%, or 0.2 (error = 0.2), on a validation dataset with 50 examples (n = 50). We can calculate the 95% confidence interval (z = 1.96) as follows:

```
# binomial confidence interval
from math import sqrt
# calculate the interval
interval = 1.96 * sqrt( (0.2 * (1 - 0.2)) / 50)
print('%.3f' % interval)
```

Listing 21.1: Example of calculating a confidence interval with 50 samples.

Running the example, we see the calculated radius of the confidence interval calculated and printed.

```
0.111
```

Listing 21.2: Sample output from calculating a confidence interval with 50 samples.

We can then make claims such as:

- The classification error of the model is 20% +/- 11%

- The true classification error of the model is likely between 9% and 31%.

We can see the impact that the sample size has on the precision of the estimate in terms of the radius of the confidence interval.

```
# binomial confidence interval
from math import sqrt
interval = 1.96 * sqrt( (0.2 * (1 - 0.2)) / 100)
print('%.3f' % interval)
```

Listing 21.3: Example of calculating a confidence interval with 100 samples.

Running the example shows that the confidence interval drops to about 7%, increasing the precision of the estimate of the models skill.

```
0.078
```

Listing 21.4: Sample output from calculating a confidence interval with 100 samples.

Remember that the confidence interval is a likelihood over a range. The true model skill may lie outside of the range.

> In fact, if we repeated this experiment over and over, each time drawing a new sample S, containing [...] new examples, we would find that for approximately 95% of these experiments, the calculated interval would contain the true error. For this reason, we call this interval the 95% confidence interval estimate
>
> — Page 131, *Machine Learning*, 1997.

The `proportion_confint()` Statsmodels function an implementation of the binomial proportion confidence interval. By default, it makes the Gaussian assumption for the Binomial distribution, although other more sophisticated variations on the calculation are supported. The function takes the count of successes (or failures), the total number of trials, and the significance level as arguments and returns the lower and upper bound of the confidence interval. The example below demonstrates this function in a hypothetical case where a model made 88 correct predictions out of a dataset with 100 instances and we are interested in the 95% confidence interval (provided to the function as a significance of 0.05).

```
# calculate the confidence interval
from statsmodels.stats.proportion import proportion_confint
# calculate the interval
lower, upper = proportion_confint(88, 100, 0.05)
print('lower=%.3f, upper=%.3f' % (lower, upper))
```

Listing 21.5: Example of calculating a confidence interval with a function.

Running the example prints the lower and upper bounds on the model's classification accuracy.

```
lower=0.816, upper=0.944
```

Listing 21.6: Sample output from calculating a confidence interval with a function.

21.4 Nonparametric Confidence Interval

Often we do not know the distribution for a chosen performance measure. Alternately, we may not know the analytical way to calculate a confidence interval for a skill score.

> The assumptions that underlie parametric confidence intervals are often violated. The predicted variable sometimes isn't normally distributed, and even when it is, the variance of the normal distribution might not be equal at all levels of the predictor variable.
>
> — Page 326, *Empirical Methods for Artificial Intelligence*, 1995.

In these cases, the bootstrap resampling method can be used as a nonparametric method for calculating confidence intervals, nominally called bootstrap confidence intervals. The bootstrap is a simulated Monte Carlo method where samples are drawn from a fixed finite dataset with replacement and a parameter is estimated on each sample. This procedure leads to a robust estimate of the true population parameter via sampling. The bootstrap method was covered in detail in Chapter 17. We can demonstrate this with the following pseudocode.

```
statistics = []
for i in bootstraps:
  sample = select_sample_with_replacement(data)
  stat = calculate_statistic(sample)
  statistics.append(stat)
```

Listing 21.7: Pseudocode for estimating a statistic using the bootstrap.

The procedure can be used to estimate the skill of a predictive model by fitting the model on each sample and evaluating the skill of the model on those samples not included in the sample. The mean or median skill of the model can then be presented as an estimate of the model skill when evaluated on unseen data. Confidence intervals can be added to this estimate by selecting observations from the sample of skill scores at specific percentiles.

Recall that a percentile is an observation value drawn from the sorted sample where a percentage of the observations in the sample fall. For example, the 70th percentile of a sample indicates that 70% of the samples fall below that value. The 50th percentile is the median or middle of the distribution. First, we must choose a significance level for the confidence level, such as 95%, represented as 5.0% (e.g. 100 - 95). Because the confidence interval is symmetric around the median, we must choose observations at the 2.5th percentile and the 97.5th percentiles to give the full range.

We can make the calculation of the bootstrap confidence interval concrete with a worked example. Let's assume we have a dataset of 1,000 observations of values between 0.5 and 1.0 drawn from a uniform distribution.

```
...
# generate dataset
dataset = 0.5 + rand(1000) * 0.5
```

Listing 21.8: Example of generating Gaussian random numbers.

We will perform the bootstrap procedure 100 times and draw samples of 1,000 observations from the dataset with replacement. We will estimate the mean of the population as the statistic we will calculate on the bootstrap samples. This could just as easily be a model evaluation.

```
...
# bootstrap
scores = list()
for _ in range(100):
  # bootstrap sample
  indices = randint(0, 1000, 1000)
  sample = dataset[indices]
  # calculate and store statistic
  statistic = mean(sample)
  scores.append(statistic)
```

Listing 21.9: Example of estimating the mean using the bootstrap.

Once we have a sample of bootstrap statistics, we can calculate the central tendency. We will use the median or 50th percentile as we do not assume any distribution.

```
...
# calculate the median
print('median=%.3f' % median(scores))
```

Listing 21.10: Example of printing the median.

We can then calculate the confidence interval as the middle 95% of observed statistical values centered around the median.

```
...
# calculate 95% confidence intervals (100 - alpha)
alpha = 5.0
```

Listing 21.11: Define the level of confidence.

First, the desired lower percentile is calculated based on the chosen confidence interval. Then the observation at this percentile is retrieved from the sample of bootstrap statistics.

```
...
# calculate lower percentile (e.g. 2.5)
lower_p = alpha / 2.0
# retrieve observation at lower percentile
lower = max(0.0, percentile(scores, lower_p))
```

Listing 21.12: Example of calculating the lower-bound on the confidence interval.

We do the same thing for the upper boundary of the confidence interval.

```
...
# calculate upper percentile (e.g. 97.5)
upper_p = (100 - alpha) + (alpha / 2.0)
# retrieve observation at upper percentile
upper = min(1.0, percentile(scores, upper_p))
```

Listing 21.13: Example of calculating the upper-bound on the confidence interval.

The complete example is listed below.

```
# bootstrap confidence intervals
from numpy.random import seed
from numpy.random import rand
from numpy.random import randint
```

```
from numpy import mean
from numpy import median
from numpy import percentile
# seed the random number generator
seed(1)
# generate dataset
dataset = 0.5 + rand(1000) * 0.5
# bootstrap
scores = list()
for _ in range(100):
  # bootstrap sample
  indices = randint(0, 1000, 1000)
  sample = dataset[indices]
  # calculate and store statistic
  statistic = mean(sample)
  scores.append(statistic)
print('50th percentile (median) = %.3f' % median(scores))
# calculate 95% confidence intervals (100 - alpha)
alpha = 5.0
# calculate lower percentile (e.g. 2.5)
lower_p = alpha / 2.0
# retrieve observation at lower percentile
lower = max(0.0, percentile(scores, lower_p))
print('%.1fth percentile = %.3f' % (lower_p, lower))
# calculate upper percentile (e.g. 97.5)
upper_p = (100 - alpha) + (alpha / 2.0)
# retrieve observation at upper percentile
upper = min(1.0, percentile(scores, upper_p))
print('%.1fth percentile = %.3f' % (upper_p, upper))
```

Listing 21.14: Example of calculating a confidence interval using the bootstrap.

Running the example summarizes the distribution of bootstrap sample statistics including the 2.5th, 50th (median) and 97.5th percentile.

```
50th percentile (median) = 0.750
2.5th percentile = 0.741
97.5th percentile = 0.757
```

Listing 21.15: Sample output from calculating a confidence interval using the bootstrap.

We can then use these observations to make a claim about the sample distribution, such as:

> There is a 95% likelihood that the range 0.741 to 0.757 covers the true statistic median.

21.5 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Test each confidence interval method on your own small contrived test datasets.
- Find 3 research papers that demonstrate the use of each confidence interval method.

- Develop a function to calculate a bootstrap confidence interval for a given sample of machine learning skill scores.

If you explore any of these extensions, I'd love to know.

21.6 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

21.6.1 Books

- *Understanding The New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis*, 2011.
  http://amzn.to/2oQW6No
- *Introduction to the New Statistics: Estimation, Open Science, and Beyond*, 2016.
  http://amzn.to/2HhrT0w
- *Statistical Intervals: A Guide for Practitioners and Researchers*, 2017.
  http://amzn.to/2G8w3IL
- *Applied Predictive Modeling*, 2013.
  http://amzn.to/2Fmrbib
- *Machine Learning*, 1997.
  http://amzn.to/2tr32Wb
- *Data Mining: Practical Machine Learning Tools and Techniques*, Second Edition, 2005.
  http://amzn.to/2G7sxhP
- *An Introduction to the Bootstrap*, 1996.
  http://amzn.to/2p2zUPl
- *Empirical Methods for Artificial Intelligence*, 1995.
  http://amzn.to/2FrFJgg

21.6.2 Papers

- *Estimation statistics should replace significance testing*, 2016.
  https://www.nature.com/articles/nmeth.3729
- *Bootstrap Confidence Intervals, Statistical Science*, 1996.
  https://projecteuclid.org/download/pdf_1/euclid.ss/1032280214

21.6.3 API

- `statsmodels.stats.proportion.proportion_confint` API.
  http://www.statsmodels.org/dev/generated/statsmodels.stats.proportion.proportion_confint.html

21.6.4 Articles

- Interval estimation on Wikipedia.
  https://en.wikipedia.org/wiki/Interval_estimation
- Confidence interval on Wikipedia.
  https://en.wikipedia.org/wiki/Confidence_interval
- Binomial proportion confidence interval on Wikipedia.
  https://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval
- Confidence interval of RMSE on Cross Validated.
  https://stats.stackexchange.com/questions/78079/confidence-interval-of-rmse
- Bootstrapping on Wikipedia.
  https://en.wikipedia.org/wiki/Bootstrapping_(statistics)

21.7 Summary

In this tutorial, you discovered confidence intervals and how to calculate confidence intervals in practice. Specifically, you learned:

- That a confidence interval is a bounds on an estimate of a population parameter.
- That the confidence interval for the estimated skill of a classification method can be calculated directly.
- That the confidence interval for any arbitrary population statistic can be estimated in a distribution-free way using the bootstrap.

21.7.1 Next

In the next section, you will discover prediction intervals for the likely range for a point prediction.

Chapter 22

Prediction Intervals

A prediction from a machine learning perspective is a single point that hides the uncertainty of that prediction. Prediction intervals provide a way to quantify and communicate the uncertainty in a prediction. They are different from confidence intervals that instead seek to quantify the uncertainty in a population parameter such as a mean or standard deviation. Prediction intervals describe the uncertainty for a single specific outcome. In this tutorial, you will discover the prediction interval and how to calculate it for a simple linear regression model. After completing this tutorial, you will know:

- That a prediction interval quantifies the uncertainty of a single point prediction.
- That prediction intervals can be estimated analytically for simple models, but are more challenging for nonlinear machine learning models.
- How to calculate the prediction interval for a simple linear regression model.

Let's get started.

22.1 Tutorial Overview

This tutorial is divided into 5 parts; they are:

1. Why Calculate a Prediction Interval?
2. What Is a Prediction Interval?
3. How to Calculate a Prediction Interval
4. Prediction Interval for Linear Regression
5. Worked Example

22.2 Why Calculate a Prediction Interval?

In predictive modeling, a prediction or a forecast is a single outcome value given some input variables. For example:

```
...
yhat = model.predict(X)
```

Listing 22.1: Make a prediction with a model.

Where `yhat` is the estimated outcome or prediction made by the trained model for the given input data X. This is a point prediction. By definition, it is an estimate or an approximation and contains some uncertainty. The uncertainty comes from the errors in the model itself and noise in the input data. The model is an approximation of the relationship between the input variables and the output variables.

Given the process used to choose and tune the model, it will be the best approximation made given available information, but it will still make errors. Data from the domain will naturally obscure the underlying and unknown relationship between the input and output variables. This will make it a challenge to fit the model, and will also make it a challenge for a fit model to make predictions. Given these two main sources of error, their point prediction from a predictive model is insufficient for describing the true uncertainty of the prediction.

22.3 What Is a Prediction Interval?

A prediction interval is a quantification of the uncertainty on a prediction. It provides a probabilistic upper and lower bounds on the estimate of an outcome variable.

> A prediction interval for a single future observation is an interval that will, with a specified degree of confidence, contain a future randomly selected observation from a distribution.
>
> — Page 27, *Statistical Intervals: A Guide for Practitioners and Researchers*, 2017.

Prediction intervals are most commonly used when making predictions or forecasts with a regression model, where a quantity is being predicted. An example of the presentation of a prediction interval is as follows:

> Given a prediction of 'y' given 'x', there is a 95% likelihood that the range 'a' to 'b' covers the true outcome.

The prediction interval surrounds the prediction made by the model and hopefully covers the range of the true outcome. The diagram below helps to visually understand the relationship between the prediction, prediction interval, and the actual outcome.

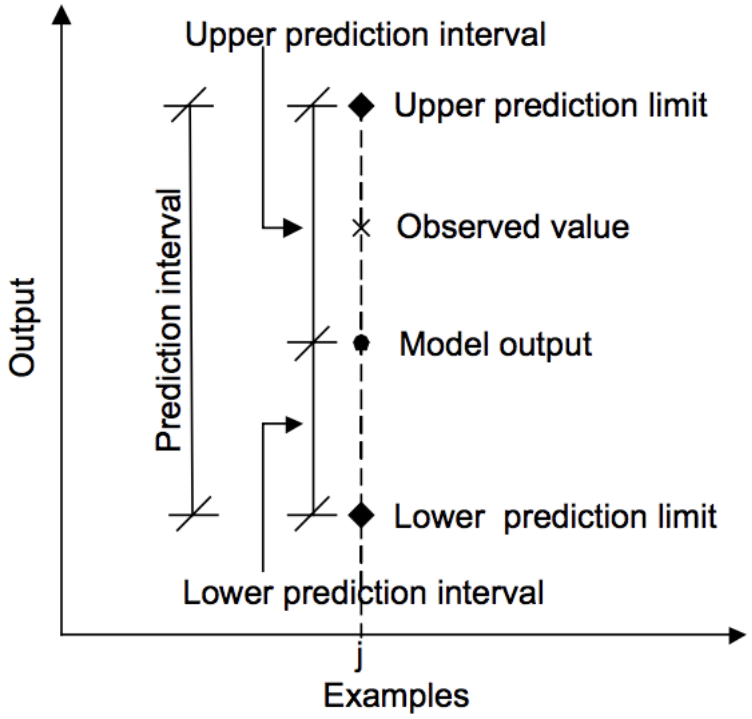

Figure 22.1: Relationship between prediction, actual value and prediction interval. Taken from *Machine learning approaches for estimation of prediction interval for the model output.*

A prediction interval is different from a confidence interval. A confidence interval quantifies the uncertainty on an estimated population variable, such as the mean or standard deviation. Whereas a prediction interval quantifies the uncertainty on a single observation estimated from the population.

In predictive modeling, a confidence interval can be used to quantify the uncertainty of the estimated skill of a model, whereas a prediction interval can be used to quantify the uncertainty of a single forecast. A prediction interval is often larger than the confidence interval as it must take the confidence interval and the variance in the output variable being predicted into account.

> Prediction intervals will always be wider than confidence intervals because they account for the uncertainty associated with e [error], the irreducible error.
>
> — Page 103, *An Introduction to Statistical Learning: with Applications in R*, 2013.

22.4 How to Calculate a Prediction Interval

A prediction interval is calculated as some combination of the estimated variance of the model and the variance of the outcome variable. Prediction intervals are easy to describe, but difficult to calculate in practice. In simple cases like linear regression, we can estimate the prediction interval directly.

In the cases of nonlinear regression algorithms, such as artificial neural networks, it is a lot more challenging and requires the choice and implementation of specialized techniques. General techniques such as the bootstrap resampling method can be used, but are computationally

expensive to calculate. The paper *A Comprehensive Review of Neural Network-based Prediction Intervals and New Advances* provides a reasonably recent study of prediction intervals for nonlinear models in the context of neural networks. The following list summarizes some methods that can be used for prediction uncertainty for nonlinear machine learning models:

- The Delta Method, from the field of nonlinear regression.
- The Bayesian Method, from Bayesian modeling and statistics.
- The Mean-Variance Estimation Method, using estimated statistics.
- The Bootstrap Method, using data resampling and developing an ensemble of models.

We can make the calculation of a prediction interval concrete with a worked example in the next section.

22.5 Prediction Interval for Linear Regression

A linear regression is a model that describes the linear combination of inputs to calculate the output variables. For example, an estimated linear regression model may be written as:

$$\hat{y} = b_0 + b_1 \times x \tag{22.1}$$

Where $\hat{y}$ (or yhat) is the prediction, b_0 and b_1 are coefficients of the model estimated from training data and x is the input variable. We do not know the true values of the coefficients b_0 and b_1. We also do not know the true population parameters such as mean and standard deviation for x or y. All of these elements must be estimated, which introduces uncertainty into the use of the model in order to make predictions. We can make some assumptions, such as the distributions of x and y and the prediction errors made by the model, called residuals, are Gaussian. The prediction interval around $\hat{y}$ can be calculated as follows:

$$\hat{y} \pm z \times \sigma \tag{22.2}$$

Where $\hat{y}$ is the predicted value, z is the critical value from the Gaussian distribution (e.g. 1.96 for a 95% interval) and σ is the standard deviation of the predicted distribution.

We do not known in practice. We can calculate an unbiased estimate of the of the predicted standard deviation as follows[1]:

$$stdev = \sqrt{\frac{1}{n-2} \times \sum_{i=1}^{n} (\hat{y}_i - y_i)^2} \tag{22.3}$$

Where $stdev$ is an unbiased estimate of the standard deviation for the predicted distribution, n are the total predictions made, $\hat{y}_i$ is the i^{th} prediction and y_i is the actual i^{th} value.

[1]Taken from *Machine learning approaches for estimation of prediction interval for the model output*, 2006.

22.6 Worked Example

Let's make the case of linear regression prediction intervals concrete with a worked example. First, let's define a simple two-variable dataset where the output variable (y) depends on the input variable (x) with some Gaussian noise. The example below defines the dataset we will use for this example.

```
# generate related variables
from numpy import mean
from numpy import std
from numpy.random import randn
from numpy.random import seed
from matplotlib import pyplot
# seed random number generator
seed(1)
# prepare data
x = 20 * randn(1000) + 100
y = x + (10 * randn(1000) + 50)
# summarize
print('x: mean=%.3f stdv=%.3f' % (mean(x), std(x)))
print('y: mean=%.3f stdv=%.3f' % (mean(y), std(y)))
# plot
pyplot.scatter(x, y)
pyplot.show()
```

Listing 22.2: Example of creating and summarizing a test dataset.

Running the example first prints the mean and standard deviations of the two variables.

```
x: mean=100.776 stdv=19.620
y: mean=151.050 stdv=22.358
```

Listing 22.3: Sample output from creating and summarizing a test dataset.

A plot of the dataset is then created. We can see the clear linear relationship between the variables with the spread of the points highlighting the noise or random error in the relationship.

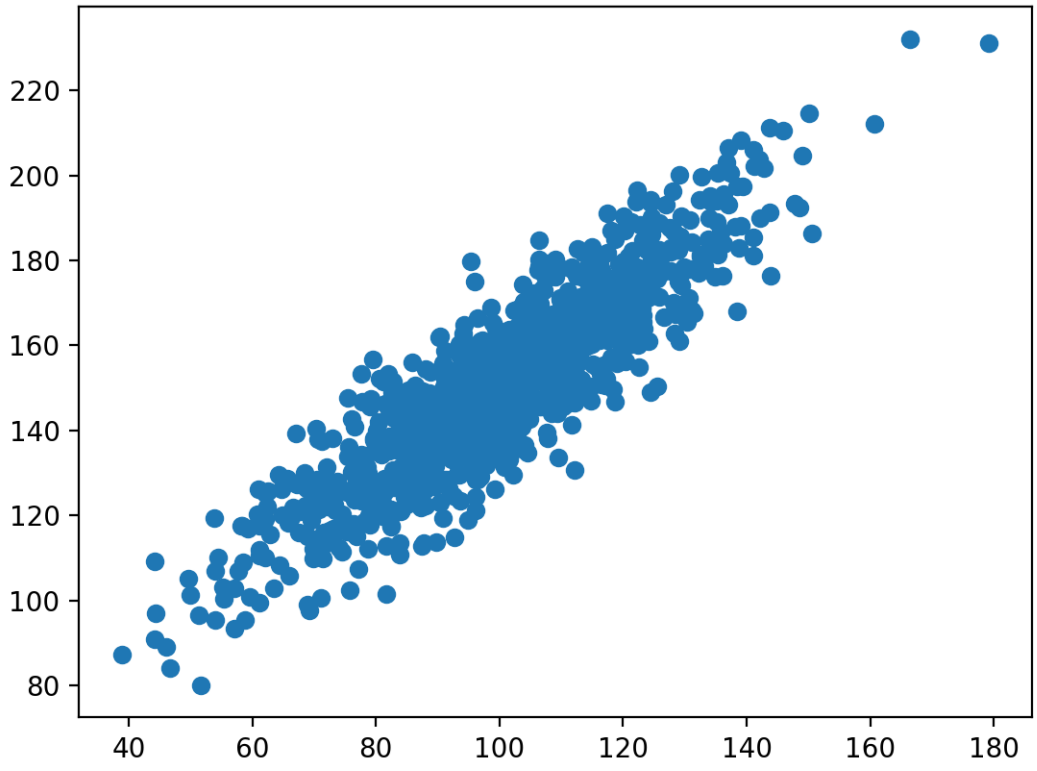

Figure 22.2: Scatter plot of related variables.

Next, we can develop a simple linear regression that given the input variable `x`, will predict the `y` variable. We can use the `linregress()` SciPy function to fit the model and return the `b0` and `b1` coefficients for the model.

```
...
# fit linear regression model
b1, b0, r_value, p_value, std_err = linregress(x, y)
```

Listing 22.4: Example of fitting a linear regression model.

