Machine Learning Mastery with Python

Understand Your Data, Create Accurate Models and Work Projects End-To-End

Jason Brownlee

Jason Brownlee

Machine Learning Mastery With Python

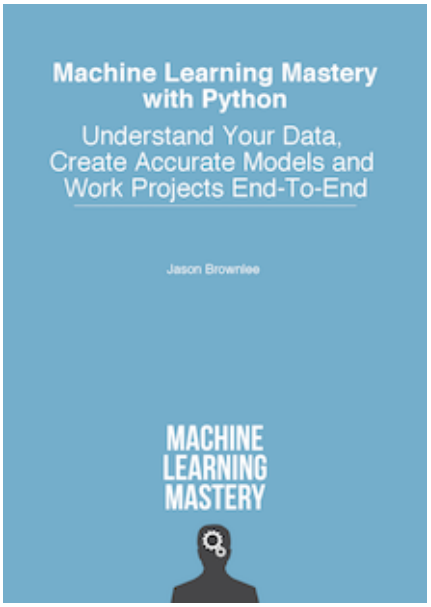
Understand Your Data, Create Accurate Models and Work Projects End-To-End

Machine Learning Mastery With Python

Edition: v1.5

This is Just a Sample

Thank-you for your interest in **Machine Learning Mastery With Python**.
This is just a sample of the full text. You can purchase the complete book online from:
https://machinelearningmastery.com/machine-learning-with-python/



Contents

Chapter 1

Welcome

Welcome to *Machine Learning Mastery With Python*. This book is your guide to applied machine learning with Python. You will discover the step-by-step process that you can use to get started and become good at machine learning for predictive modeling with the Python ecosystem.

1.1 Learn Python Machine Learning The Wrong Way

Here is what you should NOT do when you start studying machine learning in Python.

1. Get really good at Python programming and Python syntax.
2. Deeply study the underlying theory and parameters for machine learning algorithms in scikit-learn.
3. Avoid or lightly touch on all of the other tasks needed to complete a real project.

I think that this approach can work for some people, but it is a really slow and a roundabout way of getting to your goal. It teaches you that you need to spend all your time learning how to use individual machine learning algorithms. It also does not teach you the process of building predictive machine learning models in Python that you can actually use to make predictions. Sadly, this is the approach used to teach machine learning that I see in almost all books and online courses on the topic.

1.2 Machine Learning in Python

This book focuses on a specific sub-field of machine learning called predictive modeling. This is the field of machine learning that is the most useful in industry and the type of machine learning that the scikit-learn library in Python excels at facilitating. Unlike statistics, where models are used to *understand* data, predictive modeling is laser focused on developing models that make the *most accurate predictions* at the expense of explaining why predictions are made. Unlike the broader field of machine learning that could feasibly be used with data in any format, predictive modeling is primarily focused on tabular data (e.g. tables of numbers like in a spreadsheet).

This book was written around three themes designed to get you started and using Python for applied machine learning effectively and quickly. These three parts are as follows:

Lessons : Learn how the sub-tasks of a machine learning project map onto Python and the best practice way of working through each task.

Projects : Tie together all of the knowledge from the lessons by working through case study predictive modeling problems.

Recipes : Apply machine learning with a catalog of standalone recipes in Python that you can copy-and-paste as a starting point for new projects.

1.2.1 Lessons

You need to know how to complete the specific subtasks of a machine learning project using the Python ecosystem. Once you know how to complete a discrete task using the platform and get a result reliably, you can do it again and again on project after project. Let's start with an overview of the common tasks in a machine learning project. A predictive modeling machine learning project can be broken down into 6 top-level tasks:

1. **Define Problem**: Investigate and characterize the problem in order to better understand the goals of the project.
2. **Analyze Data**: Use descriptive statistics and visualization to better understand the data you have available.
3. **Prepare Data**: Use data transforms in order to better expose the structure of the prediction problem to modeling algorithms.
4. **Evaluate Algorithms**: Design a test harness to evaluate a number of standard algorithms on the data and select the top few to investigate further.
5. **Improve Results**: Use algorithm tuning and ensemble methods to get the most out of well-performing algorithms on your data.
6. **Present Results**: Finalize the model, make predictions and present results.

