Chapter 2

Python Ecosystem for Machine Learning

The Python ecosystem is growing and may become the dominant platform for machine learning. The primary rationale for adopting Python for machine learning is because it is a general purpose programming language that you can use both for R&D and in production. In this chapter you will discover the Python ecosystem for machine learning. After completing this lesson you will know:

1. Python and it's rising use for machine learning.
2. SciPy and the functionality it provides with NumPy, Matplotlib and Pandas.
3. scikit-learn that provides all of the machine learning algorithms.
4. How to setup your Python ecosystem for machine learning and what versions to use

Let's get started.

2.1 Python

Python is a general purpose interpreted programming language. It is easy to learn and use primarily because the language focuses on readability. The philosophy of Python is captured in the Zen of Python which includes phrases like:

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
```

Listing 2.1: Sample of the Zen of Python.

It is a popular language in general, consistently appearing in the top 10 programming languages in surveys on StackOverflow[1]. It's a dynamic language and very suited to interactive

[1] http://stackoverflow.com/research/developer-survey-2015

development and quick prototyping with the power to support the development of large applications. It is also widely used for machine learning and data science because of the excellent library support and because it is a general purpose programming language (unlike R or Matlab). For example, see the results of the Kaggle platform survey results in 2011[2] and the KDD Nuggets 2015 tool survey results[3].

This is a simple and very important consideration. It means that you can perform your research and development (figuring out what models to use) in the same programming language that you use for your production systems. Greatly simplifying the transition from development to production.

2.2 SciPy

SciPy is an ecosystem of Python libraries for mathematics, science and engineering. It is an add-on to Python that you will need for machine learning. The SciPy ecosystem is comprised of the following core modules relevant to machine learning:

- **NumPy**: A foundation for SciPy that allows you to efficiently work with data in arrays.
- **Matplotlib**: Allows you to create 2D charts and plots from data.
- **Pandas**: Tools and data structures to organize and analyze your data.

To be effective at machine learning in Python you must install and become familiar with SciPy. Specifically:

- You will prepare your data as NumPy arrays for modeling in machine learning algorithms.
- You will use Matplotlib (and wrappers of Matplotlib in other frameworks) to create plots and charts of your data.
- You will use Pandas to load explore and better understand your data.

2.3 scikit-learn

The scikit-learn library is how you can develop and practice machine learning in Python. It is built upon and requires the SciPy ecosystem. The name *scikit* suggests that it is a SciPy plug-in or toolkit. The focus of the library is machine learning algorithms for classification, regression, clustering and more. It also provides tools for related tasks such as evaluating models, tuning parameters and pre-processing data.

Like Python and SciPy, scikit-learn is open source and is usable commercially under the BSD license. This means that you can learn about machine learning, develop models and put them into operations all with the same ecosystem and code. A powerful reason to use scikit-learn.

[2] http://blog.kaggle.com/2011/11/27/kagglers-favorite-tools/

[3] http://www.kdnuggets.com/polls/2015/analytics-data-mining-data-science-software-used.html

2.4 Python Ecosystem Installation

There are multiple ways to install the Python ecosystem for machine learning. In this section we cover how to install the Python ecosystem for machine learning.

2.4.1 How To Install Python

The first step is to install Python. I prefer to use and recommend Python 2.7. The instructions for installing Python will be specific to your platform. For instructions see *Downloading Python*[4] in the *Python Beginners Guide*. Once installed you can confirm the installation was successful. Open a command line and type:

```
python --version
```

Listing 2.2: Print the version of Python installed.

You should see a response like the following:

```
Python 2.7.11
```

Listing 2.3: Example Python version.

The examples in this book assume that you are using this version of Python 2 or newer. The examples in this book have not been tested with Python 3.

2.4.2 How To Install SciPy

There are many ways to install SciPy. For example two popular ways are to use package management on your platform (e.g. yum on RedHat or macports on OS X) or use a Python package management tool like pip. The SciPy documentation is excellent and covers how-to instructions for many different platforms on the page *Installing the SciPy Stack*[5]. When installing SciPy, ensure that you install the following packages as a minimum:

- scipy
- numpy
- matplotlib
- pandas

Once installed, you can confirm that the installation was successful. Open the Python interactive environment by typing `python` at the command line, then type in and run the following Python code to print the versions of the installed libraries.