We can use the coefficients to calculate the predicted `y` values, called `yhat`, for each of the input variables. The resulting points will form a line that represents the learned relationship.

```
...
# make prediction
yhat = b0 + b1 * x
```

Listing 22.5: Example of making predictions with a linear regression model.

The complete example is listed below.

```
# simple linear regression model
from numpy.random import randn
from numpy.random import seed
from scipy.stats import linregress
```

```
from matplotlib import pyplot
# seed random number generator
seed(1)
# prepare data
x = 20 * randn(1000) + 100
y = x + (10 * randn(1000) + 50)
# fit linear regression model
b1, b0, r_value, p_value, std_err = linregress(x, y)
print('b0=%.3f, b1=%.3f' % (b1, b0))
# make prediction
yhat = b0 + b1 * x
# plot data and predictions
pyplot.scatter(x, y)
pyplot.plot(x, yhat, color='r')
pyplot.show()
```

Listing 22.6: Example of fitting and predicting with a linear regression model.

Running the example fits the model and prints the coefficients.

```
b0=1.011, b1=49.117
```

Listing 22.7: Sample output from fitting and predicting with a linear regression model.

The coefficients are then used with the inputs from the dataset to make a prediction. The resulting inputs and predicted y-values are plotted as a line on top of the scatter plot for the dataset. We can clearly see that the model has learned the underlying relationship in the dataset.

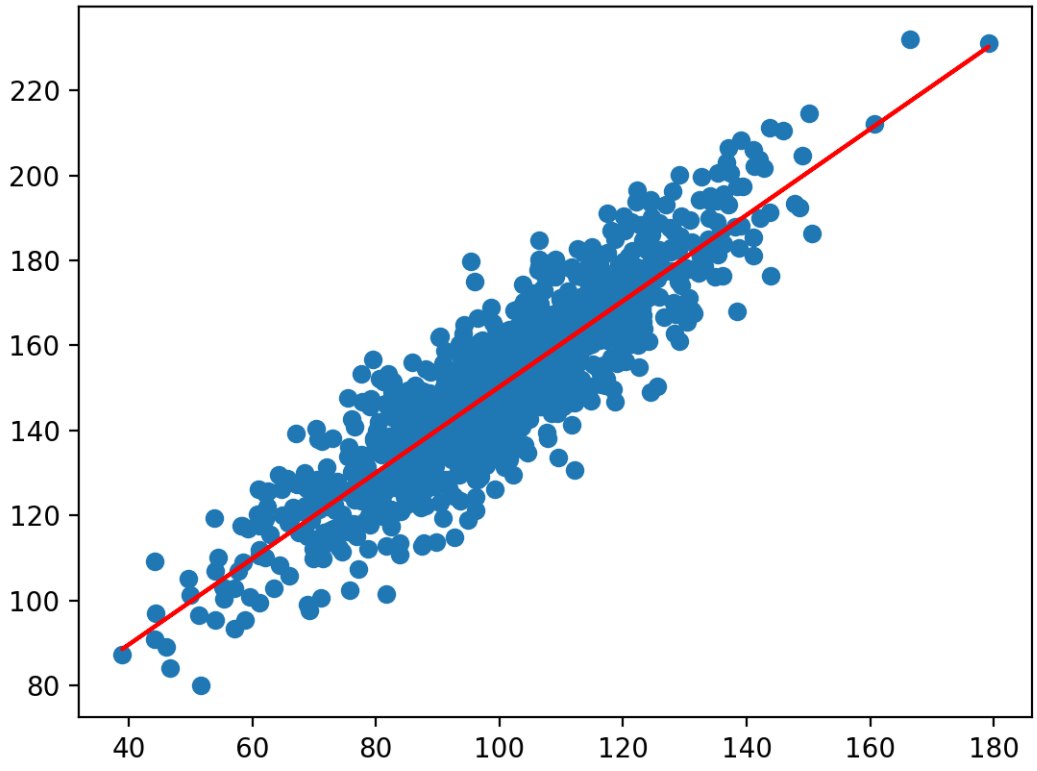

Figure 22.3: Scatter plot of dataset with line for simple linear regression model.

We are now ready to make a prediction with our simple linear regression model and add a prediction interval. We will fit the model as before. This time we will take one sample from the dataset to demonstrate the prediction interval. We will use the input to make a prediction, calculate the prediction interval for the prediction, and compare the prediction and interval to the known expected value. First, let's define the input, prediction, and expected values.

```
...
# define the prediction
x_in = x[0]
y_out = y[0]
yhat_out = yhat[0]
```

Listing 22.8: Example of defining a single prediction.

Next, we can estimate the standard deviation in the prediction direction.

$$stdev = \sqrt{\frac{1}{n-2} \times \sum_{i=1}^{n} (\hat{y}_i - y_i)^2} \tag{22.4}$$

We can calculate this directly using the NumPy arrays as follows:

```
...
# estimate stdev of yhat
```

```
sum_errs = arraysum((y - yhat)**2)
stdev = sqrt(1/(len(y)-2) * sum_errs)
```

Listing 22.9: Example of estimating the standard deviation for yhat.

Next, we can calculate the prediction interval for our chosen input:

$$\hat{y} \pm z \times \sigma \tag{22.5}$$

We will use the significance level of 95%, which is the Gaussian critical value of 1.69. Once the interval is calculated, we can summarize the bounds on the prediction to the user.

```
...
# calculate prediction interval
interval = 1.96 * stdev
lower, upper = y_out - interval, y_out + interval
```

Listing 22.10: Example of calculating the prediction interval.

We can tie all of this together. The complete example is listed below.

```
# linear regression prediction with prediction interval
from numpy.random import randn
from numpy.random import seed
from numpy import sqrt
from numpy import sum as arraysum
from scipy.stats import linregress
from matplotlib import pyplot
# seed random number generator
seed(1)
# prepare data
x = 20 * randn(1000) + 100
y = x + (10 * randn(1000) + 50)
# fit linear regression model
b1, b0, r_value, p_value, std_err = linregress(x, y)
# make predictions
yhat = b0 + b1 * x
# define new input, expected value and prediction
x_in = x[0]
y_out = y[0]
yhat_out = yhat[0]
# estimate stdev of yhat
sum_errs = arraysum((y - yhat)**2)
stdev = sqrt(1/(len(y)-2) * sum_errs)
# calculate prediction interval
interval = 1.96 * stdev
print('Prediction Interval: %.3f' % interval)
lower, upper = y_out - interval, y_out + interval
print('95%% likelihood that the true value is between %.3f and %.3f' % (lower, upper))
print('True value: %.3f' % yhat_out)
# plot dataset and prediction with interval
pyplot.scatter(x, y)
pyplot.plot(x, yhat, color='red')
pyplot.errorbar(x_in, yhat_out, yerr=interval, color='black', fmt='o')
pyplot.show()
```

Listing 22.11: Example of calculating a prediction interval for prediction with a linear regression model.

Running the example estimates the yhat standard deviation and then calculates the confidence interval. Once calculated, the prediction interval is presented to the user for the given input variable. Because we contrived this example, we know the true outcome, which we also display. We can see that in this case, the 95% prediction interval does cover the true expected value.

```
Prediction Interval: 20.204
95% likelihood that the true value is between 160.750 and 201.159
True value: 183.124
```

Listing 22.12: Sample output from calculating a prediction interval for prediction with a linear regression model.

A plot is also created showing the raw dataset as a scatter plot, the predictions for the dataset as a red line, and the prediction and prediction interval as a black dot and line respectively.

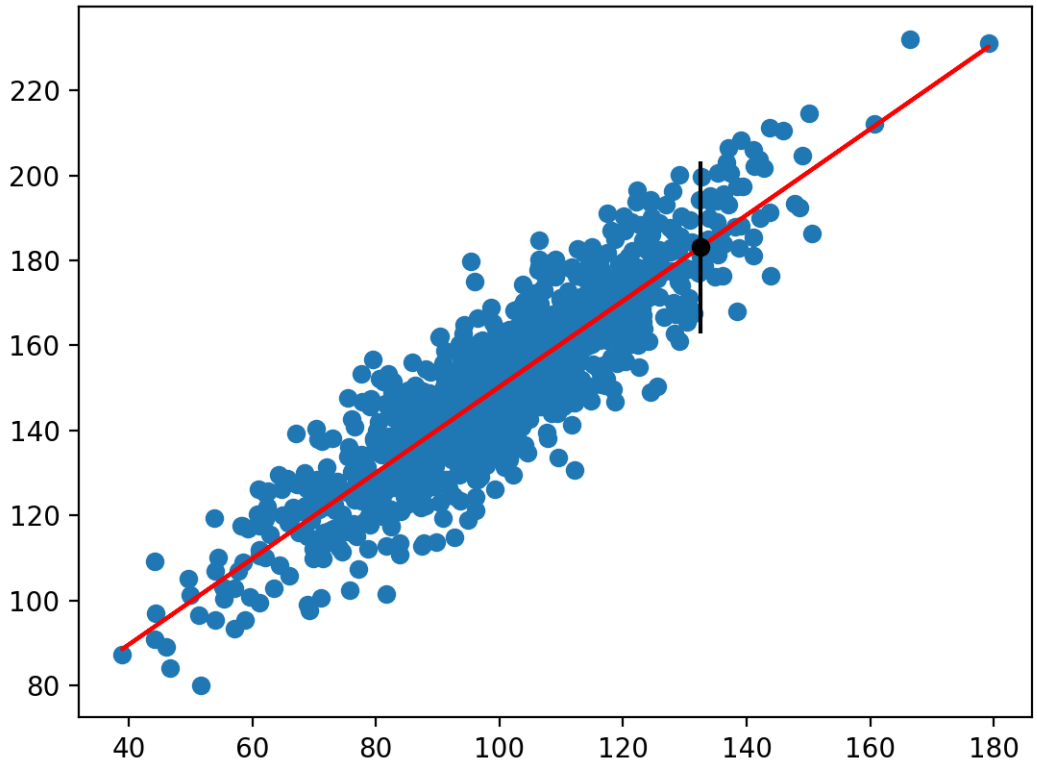

Figure 22.4: Scatter plot of dataset with linear model and prediction interval.

22.7 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Summarize the difference between tolerance, confidence, and prediction intervals.

- Develop a linear regression model for a standard machine learning dataset and calculate prediction intervals for a small test set.
- Describe in detail how one nonlinear prediction interval method works.

If you explore any of these extensions, I'd love to know.

22.8 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

22.8.1 Books

- *Understanding The New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis*, 2017.
  http://amzn.to/2oQW6No
- *Statistical Intervals: A Guide for Practitioners and Researchers*, 2017.
  http://amzn.to/2G8w3IL
- *An Introduction to Statistical Learning: with Applications in R*, 2013.
  http://amzn.to/2p1CKoB
- *Introduction to the New Statistics: Estimation, Open Science, and Beyond*, 2016.
  http://amzn.to/2FJlj5H
- *Forecasting: principles and practice*, 2013.
  http://amzn.to/2tFQ1Io

22.8.2 Papers

- A comparison of some error estimates for neural network models, 1995.
  https://pdfs.semanticscholar.org/31c8/17950a35c282c9da7ba7c7f927a6ff28a5af.pdf
- Machine learning approaches for estimation of prediction interval for the model output, 2006.
  https://www.sciencedirect.com/science/article/pii/S0893608006000153
- A Comprehensive Review of Neural Network-based Prediction Intervals and New Advances, 2010.
  http://alumnus.caltech.edu/~amir/pred-intv-2.pdf

22.8.3 API

- `scipy.stats.linregress` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html

- `matplotlib.pyplot.scatter` API.
  https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
- `matplotlib.pyplot.errorbar` API.
  https://matplotlib.org/api/_as_gen/matplotlib.pyplot.errorbar.html

22.8.4 Articles

- Prediction interval on Wikipedia.
  https://en.wikipedia.org/wiki/Prediction_interval
- Bootstrap prediction interval on Cross Validated.
  https://stats.stackexchange.com/questions/226565/bootstrap-prediction-interval

22.9 Summary

In this tutorial, you discovered the prediction interval and how to calculate it for a simple linear regression model. Specifically, you learned:

- That a prediction interval quantifies the uncertainty of a single point prediction.
- That prediction intervals can be estimated analytically for simple models but are more challenging for nonlinear machine learning models.
- How to calculate the prediction interval for a simple linear regression model.

22.9.1 Next

This is the end of part VI, in the next part you will discover nonparametric statistical methods.

Part VII

Nonparametric Methods

Chapter 23

Rank Data

A large portion of the field of statistics and statistical methods is dedicated to data where the distribution is known. Samples of data where we already know or can easily identify the distribution of are called parametric data. Often, parametric is used to refer to data that was drawn from a Gaussian distribution in common usage. Data in which the distribution is unknown or cannot be easily identified is called nonparametric.

In the case where you are working with nonparametric data, specialized nonparametric statistical methods can be used that discard all information about the distribution. As such, these methods are often referred to as distribution-free methods. In this tutorial, you will discover nonparametric statistics and their role in applied machine learning. After completing this tutorial, you will know:

- The difference between parametric and nonparametric data.
- How to rank data in order to discard all information about the data's distribution.
- Example of statistical methods that can be used for ranked data.

Let's get started.

23.1 Tutorial Overview

This tutorial is divided into 4 parts; they are:

1. Parametric Data
2. Nonparametric Data
3. Ranking Data
4. Working with Ranked Data

23.2 Parametric Data

Parametric data is a sample of data drawn from a known data distribution. This means that we already know the distribution or we have identified the distribution, and that we know the parameters of the distribution. Often, parametric is shorthand for real-valued data drawn from a Gaussian distribution. This is a useful shorthand, but strictly this is not entirely accurate. If we have parametric data, we can use parametric methods. Continuing with the shorthand of parametric meaning Gaussian. If we have parametric data, we can harness the entire suite of statistical methods developed for data assuming a Gaussian distribution, such as:

- Summary statistics.
- Correlation between variables.
- Significance tests for comparing means.

In general, we prefer to work with parametric data, and even go so far as to use data preparation methods that make data parametric, such as data transforms, so that we can harness these well-understood statistical methods.

23.3 Nonparametric Data

Data that does not fit a known or well-understood distribution is referred to as nonparametric data. Data could be nonparametric for many reasons, such as:

- Data is not real-valued, but instead is ordinal, intervals, or some other form.
- Data is real-valued but does not fit a well understood shape.
- Data is almost parametric but contains outliers, multiple peaks, a shift, or some other feature.

There are a suite of methods that we can use for nonparametric data called nonparametric statistical methods. In fact, most parametric methods have an equivalent nonparametric version. In general, the findings from nonparametric methods are less powerful than their parametric counterparts, namely because they must be generalized to work for all types of data. We can still use them for inference and make claims about findings and results, but they will not hold the same weight as similar claims with parametric methods. Information about the distribution is discarded.

In the case of ordinal or interval data, nonparametric statistics are the only type of statistics that can be used. For real-valued data, nonparametric statistical methods are required in applied machine learning when you are trying to make claims on data that does not fit the familiar Gaussian distribution.

23.4 Ranking Data

Before a nonparametric statistical method can be applied, the data must be converted into a rank format. As such, statistical methods that expect data in rank format are sometimes called rank statistics, such as rank correlation and rank statistical hypothesis tests. Ranking data is exactly as its name suggests. The procedure is as follows:

- Sort all data in the sample in ascending order.
- Assign an integer rank from 1 to N for each unique value in the data sample.

For example, imagine we have the following data sample, presented as a column:

```
0.550
0.184
0.431
0.020
0.620
```

Listing 23.1: Example of a small dataset.

We can sort it as follows:

```
0.020
0.184
0.431
0.550
0.620
```

Listing 23.2: Example of a sorted dataset.

Then assign a rank to each value, starting at 1:

```
1 = 0.021055
2 = 0.404622
3 = 0.488733
4 = 0.618510
5 = 0.832803
```

Listing 23.3: Example of a ranked sorted dataset.

We can then apply this procedure to another data sample and start using nonparametric statistical methods. There are variations on this procedure for special circumstances such as handling ties, using a reverse ranking, and using a fractional rank score, but the general properties hold. The SciPy library provides the `rankdata()` function to rank numerical data, which supports a number of variations on ranking. The example below demonstrates how to rank a numerical dataset.

```
# example of ranking real-valued observations
from numpy.random import rand
from numpy.random import seed
from scipy.stats import rankdata
# seed random number generator
seed(1)
# generate dataset
data = rand(1000)
```

```
# review first 10 samples
print(data[:10])
# rank data
ranked = rankdata(data)
# review first 10 ranked samples
print(ranked[:10])
```

Listing 23.4: Example of ranking a sample dataset.

Running the example first generates a sample of 1,000 random numbers from a uniform distribution, then ranks the data sample and prints the result.

```
[4.17022005e-01 7.20324493e-01 1.14374817e-04 3.02332573e-01
 1.46755891e-01 9.23385948e-02 1.86260211e-01 3.45560727e-01
 3.96767474e-01 5.38816734e-01]
[408. 721.  1. 300. 151. 93. 186. 342. 385. 535.]
```

Listing 23.5: Example output from ranking a sample dataset.

23.5 Working with Ranked Data

There are statistical tools that you can use to check if your sample data fits a given distribution. For example, if we take nonparametric data as data that does not look Gaussian, then you can use statistical methods that quantify how Gaussian a sample of data is and use nonparametric methods if the data fails those tests.

Three examples of statistical methods for normality testing, as it is called, are:

- Shapiro-Wilk Test.
- D'Agostino's K^2 Test.
- Anderson-Darling Test.

Once you have decided to use nonparametric statistics, you must then rank your data. In fact, most of the tools that you use for inference will perform the ranking of the sample data automatically. Nevertheless, it is important to understand how your sample data is being transformed prior to performing the tests. In applied machine learning, there are two main types of questions that you may have about your data that you can address with nonparametric statistical methods.

23.5.1 Relationship Between Variables

Methods for quantifying the dependency between variables are called correlation methods. Two nonparametric statistical correlation methods that you can use are:

- Spearman's Rank Correlation.
- Kendall's Rank Correlation.
- Goodman and Kruskal's Rank Correlation.
- Somers' Rank Correlation.

23.5.2 Compare Sample Means

Methods for quantifying whether the mean between two populations is significantly different are called statistical significance tests. Three nonparametric statistical significance tests that you can use are:

- Mann-Whitney U Test.
- Wilcoxon Signed-Rank Test.
- Kruskal-Wallis H Test.
- Friedman Test.

23.6 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- List three examples of when you think you might need to use nonparametric statistical methods in an applied machine learning project.
- Develop your own example to demonstrate the capabilities of the `rankdata()` function.
- Write your own function to rank a provided univariate dataset.

If you explore any of these extensions, I'd love to know.

23.7 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

23.7.1 Books

- *All of Nonparametric Statistics*, 2007.
  http://amzn.to/2oGv2A6
- *Practical Nonparametric Statistics*, 1999.
  http://amzn.to/2CXUe9y
- *Applied Nonparametric Statistics*, 2000.
  http://amzn.to/2t9iMN6

23.7.2 API

- `scipy.stats.rankdata` API.
  http://scipy.github.io/devdocs/generated/scipy.stats.rankdata.html

23.7.3 Articles

- Parametric statistics on Wikipedia.
  https://en.wikipedia.org/wiki/Parametric_statistics
- Nonparametric statistics on Wikipedia.
  https://en.wikipedia.org/wiki/Nonparametric_statistics
- Ranking on Wikipedia.
  https://en.wikipedia.org/wiki/Ranking
- Rank correlation on Wikipedia.
  https://en.wikipedia.org/wiki/Rank_correlation
- Normality test on Wikipedia.
  https://en.wikipedia.org/wiki/Normality_test

23.8 Summary

In this tutorial, you discovered nonparametric statistics and their role in applied machine learning. Specifically, you learned:

- The difference between parametric and nonparametric data.
- How to rank data in order to discard all information about the data's distribution.
- Example of statistical methods that can be used for ranked data.

23.8.1 Next

In the next section, you will discover statistical methods for determining whether a sample of data was drawn from a Gaussian distribution or not.

Chapter 24

Normality Tests

An important decision point when working with a sample of data is whether to use parametric or nonparametric statistical methods. Parametric statistical methods assume that the data has a known and specific distribution, often a Gaussian distribution. If a data sample is not Gaussian, then the assumptions of parametric statistical tests are violated and nonparametric statistical methods must be used.

There are a range of techniques that you can use to check if your data sample deviates from a Gaussian distribution, called normality tests. In this tutorial, you will discover the importance of checking whether a data sample deviates from the normal distribution and a suite of techniques that you can use to evaluate your data sample. After completing this tutorial, you will know:

- How whether a sample is normal dictates the types of statistical methods to use with a data sample.
- Graphical methods for qualifying deviations from normal, such as histograms and the Q-Q plot.
- Statistical normality tests for quantifying deviations from normal.

Let's get started.

24.1 Tutorial Overview

This tutorial is divided into 5 parts; they are:

1. Normality Assumption
2. Test Dataset
3. Visual Normality Checks
4. Statistical Normality Tests
5. What Test Should You Use?

24.2 Normality Assumption

A large fraction of the field of statistics is concerned with data that assumes that it was drawn from a Gaussian distribution. If methods are used that assume a Gaussian distribution, and your data was drawn from a different distribution, the findings may be misleading or plain wrong.

There are a number of techniques that you can check if your data sample is Gaussian or sufficiently Gaussian-like to use the standard techniques, or sufficiently non-Gaussian to instead use nonparametric statistical methods. This is a key decision point when it comes to choosing statistical methods for your data sample. We can summarize this decision as follows:

```
If Data Is Gaussian:
  Use Parametric Statistical Methods
Else:
  Use Nonparametric Statistical Methods
```

Listing 24.1: Pseudocode for the choice of statistical hypothesis tests.

There is also some middle ground where we can assume that the data is Gaussian-enough to use parametric methods or that we can use data preparation techniques to transform the data to be sufficiently Gaussian to use the parametric methods. There are three main areas where you may need to make this evaluation of a data sample in a machine learning project; they are:

- Input data to the model in the case of fitting models.
- Model evaluation results in the case of model selection.
- Residual errors from model predictions in the case of regression.

In this tutorial, we will look at two classes of techniques for checking whether a sample of data is Gaussian:

- **Graphical Methods**. These are methods for plotting the data and qualitatively evaluating whether the data looks Gaussian.
- **Statistical Tests**. These are methods that calculate statistics on the data and quantify how likely it is that the data was drawn from a Gaussian distribution.

Methods of this type are often called normality tests.

24.3 Test Dataset

Before we start looking at normality tests, let's first develop a test dataset that we can use throughout this tutorial. We will generate a small sample of random numbers drawn from a Gaussian distribution. The choice of Gaussian random numbers for the test dataset means that we do expect each test to correctly identify the distribution, nevertheless, the small-ish sample size may introduce some noise into the results.