A blessing and a curse with Python is that there are so many techniques and so many ways to do the same thing with the platform. In part II of this book you will discover one easy or best practice way to complete each subtask of a general machine learning project. Below is a summary of the Lessons from Part II and the sub-tasks that you will learn about.

- Lesson 1: Python Ecosystem for Machine Learning.
- Lesson 2: Python and SciPy Crash Course.
- Lesson 3: Load Datasets from CSV.
- Lesson 4: Understand Data With Descriptive Statistics. (**Analyze Data**)
- Lesson 5: Understand Data With Visualization. (**Analyze Data**)
- Lesson 6: Pre-Process Data. (**Prepare Data**)

- Lesson 7: Feature Selection. (**Prepare Data**)
- Lesson 8: Resampling Methods. (**Evaluate Algorithms**)
- Lesson 9: Algorithm Evaluation Metrics. (**Evaluate Algorithms**)
- Lesson 10: Spot-Check Classification Algorithms. (**Evaluate Algorithms**)
- Lesson 11: Spot-Check Regression Algorithms. (**Evaluate Algorithms**)
- Lesson 12: Model Selection. (**Evaluate Algorithms**)
- Lesson 13: Pipelines. (**Evaluate Algorithms**)
- Lesson 14: Ensemble Methods. (**Improve Results**)
- Lesson 15: Algorithm Parameter Tuning. (**Improve Results**)
- Lesson 16: Model Finalization. (**Present Results**)

These lessons are intended to be read from beginning to end in order, showing you exactly how to complete each task in a predictive modeling machine learning project. Of course, you can dip into specific lessons again later to refresh yourself. Lessons are structured to demonstrate key API classes and functions, showing you how to use specific techniques for a common machine learning task. Each lesson was designed to be completed in under 30 minutes (depending on your level of skill and enthusiasm). It is possible to work through the entire book in one weekend. It also works if you want to dip into specific sections and use the book as a reference.

1.2.2 Projects

Recipes for common predictive modeling tasks are critically important, but they are also just the starting point. This is where most books and courses stop.

You need to piece the recipes together into end-to-end projects. This will show you how to actually deliver a model or make predictions on new data using Python. This book uses small well-understood machine learning datasets from the UCI Machine learning repository[1] in both the lessons and in the example projects. These datasets are available for free as CSV downloads. These datasets are excellent for practicing applied machine learning because:

- **They are small**, meaning they fit into memory and algorithms can model them in reasonable time.
- **They are well behaved**, meaning you often don't need to do a lot of feature engineering to get a good result.
- **They are benchmarks**, meaning that many people have used them before and you can get ideas of good algorithms to try and accuracy levels you should expect.

In Part III you will work through three projects:

[1] http://archive.ics.uci.edu/ml

Hello World Project (Iris flowers dataset) : This is a quick pass through the project steps without much tuning or optimizing on a dataset that is widely used as the *hello world* of machine learning.

Regression (Boston House Price dataset) : Work through each step of the project process with a regression problem.

Binary Classification (Sonar dataset) : Work through each step of the project process using all of the methods on a binary classification problem.

These projects unify all of the lessons from Part II. They also give you insight into the process for working through predictive modeling machine learning problems which is invaluable when you are trying to get a feeling for how to do this in practice. Also included in this section is a template for working through predictive modeling machine learning problems which you can use as a starting point for current and future projects. I find this useful myself to set the direction and setup important tasks (which are easy to forget) on new projects.

1.2.3 Recipes

Recipes are small standalone examples in Python that show you how to do one specific thing and get a result. For example, you could have a recipe that demonstrates how to use the Random Forest algorithm for classification. You could have another for normalizing the attributes of a dataset.

Recipes make the difference between a beginner who is having trouble and a fast learner capable of making accurate predictions quickly on any new project. A catalog of recipes provides a repertoire of skills that you can draw from when starting a new project. More formally, recipes are defined as follows:

- Recipes are code snippets not tutorials.
- Recipes provide just enough code to work.
- Recipes are demonstrative not exhaustive.
- Recipes run as-is and produce a result.
- Recipes assume that required libraries are installed.
- Recipes use built-in datasets or datasets provided in specific libraries.