We will use the `randn()` NumPy function to generate random Gaussian numbers with a mean of 0 and a standard deviation of 1, so-called standard, normal variables. We will then shift them to have a mean of 50 and a standard deviation of 5. The complete example is listed below.

```
# generate gaussian data
from numpy.random import seed
from numpy.random import randn
from numpy import mean
from numpy import std
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(100) + 50
# summarize
print('mean=%.3f stdv=%.3f' % (mean(data), std(data)))
```

Listing 24.2: Example of a small sample of Gaussian random numbers.

Running the example generates the sample and prints the mean and standard deviation of the sample. We can see that the mean and standard deviation are reasonable but rough estimations of the true underlying population mean and standard deviation, given the small-ish sample size.

```
mean=50.303 stdv=4.426
```

Listing 24.3: Example output from summarizing a small sample of Gaussian random numbers.

24.4 Visual Normality Checks

We can create plots of the data to check whether it is Gaussian. These checks are qualitative, so less accurate than the statistical methods we will calculate in the next section. Nevertheless, they are fast and like the statistical tests, must still be interpreted before you can make a call about your data sample. In this section, we will look at two common methods for visually inspecting a dataset to check if it was drawn from a Gaussian distribution.

24.4.1 Histogram Plot

A simple and commonly used plot to quickly check the distribution of a sample of data is the histogram. In the histogram, the data is divided into a pre-specified number of groups called bins. The data is then sorted into each bin and the count of the number of observations in each bin is retained. The plot shows the bins across the x-axis maintaining their ordinal relationship, and the count in each bin on the y-axis.

A sample of data has a Gaussian distribution of the histogram plot, showing the familiar bell shape. A histogram can be created using the `hist()` Matplotlib function. By default, the number of bins is automatically estimated from the data sample. A complete example demonstrating the histogram plot on the test problem is listed below.

```
# histogram plot
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(100) + 50
```

```
# histogram plot
pyplot.hist(data)
pyplot.show()
```

Listing 24.4: Example of creating a histogram of a data sample.

Running the example creates a histogram plot showing the number of observations in each bin. We can see a Gaussian-like shape to the data, that although is not strongly the familiar bell-shape, is a rough approximation.

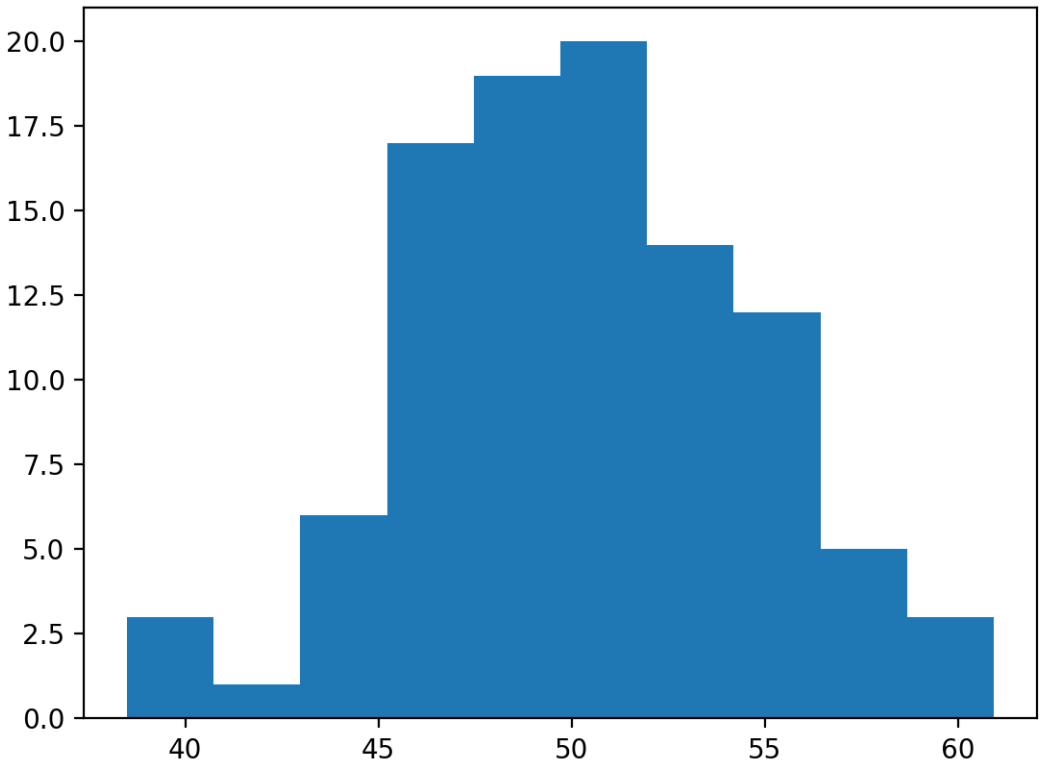

Figure 24.1: Histogram plot as a visual normality check.

24.4.2 Quantile-Quantile Plot

Another popular plot for checking the distribution of a data sample is the quantile-quantile plot, Q-Q plot, or QQ plot for short. This plot generates its own sample of the idealized distribution that we are comparing with, in this case the Gaussian distribution. The idealized samples are divided into groups (e.g. 5), called quantiles. Each data point in the sample is paired with a similar member from the idealized distribution at the same cumulative distribution. The resulting points are plotted as a scatter plot with the idealized value on the x-axis and the data sample on the y-axis.

A perfect match for the distribution will be shown by a line of dots on a 45-degree angle from the bottom left of the plot to the top right. Often a line is drawn on the plot to help make

this expectation clear. Deviations by the dots from the line shows a deviation from the expected distribution. We can develop a QQ plot in Python using the `qqplot()` Statsmodels function. The function takes the data sample and by default assumes we are comparing it to a Gaussian distribution. We can draw the standardized line by setting the `line` argument to the value 's'. A complete example of plotting the test dataset as a QQ plot is provided below.

```
# q-q plot
from numpy.random import seed
from numpy.random import randn
from statsmodels.graphics.gofplots import qqplot
from matplotlib import pyplot
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(100) + 50
# q-q plot
qqplot(data, line='s')
pyplot.show()
```

Listing 24.5: Example of creating a QQ plot of a data sample.

Running the example creates the QQ plot showing the scatter plot of points in a diagonal line, closely fitting the expected diagonal pattern for a sample from a Gaussian distribution. There are a few small deviations, especially at the bottom of the plot, which is to be expected given the small data sample.

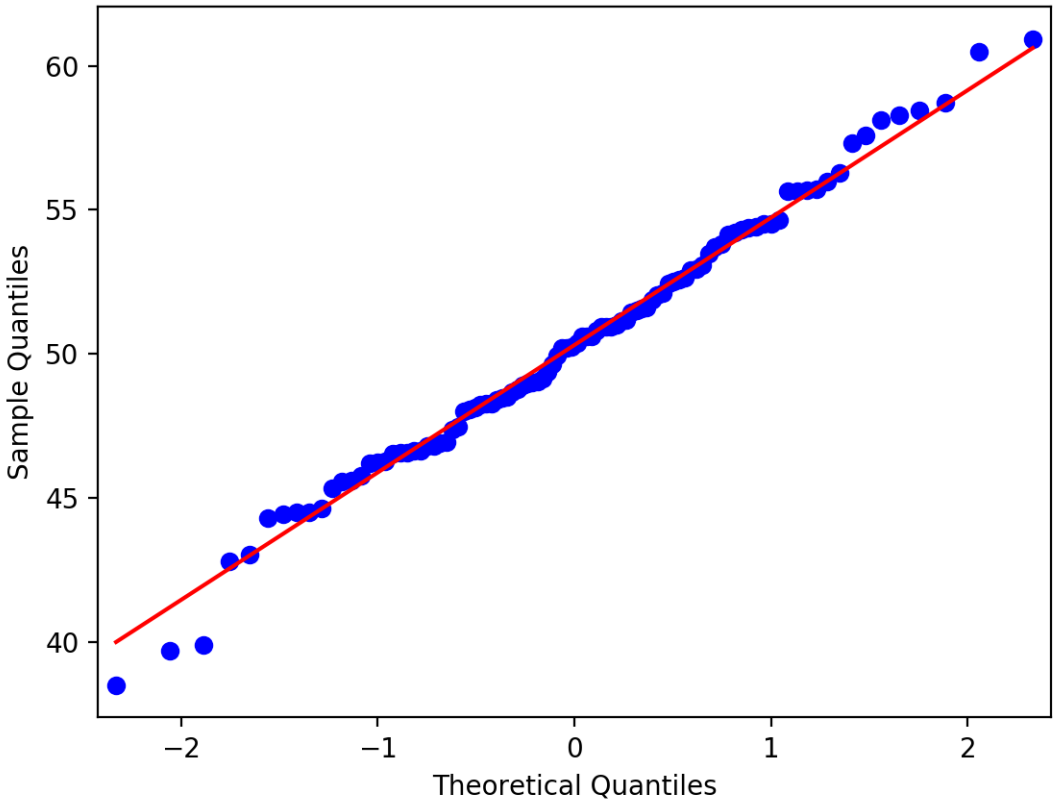

Figure 24.2: QQ plot as a visual normality check.

24.5 Statistical Normality Tests

There are many statistical tests that we can use to quantify whether a sample of data looks as though it was drawn from a Gaussian distribution. Each test makes different assumptions and considers different aspects of the data. We will look at 3 commonly used tests in this section that you can apply to your own data samples.

24.5.1 Interpretation of a Test

Before you can apply the statistical tests, you must know how to interpret the results. Each test will return at least two things:

- **Statistic**: A quantity calculated by the test that can be interpreted in the context of the test via comparing it to critical values from the distribution of the test statistic.
- **p-value**: Used to interpret the test, in this case whether the sample was drawn from a Gaussian distribution.

Each test calculates a test-specific statistic. This statistic can aid in the interpretation of the result, although it may require a deeper proficiency with statistics and a deeper knowledge of

the specific statistical test. Instead, the p-value can be used to quickly and accurately interpret the statistic in practical applications. The tests assume that the sample was drawn from a Gaussian distribution. Technically this is called the null hypothesis, or H0. A threshold level is chosen called alpha, typically 5% (or 0.05), that is used to interpret the p-value. In the SciPy implementation of these tests, you can interpret the p value as follows.

- `p-value` $\leq$ `alpha`: significant result, reject null hypothesis, not Gaussian (H1).
- `p-value` $>$ `alpha`: not significant result, fail to reject null hypothesis, Gaussian (H0).

This means that, in general, we are seeking results with a larger p-value to confirm that our sample was likely drawn from a Gaussian distribution. A result above 5% does not mean that the null hypothesis is true. It means that it is very likely true given available evidence. The p-value is not the probability of the data fitting a Gaussian distribution; it can be thought of as a value that helps us interpret the statistical test.

24.5.2 Shapiro-Wilk Test

The Shapiro-Wilk test evaluates a data sample and quantifies how likely it is that the data was drawn from a Gaussian distribution, named for Samuel Shapiro and Martin Wilk. In practice, the Shapiro-Wilk test is believed to be a reliable test of normality, although there is some suggestion that the test may be suitable for smaller samples of data, e.g. thousands of observations or fewer. The `shapiro()` SciPy function will calculate the Shapiro-Wilk on a given dataset. The function returns both the W-statistic calculated by the test and the p-value. The complete example of performing the Shapiro-Wilk test on the dataset is listed below.

```
# example of the shapiro-wilk test
from numpy.random import seed
from numpy.random import randn
from scipy.stats import shapiro
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(100) + 50
# normality test
stat, p = shapiro(data)
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
  print('Sample looks Gaussian (fail to reject H0)')
else:
  print('Sample does not look Gaussian (reject H0)')
```

Listing 24.6: Example calculating the Shapiro-Wilk test on the test dataset.

Running the example first calculates the test on the data sample, then prints the statistic and calculated p-value. The p-value is interpreted and finds that the data is likely drawn from a Gaussian distribution.

```
Statistics=0.992, p=0.822
Sample looks Gaussian (fail to reject H0)
```

Listing 24.7: Example output from calculating the Shapiro-Wilk test on the test dataset.

24.5.3 D'Agostino's K^2 Test

The D'Agostino's K^2 test calculates summary statistics from the data, namely kurtosis and skewness, to determine if the data distribution departs from the normal distribution, named for Ralph D'Agostino. Skew is a quantification of how much a distribution is pushed left or right, a measure of asymmetry in the distribution. Kurtosis quantifies how much of the distribution is in the tail. It is a simple and commonly used statistical test for normality. The D'Agostino's K^2 test is available via the `normaltest()` SciPy function and returns the test statistic and the p-value. The complete example of the D'Agostino's K^2 test on the dataset is listed below.

```
# example of the d'agostino and pearson's test
from numpy.random import seed
from numpy.random import randn
from scipy.stats import normaltest
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(100) + 50
# normality test
stat, p = normaltest(data)
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
  print('Sample looks Gaussian (fail to reject H0)')
else:
  print('Sample does not look Gaussian (reject H0)')
```

Listing 24.8: Example calculating D'Agostino's K^2 test on the test dataset.

Running the example calculates the statistic and prints the statistic and p-value. The p-value is interpreted against an alpha of 5% and finds that the test dataset does not significantly deviate from normal.

```
Statistics=0.102, p=0.950
Sample looks Gaussian (fail to reject H0)
```

Listing 24.9: Example output from calculating D'Agostino's K^2 test on the test dataset.

24.5.4 Anderson-Darling Test

Anderson-Darling Test is a statistical test that can be used to evaluate whether a data sample comes from one of among many known data samples, named for Theodore Anderson and Donald Darling. It can be used to check whether a data sample is normal. The test is a modified version of a more sophisticated nonparametric goodness-of-fit statistical test called the Kolmogorov-Smirnov test. A feature of the Anderson-Darling test is that it returns a list of critical values rather than a single p-value. This can provide the basis for a more thorough interpretation of the result.

The `anderson()` SciPy function implements the Anderson-Darling test. It takes as parameters the data sample and the name of the distribution to test it against. By default, the test will check against the Gaussian distribution (dist='norm'). The complete example of calculating the Anderson-Darling test on the sample problem is listed below.

```
# example of the anderson-darling test
from numpy.random import seed
from numpy.random import randn
from scipy.stats import anderson
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(100) + 50
# normality test
result = anderson(data)
print('Statistic: %.3f' % result.statistic)
p = 0
for i in range(len(result.critical_values)):
  sl, cv = result.significance_level[i], result.critical_values[i]
  if result.statistic < result.critical_values[i]:
    print('%.3f: %.3f, data looks normal (fail to reject H0)' % (sl, cv))
  else:
    print('%.3f: %.3f, data does not look normal (reject H0)' % (sl, cv))
```

Listing 24.10: Example calculating the Anderson-Darling test on the test dataset.

Running the example calculates the statistic on the test data set and prints the critical values. Critical values in a statistical test are a range of pre-defined significance boundaries at which the H0 is failed to be rejected if the calculated statistic is less than the critical value. Rather than just a single p-value, the test returns a critical value for a range of different commonly used significance levels. We can interpret the results by failing to reject the null hypothesis that the data is normal if the calculated test statistic is less than the critical value at a chosen significance level. We can see that at each significance level, the test has found that the data follows a normal distribution

```
Statistic: 0.220
15.000: 0.555, data looks normal (fail to reject H0)
10.000: 0.632, data looks normal (fail to reject H0)
5.000: 0.759, data looks normal (fail to reject H0)
2.500: 0.885, data looks normal (fail to reject H0)
1.000: 1.053, data looks normal (fail to reject H0)
```

Listing 24.11: Example output from calculating the Anderson-Darling test on the test dataset.

24.6 What Test Should You Use?

We have covered a few normality tests, but this is not all of the tests that exist. So which test do you use? I recommend using them all on your data, where appropriate. The question then becomes, how do you interpret the results? What if the tests disagree, which they often will? I have two suggestions for you to help think about this question.

24.6.1 Hard Fail

Your data may not be normal for lots of different reasons. Each test looks at the question of whether a sample was drawn from a Gaussian distribution from a slightly different perspective. A failure of one normality test means that your data is not normal. As simple as that. You can

either investigate why your data is not normal and perhaps use data preparation techniques to make the data more normal. Or you can start looking into the use of nonparametric statistical methods instead of the parametric methods.

24.6.2 Soft Fail

If some of the methods suggest that the sample is Gaussian and some not, then perhaps take this as an indication that your data is Gaussian-like. In many situations, you can treat your data as though it is Gaussian and proceed with your chosen parametric statistical methods.

24.7 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- List two additional examples of when you think a normality test might be useful in a machine learning project.
- Develop your own contrived dataset and apply each normality test.
- Load a standard machine learning dataset and apply normality tests to each real-valued variable.

If you explore any of these extensions, I'd love to know.

24.8 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

24.8.1 API

- `scipy.stats.normaltest` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.normaltest.html
- `scipy.stats.shapiro` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.shapiro.html
- `scipy.stats.anderson` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.anderson.html
- `statsmodels.graphics.gofplots.qqplot` API.
  http://www.statsmodels.org/dev/generated/statsmodels.graphics.gofplots.qqplot.html
- `matplotlib.pyplot.hist` API.
  https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html

24.8.2 Articles

- Normality test on Wikipedia.
  https://en.wikipedia.org/wiki/Normality_test
- Histogram on Wikipedia.
  https://en.wikipedia.org/wiki/Histogram
- Q-Q plot on Wikipedia.
  https://en.wikipedia.org/wiki/Q%E2%80%93Q_plot
- D'Agostino's K-squared test on Wikipedia.
  https://en.wikipedia.org/wiki/D%27Agostino%27s_K-squared_test
- Anderson-Darling test on Wikipedia.
  https://en.wikipedia.org/wiki/Anderson%E2%80%93Darling_test
- Shapiro-Wilk test on Wikipedia.
  https://en.wikipedia.org/wiki/Shapiro%E2%80%93Wilk_test

24.9 Summary

In this tutorial, you discovered the importance of checking whether a data sample deviates from the normal distribution and a suite of techniques that you can use to evaluate your data sample. Specifically, you learned:

- How whether a sample is normal dictates the types of statistical methods to use with a data sample.
- Graphical methods for qualifying deviations from normal such as histograms and the Q-Q plot.
- Statistical normality tests for quantifying deviations from normal.

24.9.1 Next

In the next section, you will discover statistical methods for making nearly-Gaussian data Gaussian.

Chapter 25

Make Data Normal

A large portion of the field of statistics is concerned with methods that assume a Gaussian distribution: the familiar bell curve. If your data has a Gaussian distribution, the parametric methods are powerful and well understood. This gives some incentive to use them if possible. Even if your data does not have a Gaussian distribution.

It is possible that your data does not look Gaussian or fails a normality test, but can be transformed to make it fit a Gaussian distribution. This is more likely if you are familiar with the process that generated the observations and you believe it to be a Gaussian process, or the distribution looks almost Gaussian, except for some distortion. In this tutorial, you will discover the reasons why a Gaussian-like distribution may be distorted and techniques that you can use to make a data sample more normal. After completing this tutorial, you will know:

- How to consider the size of the sample and whether the law of large numbers may help improve the distribution of a sample.
- How to identify and remove extreme values and long tails from a distribution.
- Power transforms and the Box-Cox transform that can be used to control for quadratic or exponential distributions.

Let's get started.

25.1 Tutorial Overview

This tutorial is divided into 7 parts; they are:

1. Gaussian and Gaussian-Like
2. Sample Size
3. Data Resolution
4. Extreme Values
5. Long Tails
6. Power Transforms
7. Use Anyway

25.2 Gaussian and Gaussian-Like

There may be occasions when you are working with a non-Gaussian distribution, but wish to use parametric statistical methods instead of nonparametric methods. For example, you may have a data sample that has the familiar bell-shape, meaning that it looks Gaussian, but it fails one or more statistical normality tests. This suggests that the data may be Gaussian-like. You would prefer to use parametric statistics in this situation given that better statistical power and because the data is clearly Gaussian, or could be, after the right data transform.

There are many reasons why the dataset may not be technically Gaussian. In this tutorial, we will look at some simple techniques that you may be able to use to transform a data sample with a Gaussian-like distribution into a Gaussian distribution. There is no silver bullet for this process; some experimentation and judgment may be required.

25.3 Sample Size

One common reason that a data sample is non-Gaussian is because the size of the data sample is too small. Many statistical methods were developed where data was scarce. Hence, the minimum. number of samples for many methods may be as low as 20 or 30 observations. Nevertheless, given the noise in your data, you may not see the familiar bell-shape or fail normality tests with a modest number of samples, such as 50 or 100. If this is the case, perhaps you can collect more data. Thanks to the law of large numbers, the more data that you collect, the more likely your data will be able to used to describe the underlying population distribution.

To make this concrete, below is an example of a plot of a small sample of 10 observations drawn from a Gaussian distribution with a mean of 100 and a standard deviation of 50.

```
# histogram plot of a small sample
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# generate a univariate data sample
data = 50 * randn(10) + 100
# histogram
pyplot.hist(data)
pyplot.show()
```

Listing 25.1: Example of plotting a small sample of Gaussian random numbers.

Running the example creates a histogram plot of the data showing no clear Gaussian distribution, not even Gaussian-like.

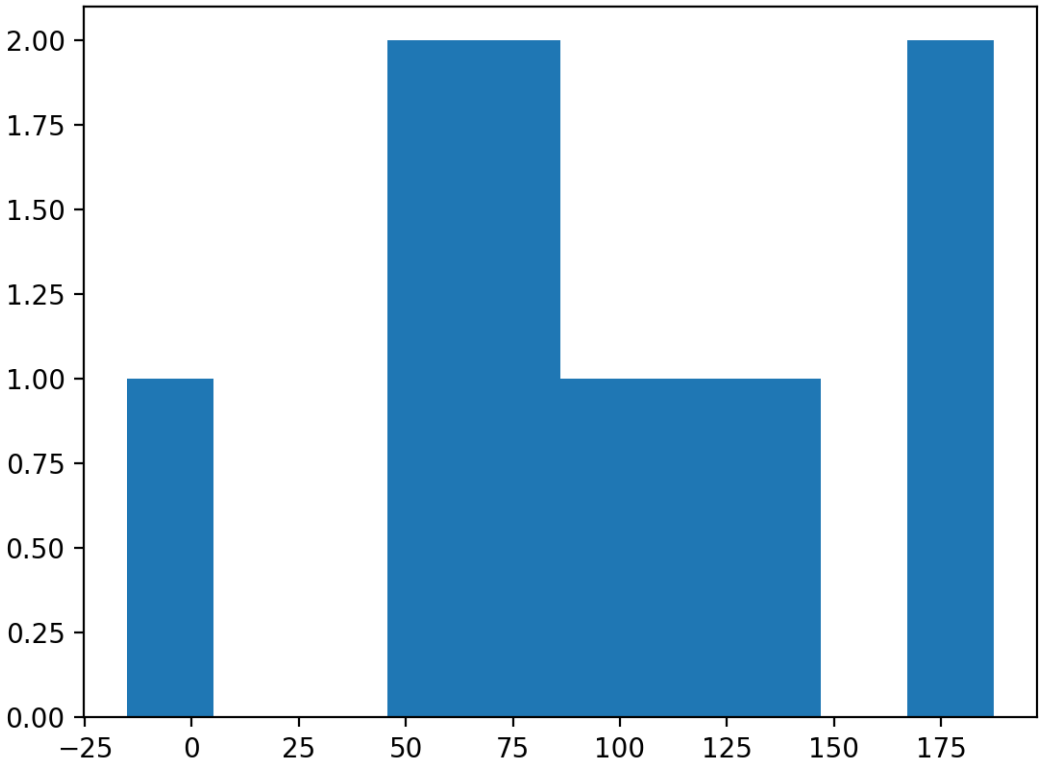

Figure 25.1: Histogram plot as small sample size.