You are starting your journey into machine learning with Python with a catalog of machine learning recipes used throughout this book. All of the code from the lessons in Part II and projects in Part III are available in your Python recipe catalog. Recipes are organized by chapter so that you can quickly locate a specific example used in the book. This is an valuable resource that you can use to jump-start your current and future machine learning projects. You can also build upon this recipe catalog as you discover new techniques.

1.2.4 Your Outcomes From Reading This Book

This book will lead you from being a developer who is interested in machine learning with Python to a developer who has the resources and capability to work through a new dataset end-to-end using Python and develop accurate predictive models. Specifically, you will know:

- How to work through a small to medium sized dataset end-to-end.
- How to deliver a model that can make accurate predictions on new unseen data.
- How to complete all subtasks of a predictive modeling problem with Python.
- How to learn new and different techniques in Python and SciPy.
- How to get help with Python machine learning.

From here you can start to dive into the specifics of the functions, techniques and algorithms used with the goal of learning how to use them better in order to deliver more accurate predictive models, more reliably in less time.

1.3 What This Book is Not

This book was written for professional developers who want to know how to build reliable and accurate machine learning models in Python.

- **This is not a machine learning textbook**. We will not be getting into the basic theory of machine learning (e.g. induction, bias-variance trade-off, etc.). You are expected to have some familiarity with machine learning basics, or be able to pick them up yourself.
- **This is not an algorithm book**. We will not be working through the details of how specific machine learning algorithms work (e.g. Random Forests). You are expected to have some basic knowledge of machine learning algorithms or how to pick up this knowledge yourself.
- **This is not a Python programming book**. We will not be spending a lot of time on Python syntax and programming (e.g. basic programming tasks in Python). You are expected to be a developer who can pick up a new C-like language relatively quickly.

You can still get a lot out of this book if you are weak in one or two of these areas, but you may struggle picking up the language or require some more explanation of the techniques. If this is the case, see the *Getting More Help* chapter at the end of the book and seek out a good companion reference text.

1.4 Summary

I hope you are as excited as me to get started. In this introduction chapter you learned that this book is unconventional. Unlike other books and courses that focus heavily on machine learning algorithms in Python and focus on little else, this book will walk you through each step of a predictive modeling machine learning project.

- Part II of this book provides standalone lessons including a mixture of recipes and tutorials to build up your basic working skills and confidence in Python.
- Part III of this book will introduce a machine learning project template that you can use as a starting point on your own projects and walks you through three end-to-end projects.
- The recipes companion to this book provides a catalog of machine learning code in Python. You can browse this invaluable resource, find useful recipes and copy-and-paste them into your current and future machine learning projects.
- Part IV will finish out the book. It will look back at how far you have come in developing your new found skills in applied machine learning with Python. You will also discover resources that you can use to get help if and when you have any questions about Python or the ecosystem.

1.4.1 Next Step

Next you will start Part II and your first lesson. You will take a closer look at the Python ecosystem for machine learning. You will discover what Python and SciPy are, why it is so powerful as a platform for machine learning and the different ways you should and should not use the platform.

Chapter 2

Understand Your Data With Descriptive Statistics

You must understand your data in order to get the best results. In this chapter you will discover 7 recipes that you can use in Python to better understand your machine learning data. After reading this lesson you will know how to:

1. Take a peek at your raw data.
2. Review the dimensions of your dataset.
3. Review the data types of attributes in your data.
4. Summarize the distribution of instances across classes in your dataset.
5. Summarize your data using descriptive statistics.
6. Understand the relationships in your data using correlations.
7. Review the skew of the distributions of each attribute.

Each recipe is demonstrated by loading the Pima Indians Diabetes classification dataset from the UCI Machine Learning repository. Open your Python interactive environment and try each recipe out in turn. Let's get started.

2.1 Peek at Your Data

There is no substitute for looking at the raw data. Looking at the raw data can reveal insights that you cannot get any other way. It can also plant seeds that may later grow into ideas on how to better pre-process and handle the data for machine learning tasks. You can review the first 20 rows of your data using the `head()` function on the Pandas DataFrame.

```
# View first 20 rows
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
peek = data.head(20)
```

```
print(peek)
```

Listing 2.1: Example of reviewing the first few rows of data.