Increasing the size of the sample from 10 to 100 can help to better expose the Gaussian shape of the data distribution.

```
# histogram plot of a small sample
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# generate a univariate data sample
data = 50 * randn(100) + 100
# histogram
pyplot.hist(data)
pyplot.show()
```

Listing 25.2: Example of plotting a large sample of Gaussian random numbers.

Running the example, we can better see the Gaussian distribution of the data that would pass both statistical tests and eye-ball checks.

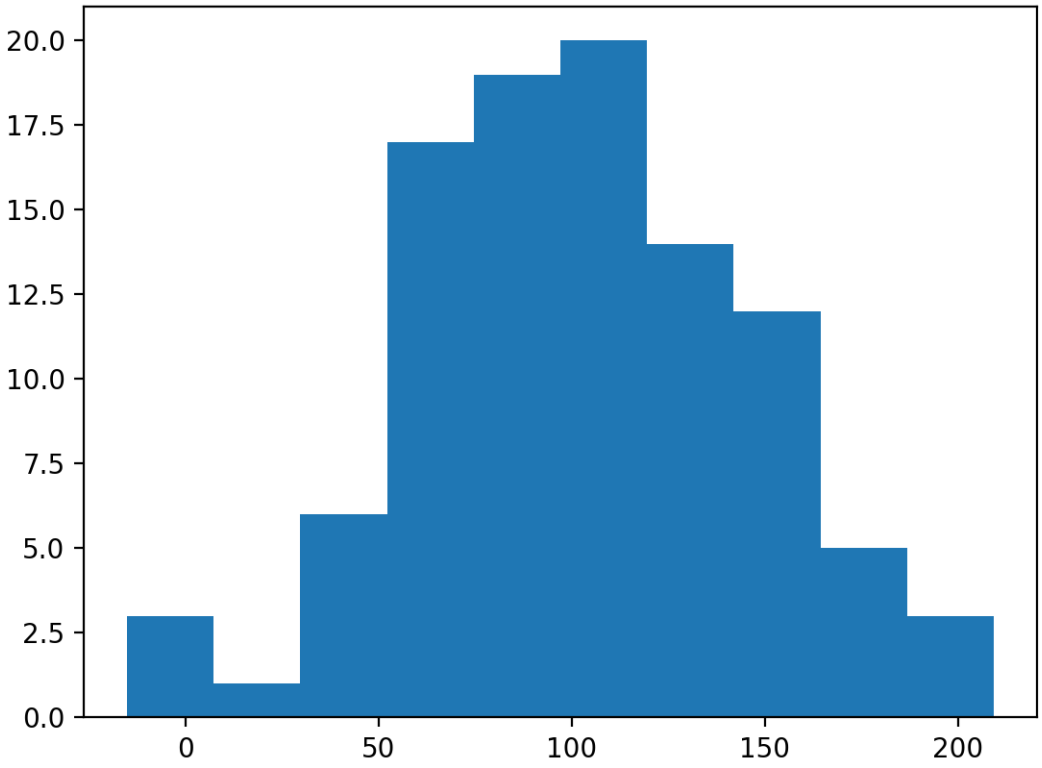

Figure 25.2: Histogram plot as large sample size.

25.4 Data Resolution

Perhaps you expect a Gaussian distribution from the data, but no matter the size of the sample that you collect, it does not materialize. A common reason for this is the resolution that you are using to collect the observations. The distribution of the data may be obscured by the chosen resolution of the data or the fidelity of the observations. There may be many reasons why the resolution of the data is being modified prior to modeling, such as:

- The configuration of the mechanism making the observation.
- The data is passing through a quality-control process.
- The resolution of the database used to store the data.

To make this concrete, we can make a sample of 100 random Gaussian numbers with a mean of 0 and a standard deviation of 1 and remove all of the decimal places.

```
# histogram plot of a low res sample
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
```

```
# seed the random number generator
seed(1)
# generate a univariate data sample
data = randn(100)
# remove decimal component
data = data.round(0)
# histogram
pyplot.hist(data)
pyplot.show()
```

Listing 25.3: Example of plotting a sample of Gaussian random numbers with low resolution.

Running the example results in a distribution that appears discrete although Gaussian-like. Adding the resolution back to the observations would result in a fuller distribution of the data.

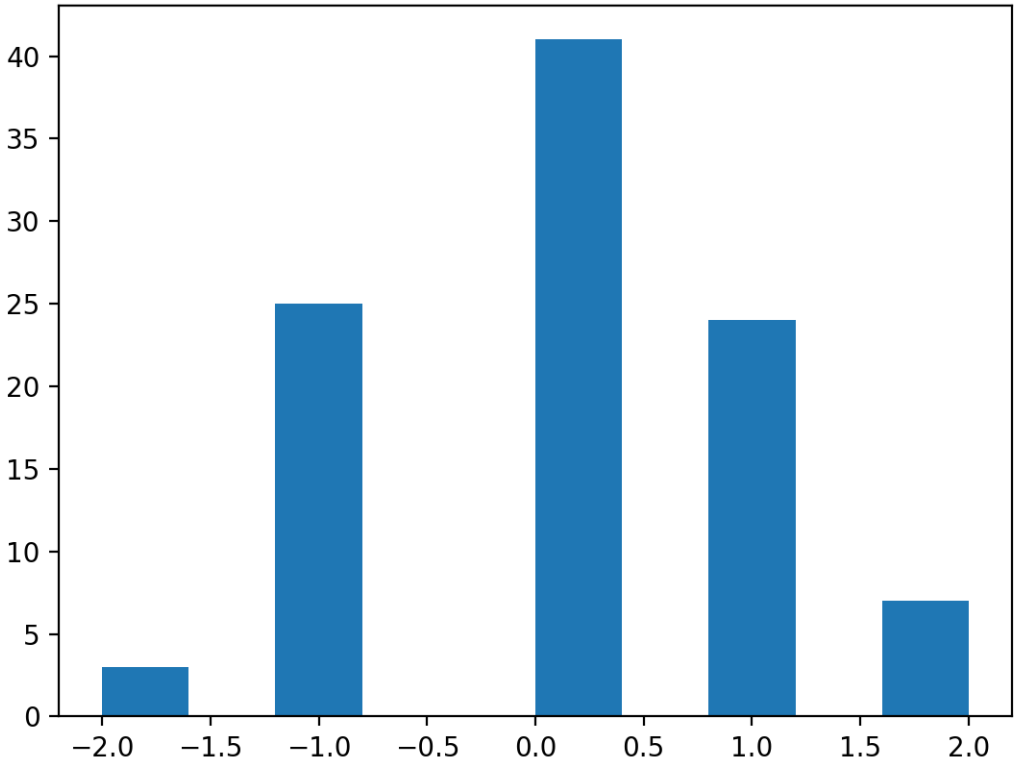

Figure 25.3: Histogram plot of low resolution data sample.

25.5 Extreme Values

A data sample may have a Gaussian distribution, but may be distorted for a number of reasons. A common reason is the presence of extreme values at the edge of the distribution. Extreme values could be present for a number of reasons, such as:

- Measurement error.

- Missing data.
- Data corruption.
- Rare events.

In such cases, the extreme values could be identified and removed in order to make the distribution more Gaussian. These extreme values are often called outliers. This may require domain expertise or consultation with a domain expert in order to both design the criteria for identifying outliers and then removing them from the data sample and all data samples that you or your model expect to work with in the future. We can demonstrate how easy it is to have extreme values disrupt the distribution of data.

The example below creates a data sample with 100 random Gaussian numbers scaled to have a mean of 10 and a standard deviation of 5. An additional 10 zero-valued observations are then added to the distribution. This can happen if missing or corrupt values are assigned the value of zero. This is a common behavior in publicly available machine learning datasets; for example.

```
# histogram plot of data with outliers
from numpy.random import seed
from numpy.random import randn
from numpy import zeros
from numpy import append
from matplotlib import pyplot
# seed the random number generator
seed(1)
# generate a univariate data sample
data = 5 * randn(100) + 10
# add extreme values
data = append(data, zeros(10))
# histogram
pyplot.hist(data)
pyplot.show()
```

Listing 25.4: Example of plotting a sample of Gaussian random numbers with many rare events.

Running the example creates and plots the data sample. You can clearly see how the unexpected high frequency of zero-valued observations disrupts the distribution.

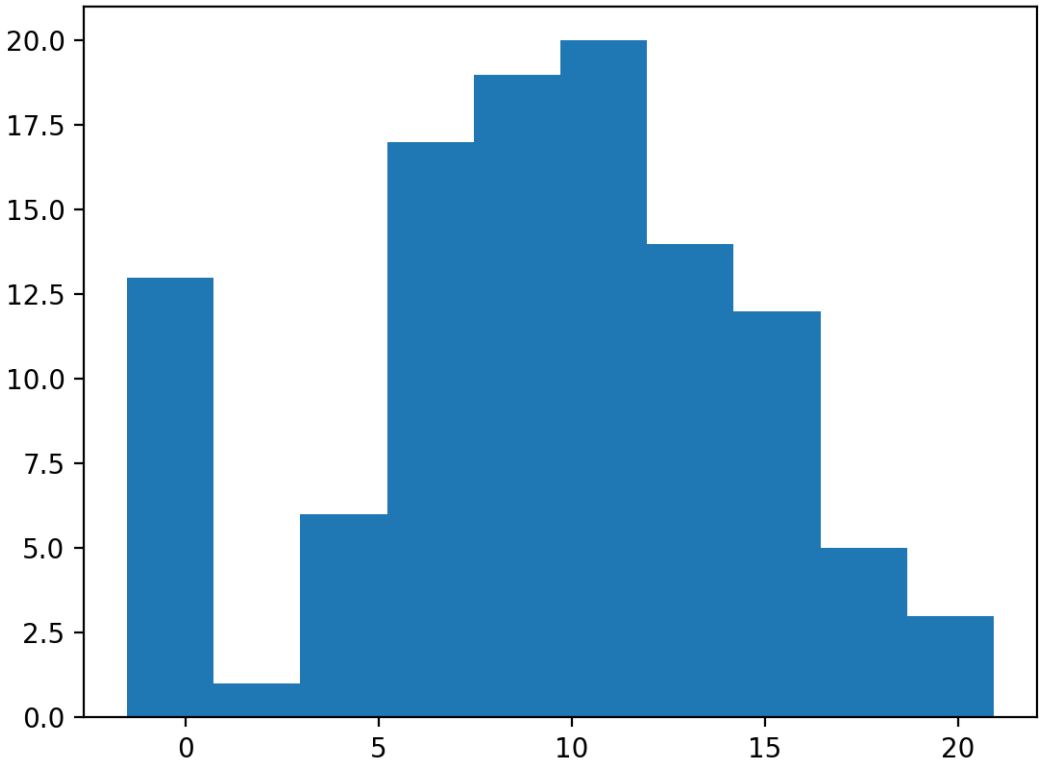

Figure 25.4: Histogram plot of data sample with many rare events.

25.6 Long Tails

Extreme values can manifest in many ways. In addition to an abundance of rare events at the edge of the distribution, you may see a long tail on the distribution in one or both directions. In plots, this can make the distribution look like it is exponential, when in fact it might be Gaussian with an abundance of rare events in one direction. You could use simple threshold values, perhaps based on the number of standard deviations from the mean, to identify and remove long tail values.

We can demonstrate this with a contrived example. The data sample contains 100 Gaussian random numbers with a mean of 10 and a standard deviation of 5. An additional 50 uniformly random values in the range 10-to-110 are added. This creates a long tail on the distribution.

```
# histogram plot of data with a long tail
from numpy.random import seed
from numpy.random import randn
from numpy.random import rand
from numpy import append
from matplotlib import pyplot
# seed the random number generator
seed(1)
# generate a univariate data sample
```

```
data = 5 * randn(100) + 10
tail = 10 + (rand(50) * 100)
# add long tail
data = append(data, tail)
# histogram
pyplot.hist(data)
pyplot.show()
```

Listing 25.5: Example of plotting a sample of Gaussian random numbers with a long tail.

Running the example you can see how the long tail distorts the Gaussian distribution and makes it look almost exponential or perhaps even bimodal (two bumps).

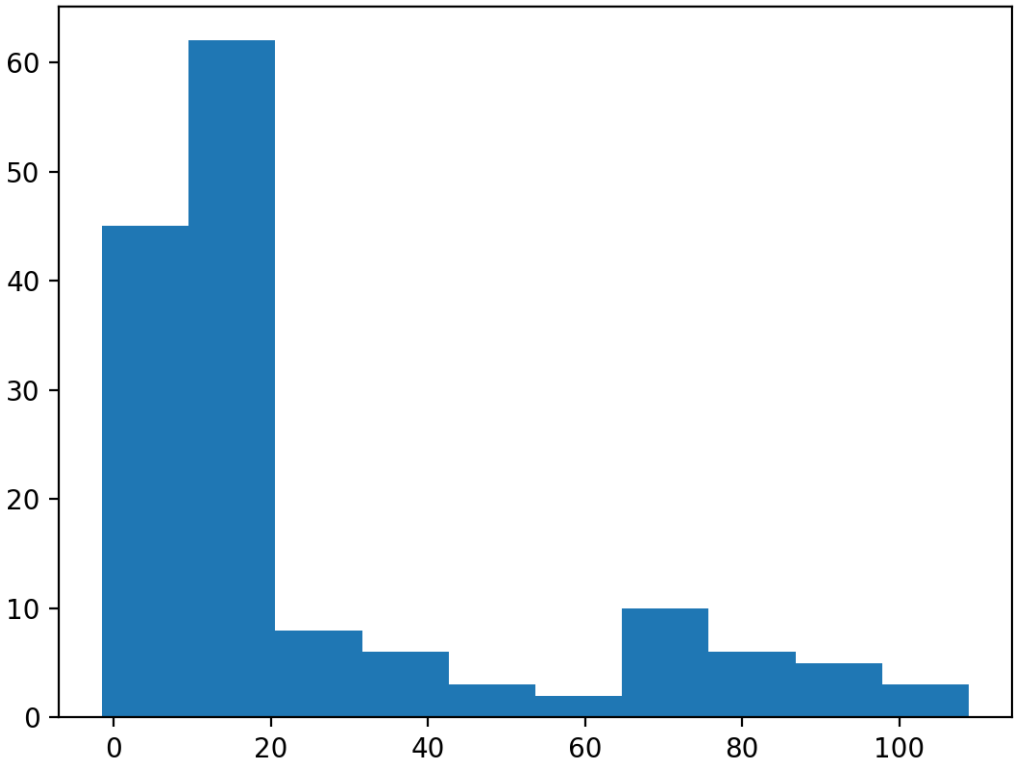

Figure 25.5: Histogram plot of data sample with a long tail.

We can use a simple threshold, such as a value of 25, on this dataset as a cutoff and remove all observations higher than this threshold. We did choose this threshold with prior knowledge of how the data sample was contrived, but you can imagine testing different thresholds on your own dataset and evaluating their effect.

```
# histogram plot of data with a long tail
from numpy.random import seed
from numpy.random import randn
from numpy.random import rand
from numpy import append
```

```
from matplotlib import pyplot
# seed the random number generator
seed(1)
# generate a univariate data sample
data = 5 * randn(100) + 10
tail = 10 + (rand(10) * 100)
# add long tail
data = append(data, tail)
# trim values
data = [x for x in data if x < 25]
# histogram
pyplot.hist(data)
pyplot.show()
```

Listing 25.6: Example of plotting a sample of Gaussian random numbers with a truncated long tail.

Running the code shows how this simple trimming of the long tail returns the data to a Gaussian distribution.

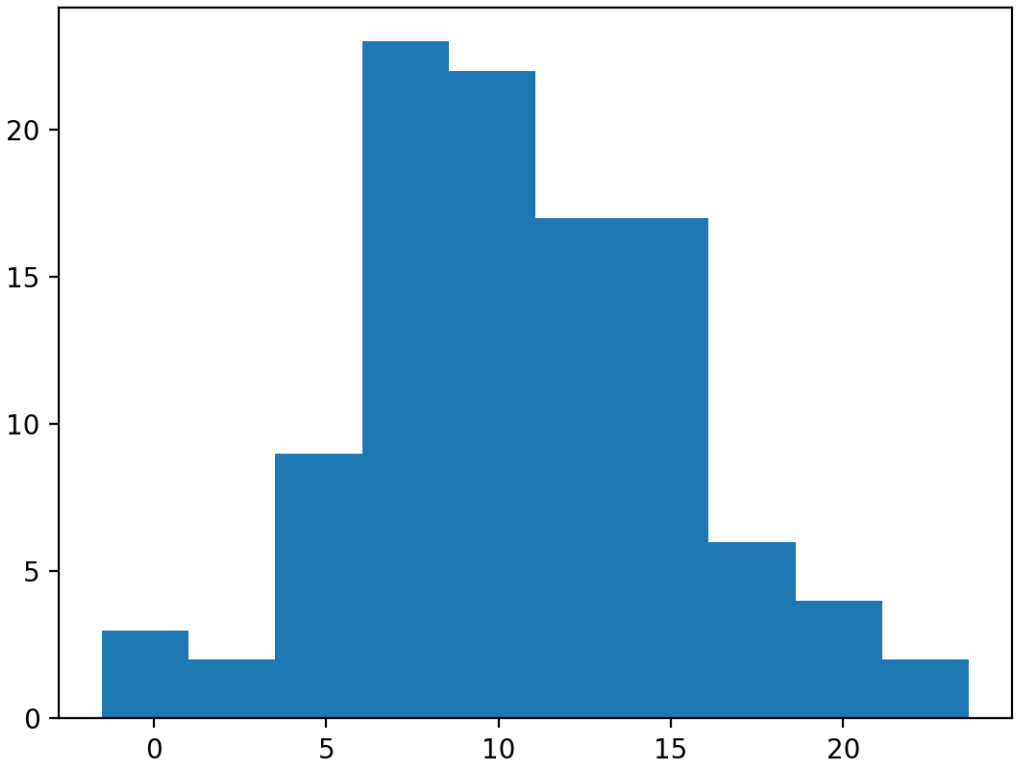

Figure 25.6: Histogram plot of data sample with a truncated long tail.

25.7 Power Transforms

The distribution of the data may be normal, but the data may require a transform in order to help expose it. For example, the data may have a skew, meaning that the bell in the bell shape may be pushed one way or another. In some cases, this can be corrected by transforming the data via calculating the square root of the observations. Alternately, the distribution may be exponential, but may look normal if the observations are transformed by taking the natural logarithm of the values. Data with this distribution is called log-normal. To make this concrete, below is an example of a sample of Gaussian numbers transformed to have an exponential distribution.

```
# log-normal distribution
from numpy.random import seed
from numpy.random import randn
from numpy import exp
from matplotlib import pyplot
# seed the random number generator
seed(1)
# generate two sets of univariate observations
data = 5 * randn(100) + 50
# transform to be exponential
data = exp(data)
# histogram
pyplot.hist(data)
pyplot.show()
```

Listing 25.7: Example of plotting a data sample with an exponential distribution.

Running the example creates a histogram showing the exponential distribution. It is not obvious that the data is in fact log-normal.

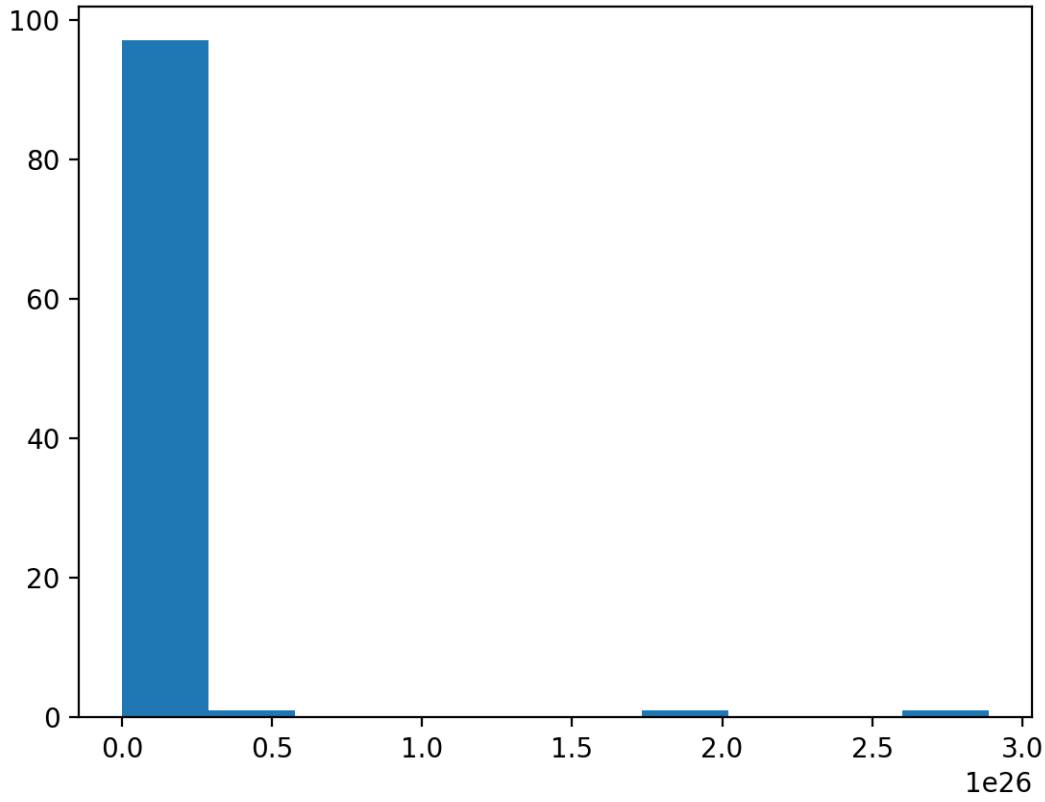

Figure 25.7: Histogram plot of data sample with an exponential distribution.

Taking the square root and the logarithm of the observation in order to make the distribution normal belongs to a class of transforms called power transforms. The Box-Cox method is a data transform method that is able to perform a range of power transforms, including the log and the square root. The method is named for George Box and David Cox. More than that, it can be configured to evaluate a suite of transforms automatically and select a best fit. It can be thought of as a power tool to iron out power-based change in your data sample. The resulting data sample may be more linear and will better represent the underlying non-power distribution, including Gaussian.

The `boxcox()` SciPy function implements the Box-Cox method. It takes an argument, called `lambda`, that controls the type of transform to perform. Below are some common values for `lambda`:

- `lambda = -1.0` is a reciprocal transform.
- `lambda = -0.5` is a reciprocal square root transform.
- `lambda = 0.0` is a log transform.
- `lambda = 0.5` is a square root transform.
- `lambda = 1.0` is no transform.

For example, because we know that the data is lognormal, we can use the Box-Cox to perform the log transform by setting lambda explicitly to 0.

```
...
# power transform
data = boxcox(data, 0)
```

Listing 25.8: Example of calculating a Box-Cox transform.

The complete example of applying the Box-Cox transform on the exponential data sample is listed below.

```
# box-cox transform
from numpy.random import seed
from numpy.random import randn
from numpy import exp
from scipy.stats import boxcox
from matplotlib import pyplot
# seed the random number generator
seed(1)
# generate two sets of univariate observations
data = 5 * randn(100) + 100
# transform to be exponential
data = exp(data)
# power transform
data = boxcox(data, 0)
# histogram
pyplot.hist(data)
pyplot.show()
```

Listing 25.9: Example of calculating a Box-Cox transform on a data sample.

Running the example performs the Box-Cox transform on the data sample and plots the result, clearly showing the Gaussian distribution.

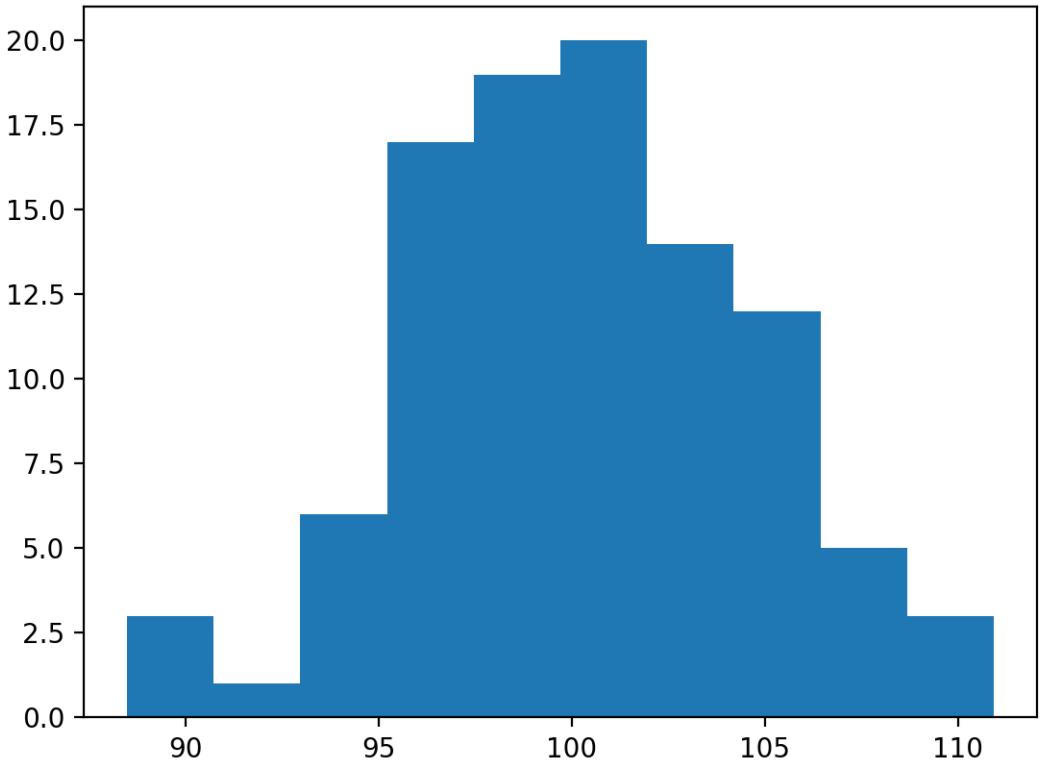

Figure 25.8: Histogram plot of data sample after a Box-Cox transform.

A limitation of the Box-Cox transform is that it assumes that all values in the data sample are positive. An alternative method that does not make this assumption is the Yeo-Johnson transformation.

25.8 Use Anyway

Finally, you may wish to treat the data as Gaussian anyway, especially if the data is already Gaussian-like. In some cases, such as the use of parametric statistical methods, this may lead to optimistic findings. In other cases, such as machine learning methods that make Gaussian expectations on input data, you may still see good results. This is a choice you can make, as long as you are aware of the possible downsides.

25.9 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- List 3 possible additional ways that a Gaussian distribution may have been distorted

- Develop a data sample and experiment with the 5 common values for lambda in the Box-Cox transform.
- Load a machine learning dataset where at least one variable has a Gaussian-like distribution and experiment.

If you explore any of these extensions, I'd love to know.

25.10 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

25.10.1 API

- `matplotlib.pyplot.hist` API.
  https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html
- `scipy.stats.boxcox` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.boxcox.html

25.10.2 Articles

- Normal distribution on Wikipedia.
  https://en.wikipedia.org/wiki/Normal_distribution
- Outlier on Wikipedia.
  https://en.wikipedia.org/wiki/Outlier
- Log-normal distribution on Wikipedia.
  https://en.wikipedia.org/wiki/Log-normal_distribution
- Power transform on Wikipedia.
  https://en.wikipedia.org/wiki/Power_transform

25.11 Summary

In this tutorial, you discovered the reasons why a Gaussian-like distribution may be distorted and techniques that you can use to make a data sample more normal. Specifically, you learned:

- How to consider the size of the sample and whether the law of large numbers may help improve the distribution of a sample.
- How to identify and remove extreme values and long tails from a distribution.
- Power transforms and the Box-Cox transform that can be used to control for quadratic or exponential distributions.

25.11.1 Next

In the next section, you will discover how to summarize a data sample regardless of it's distribution.

Chapter 26

5-Number Summary

Data summarization provides a convenient way to describe all of the values in a data sample with just a few statistical values. The mean and standard deviation are used to summarize data with a Gaussian distribution, but may not be meaningful, or could even be misleading, if your data sample has a non-Gaussian distribution. In this tutorial, you will discover the five-number summary for describing the distribution of a data sample without assuming a specific data distribution. After completing this tutorial, you will know:

- Data summarization, such as calculating the mean and standard deviation, are only meaningful for the Gaussian distribution.
- The five-number summary can be used to describe a data sample with any distribution.
- How to calculate the five-number summary in Python.

Let's get started.

26.1 Tutorial Overview

This tutorial is divided into 4 parts; they are:

1. Nonparametric Data Summarization
2. Five-Number Summary
3. How to Calculate the Five-Number Summary
4. Use of the Five-Number Summary

26.2 Nonparametric Data Summarization

Data summarization techniques provide a way to describe the distribution of data using a few key measurements. The most common example of data summarization is the calculation of the mean and standard deviation for data that has a Gaussian distribution. With these two parameters alone, you can understand and re-create the distribution of the data. The data summary can compress as few as tens or as many as millions individual observations.

The problem is, you cannot easily calculate the mean and standard deviation of data that does not have a Gaussian distribution. Technically, you can calculate these quantities, but they do not summarize the data distribution; in fact, they can be very misleading. In the case of data that does not have a Gaussian distribution, you can summarize the data sample using the five-number summary.

26.3 Five-Number Summary

The five-number summary, or 5-number summary for short, is a nonparametric data summarization technique. It is sometimes called the Tukey 5-number summary because it was recommended by John Tukey. It can be used to describe the distribution of data samples for data with any distribution.

> As a standard summary for general use, the 5-number summary provides about the right amount of detail.
>
> — Page 37, *Understanding Robust and Exploratory Data Analysis*, 2000.

The five-number summary involves the calculation of 5 summary statistical quantities: namely:

- **Median**: The middle value in the sample, also called the 50th percentile or the 2nd quartile.
- **1st Quartile**: The 25th percentile.
- **3rd Quartile**: The 75th percentile.
- **Minimum**: The smallest observation in the sample.
- **Maximum**: The largest observation in the sample.

A quartile is an observed value at a point that aids in splitting the ordered data sample into four equally sized parts. The median, or 2nd Quartile, splits the ordered data sample into two parts, and the 1st and 3rd quartiles split each of those halves into quarters. A percentile is an observed value at a point that aids in splitting the ordered data sample into 100 equally sized portions. Quartiles are often also expressed as percentiles.

Both the quartile and percentile values are examples of rank statistics that can be calculated on a data sample with any distribution. They are used to quickly summarize how much of the data in the distribution is behind or in front of a given observed value. For example, half of the observations are behind and in front of the median of a distribution. Note that quartiles are also calculated in the box and whisker plot, a nonparametric method to graphically summarize the distribution of a data sample.

26.4 How to Calculate the Five-Number Summary

Calculating the five-number summary involves finding the observations for each quartile as well as the minimum and maximum observed values from the data sample. If there is no specific value in the ordered data sample for the quartile, such as if there are an even number of observations and we are trying to find the median, then we can calculate the mean of the two closest values, such as the two middle values.

We can calculate arbitrary percentile values in Python using the `percentile()` NumPy function. We can use this function to calculate the 1st, 2nd (median), and 3rd quartile values. The function takes both an array of observations and a floating point value to specify the percentile to calculate in the range of 0 to 100. It can also takes a list of percentile values to calculate multiple percentiles; for example:

```
...
# calculate quartiles
quartiles = percentile(data, [25, 50, 75])
```

Listing 26.1: Example of calculating quartiles.

By default, the function will calculate a linear interpolation (average) between observations if needed, such as in the case of calculating the median on a sample with an even number of values. The NumPy functions `min()` and `max()` can be used to return the smallest and largest values in the data sample; for example:

```
...
# calculate min and max
data_min, data_max = data.min(), data.max()
```

Listing 26.2: Example of calculating min and max.

We can put all of this together. The example below generates a data sample drawn from a uniform distribution between 0 and 1 and summarizes it using the five-number summary.

```
# calculate a 5-number summary
from numpy import percentile
from numpy.random import seed
from numpy.random import rand
# seed random number generator
seed(1)
# generate data sample
data = rand(1000)
# calculate quartiles
quartiles = percentile(data, [25, 50, 75])
# calculate min/max
data_min, data_max = data.min(), data.max()
# display 5-number summary
print('Min: %.3f' % data_min)
print('Q1: %.3f' % quartiles[0])
print('Median: %.3f' % quartiles[1])
print('Q3: %.3f' % quartiles[2])
print('Max: %.3f' % data_max)
```

Listing 26.3: Example of calculating a 5 number summary of a data sample.

Running the example generates the data sample and calculates the five-number summary to describe the sample distribution. We can see that the spread of observations is close to our

expectations showing 0.252 for the 25th percentile 0.508 for the 50th percentile, and 0.751 for the 75th percentile, close to the idealized values of 0.250, 0.500, and 0.750 respectively.

```
Min: 0.000
Q1: 0.252
Median: 0.508
Q3: 0.751
Max: 0.997
```

Listing 26.4: Example output from calculating a 5 number summary of a data sample.

26.5 Use of the Five-Number Summary

The five-number summary can be calculated for a data sample with any distribution. This includes data that has a known distribution, such as a Gaussian or Gaussian-like distribution. I would recommend always calculating the five-number summary, and only moving on to distribution specific summaries, such as mean and standard deviation for the Gaussian, in the case that you can identify the distribution to which the data belongs.

26.6 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Describe three examples in a machine learning project where a five-number summary could be calculated.
- Generate a data sample with a Gaussian distribution and calculate the five-number summary.
- Write a function to calculate a 5-number summary for any data sample.

If you explore any of these extensions, I'd love to know.

26.7 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

26.7.1 Books

- *Understanding Robust and Exploratory Data Analysis*, 2000.
  https://amzn.to/2Gp2sNW

26.7.2 API

- `numpy.percentile` API.
  https://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.percentile.html

- `numpy.ndarray.min` API.
  https://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.ndarray.min.html

- `numpy.ndarray.max` API.
  https://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.ndarray.max.html

26.7.3 Articles

- Five-number summary on Wikipedia.
  https://en.wikipedia.org/wiki/Five-number_summary

- Quartile on Wikipedia.
  https://en.wikipedia.org/wiki/Quartile

- Percentile on Wikipedia.
  https://en.wikipedia.org/wiki/Percentile

26.8 Summary

In this tutorial, you discovered the five-number summary for describing the distribution of a data sample without assuming a specific data distribution. Specifically, you learned:

- Data summarization, such as calculating the mean and standard deviation, are only meaningful for the Gaussian distribution.
- The five-number summary can be used to describe a data sample with any distribution.
- How to calculate the five-number summary in Python.

26.8.1 Next

In the next section, you will discover how to quantify the relationship between two samples regardless of their distribution.

Chapter 27

Rank Correlation

Correlation is a measure of the association between two variables. It is easy to calculate and interpret when both variables have a well understood Gaussian distribution. When we do not know the distribution of the variables, we must use nonparametric rank correlation methods. In this tutorial, you will discover rank correlation methods for quantifying the association between variables with a non-Gaussian distribution. After completing this tutorial, you will know:

- How rank correlation methods work and the methods are that are available.
- How to calculate and interpret the Spearman's rank correlation coefficient in Python.
- How to calculate and interpret the Kendall's rank correlation coefficient in Python.

Let's get started.

27.1 Tutorial Overview

This tutorial is divided into 4 parts; they are:

1. Rank Correlation
2. Test Dataset
3. Spearman's Rank Correlation
4. Kendall's Rank Correlation

27.2 Rank Correlation

Correlation refers to the association between the observed values of two variables. The variables may have a positive association, meaning that as the values for one variable increase, so do the values of the other variable. The association may also be negative, meaning that as the values of one variable increase, the values of the others decrease. Finally, the association may be neutral, meaning that the variables are not associated. Correlation quantifies this association, often as a measure between the values -1 to 1 for perfectly negatively correlated and perfectly

positively correlated. The calculated correlation is referred to as the *correlation coefficient*. This correlation coefficient can then be interpreted to describe the measures.

See the table below to help with interpretation the correlation coefficient.

TABLE 7.1

Correlation Coefficient for a Direct Relationship	Correlation Coefficient for an Indirect Relationship	Relationship Strength of the Variables
0.0	0.0	None/trivial
0.1	−0.1	Weak/small
0.3	−0.3	Moderate/medium
0.5	−0.5	Strong/large
1.0	−1.0	Perfect

Figure 27.1: Table of correlation coefficient values and their interpretation. Taken from *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*.

The correlation between two variables that each have a Gaussian distribution can be calculated using standard methods such as the Pearson's correlation. This procedure cannot be used for data that does not have a Gaussian distribution. Instead, rank correlation methods must be used. Rank correlation refers to methods that quantify the association between variables using the ordinal relationship between the values rather than the specific values. Ordinal data is data that has label values and has an order or rank relationship; for example: `low`, `medium`, and `high`.

Rank correlation can be calculated for real-valued variables. This is done by first converting the values for each variable into rank data. This is where the values are ordered and assigned an integer rank value. Rank correlation coefficients can then be calculated in order to quantify the association between the two ranked variables. Because no distribution for the values is assumed, rank correlation methods are referred to as distribution-free correlation or nonparametric correlation. Interestingly, rank correlation measures are often used as the basis for other statistical hypothesis tests, such as determining whether two samples were likely drawn from the same (or different) population distributions.

Rank correlation methods are often named after the researcher or researchers that developed the method. Four examples of rank correlation methods are as follows:

- Spearman's Rank Correlation.
- Kendall's Rank Correlation.
- Goodman and Kruskal's Rank Correlation.
- Somers' Rank Correlation.

In the following sections, we will take a closer look at two of the more common rank correlation methods: Spearman's and Kendall's.

27.3 Test Dataset

Before we demonstrate rank correlation methods, we must first define a test problem. In this section, we will define a simple two-variable dataset where each variable is drawn from a uniform distribution (e.g. non-Gaussian) and the values of the second variable depend on the values of the first value. Specifically, a sample of 1,000 random floating point values are drawn from a uniform distribution and scaled to the range 0 to 20. A second sample of 1,000 random floating point values are drawn from a uniform distribution between 0 and 10 and added to values in the first sample to create an association.

```
...
# prepare data
data1 = rand(1000) * 20
data2 = data1 + (rand(1000) * 10)
```

Listing 27.1: Generate data sample with linearly related variables.

The complete example is listed below.

```
# generate related variables
from numpy.random import rand
from numpy.random import seed
from matplotlib import pyplot
# seed random number generator
seed(1)
# prepare data
data1 = rand(1000) * 20
data2 = data1 + (rand(1000) * 10)
# plot
pyplot.scatter(data1, data2)
pyplot.show()
```

Listing 27.2: Example generating and plotting a dataset with dependent variables.

Running the example generates the data sample and graphs the points on a scatter plot. We can clearly see that each variable has a uniform distribution and the positive association between the variables is visible by the diagonal grouping of the points from the bottom left to the top right of the plot.

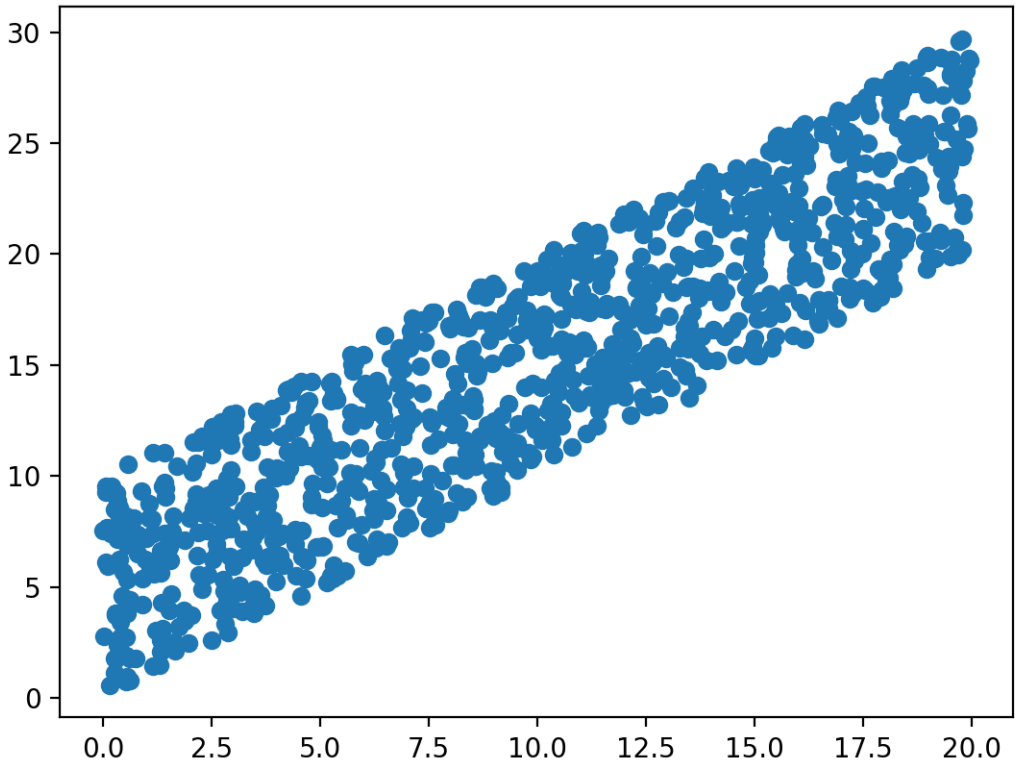

Figure 27.2: Scatter plot of associated variables drawn from a uniform distribution.

27.4 Spearman's Rank Correlation

Spearman's rank correlation is named for Charles Spearman. It may also be called Spearman's correlation coefficient and is denoted by the lowercase Greek letter rho (ρ). As such, it may be referred to as Spearman's rho. This statistical method quantifies the degree to which ranked variables are associated by a monotonic function, meaning an increasing or decreasing relationship. As a statistical hypothesis test, the method assumes that the samples are uncorrelated (fail to reject H0).

> The Spearman rank-order correlation is a statistical procedure that is designed to measure the relationship between two variables on an ordinal scale of measurement.

— Page 124, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*, 2009.

The intuition for the Spearman's rank correlation is that it calculates a Pearson's correlation (e.g. a parametric measure of correlation) using the rank values instead of the real values. Where the Pearson's correlation is the calculation of the covariance (or expected difference of observations from the mean) between the two variables normalized by the variance or spread of both variables. Spearman's rank correlation can be calculated in Python using the `spearmanr()`

SciPy function. The function takes two real-valued samples as arguments and returns both the correlation coefficient in the range between -1 and 1 and the p-value for interpreting the significance of the coefficient.

```
...
# calculate spearman's correlation
coef, p = spearmanr(data1, data2)
```

Listing 27.3: Example of calculating Spearman's rank correlation.

We can demonstrate the Spearman's rank correlation on the test dataset. We know that there is a strong association between the variables in the dataset and we would expect the Spearman's test to find this association. The complete example is listed below.

```
# calculate the spearman's correlation between two variables
from numpy.random import rand
from numpy.random import seed
from scipy.stats import spearmanr
# seed random number generator
seed(1)
# prepare data
data1 = rand(1000) * 20
data2 = data1 + (rand(1000) * 10)
# calculate spearman's correlation
coef, p = spearmanr(data1, data2)
print('Spearmans correlation coefficient: %.3f' % coef)
# interpret the significance
alpha = 0.05
if p > alpha:
  print('Samples are uncorrelated (fail to reject H0) p=%.3f' % p)
else:
  print('Samples are correlated (reject H0) p=%.3f' % p)
```

Listing 27.4: Example calculating Spearman's rank correlation on the test dataset.