You can see that the first column lists the row number, which is handy for referencing a specific observation.

```
    preg  plas  pres  skin  test  mass   pedi  age  class
0      6   148    72    35     0  33.6  0.627   50      1
1      1    85    66    29     0  26.6  0.351   31      0
2      8   183    64     0     0  23.3  0.672   32      1
3      1    89    66    23    94  28.1  0.167   21      0
4      0   137    40    35   168  43.1  2.288   33      1
5      5   116    74     0     0  25.6  0.201   30      0
6      3    78    50    32    88  31.0  0.248   26      1
7     10   115     0     0     0  35.3  0.134   29      0
8      2   197    70    45   543  30.5  0.158   53      1
9      8   125    96     0     0   0.0  0.232   54      1
10     4   110    92     0     0  37.6  0.191   30      0
11    10   168    74     0     0  38.0  0.537   34      1
12    10   139    80     0     0  27.1  1.441   57      0
13     1   189    60    23   846  30.1  0.398   59      1
14     5   166    72    19   175  25.8  0.587   51      1
15     7   100     0     0     0  30.0  0.484   32      1
16     0   118    84    47   230  45.8  0.551   31      1
17     7   107    74     0     0  29.6  0.254   31      1
18     1   103    30    38    83  43.3  0.183   33      0
19     1   115    70    30    96  34.6  0.529   32      1
```

Listing 2.2: Output of reviewing the first few rows of data.

2.2 Dimensions of Your Data

You must have a very good handle on how much data you have, both in terms of rows and columns.

- Too many rows and algorithms may take too long to train. Too few and perhaps you do not have enough data to train the algorithms.
- Too many features and some algorithms can be distracted or suffer poor performance due to the curse of dimensionality.

You can review the shape and size of your dataset by printing the `shape` property on the Pandas DataFrame.

```
# Dimensions of your data
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
shape = data.shape
print(shape)
```

Listing 2.3: Example of reviewing the shape of the data.

The results are listed in rows then columns. You can see that the dataset has 768 rows and 9 columns.

```
(768, 9)
```

Listing 2.4: Output of reviewing the shape of the data.

2.3 Data Type For Each Attribute

The type of each attribute is important. Strings may need to be converted to floating point values or integers to represent categorical or ordinal values. You can get an idea of the types of attributes by peeking at the raw data, as above. You can also list the data types used by the DataFrame to characterize each attribute using the `dtypes` property.

```
# Data Types for Each Attribute
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
types = data.dtypes
print(types)
```

Listing 2.5: Example of reviewing the data types of the data.

You can see that most of the attributes are integers and that `mass` and `pedi` are floating point types.

```
preg      int64
plas      int64
pres      int64
skin      int64
test      int64
mass    float64
pedi    float64
age       int64
class     int64
dtype: object
```

Listing 2.6: Output of reviewing the data types of the data.

2.4 Descriptive Statistics

Descriptive statistics can give you great insight into the shape of each attribute. Often you can create more summaries than you have time to review. The `describe()` function on the Pandas DataFrame lists 8 statistical properties of each attribute. They are:

- Count.
- Mean.
- Standard Deviation.

- Minimum Value.
- 25th Percentile.
- 50th Percentile (Median).
- 75th Percentile.
- Maximum Value.

```
# Statistical Summary
from pandas import read_csv
from pandas import set_option
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
set_option('display.width', 100)
set_option('precision', 3)
description = data.describe()
print(description)
```

Listing 2.7: Example of reviewing a statistical summary of the data.

You can see that you do get a lot of data. You will note some calls to `pandas.set_option()` in the recipe to change the precision of the numbers and the preferred width of the output. This is to make it more readable for this example. When describing your data this way, it is worth taking some time and reviewing observations from the results. This might include the presence of `NA` values for missing data or surprising distributions for attributes.

```
        preg    plas    pres    skin    test    mass    pedi     age   class
count 768.000 768.000 768.000 768.000 768.000 768.000 768.000 768.000 768.000
mean    3.845 120.895  69.105  20.536  79.799  31.993   0.472  33.241   0.349
std     3.370  31.973  19.356  15.952 115.244   7.884   0.331  11.760   0.477
min     0.000   0.000   0.000   0.000   0.000   0.000   0.078  21.000   0.000
25%     1.000  99.000  62.000   0.000   0.000  27.300   0.244  24.000   0.000
50%     3.000 117.000  72.000  23.000  30.500  32.000   0.372  29.000   0.000
75%     6.000 140.250  80.000  32.000 127.250  36.600   0.626  41.000   1.000
max    17.000 199.000 122.000  99.000 846.000  67.100   2.420  81.000   1.000
```

Listing 2.8: Output of reviewing a statistical summary of the data.

2.5 Class Distribution (Classification Only)

On classification problems you need to know how balanced the class values are. Highly imbalanced problems (a lot more observations for one class than another) are common and may need special handling in the data preparation stage of your project. You can quickly get an idea of the distribution of the class attribute in Pandas.

```
# Class Distribution
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
```

```
class_counts = data.groupby('class').size()
print(class_counts)
```

Listing 2.9: Example of reviewing a class breakdown of the data.

You can see that there are nearly double the number of observations with class 0 (no onset of diabetes) than there are with class 1 (onset of diabetes).

```
class
0    500
1    268
```

Listing 2.10: Output of reviewing a class breakdown of the data.

2.6 Correlations Between Attributes

Correlation refers to the relationship between two variables and how they may or may not change together. The most common method for calculating correlation is Pearson's Correlation Coefficient, that assumes a normal distribution of the attributes involved. A correlation of -1 or 1 shows a full negative or positive correlation respectively. Whereas a value of 0 shows no correlation at all. Some machine learning algorithms like linear and logistic regression can suffer poor performance if there are highly correlated attributes in your dataset. As such, it is a good idea to review all of the pairwise correlations of the attributes in your dataset. You can use the `corr()` function on the Pandas DataFrame to calculate a correlation matrix.

```
# Pairwise Pearson correlations
from pandas import read_csv
from pandas import set_option
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
set_option('display.width', 100)
set_option('precision', 3)
correlations = data.corr(method='pearson')
print(correlations)
```

Listing 2.11: Example of reviewing correlations of attributes in the data.

The matrix lists all attributes across the top and down the side, to give correlation between all pairs of attributes (twice, because the matrix is symmetrical). You can see the diagonal line through the matrix from the top left to bottom right corners of the matrix shows perfect correlation of each attribute with itself.

```
        preg   plas   pres   skin   test   mass   pedi    age  class
preg   1.000  0.129  0.141 -0.082 -0.074  0.018 -0.034  0.544  0.222
plas   0.129  1.000  0.153  0.057  0.331  0.221  0.137  0.264  0.467
pres   0.141  0.153  1.000  0.207  0.089  0.282  0.041  0.240  0.065
skin  -0.082  0.057  0.207  1.000  0.437  0.393  0.184 -0.114  0.075
test  -0.074  0.331  0.089  0.437  1.000  0.198  0.185 -0.042  0.131
mass   0.018  0.221  0.282  0.393  0.198  1.000  0.141  0.036  0.293
pedi  -0.034  0.137  0.041  0.184  0.185  0.141  1.000  0.034  0.174
age    0.544  0.264  0.240 -0.114 -0.042  0.036  0.034  1.000  0.238
class  0.222  0.467  0.065  0.075  0.131  0.293  0.174  0.238  1.000
```

Listing 2.12: Output of reviewing correlations of attributes in the data.

2.7 Skew of Univariate Distributions

Skew refers to a distribution that is assumed Gaussian (normal or bell curve) that is shifted or squashed in one direction or another. Many machine learning algorithms assume a Gaussian distribution. Knowing that an attribute has a skew may allow you to perform data preparation to correct the skew and later improve the accuracy of your models. You can calculate the skew of each attribute using the `skew()` function on the Pandas DataFrame.

```
# Skew for each attribute
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
skew = data.skew()
print(skew)
```

Listing 2.13: Example of reviewing skew of attribute distributions in the data.

The skew result show a positive (right) or negative (left) skew. Values closer to zero show less skew.

```
preg     0.901674
plas     0.173754
pres    -1.843608
skin     0.109372
test     2.272251
mass    -0.428982
pedi     1.919911
age      1.129597
class    0.635017
```

Listing 2.14: Output of reviewing skew of attribute distributions in the data.

2.8 Tips To Remember

This section gives you some tips to remember when reviewing your data using summary statistics.