Running the example calculates the Spearman's correlation coefficient between the two variables in the test dataset. The statistical test reports a strong positive correlation with a value of 0.9. The p-value is close to zero, which means that the likelihood of observing the data given that the samples are uncorrelated is very unlikely (e.g. 95% confidence) and that we can reject the null hypothesis that the samples are uncorrelated.

```
Spearmans correlation coefficient: 0.900
Samples are correlated (reject H0) p=0.000
```

Listing 27.5: Example output from calculating Spearman's rank correlation on the test dataset.

27.5 Kendall's Rank Correlation

Kendall's rank correlation is named for Maurice Kendall. It is also called Kendall's correlation coefficient, and the coefficient is often referred to by the lowercase Greek letter tau (τ). In turn, the test may be called Kendall's tau. The intuition for the test is that it calculates a normalized score for the number of matching or concordant rankings between the two samples. As such, the test is also referred to as Kendall's concordance test. The Kendall's rank correlation

coefficient can be calculated in Python using the `kendalltau()` SciPy function. The test takes the two data samples as arguments and returns the correlation coefficient and the p-value. As a statistical hypothesis test, the method assumes (H0) that there is no association between the two samples.

```
...
# calculate kendall's correlation
coef, p = kendalltau(data1, data2)
```

Listing 27.6: Example of calculating Kendall's rank correlation.

We can demonstrate the calculation on the test dataset, where we do expect a significant positive association to be reported. The complete example is listed below.

```
# calculate the kendall's correlation between two variables
from numpy.random import rand
from numpy.random import seed
from scipy.stats import kendalltau
# seed random number generator
seed(1)
# prepare data
data1 = rand(1000) * 20
data2 = data1 + (rand(1000) * 10)
# calculate kendall's correlation
coef, p = kendalltau(data1, data2)
print('Kendall correlation coefficient: %.3f' % coef)
# interpret the significance
alpha = 0.05
if p > alpha:
  print('Samples are uncorrelated (fail to reject H0) p=%.3f' % p)
else:
  print('Samples are correlated (reject H0) p=%.3f' % p)
```

Listing 27.7: Example calculating Kendall's rank correlation on the test dataset.

Running the example calculates the Kendall's correlation coefficient as 0.7, which is highly correlated. The p-value is close to zero (and printed as zero), as with the Spearman's test, meaning that we can confidently reject the null hypothesis that the samples are uncorrelated.

```
Kendall correlation coefficient: 0.709
Samples are correlated (reject H0) p=0.000
```

Listing 27.8: Example output from calculating Kendall's rank correlation on the test dataset.

27.6 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- List three examples where calculating a nonparametric correlation coefficient might be useful during a machine learning project.
- Update each example to calculate the correlation between uncorrelated data samples drawn from a non-Gaussian distribution.

- Load a standard machine learning dataset and calculate the pairwise nonparametric correlation between all variables.

If you explore any of these extensions, I'd love to know.

27.7 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

27.7.1 Books

- *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*, 2009.
  https://amzn.to/2HevldG
- *Applied Nonparametric Statistical Methods*, Fourth Edition, 2007.
  https://amzn.to/2GCKnfW
- *Rank Correlation Methods*, 1990.
  https://amzn.to/2JofYzY

27.7.2 API

- `scipy.stats.spearmanr` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.spearmanr.html
- `scipy.stats.kendalltau` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kendalltau.html

27.7.3 Articles

- Nonparametric statistics on Wikipedia.
  https://en.wikipedia.org/wiki/Nonparametric_statistics
- Rank correlation on Wikipedia.
  https://en.wikipedia.org/wiki/Rank_correlation
- Spearman's rank correlation coefficient on Wikipedia.
  https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient
- Kendall rank correlation coefficient on Wikipedia.
  https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient
- Goodman and Kruskal's gamma on Wikipedia.
  https://en.wikipedia.org/wiki/Goodman_and_Kruskal%27s_gamma
- Somers' D on Wikipedia.
  https://en.wikipedia.org/wiki/Somers%27_D

27.8 Summary

In this tutorial, you discovered rank correlation methods for quantifying the association between variables with a non-Gaussian distribution. Specifically, you learned:

- How rank correlation methods work and the methods are that are available.
- How to calculate and interpret the Spearman's rank correlation coefficient in Python.
- How to calculate and interpret the Kendall's rank correlation coefficient in Python.

27.8.1 Next

In the next section, you will discover how to calculate nonparametric statistical hypothesis tests.

Chapter 28

Rank Significance Tests

In applied machine learning, we often need to determine whether two data samples have the same or different distributions. We can answer this question using statistical significance tests that can quantify the likelihood that the samples have the same distribution.

If the data does not have the familiar Gaussian distribution, we must resort to nonparametric version of the significance tests. These tests operate in a similar manner, but are distribution free, requiring that real valued data be first transformed into rank data before the test can be performed. In this tutorial, you will discover nonparametric statistical tests that you can use to determine if data samples were drawn from populations with the same or different distributions. After completing this tutorial, you will know:

- The Mann-Whitney U test for comparing independent data samples: the nonparametric version of the Student's t-test.
- The Wilcoxon signed-rank test for comparing paired data samples: the nonparametric version of the paired Student's t-test.
- The Kruskal-Wallis H and Friedman tests for comparing more than two data samples: the nonparametric version of the ANOVA and repeated measures ANOVA tests.

Let's get started.

28.1 Tutorial Overview

This tutorial is divided into 6 parts; they are:

1. Nonparametric Statistical Significance Tests
2. Test Data
3. Mann-Whitney U Test
4. Wilcoxon Signed-Rank Test
5. Kruskal-Wallis H Test
6. Friedman Test

28.2 Nonparametric Statistical Significance Tests

Nonparametric statistics are those methods that do not assume a specific distribution to the data. Often, they refer to statistical methods that do not assume a Gaussian distribution. They were developed for use with ordinal or interval data, but in practice can also be used with a ranking of real-valued observations in a data sample rather than on the observation values themselves. A common question about two or more datasets is whether they are different. Specifically, whether the difference between their central tendency (e.g. mean or median) is statistically significant.

This question can be answered for data samples that do not have a Gaussian distribution by using nonparametric statistical significance tests. The null hypothesis of these tests is often the assumption that both samples were drawn from a population with the same distribution, and therefore the same population parameters, such as mean or median.

If after calculating the significance test on two or more samples the null hypothesis is rejected, it indicates that there is evidence to suggest that samples were drawn from different populations, and in turn the difference between sample estimates of population parameters, such as means or medians may be significant. These tests are often used on samples of model skill scores in order to confirm that the difference in skill between machine learning models is significant.

In general, each test calculates a test statistic, that must be interpreted with some background in statistics and a deeper knowledge of the statistical test itself. Tests also return a p-value that can be used to interpret the result of the test. The p-value can be thought of as the probability of observing the two data samples given the base assumption (null hypothesis) that the two samples were drawn from a population with the same distribution. The p-value can be interpreted in the context of a chosen significance level called alpha. A common value for alpha is 5% or 0.05. If the p-value is below the significance level, then the test says there is enough evidence to reject the null hypothesis and that the samples were likely drawn from populations with differing distributions.

- `p-value` $\leq$ `alpha`: significant result, reject null hypothesis, distributions differ (H1).
- `p-value` $>$ `alpha`: not significant result, fail to reject null hypothesis, distributions same (H0).

28.3 Test Dataset

Before we look at specific nonparametric significance tests, let's first define a test dataset that we can use to demonstrate each test. We will generate two samples drawn from different distributions. We will use the `rand()` NumPy function to generate two samples of 100 uniform random numbers between 50-59 and 51-60 respectively. We expect the statistical tests to discover that the samples were drawn from differing distributions, although the small sample size of 100 observations per sample will add some noise to this decision. The complete code example is listed below.

```
# generate data samples
from numpy.random import seed
from numpy.random import rand
# seed the random number generator
seed(1)
```

```
# generate two sets of univariate observations
data1 = 50 + (rand(100) * 10)
data2 = 51 + (rand(100) * 10)
# summarize
print('data1: min=%.3f max=%.3f' % (min(data1), max(data1)))
print('data2: min=%.3f max=%.3f' % (min(data2), max(data2)))
```

Listing 28.1: Example generating a small test dataset.

Running the example generates the data samples, then calculates and prints the min and max values for each sample, confirming their different distribution.

```
data1: min=50.001 max=59.889
data2: min=51.126 max=60.973
```

Listing 28.2: Example output from creating a small test dataset.

28.4 Mann-Whitney U Test

The Mann-Whitney U test is a nonparametric statistical significance test for determining whether two independent samples were drawn from a population with the same distribution. The test was named for Henry Mann and Donald Whitney, although it is sometimes called the Wilcoxon-Mann-Whitney test, also named for Frank Wilcoxon, who also developed a variation of the test.

> The two samples are combined and rank ordered together. The strategy is to determine if the values from the two samples are randomly mixed in the rank ordering or if they are clustered at opposite ends when combined. A random rank order would mean that the two samples are not different, while a cluster of one sample values would indicate a difference between them.

— Page 58, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*, 2009.

The default assumption or null hypothesis is that there is no difference between the distributions of the data samples. Rejection of this hypothesis suggests that there is likely some difference between the samples. More specifically, the test determines whether it is equally likely that any randomly selected observation from one sample will be greater or less than a sample in the other distribution. If violated, it suggests differing distributions.

- **Fail to Reject H0**: Sample distributions are equal.
- **Reject H0**: Sample distributions are not equal.

For the test to be effective, it requires at least 20 observations in each data sample. We can implement the Mann-Whitney U test in Python using the `mannwhitneyu()` SciPy function. The functions takes as arguments the two data samples. It returns the test statistic and the p-value. The example below demonstrates the Mann-Whitney U test on the test dataset.

```
# example of the mann-whitney u test
from numpy.random import seed
from numpy.random import rand
from scipy.stats import mannwhitneyu
# seed the random number generator
seed(1)
# generate two independent samples
data1 = 50 + (rand(100) * 10)
data2 = 51 + (rand(100) * 10)
# compare samples
stat, p = mannwhitneyu(data1, data2)
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
  print('Same distribution (fail to reject H0)')
else:
  print('Different distribution (reject H0)')
```

Listing 28.3: Example of calculating the Mann-Whitney U test on the test dataset.

Running the example calculates the test on the datasets and prints the statistic and p-value. The p-value strongly suggests that the sample distributions are different, as is expected.

```
Statistics=4077.000, p=0.012
Different distribution (reject H0)
```

Listing 28.4: Example output from calculating the Mann-Whitney U test on the test dataset.

28.5 Wilcoxon Signed-Rank Test

In some cases, the data samples may be paired. There are many reasons why this may be the case, for example, the samples are related or matched in some way or represent two measurements of the same technique. More specifically, each sample is independent, but comes from the same population. Examples of paired samples in machine learning might be the same algorithm evaluated on different datasets or different algorithms evaluated on exactly the same training and test data. The samples are not independent, therefore the Mann-Whitney U test cannot be used. Instead, the Wilcoxon signed-rank test is used, also called the Wilcoxon t-test, named for Frank Wilcoxon. It is the equivalent of the paired Student's t-test, but for ranked data instead of real valued data with a Gaussian distribution.

> The Wilcoxon signed ranks test is a nonparametric statistical procedure for comparing two samples that are paired, or related. The parametric equivalent to the Wilcoxon signed ranks test goes by names such as the Student's t-test, t-test for matched pairs, t-test for paired samples, or t-test for dependent samples.

— Pages 38-39, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*, 2009.

The default assumption for the test, the null hypothesis, is that the two samples have the same distribution.

- **Fail to Reject H0**: Sample distributions are equal.
- **Reject H0**: Sample distributions are not equal.

For the test to be effective, it requires at least 20 observations in each data sample. The Wilcoxon signed-rank test can be implemented in Python using the `wilcoxon()` SciPy function. The function takes the two samples as arguments and returns the calculated statistic and p-value. The complete example is below, demonstrating the calculation of the Wilcoxon signed-rank test on the test problem. The two samples are technically not paired, but we can pretend they are for the sake of demonstrating the calculation of this significance test.

```
# example of the wilcoxon signed-rank test
from numpy.random import seed
from numpy.random import rand
from scipy.stats import wilcoxon
# seed the random number generator
seed(1)
# generate two independent samples
data1 = 50 + (rand(100) * 10)
data2 = 51 + (rand(100) * 10)
# compare samples
stat, p = wilcoxon(data1, data2)
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
  print('Same distribution (fail to reject H0)')
else:
  print('Different distribution (reject H0)')
```

Listing 28.5: Example of calculating the Wilcoxon Signed-Rank test on the test dataset.

Running the example calculates and prints the statistic and prints the result. The p-value is interpreted strongly suggesting that the samples are drawn from different distributions.

```
Statistics=1937.000, p=0.043
Different distribution (reject H0)
```

Listing 28.6: Example output from calculating the Wilcoxon Signed-Rank test on the test dataset.

28.6 Kruskal-Wallis H Test

When working with significance tests, such as Mann-Whitney U and the Wilcoxon signed-rank tests, comparisons between data samples must be performed pairwise. This can be inefficient if you have many data samples and you are only interested in whether two or more samples have a different distribution. The Kruskal-Wallis test is a nonparametric version of the one-way analysis of variance test or ANOVA for short. It is named for the developers of the method, William Kruskal and Wilson Wallis. This test can be used to determine whether more than two independent samples have a different distribution. It can be thought of as the generalization of the Mann-Whitney U test.

The default assumption or the null hypothesis is that all data samples were drawn from the same distribution. Specifically, that the population medians of all groups are equal. A rejection of the null hypothesis indicates that there is enough evidence to suggest that one or more samples dominate another sample, but the test does not indicate which samples or by how much.

> When the Kruskal-Wallis H-test leads to significant results, then at least one of the samples is different from the other samples. However, the test does not identify where the difference(s) occur. Moreover, it does not identify how many differences occur. To identify the particular differences between sample pairs, a researcher might use sample contrasts, or post hoc tests, to analyze the specific sample pairs for significant difference(s). The Mann-Whitney U-test is a useful method for performing sample contrasts between individual sample sets.

— Page 100, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*, 2009.

- **Fail to Reject H0**: All sample distributions are equal.
- **Reject H0**: One or more sample distributions are not equal.

Each data sample must be independent, have 5 or more observations, and the data samples can differ in size. We can update the test problem to have 3 data samples, each of which has a slightly different sample mean. We would expect the test to discover the difference and reject the null hypothesis.

```
...
# generate three independent samples
data1 = 50 + (rand(100) * 10)
data2 = 51 + (rand(100) * 10)
data3 = 52 + (rand(100) * 10)
```

Listing 28.7: Example of updated dataset with multiple independent samples.

The Kruskal-Wallis H-test can be implemented in Python using the `kruskal()` SciPy function. It takes two or more data samples as arguments and returns the test statistic and p-value as the result. The complete example is listed below.

```
# example of the kruskal-wallis h-test
from numpy.random import seed
from numpy.random import rand
from scipy.stats import kruskal
# seed the random number generator
seed(1)
# generate three independent samples
data1 = 50 + (rand(100) * 10)
data2 = 51 + (rand(100) * 10)
data3 = 52 + (rand(100) * 10)
# compare samples
stat, p = kruskal(data1, data2, data3)
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
```

```
  print('Same distributions (fail to reject H0)')
else:
  print('Different distributions (reject H0)')
```

Listing 28.8: Example of calculating the Kruskal-Wallis H test on the test dataset.

Running the example calculates the test and prints the results. The p-value is interpreted, correctly rejecting the null hypothesis that all samples have the same distribution.

```
Statistics=34.747, p=0.000
Different distributions (reject H0)
```

Listing 28.9: Example output from calculating the Kruskal-Wallis H test on the test dataset.

28.7 Friedman Test

As in the previous example, we may have more than two different samples and an interest in whether all samples have the same distribution or not. If the samples are paired in some way, such as repeated measures, then the Kruskal-Wallis H test would not be appropriate. Instead, the Friedman test can be used, named for Milton Friedman. The Friedman test is the nonparametric version of the repeated measures analysis of variance test, or repeated measures ANOVA. The test can be thought of as a generalization of the Kruskal-Wallis H Test to more than two samples. The default assumption, or null hypothesis, is that the multiple paired samples have the same distribution. A rejection of the null hypothesis indicates that one or more of the paired samples has a different distribution.

- **Fail to Reject H0**: Paired sample distributions are equal.
- **Reject H0**: Paired sample distributions are not equal.

The test assumes two or more paired data samples with 10 or more samples per group.

> The Friedman test is a nonparametric statistical procedure for comparing more than two samples that are related. The parametric equivalent to this test is the repeated measures analysis of variance (ANOVA). When the Friedman test leads to significant results, at least one of the samples is different from the other samples.

— Pages 79-80, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*, 2009.

We can implement the Friedman test in Python using the `friedmanchisquare()` SciPy function. This function takes as arguments the data samples to compare and returns the calculated statistic and p-value. This significance test can be demonstrated on the same variation of the test dataset as was used in the previous section. Namely three samples, each with a slightly different sample mean. Although the samples are not paired, we expect the test to discover that not all of the samples have the same distribution. The complete code example is listed below.

```
# example of the friedman test
from numpy.random import seed
from numpy.random import rand
from scipy.stats import friedmanchisquare
# seed the random number generator
seed(1)
# generate three independent samples
data1 = 50 + (rand(100) * 10)
data2 = 51 + (rand(100) * 10)
data3 = 52 + (rand(100) * 10)
# compare samples
stat, p = friedmanchisquare(data1, data2, data3)
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
  print('Same distributions (fail to reject H0)')
else:
  print('Different distributions (reject H0)')
```

Listing 28.10: Example of calculating the Friedman test on the test dataset.

Running the example calculates the test on the three data samples and prints the test statistic and p-value. The interpretation of the p-value correctly indicates that at least one sample has a different distribution.

```
Statistics=36.240, p=0.000
Different distributions (reject H0)
```

Listing 28.11: Example output from calculating the Friedman test on the test dataset.

28.8 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Update all examples to operate on data samples that have the same distribution.
- Create a flowchart for choosing each of the statistical significance tests given the requirements and behavior of each test.
- Consider 3 cases of comparing data samples in a machine learning project, assume a non-Gaussian distribution for the samples, and suggest the type of test that could be used in each case.

If you explore any of these extensions, I'd love to know.

28.9 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

28.9.1 Books

- *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*, 2009. http://amzn.to/2CZcXBz

28.9.2 Books

- `scipy.stats.mannwhitneyu` API. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mannwhitneyu.html
- `scipy.stats.wilcoxon` API. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wilcoxon.html
- `scipy.stats.kruskal` API. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kruskal.html
- `scipy.stats.friedmanchisquare` API. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kruskal.html

28.9.3 Articles

- Nonparametric statistics on Wikipedia. https://en.wikipedia.org/wiki/Nonparametric_statistics
- Paired difference test on Wikipedia. https://en.wikipedia.org/wiki/Paired_difference_test
- Mann-Whitney U test on Wikipedia. https://en.wikipedia.org/wiki/Mann%E2%80%93Whitney_U_test
- Wilcoxon signed-rank test on Wikipedia. https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test
- Kruskal-Wallis one-way analysis of variance on Wikipedia. https://en.wikipedia.org/wiki/Kruskal%E2%80%93Wallis_one-way_analysis_of_variance
- Friedman test on Wikipedia. https://en.wikipedia.org/wiki/Friedman_test

28.9.4 Summary

In this tutorial, you discovered nonparametric statistical tests that you can use to determine if data samples were drawn from populations with the same or different distributions. Specifically, you learned:

- The Mann-Whitney U test for comparing independent data samples: the nonparametric version of the Student's t-test.
- The Wilcoxon signed-rank test for comparing paired data samples: the nonparametric version of the paired Student's t-test.

- The Kruskal-Wallis H and Friedman tests for comparing more than two data samples: the nonparametric version of the ANOVA and repeated measures ANOVA tests.

28.9.5 Next

In the next section, you will discover how to calculate tests of independence for categorical variables.

Chapter 29

Independence Test

A common problem in applied machine learning is determining whether input features are relevant to the outcome to be predicted. This is the problem of feature selection. In the case of classification problems where input variables are also categorical, we can use statistical tests to determine whether the output variable is dependent or independent of the input variables. If independent, then the input variable is a candidate for a feature that may be irrelevant to the problem and removed from the dataset.

The Pearson's Chi-Squared statistical hypothesis is an example of a test for independence between categorical variables. In this tutorial, you will discover the Chi-Squared statistical hypothesis test for quantifying the independence of pairs of categorical variables. After completing this tutorial, you will know:

- Pairs of categorical variables can be summarized using a contingency table.
- The Chi-Squared test can compare an observed contingency table to an expected table and determine if the categorical variables are independent.
- How to calculate and interpret the Chi-Squared test for categorical variables in Python.

Let's get started.

29.1 Tutorial Overview

This tutorial is divided into 3 parts; they are:

1. Contingency Table
2. Pearson's Chi-Squared Test
3. Example Chi-Squared Test

29.2 Contingency Table

A categorical variable is a variable that may take on one of a set of labels. An example might be sex, which may be summarized as male or female. The variable is `sex` and the labels or

factors of the variable are `male` and `female` in this case. We may wish to look at a summary of a categorical variable as it pertains to another categorical variable. For example, sex and interest, where interest may have the labels `science`, `math`, or `art`. We can collect observations from people collected with regard to these two categorical variables; for example:

```
Sex,     Interest
Male,    Art
Female,  Math
Male,    Science
Male,    Math
...
```

Listing 29.1: Example of a small categorical dataset.

We can summarize the collected observations in a table with one variable corresponding to columns and another variable corresponding to rows. Each cell in the table corresponds to the count or frequency of observations that correspond to the row and column categories. Historically, a table summarization of two categorical variables in this form is called a contingency table. For example, the Sex=rows and Interest=columns table with contrived counts might look as follows:

```
       Science,   Math,  Art
Male       20,     30,   15
Female     20,     15,   30
```

Listing 29.2: Example of a contingency table.

The table was called a contingency table, by Karl Pearson, because the intent is to help determine whether one variable is contingent upon or depends upon the other variable. For example, does an interest in math or science depend on gender, or are they independent? This is challenging to determine from the table alone; instead, we can use a statistical method called the Pearson's Chi-Squared test.

29.3 Pearson's Chi-Squared Test

The Pearson's Chi-Squared test, or just Chi-Squared test for short, is named for Karl Pearson, although there are variations on the test. The Chi-Squared test is a statistical hypothesis test that assumes (the null hypothesis) that the observed frequencies for a categorical variable match the expected frequencies for the categorical variable. The test calculates a statistic that has a Chi-Squared distribution, named for the Greek lowercase letter chi (χ) pronounced "ki" as in "kite".