- **Review the numbers**. Generating the summary statistics is not enough. Take a moment to pause, read and really think about the numbers you are seeing.
- **Ask why**. Review your numbers and ask a lot of questions. How and why are you seeing specific numbers. Think about how the numbers relate to the problem domain in general and specific entities that observations relate to.
- **Write down ideas**. Write down your observations and ideas. Keep a small text file or note pad and jot down all of the ideas for how variables may relate, for what numbers mean, and ideas for techniques to try later. The things you write down now while the data is fresh will be very valuable later when you are trying to think up new things to try.

2.9 Summary

In this chapter you discovered the importance of describing your dataset before you start work on your machine learning project. You discovered 7 different ways to summarize your dataset using Python and Pandas:

- Peek At Your Data.
- Dimensions of Your Data.
- Data Types.
- Class Distribution.
- Data Summary.
- Correlations.
- Skewness.

2.9.1 Next

Another excellent way that you can use to better understand your data is by generating plots and charts. In the next lesson you will discover how you can visualize your data for machine learning in Python.

Chapter 3

Prepare Your Data For Machine Learning

Many machine learning algorithms make assumptions about your data. It is often a very good idea to prepare your data in such a way to best expose the structure of the problem to the machine learning algorithms that you intend to use. In this chapter you will discover how to prepare your data for machine learning in Python using scikit-learn. After completing this lesson you will know how to:

1. Rescale data.
2. Standardize data.
3. Normalize data.
4. Binarize data.

Let's get started.

3.1 Need For Data Pre-processing

You almost always need to pre-process your data. It is a required step. A difficulty is that different algorithms make different assumptions about your data and may require different transforms. Further, when you follow all of the rules and prepare your data, sometimes algorithms can deliver better results without pre-processing.

Generally, I would recommend creating many different views and transforms of your data, then exercise a handful of algorithms on each view of your dataset. This will help you to flush out which data transforms might be better at exposing the structure of your problem in general.

3.2 Data Transforms

In this lesson you will work through 4 different data pre-processing recipes for machine learning. The Pima Indian diabetes dataset is used in each recipe. Each recipe follows the same structure:

- Load the dataset from a URL.

- Split the dataset into the input and output variables for machine learning.
- Apply a pre-processing transform to the input variables.
- Summarize the data to show the change.

The scikit-learn library provides two standard idioms for transforming data. Each are useful in different circumstances. The transforms are calculated in such a way that they can be applied to your training data and any samples of data you may have in the future. The scikit-learn documentation has some information on how to use various different pre-processing methods:

- Fit and Multiple Transform.
- Combined Fit-And-Transform.

The Fit and Multiple Transform method is the preferred approach. You call the `fit()` function to prepare the parameters of the transform once on your data. Then later you can use the `transform()` function on the same data to prepare it for modeling and again on the test or validation dataset or new data that you may see in the future. The Combined Fit-And-Transform is a convenience that you can use for one off tasks. This might be useful if you are interested in plotting or summarizing the transformed data. You can review the `preprocess` API in scikit-learn here[1].

3.3 Rescale Data

When your data is comprised of attributes with varying scales, many machine learning algorithms can benefit from rescaling the attributes to all have the same scale. Often this is referred to as normalization and attributes are often rescaled into the range between 0 and 1. This is useful for optimization algorithms used in the core of machine learning algorithms like gradient descent. It is also useful for algorithms that weight inputs like regression and neural networks and algorithms that use distance measures like k-Nearest Neighbors. You can rescale your data using scikit-learn using the `MinMaxScaler` class[2].

```
# Rescale data (between 0 and 1)
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import MinMaxScaler
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX = scaler.fit_transform(X)
# summarize transformed data
set_printoptions(precision=3)
```

[1] http://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing
[2] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

```
print(rescaledX[0:5,:])
```

Listing 3.1: Example of rescaling data.

After rescaling you can see that all of the values are in the range between 0 and 1.

```
[[ 0.353 0.744 0.59  0.354 0.    0.501 0.234 0.483]
 [ 0.059 0.427 0.541 0.293 0.    0.396 0.117 0.167]
 [ 0.471 0.92  0.525 0.    0.    0.347 0.254 0.183]
 [ 0.059 0.447 0.541 0.232 0.111 0.419 0.038 0.   ]
 [ 0.    0.688 0.328 0.354 0.199 0.642 0.944 0.2 ]]
```

Listing 3.2: Output of rescaling data.