Given the `Sex/Interest` example above, the number of observations for a category (such as male and female) may or may not be the same. Nevertheless, we can calculate the expected frequency of observations in each `Interest` group and see whether the partitioning of interests by `Sex` results in similar or different frequencies. The Chi-Squared test does this for a contingency table, first calculating the expected frequencies for the groups, then determining whether the division of the groups, called the observed frequencies, matches the expected frequencies. The result of the test is a test statistic that has a Chi-Squared distribution and can be interpreted to reject or fail to reject the assumption or null hypothesis that the observed and expected frequencies are the same, that the variables are independent of each other.

> When observed frequency is far from the expected frequency, the corresponding term in the sum is large; when the two are close, this term is small. Large values of χ^2 indicate that observed and expected frequencies are far apart. Small values of χ^2 mean the opposite: observeds are close to expecteds. So χ^2 does give a measure of the distance between observed and expected frequencies.
>
> — Page 525, *Statistics*, Fourth Edition, 2007.

The variables are considered independent if the observed and expected frequencies are similar, that the levels of the variables do not interact, are not dependent.

> The chi-square test of independence works by comparing the categorically coded data that you have collected (known as the observed frequencies) with the frequencies that you would expect to get in each cell of a table by chance alone (known as the expected frequencies).
>
> — Page 162, *Statistics in Plain English*, Third Edition, 2010.

We can interpret the test statistic in the context of the Chi-Squared distribution with the requisite number of degrees of freedom as follows:

- `Test Statistic` $\geq$ `Critical Value`: significant result, reject null hypothesis, dependent (H1).
- `Test Statistic` $<$ `Critical Value`: not significant result, fail to reject null hypothesis, independent (H0).

The degrees of freedom for the Chi-Squared distribution is calculated based on the size of the contingency table as:

$$\text{degrees of freedom} = (rows - 1) \times (cols - 1) \tag{29.1}$$

In terms of a p-value and a chosen significance level (alpha), the test can be interpreted as follows:

- `p-value` $\leq$ `alpha`: significant result, reject null hypothesis, dependent (H1).
- `p-value` $>$ `alpha`: not significant result, fail to reject null hypothesis, independent (H0).

For the test to be effective, at least five observations are required in each cell of the contingency table. Next, let's look at how we can calculate the Chi-Squared test.

29.4 Example Chi-Squared Test

The Pearson's Chi-Squared test for independence can be calculated in Python using the `chi2_contingency()` SciPy function. The function takes an array as input representing the contingency table for the two categorical variables. It returns the calculated statistic and p-value for interpretation as well as the calculated degrees of freedom and table of expected frequencies.

```
...
# calculate Chi-Squared test
stat, p, dof, expected = chi2_contingency(table)
```

Listing 29.3: Example of calculating the Chi-Squared test.

We can interpret the statistic by retrieving the critical value from the Chi-Squared distribution for the probability and number of degrees of freedom. For example, a probability of 95% can be used, suggesting that the finding of the test is quite likely given the assumption of the test that the variables are independent. If the statistic is less than or equal to the critical value, we can fail to reject this assumption, otherwise it can be rejected.

```
...
# interpret test-statistic
prob = 0.95
critical = chi2.ppf(prob, dof)
if abs(stat) >= critical:
  print('Dependent (reject H0)')
else:
  print('Independent (fail to reject H0)')
```

Listing 29.4: Example of interpreting the Chi-Squared test statistic.

We can also interpret the p-value by comparing it to a chosen significance level, which would be 5%, calculated by inverting the 95% probability used in the critical value interpretation.

```
...
# interpret p-value
alpha = 1.0 - prob
if p <= alpha:
  print('Dependent (reject H0)')
else:
  print('Independent (fail to reject H0)')
```

Listing 29.5: Example of interpreting the Chi-Squared p-value.

We can tie all of this together and demonstrate the Chi-Squared significance test using a contrived contingency table. A contingency table is defined below that has a different number of observations for each population (row), but a similar proportion across each group (column). Given the similar proportions, we would expect the test to find that the groups are similar and that the variables are independent (fail to reject the null hypothesis, or H0).

```
...
# contingency table
table = [ [10, 20, 30],
      [6,  9,  17]]
```

Listing 29.6: Example of a contingency table.

The complete example is listed below.

```
# chi-squared test with similar proportions
from scipy.stats import chi2_contingency
from scipy.stats import chi2
# contingency table
table = [ [10, 20, 30],
```

```
     [6,  9,  17]]
print(table)
stat, p, dof, expected = chi2_contingency(table)
print('dof=%d' % dof)
print(expected)
# interpret test-statistic
prob = 0.95
critical = chi2.ppf(prob, dof)
print('probability=%.3f, critical=%.3f, stat=%.3f' % (prob, critical, stat))
if abs(stat) >= critical:
  print('Dependent (reject H0)')
else:
  print('Independent (fail to reject H0)')
# interpret p-value
alpha = 1.0 - prob
print('significance=%.3f, p=%.3f' % (alpha, p))
if p <= alpha:
  print('Dependent (reject H0)')
else:
  print('Independent (fail to reject H0)')
```

Listing 29.7: Example of calculating the Chi-Squared test on the contingency table.

Running the example first prints the contingency table. The test is calculated and the degrees of freedom (`dof`) is reported as 2, which makes sense given:

$$\begin{aligned} dof &= (rows - 1) \times (cols - 1) \\ &= (2 - 1) \times (3 - 1) \\ &= 1 \times 2 \\ &= 2 \end{aligned} \tag{29.2}$$

Next, the calculated expected frequency table is printed and we can see that indeed the observed contingency table does appear to match via an eyeball check of the numbers. The critical value is calculated and interpreted, finding that indeed the variables are independent (fail to reject H0). The interpretation of the p-value makes the same finding.

```
[[10, 20, 30], [6, 9, 17]]

dof=2

[[10.43478261 18.91304348 30.65217391]
 [ 5.56521739 10.08695652 16.34782609]]

probability=0.950, critical=5.991, stat=0.272
Independent (fail to reject H0)

significance=0.050, p=0.873
Independent (fail to reject H0)
```

Listing 29.8: Example output from calculating the Chi-Squared test on the contingency table.

29.5 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Update the Chi-Squared test to use your own contingency table.
- Write a function to report on the independence given observations from two categorical variables
- Load a standard machine learning dataset containing categorical variables and report on the independence of each.

If you explore any of these extensions, I'd love to know.

29.6 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

29.6.1 Books

- *Statistics in Plain English*, Third Edition, 2010.
  http://amzn.to/2IFyS4P
- *Statistics*, Fourth Edition, 2007.
  http://amzn.to/2u44zll

29.6.2 API

- `scipy.stats.chisquare` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chisquare.html
- `scipy.stats.chi2_contingency` API.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chi2_contingency.html
- `sklearn.feature_selection.chi2` API.
  http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html

29.6.3 Articles

- Chi-Squared test on Wikipedia.
  https://en.wikipedia.org/wiki/Chi-squared_test
- Pearson's Chi-Squared test on Wikipedia.
  https://en.wikipedia.org/wiki/Pearson%27s_chi-squared_test
- Contingency table on Wikipedia.
  https://en.wikipedia.org/wiki/Contingency_table
- How is chi test used for feature selection in machine learning? on Quora.
  https://www.quora.com/How-is-chi-test-used-for-feature-selection-in-machine-learni

29.7 Summary

In this tutorial, you discovered the Chi-Squared statistical hypothesis test for quantifying the independence of pairs of categorical variables. Specifically, you learned:

- Pairs of categorical variables can be summarized using a contingency table.
- The Chi-Squared test can compare an observed contingency table to an expected table and determine if the categorical variables are independent.
- How to calculate and interpret the Chi-Squared test for categorical variables in Python.

29.7.1 Next

This is the end of part VII, in the next part you will discover additional resources to help you on your journey with statistical methods.

Part VIII

Appendix

Appendix A

Getting Help

This is just the beginning of your journey with statistics. As you start to work on projects and expand your existing knowledge of the techniques, you may need help. This appendix points out some of the best sources of help.

A.1 Statistics on Wikipedia

Wikipedia is a great place to start. All of the important topics are covered, the descriptions are concise, and the equations are consistent and readable. What is missing is the more human level descriptions such as analogies and intuitions. Nevertheless, when you have questions about statistics, I recommend stopping by Wikipedia first. Some good high-level pages to start on include:

- Statistics.
  https://en.wikipedia.org/wiki/Statistics
- Descriptive statistics.
  https://en.wikipedia.org/wiki/Descriptive_statistics
- Statistical inference.
  https://en.wikipedia.org/wiki/Statistical_inference
- Glossary of probability and statistics.
  https://en.wikipedia.org/wiki/Glossary_of_probability_and_statistics
- List of statistics articles.
  https://en.wikipedia.org/wiki/List_of_statistics_articles

A.2 Statistics Textbooks

I strongly recommend getting a good textbook on the topic of statistics and using it as a reference. The benefit of a good textbook is that the explanations of the various operations you require will be consistent (or should be). The downside of textbooks is that they can be very expensive. A good textbook is often easy to spot because it will be the basis for a range of undergraduate or postgraduate courses at top universities. Some introductory textbooks on statistics I recommend include:

- *Statistics in Plain English*, 2016.
  http://amzn.to/2t3Nt6q
- *All of Statistics: A Concise Course in Statistical Inference*, 2004.
  http://amzn.to/2tbJJQC
- *Practical Statistics for Data Scientists: 50 Essential Concepts*, 2017.
  http://amzn.to/2F6WT6L

A good book on the application of statistics for experiments in artificial intelligence is:

- *Empirical Methods for Artificial Intelligence*, 1995.
  http://amzn.to/2GTFRWi

I'd also recommend reading popular science books on statistics. They provide good context and intuitions for the methods. Some recommendations include:

- *Naked Statistics: Stripping the Dread from the Data*, 2014.
  http://amzn.to/2t7yXdV
- *The Drunkard's Walk: How Randomness Rules Our Lives*, 2009.
  http://amzn.to/2CT4rUD
- *The Signal and the Noise: Why So Many Predictions Fail - but Some Don't*, 2015.
  http://amzn.to/2FQdYyX

A.3 Python API Resources

You may need help with NumPy, SciPy, Statsmodels or Matplotlib when implementing statistics in Python. The API documentation is good, below are a few resources that you can use to use specific Python APIs.

- NumPy Statistical functions.
  https://docs.scipy.org/doc/numpy/reference/routines.statistics.html
- SciPy Statistical functions (scipy.stats).
  https://docs.scipy.org/doc/scipy/reference/stats.html
- Statsmodels API.
  https://www.statsmodels.org/stable/index.html
- Matplotlib API.
  https://matplotlib.org/api/index.html
- Scikit-Learn API.
  http://scikit-learn.org/stable/modules/classes.html

A.4 Ask Questions About Statistics

There are a lot of places that you can ask questions about statistics online given the current abundance of question-and-answer platforms. Below is a list of the top places I recommend posting a question. Remember to search for your question before posting in case it has been asked and answered before.

- Statistics tag on the Mathematics Stack Exchange.
  https://math.stackexchange.com/?tags=statistics
- Cross Validated.
  https://stats.stackexchange.com
- Statistics tag on Stack Overflow.
  https://stackoverflow.com/questions/tagged/statistics
- Statistics on Quora.
  https://www.quora.com/topic/Statistics-academic-discipline
- Statistics Subreddit.
  https://www.reddit.com/r/statistics/

A.5 How to Ask Questions

Knowing where to get help is the first step, but you need to know how to get the most out of these resources. Below are some tips that you can use:

- Boil your question down to the simplest form. E.g. not something broad like *my model does not work* or *how does x work*.
- Search for answers before asking questions.
- Provide complete code and error messages.
- Boil your code down to the smallest possible working example that demonstrates the issue.

A.6 Contact the Author

You are not alone. If you ever have any questions about deep learning, natural language processing, or this book, please contact me directly. I will do my best to help.

Jason Brownlee
Jason@MachineLearningMastery.com

Appendix B

How to Setup a Workstation for Python

It can be difficult to install a Python machine learning environment on some platforms. Python itself must be installed first and then there are many packages to install, and it can be confusing for beginners. In this tutorial, you will discover how to setup a Python machine learning development environment using Anaconda.

After completing this tutorial, you will have a working Python environment to begin learning, practicing, and developing machine learning software. These instructions are suitable for Windows, Mac OS X, and Linux platforms. I will demonstrate them on OS X, so you may see some mac dialogs and file extensions.

B.1 Overview

In this tutorial, we will cover the following steps:

1. Download Anaconda
2. Install Anaconda
3. Start and Update Anaconda

Note: The specific versions may differ as the software and libraries are updated frequently.

B.2 Download Anaconda

In this step, we will download the Anaconda Python package for your platform. Anaconda is a free and easy-to-use environment for scientific Python.

- 1. Visit the Anaconda homepage.
  https://www.continuum.io/
- 2. Click `Anaconda` from the menu and click `Download` to go to the download page.
  https://www.continuum.io/downloads

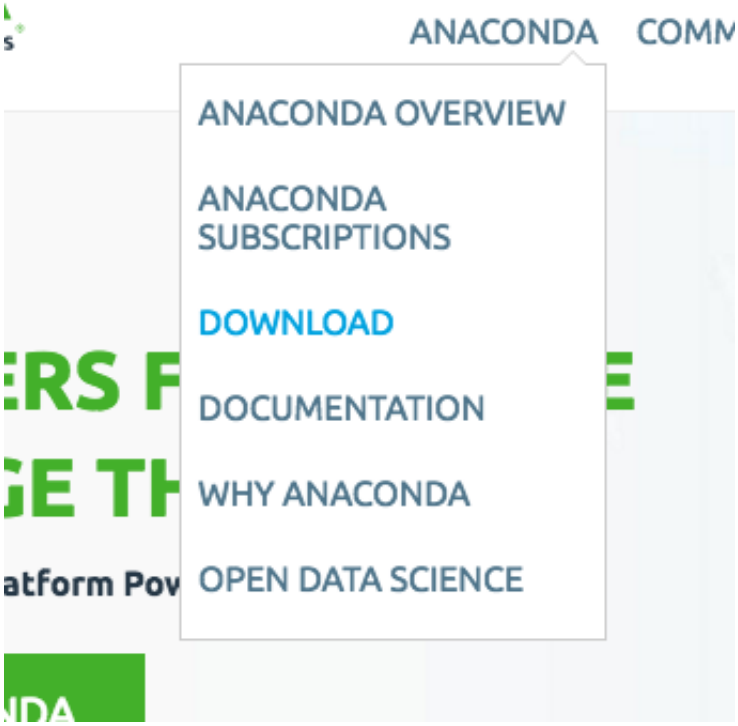

Figure B.1: Click Anaconda and Download.

- 3. Choose the download suitable for your platform (Windows, OSX, or Linux):
  - Choose Python 3.6
  - Choose the Graphical Installer

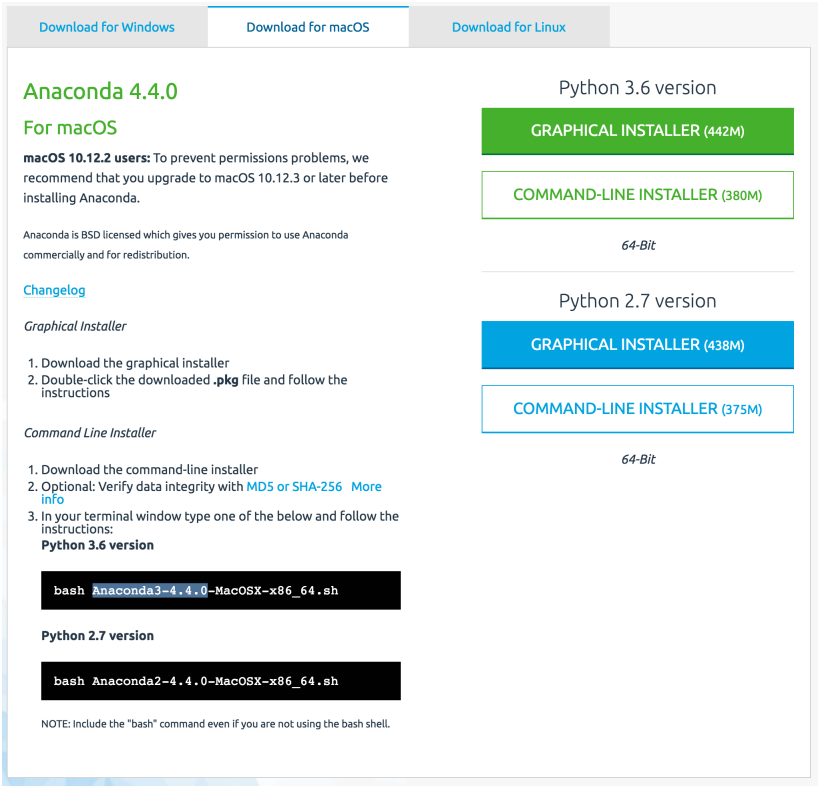

Figure B.2: Choose Anaconda Download for Your Platform.

This will download the Anaconda Python package to your workstation. I'm on OS X, so I chose the OS X version. The file is about 426 MB. You should have a file with a name like:

```
Anaconda3-4.4.0-MacOSX-x86_64.pkg
```

Listing B.1: Example filename on Mac OS X.

B.3 Install Anaconda

In this step, we will install the Anaconda Python software on your system. This step assumes you have sufficient administrative privileges to install software on your system.

- 1. Double click the downloaded file.
- 2. Follow the installation wizard.

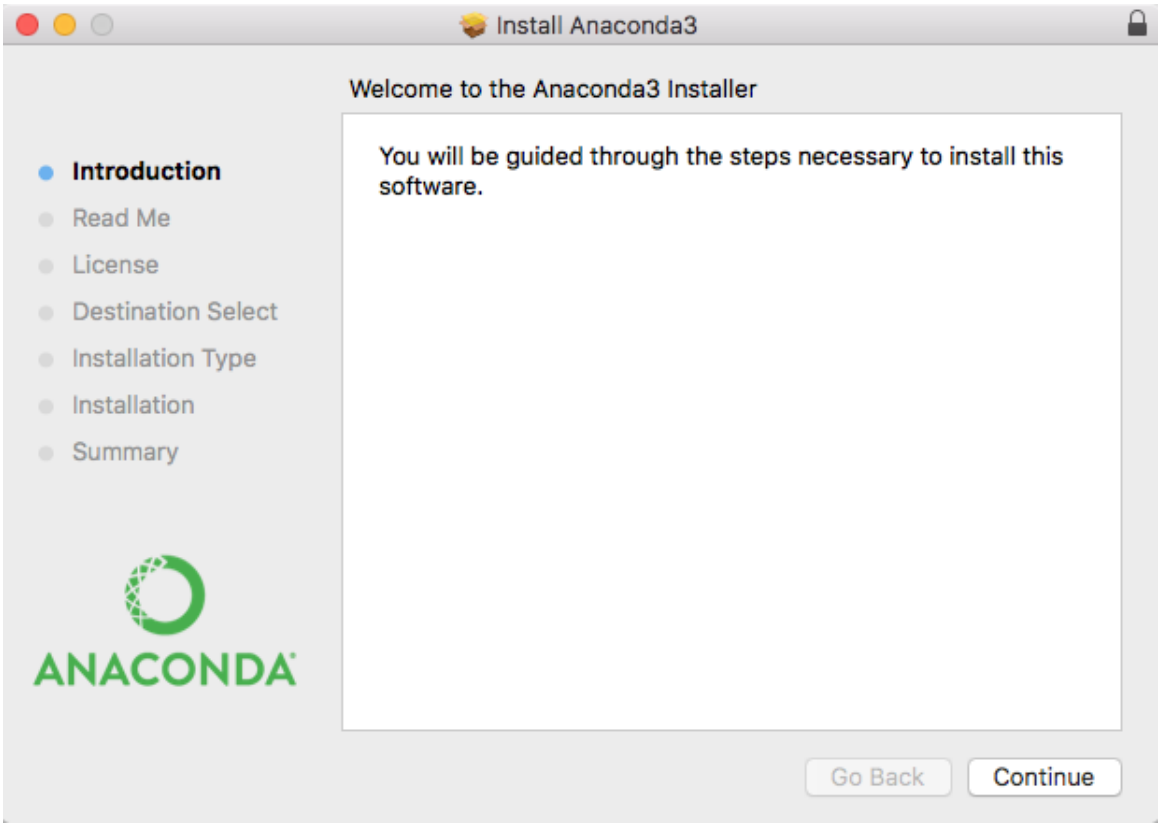

Figure B.3: Anaconda Python Installation Wizard.

Installation is quick and painless. There should be no tricky questions or sticking points.

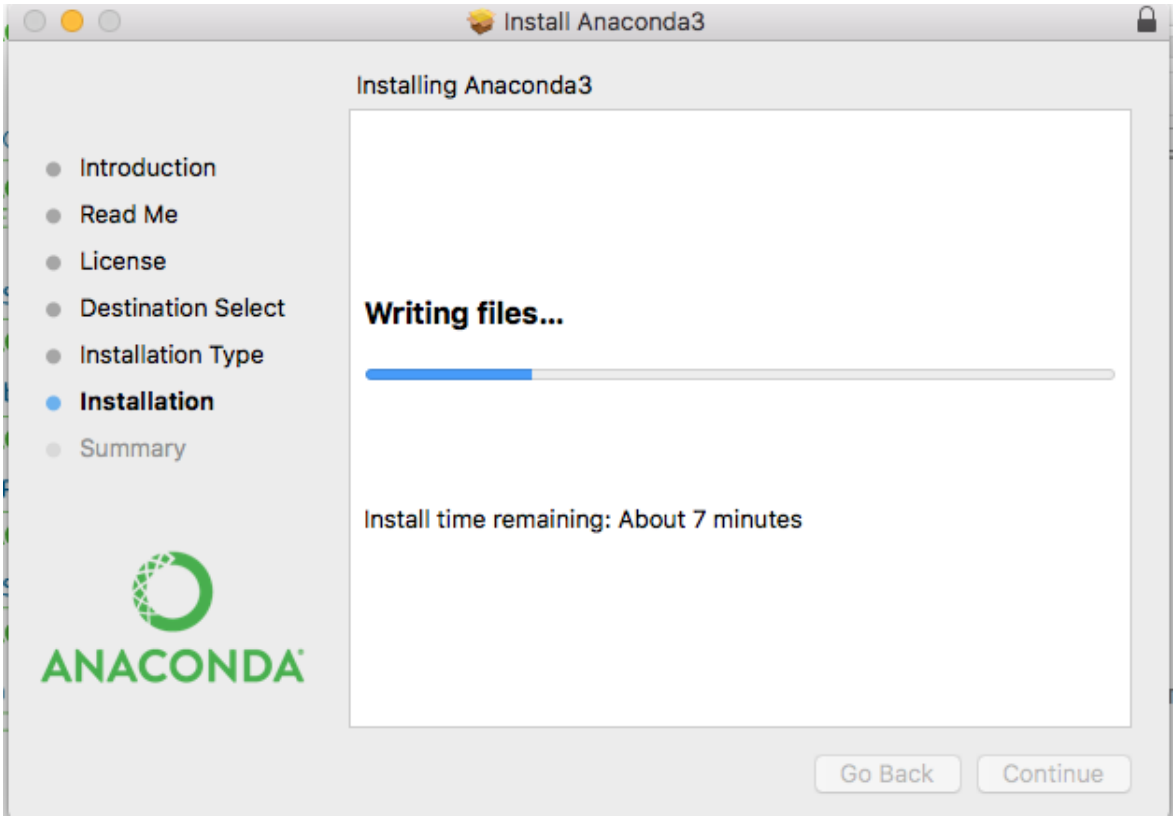

Figure B.4: Anaconda Python Installation Wizard Writing Files.

The installation should take less than 10 minutes and take up a little more than 1 GB of space on your hard drive.

B.4 Start and Update Anaconda

In this step, we will confirm that your Anaconda Python environment is up to date. Anaconda comes with a suite of graphical tools called Anaconda Navigator. You can start Anaconda Navigator by opening it from your application launcher.

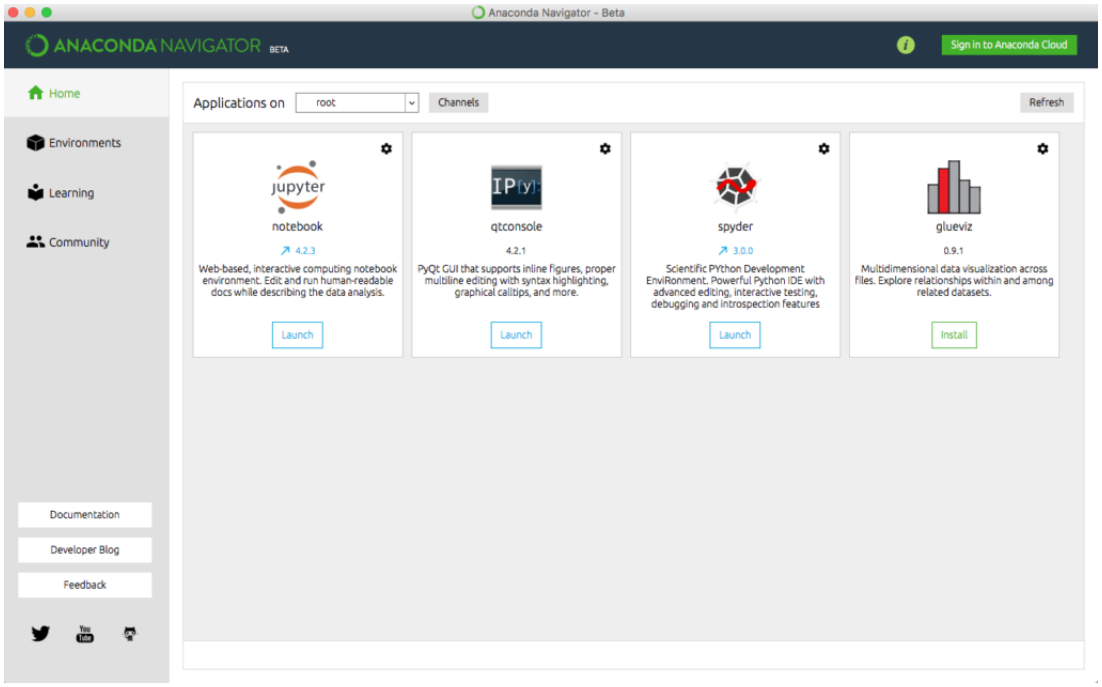

Figure B.5: Anaconda Navigator GUI.

You can use the Anaconda Navigator and graphical development environments later; for now, I recommend starting with the Anaconda command line environment called conda. Conda is fast, simple, it's hard for error messages to hide, and you can quickly confirm your environment is installed and working correctly.

- 1. Open a terminal (command line window).
- 2. Confirm conda is installed correctly, by typing:

```
conda -V
```

Listing B.2: Check the conda version.

You should see the following (or something similar):

```
conda 4.3.21
```

Listing B.3: Example conda version.

- 3. Confirm Python is installed correctly by typing:

```
python -V
```

Listing B.4: Check the Python version.

You should see the following (or something similar):

```
Python 3.6.1 :: Anaconda 4.4.0 (x86_64)
```

Listing B.5: Example Python version.

If the commands do not work or have an error, please check the documentation for help for your platform. See some of the resources in the *Further Reading* section.

- 4. Confirm your conda environment is up-to-date, type:

```
conda update conda
conda update anaconda
```

Listing B.6: Update conda and anaconda.

You may need to install some packages and confirm the updates.

- 5. Confirm your SciPy environment.

The script below will print the version number of the key SciPy libraries you require for machine learning development, specifically: SciPy, NumPy, Matplotlib, Pandas, Statsmodels, and Scikit-learn. You can type `python` and type the commands in directly. Alternatively, I recommend opening a text editor and copy-pasting the script into your editor.

```
# check library version numbers
# scipy
import scipy
print('scipy: %s' % scipy.__version__)
# numpy
import numpy
print('numpy: %s' % numpy.__version__)
# matplotlib
import matplotlib
print('matplotlib: %s' % matplotlib.__version__)
# pandas
import pandas
print('pandas: %s' % pandas.__version__)
# statsmodels
import statsmodels
print('statsmodels: %s' % statsmodels.__version__)
# scikit-learn
import sklearn
print('sklearn: %s' % sklearn.__version__)
```

Listing B.7: Code to check that key Python libraries are installed.

Save the script as a file with the name: `versions.py`. On the command line, change your directory to where you saved the script and type:

```
python versions.py
```

Listing B.8: Run the script from the command line.

You should see output like the following:

```
scipy: 1.0.1
numpy: 1.14.2
matplotlib: 2.2.2
pandas: 0.22.0
statsmodels: 0.8.0
sklearn: 0.19.1
```

Listing B.9: Sample output of versions script.

B.5 Further Reading

This section provides resources if you want to know more about Anaconda.

- Anaconda homepage.
  https://www.continuum.io/
- Anaconda Navigator.
  https://docs.continuum.io/anaconda/navigator.html
- The conda command line tool.
  http://conda.pydata.org/docs/index.html

B.6 Summary

Congratulations, you now have a working Python development environment for machine learning. You can now learn and practice machine learning on your workstation.

Appendix C

Basic Math Notation

You cannot avoid mathematical notation when reading the descriptions of machine learning methods. Often, all it takes is one term or one fragment of notation in an equation to completely derail your understanding of the entire procedure. This can be extremely frustrating, especially for machine learning beginners coming from the world of development. You can make great progress if you know a few basic areas of mathematical notation and some tricks for working through the description of machine learning methods in papers and books. In this tutorial, you will discover the basics of mathematical notation that you may come across when reading descriptions of techniques in machine learning. After completing this tutorial, you will know:

- Notation for arithmetic including variations of multiplication, exponents, roots and logarithms.
- Notation for sequences and sets including indexing, summation and set membership.
- 5 Techniques you can use to get help if you are struggling with mathematical notation.

Let's get started.

C.1 Tutorial Overview

This tutorial is divided into 7 parts; they are:

1. The Frustration with Math Notation
2. Arithmetic Notation
3. Greek Alphabet
4. Sequence Notation
5. Set Notation
6. Other Notation
7. Tips for Getting More Help

C.2 The Frustration with Math Notation

You will encounter mathematical notation when reading about machine learning algorithms. For example, notation may be used to:

- Describe an algorithm.
- Describe data preparation.
- Describe results.
- Describe a test harness.
- Describe implications.

These descriptions may be in research papers, textbooks, blog posts and elsewhere. Often the terms are well defined, but there are also mathematical notation norms that you may not be familiar with. All it takes is one term or one equation that you do not understand and your understanding of the entire method will be lost. I've suffered this problem myself many times and it is incredibly frustrating! In this tutorial we will review some basic mathematical notation that will help you when reading descriptions of machine learning methods.

C.3 Arithmetic Notation

In this section we will go over some less obvious notations for basic arithmetic as well as a few concepts you may have forgotten since school.

C.3.1 Simple Arithmetic

The notation for basic arithmetic is as you would write it. For example:

- Addition: $1 + 1 = 2$
- Subtraction: $2 - 1 = 1$
- Multiplication: $2 \times 2 = 4$
- Division: $\frac{2}{2} = 1$

Most mathematical operations have a sister operation that performs the inverse operation, for example subtraction is the inverse of addition and division is the inverse of multiplication.

C.3.2 Algebra

We often want to describe operations abstractly to separate them from specific data or specific implementations. For this reason we see heavy use of algebra, that is uppercase and/or lowercase letters or words to represents terms or concepts in mathematical notation. It is also common to use letters from the Greek alphabet. Each sub-field of math may have reserved letters, that is terms or letters that always mean the same thing. Nevertheless, algebraic terms should be defined as part of the description and if they are not, it may just be a poor description, not your fault.

C.3.3 Multiplication Notation

Multiplication is a common notation and has a few short hands. Often a little "x" ($\times$) or an asterisk "*" is used to represent multiplication:

$$c = a \times b \tag{C.1}$$

Or

$$c = a * b \tag{C.2}$$

You may see a dot notation used, for example:

$$c = a \cdot b \tag{C.3}$$

Alternately, you may see no operation and no white space separation between previously defined terms, for example:

$$c = ab \tag{C.4}$$

Which again is the same thing.

C.3.4 Exponents and Square Roots

An exponent is a number raised to a power. The notation is written as the original number or the base with a second number or the exponent shown as a superscript, for example:

$$2^3 \tag{C.5}$$

Which would be calculated as 2 multiplied by itself 3 times or cubing:

$$2 \times 2 \times 2 = 8 \tag{C.6}$$

A number raised to the power 2 to is said to be it's square.

$$2^2 = 2 \times 2 = 4 \tag{C.7}$$

The square of a number can be inverted by calculating the square root. This is shown using the notation of a number and with a tick above $\sqrt{x}$.

$$\sqrt{4} = 2 \tag{C.8}$$

Here, we know the result and the exponent and we wish to find the base. In fact, the root operation can be used to inverse any exponent, it just so happens that the default square root assumes an exponent of 2, represented by a subscript 2 in front of the square root tick. For example, we can invert the cubing of a number by taking the cube root:

$$2^3 = 8 \tag{C.9}$$

$$\sqrt[3]{8} = 2 \tag{C.10}$$

C.3.5 Logarithms and e

When we raise 10 to an integer exponent we often call this an order of magnitude.

$$10^2 = 10 \times 10 \tag{C.11}$$

Another way to reverse this operation is by calculating the logarithm of the result 100 assuming a base of 10, in notation this is written as `log10()`.

$$log10(100) = 2 \tag{C.12}$$

Here, we know the result and the base and wish to find the exponent. This allows us to move up and down orders of magnitude very easily. Taking the logarithm assuming the base of 2 is also commonly used, given the use of binary arithmetic used in computers. For example:

$$2^6 = 64 \tag{C.13}$$

$$log2(64) = 6 \tag{C.14}$$

Another popular logarithm is to assume the natural base called e. The e is reserved and is a special number or a constant called Euler's number (pronounced *oy-ler*) that refers to a value with practically infinite precision.

$$e = 2.71828\cdots \tag{C.15}$$

Raising e to a power is called a natural exponential function:

$$e^2 = 7.38905\cdots \tag{C.16}$$

It can be inverted using the natural logarithm which is denoted as `ln()`:

$$ln(7.38905\cdots) = 2 \tag{C.17}$$

Without going into detail, the natural exponent and natural logarithm prove useful throughout mathematics to abstractly describe the continuous growth of some systems, e.g. systems that grow exponentially such as compound interest.

C.4 Greek Alphabet

Greek letters are used throughout mathematical notation for variables, constants, functions and more. For example in statistics we talk about the mean using the lowercase Greek letter mu (μ), and the standard deviation as the lowercase Greek letter sigma (σ). In linear regression we talk about the coefficients as the lowercase letter beta (β). And so on. It is useful to know all of the uppercase and lowercase Greek letters and how to pronounce them. When I was a grad student, I printed the Greek alphabet and stuck it on my computer monitor so that I could memorize it. A useful trick! Below is the full Greek alphabet.

Greek letters															
Name	TeX	HTML	Name	TeX	HTML	Name	TeX	HTML	Name	TeX	HTML	Name	TeX	HTML
Alpha	$A\,\alpha$	Α α	Digamma	$F\,\digamma$	Ϝ ϝ	Kappa	$K\,\kappa\,\varkappa$	Κ κ ϰ	Omicron	$O\,o$	Ο ο	Upsilon	$\Upsilon\,\upsilon$	Υ υ
Beta	$B\,\beta$	Β β	Zeta	$Z\,\zeta$	Ζ ζ	Lambda	$\Lambda\,\lambda$	Λ λ	Pi	$\Pi\,\pi\,\varpi$	Π π ϖ	Phi	$\Phi\,\phi\,\varphi$	Φ ϕ φ
Gamma	$\Gamma\,\gamma$	Γ γ	Eta	$H\,\eta$	Η η	Mu	$M\,\mu$	Μ μ	Rho	$P\,\rho\,\varrho$	Ρ ρ ϱ	Chi	$X\,\chi$	Χ χ
Delta	$\Delta\,\delta$	Δ δ	Theta	$\Theta\,\theta\,\vartheta$	Θ θ ϑ	Nu	$N\,\nu$	Ν ν	Sigma	$\Sigma\,\sigma\,\varsigma$	Σ σ ς	Psi	$\Psi\,\psi$	Ψ ψ
Epsilon	$E\,\epsilon\,\varepsilon$	Ε ε ε	Iota	$I\,\iota$	Ι ι	Xi	$\Xi\,\xi$	Ξ ξ	Tau	$T\,\tau$	Τ τ	Omega	$\Omega\,\omega$	Ω ω

Figure C.1: Greek Alphabet, Taken from Wikipedia.

The Wikipedia page titled *Greek letters used in mathematics, science, and engineering*[1] is also a useful guide as it lists common uses for each Greek letter in different sub-fields of math and science.

C.5 Sequence Notation

Machine learning notation often describes an operation on a sequence. A sequence may be an array of data or a list of terms.

C.5.1 Indexing

A key to reading notation for sequences is the notation of indexing elements in the sequence. Often the notation will specify the beginning and end of the sequence, such as 1 to n, where n will be the extent or length of the sequence. Items in the sequence are index by a variable such as i, j, k as a subscript. This is just like array notation. For example a_i is the i^{th} element of the sequence a. If the sequence is two dimensional, two indices may be used, for example: $b_{i,j}$ is the $(i,j)^{th}$ element of the sequence b.

C.5.2 Sequence Operations

Mathematical operations can be performed over a sequence. Two operations are performed on sequences so often that they have their own shorthand, the sum and the multiplication.

Sequence Summation

The sum over a sequence is denoted as the uppercase Greek letter sigma (Σ). It is specified with the variable and start of the sequence summation below the sigma (e.g. $i = 1$) and the index of the end of the summation above the sigma (e.g. n).

$$\sum_{i=1}^{n} a_i \tag{C.18}$$

This is the sum of the sequence a starting at element 1 to element n.

[1] https://en.wikipedia.org/wiki/Greek_letters_used_in_mathematics,_science,_and_engineering

Sequence Multiplication

The multiplication over a sequence is denoted as the uppercase Greek letter pi (Π). It is specified in the same way as the sequence summation with the beginning and end of the operation below and above the letter respectively.

$$\prod_{i=1}^{n} a_i \tag{C.19}$$

This is the product of the sequence a starting at element 1 to element n.

C.6 Set Notation

A set is a group of unique items. We may see set notation used when defining terms in machine learning.

C.6.1 Set of Numbers

A common set you may see is a set of numbers, such as a term defined as being within the set of integers or the set of real numbers. Some common sets of numbers you may see include:

- Set of all natural numbers: $\mathbb{N}$
- Set of all integers: $\mathbb{Z}$
- Set of all real numbers: $\mathbb{R}$

There are other sets, see *Special sets on Wikipedia*[2]. We often talk about real-values or real numbers when defining terms rather than floating point values, which are really discrete creations for operations in computers.

C.6.2 Set Membership

It is common to see set membership in definitions of terms. Set membership is denoted as a symbol that looks like an uppercase "E" ($\in$).

$$a \in \mathbb{R} \tag{C.20}$$

Which means a is defined as being a member of the set $\mathbb{R}$ or the set of real numbers. There is also a host of set operations, two common set operations include:

- Union, or aggregation: $A \cup B$
- Intersection, or overlap: $A \cap B$

Learn more about sets on Wikipedia[3].

[2] https://en.wikipedia.org/wiki/Set_(mathematics)#Special_sets
[3] https://en.wikipedia.org/wiki/Set_(mathematics)

C.7 Other Notation

There is other notation that you may come across. I try to lay some of it out in this section. It is common to define a method in the abstract and then define it again as a specific implementation with separate notation. For example, if we are estimating a variable x we may represent it using a notation that modifies the x, for example:

- x-bar ($\bar{x}$)
- x-prime ($\grave{x}$)
- x-hat ($\hat{x}$)
- x-tilde ($\tilde{x}$)

The same notation may have different meaning in a different context, such as use on different objects or sub-fields of mathematics. For example, a common point of confusion is $|x|$, which, depending on context can mean:

- $|x|$: The absolute or positive value of x.
- $|x|$: The length of the vector x.
- $|x|$: The cardinality of the set x.

This tutorial only covered the basics of mathematical notation. There are some subfields of mathematics that are more relevant to machine learning and should be reviewed in more detail. They are:

- Linear Algebra.
- Statistics.
- Probability.
- Calculus.

And perhaps a little bit of multivariate analysis and information theory.

C.8 Tips for Getting More Help

This section lists some tips that you can use when you are struggling with mathematical notation in machine learning.

C.8.1 Think About the Author

People wrote the paper or book you are reading. People that can make mistakes, make omissions, and even make things confusing because they don't fully understand what they are writing. Relax the constraints of the notation you are reading slightly and think about the intent of the author. What are they trying to get across? Perhaps you can even contact the author via email, Twitter, Facebook, Linked-in, etc, and seek clarification. Remember that academics want other people to understand and use their work (mostly).

C.8.2 Check Wikipedia

Wikipedia has lists of notation which can help narrow down on the meaning or intent of the notation you are reading. Two places I recommend you start are:

- List of mathematical symbols on Wikipedia.
  https://en.wikipedia.org/wiki/List_of_mathematical_symbols
- Greek letters used in mathematics, science, and engineering on Wikipedia.
  https://en.wikipedia.org/wiki/Greek_letters_used_in_mathematics,_science,_and_engineering

C.8.3 Sketch in Code

Mathematical operations are just functions on data. Map everything you're reading to pseudocode with variables, for-loops and more. You might want to use a scripting language as you go along with small arrays of contrived data or even an Excel spreadsheet. As your reading and understanding of the technique improves, your code-sketch of the technique will make more sense and at the end you will have a mini prototype to play with.

I never used to take much stock in this approach until I saw an academic sketch out a very complex paper in a few lines of Matlab with some contrived data. It knocked my socks off because I believed the system had to be coded completely and run with a *real* dataset and that the only option was to get the original code and data. I was very wrong. Also, looking back, the guy was gifted. I now use this method all the time and sketch techniques in Python.

C.8.4 Seek Alternatives

There is a trick I use when I'm trying to understand a new technique. I find and read all the papers that reference the paper I'm reading with the new technique. Reading other academics interpretation and re-explanation of the technique can often clarify my misunderstandings in the original description. Not always though. Sometimes it can muddy the waters and introduce misleading explanations or new notation. But more often than not, it helps. After circling back to the original paper and re-reading it, I can often find cases where subsequent papers have actually made errors and misinterpretations of the original method.

C.8.5 Post a Question

There are places online where people love to explain math to others. Seriously! Consider taking a screen shot of the notation you are struggling with, write out the full reference or link to it and put it and your area of misunderstanding to a question and answer site. Two great places to start are:

- Mathematics Stack Exchange.
  https://math.stackexchange.com/
- Cross Validated.
  https://stats.stackexchange.com/

C.9 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

- Section 0.1. Reading Mathematics, *Vector Calculus, Linear Algebra, and Differential Forms*, 2009.
  http://amzn.to/2qarp8L
- The Language and Grammar of Mathematics, Timothy Gowers.
  http://assets.press.princeton.edu/chapters/gowers/gowers_I_2.pdf
- Understanding Mathematics, a guide, Peter Alfeld.
  http://www.math.utah.edu/~pa/math.html

C.10 Summary

In this tutorial, you discovered the basics of mathematical notation that you may come across when reading descriptions of techniques in machine learning. Specifically, you learned:

- Notation for arithmetic including variations of multiplication, exponents, roots and logarithms.
- Notation for sequences and sets including indexing, summation and set membership.
- 5 Techniques you can use to get help if you are struggling with mathematical notation.

Part IX

Conclusions

How Far You Have Come

You made it. Well done. Take a moment and look back at how far you have come. You now know:

- About the field of statistics, how it relates to machine learning, and how to harness statistical methods on a machine learning project.
- How to calculate and interpret common summary statistics and how to present data using standard data visualization techniques.
- Findings from mathematical statistics that underlie much of the field, such as the central limit theorem and the law of large numbers.
- How to evaluate and interpret the relationship between variables and the independence of variables.
- How to calculate and interpret parametric statistical hypothesis tests for comparing two or more data samples.
- How to calculate and interpret interval statistics for distributions, population parameters, and observations.
- How to use statistical resampling to make good economic use of available data in order to evaluate predictive models.
- How to calculate and interpret nonparametric statistical hypothesis tests for comparing two or more data samples that do not conform to the expectations of parametric tests.

Don't make light of this. You have come a long way in a short amount of time. You have developed the important and valuable foundational skills in statistical methods. You can now confidently:

- Use descriptive statistics and data visualizations to quickly and more deeply understand the shape and relationships in data.
- Use inferential statistical tests to quickly and effectively quantify the relationships between samples, such as the results of experiments with different predictive algorithms or differing configurations.
- Use estimation statistics to quickly and effectively quantify the confidence in estimated model skill and model predictions.

The sky's the limit.

Thank You!

I want to take a moment and sincerely thank you for letting me help you start your journey with statistical methods. I hope you keep learning and have fun as you continue to master machine learning.

Jason Brownlee
2018