3.4 Standardize Data

Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. It is most suitable for techniques that assume a Gaussian distribution in the input variables and work better with rescaled data, such as linear regression, logistic regression and linear discriminate analysis. You can standardize data using scikit-learn with the `StandardScaler` class[3].

```
# Standardize data (0 mean, 1 stdev)
from sklearn.preprocessing import StandardScaler
from pandas import read_csv
from numpy import set_printoptions
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

Listing 3.3: Example of standardizing data.

The values for each attribute now have a mean value of 0 and a standard deviation of 1.

```
[[ 0.64  0.848  0.15   0.907 -0.693 0.204 0.468  1.426]
 [-0.845 -1.123 -0.161 0.531 -0.693 -0.684 -0.365 -0.191]
 [ 1.234 1.944 -0.264 -1.288 -0.693 -1.103 0.604 -0.106]
 [-0.845 -0.998 -0.161 0.155 0.123 -0.494 -0.921 -1.042]
 [-1.142 0.504 -1.505 0.907 0.766  1.41   5.485 -0.02 ]]
```

Listing 3.4: Output of rescaling data.

[3] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

3.5 Normalize Data

Normalizing in scikit-learn refers to rescaling each observation (row) to have a length of 1 (called a unit norm or a vector with the length of 1 in linear algebra). This pre-processing method can be useful for sparse datasets (lots of zeros) with attributes of varying scales when using algorithms that weight input values such as neural networks and algorithms that use distance measures such as k-Nearest Neighbors. You can normalize data in Python with scikit-learn using the `Normalizer` class[4].

```
# Normalize data (length of 1)
from sklearn.preprocessing import Normalizer
from pandas import read_csv
from numpy import set_printoptions
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = Normalizer().fit(X)
normalizedX = scaler.transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(normalizedX[0:5,:])
```

Listing 3.5: Example of normalizing data.

The rows are normalized to length 1.

```
[[ 0.034 0.828 0.403 0.196 0.    0.188 0.004 0.28 ]
 [ 0.008 0.716 0.556 0.244 0.    0.224 0.003 0.261]
 [ 0.04  0.924 0.323 0.    0.    0.118 0.003 0.162]
 [ 0.007 0.588 0.436 0.152 0.622 0.186 0.001 0.139]
 [ 0.    0.596 0.174 0.152 0.731 0.188 0.01  0.144]]
```

Listing 3.6: Output of normalizing data.

3.6 Binarize Data (Make Binary)

You can transform your data using a binary threshold. All values above the threshold are marked 1 and all equal to or below are marked as 0. This is called *binarizing* your data or *thresholding* your data. It can be useful when you have probabilities that you want to make crisp values. It is also useful when feature engineering and you want to add new features that indicate something meaningful. You can create new binary attributes in Python using scikit-learn with the `Binarizer` class[5].

```
# binarization
from sklearn.preprocessing import Binarizer
from pandas import read_csv
```

[4] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html
[5] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Binarizer.html

```
from numpy import set_printoptions
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
binarizer = Binarizer(threshold=0.0).fit(X)
binaryX = binarizer.transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(binaryX[0:5,:])
```

Listing 3.7: Example of binarizing data.

You can see that all values equal or less than 0 are marked 0 and all of those above 0 are marked 1.

```
[[ 1.  1.  1.  1.  0.  1.  1.  1.]
 [ 1.  1.  1.  1.  0.  1.  1.  1.]
 [ 1.  1.  1.  0.  0.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.]
 [ 0.  1.  1.  1.  1.  1.  1.  1.]]
```

Listing 3.8: Output of normalizing data.

3.7 Summary

In this chapter you discovered how you can prepare your data for machine learning in Python using scikit-learn. You now have recipes to:

- Rescale data.
- Standardize data.
- Normalize data.
- Binarize data.

3.7.1 Next

You now know how to transform your data to best expose the structure of your problem to the modeling algorithms. In the next lesson you will discover how to select the features of your data that are most relevant to making predictions.

This is Just a Sample

Thank-you for your interest in **Machine Learning Mastery With Python**.
This is just a sample of the full text. You can purchase the complete book online from:
https://machinelearningmastery.com/machine-learning-with-python/