```
# scipy
import scipy
print('scipy: {}'.format(scipy.__version__))
# numpy
import numpy
print('numpy: {}'.format(numpy.__version__))
```

[4] https://wiki.python.org/moin/BeginnersGuide/Download
[5] http://scipy.org/install.html

```
# matplotlib
import matplotlib
print('matplotlib: {}'.format(matplotlib.__version__))
# pandas
import pandas
print('pandas: {}'.format(pandas.__version__))
```

Listing 2.4: Print the versions of the SciPy stack.

On my workstation at the time of writing I see the following output.

```
scipy: 0.18.1
numpy: 1.11.2
matplotlib: 1.5.1
pandas: 0.18.0
```

Listing 2.5: Example versions of the SciPy stack.

The examples in this book assume you have these version of the SciPy libraries or newer. If you have an error, you may need to consult the documentation for your platform.

2.4.3 How To Install scikit-learn

I would suggest that you use the same method to install scikit-learn as you used to install SciPy. There are instructions for installing scikit-learn[6], but they are limited to using the Python `pip` and `conda` package managers. Like SciPy, you can confirm that scikit-learn was installed successfully. Start your Python interactive environment and type and run the following code.

```
# scikit-learn
import sklearn
print('sklearn: {}'.format(sklearn.__version__))
```

Listing 2.6: Print the version of scikit-learn.

It will print the version of the scikit-learn library installed. On my workstation at the time of writing I see the following output:

```
sklearn: 0.18
```

Listing 2.7: Example versions of scikit-learn.

The examples in this book assume you have this version of scikit-learn or newer.

2.4.4 How To Install The Ecosystem: An Easier Way

If you are not confident at installing software on your machine, there is an easier option for you. There is a distribution called Anaconda that you can download and install for free[7]. It supports the three main platforms of Microsoft Windows, Mac OS X and Linux. It includes Python, SciPy and scikit-learn. Everything you need to learn, practice and use machine learning with the Python Environment.

[6] http://scikit-learn.org/stable/install.html
[7] https://www.continuum.io/downloads

2.5 Summary

In this chapter you discovered the Python ecosystem for machine learning. You learned about:

- Python and it's rising use for machine learning.
- SciPy and the functionality it provides with NumPy, Matplotlib and Pandas.
- scikit-learn that provides all of the machine learning algorithms.

You also learned how to install the Python ecosystem for machine learning on your workstation.

2.5.1 Next

In the next lesson you will get a crash course in the Python and SciPy ecosystem, designed specifically to get a developer like you up to speed with ecosystem very fast.

Chapter 3

Crash Course in Python and SciPy

You do not need to be a Python developer to get started using the Python ecosystem for machine learning. As a developer who already knows how to program in one or more programming languages, you are able to pick up a new language like Python very quickly. You just need to know a few properties of the language to transfer what you already know to the new language. After completing this lesson you will know:

1. How to navigate Python language syntax.
2. Enough NumPy, Matplotlib and Pandas to read and write machine learning Python scripts.
3. A foundation from which to build a deeper understanding of machine learning tasks in Python.

If you already know a little Python, this chapter will be a friendly reminder for you. Let's get started.

3.1 Python Crash Course

When getting started in Python you need to know a few key details about the language syntax to be able to read and understand Python code. This includes:

- Assignment.
- Flow Control.
- Data Structures.
- Functions.

We will cover each of these topics in turn with small standalone examples that you can type and run. Remember, whitespace has meaning in Python.

3.1.1 Assignment

As a programmer, assignment and types should not be surprising to you.

Strings

```
# Strings
data = 'hello world'
print(data[0])
print(len(data))
print(data)
```

Listing 3.1: Example of working with strings.

Notice how you can access characters in the string using array syntax. Running the example prints:

```
h
11
hello world
```

Listing 3.2: Output of example working with strings.

Numbers

```
# Numbers
value = 123.1
print(value)
value = 10
print(value)
```

Listing 3.3: Example of working with numbers.

Running the example prints:

```
123.1
10
```

Listing 3.4: Output of example working with numbers.

Boolean

```
# Boolean
a = True
b = False
print(a, b)
```

Listing 3.5: Example of working with booleans.

Running the example prints:

```
(True, False)
```

Listing 3.6: Output of example working with booleans.

Multiple Assignment

```
# Multiple Assignment
a, b, c = 1, 2, 3
print(a, b, c)
```

Listing 3.7: Example of working with multiple assignment.

This can also be very handy for unpacking data in simple data structures. Running the example prints:

```
(1, 2, 3)
```

Listing 3.8: Output of example working with multiple assignment.

No Value

```
# No value
a = None
print(a)
```

Listing 3.9: Example of working with no value.

Running the example prints:

```
None
```

Listing 3.10: Output of example working with no value.

3.1.2 Flow Control

There are three main types of flow control that you need to learn: If-Then-Else conditions, For-Loops and While-Loops.

If-Then-Else Conditional

```
value = 99
if value == 99:
  print 'That is fast'
elif value > 200:
  print 'That is too fast'
else:
  print 'That is safe'
```

Listing 3.11: Example of working with an If-Then-Else conditional.

Notice the colon (:) at the end of the condition and the meaningful tab intend for the code block under the condition. Running the example prints:

```
If-Then-Else conditional
```

Listing 3.12: Output of example working with an If-Then-Else conditional.

For-Loop

```
# For-Loop
for i in range(10):
  print i
```

Listing 3.13: Example of working with a For-Loop.

Running the example prints:

```
0
1
2
3
4
5
6
7
8
9
```

Listing 3.14: Output of example working with a For-Loop.

While-Loop

```
# While-Loop
i = 0
while i < 10:
  print i
  i += 1
```

Listing 3.15: Example of working with a While-Loop.

Running the example prints:

```
0
1
2
3
4
5
6
7
8
9
```

Listing 3.16: Output of example working with a While-Loop.

3.1.3 Data Structures

There are three data structures in Python that you will find the most used and useful. They are tuples, lists and dictionaries.

Tuple

Tuples are read-only collections of items.

```
a = (1, 2, 3)
print a
```

Listing 3.17: Example of working with a Tuple.

Running the example prints:

```
(1, 2, 3)
```

Listing 3.18: Output of example working with a Tuple.

List

Lists use the square bracket notation and can be index using array notation.

```
mylist = [1, 2, 3]
print("Zeroth Value: %d") % mylist[0]
mylist.append(4)
print("List Length: %d") % len(mylist)
for value in mylist:
  print value
```

Listing 3.19: Example of working with a List.

Notice that we are using some simple `printf`-like functionality to combine strings and variables when printing. Running the example prints:

```
Zeroth Value: 1
List Length: 4
1
2
3
4
```

Listing 3.20: Output of example working with a List.

Dictionary

Dictionaries are mappings of names to values, like key-value pairs. Note the use of the curly bracket and colon notations when defining the dictionary.

```
mydict = {'a': 1, 'b': 2, 'c': 3}
print("A value: %d") % mydict['a']
mydict['a'] = 11
print("A value: %d") % mydict['a']
print("Keys: %s") % mydict.keys()
print("Values: %s") % mydict.values()
for key in mydict.keys():
  print mydict[key]
```

Listing 3.21: Example of working with a Dictionary.

Running the example prints:

```
A value: 1
A value: 11
Keys: ['a', 'c', 'b']
Values: [11, 3, 2]
11
3
2
```

Listing 3.22: Output of example working with a Dictionary.

Functions

The biggest gotcha with Python is the whitespace. Ensure that you have an empty new line after indented code. The example below defines a new function to calculate the sum of two values and calls the function with two arguments.

```
# Sum function
def mysum(x, y):
  return x + y

# Test sum function
result = mysum(1, 3)
print(result)
```

Listing 3.23: Example of working with a custom function.

Running the example prints:

```
4
```

Listing 3.24: Output of example working with a custom function.

3.2 NumPy Crash Course

NumPy provides the foundation data structures and operations for SciPy. These are arrays (`ndarrays`) that are efficient to define and manipulate.

3.2.1 Create Array

```
# define an array
import numpy
mylist = [1, 2, 3]
myarray = numpy.array(mylist)
print(myarray)
print(myarray.shape)
```

Listing 3.25: Example of creating a NumPy array.

Notice how we easily converted a Python list to a NumPy array. Running the example prints:

```
[1 2 3]
(3,)
```

Listing 3.26: Output of example creating a NumPy array.

3.2.2 Access Data

Array notation and ranges can be used to efficiently access data in a NumPy array.

```
# access values
import numpy
mylist = [[1, 2, 3], [3, 4, 5]]
myarray = numpy.array(mylist)
print(myarray)
print(myarray.shape)
print("First row: %s") % myarray[0]
print("Last row: %s") % myarray[-1]
print("Specific row and col: %s") % myarray[0, 2]
print("Whole col: %s") % myarray[:, 2]
```

Listing 3.27: Example of working with a NumPy array.

Running the example prints:

```
[[1 2 3]
 [3 4 5]]
(2, 3)
First row: [1 2 3]
Last row: [3 4 5]
Specific row and col: 3
Whole col: [3 5]
```

Listing 3.28: Output of example working with a NumPy array.

3.2.3 Arithmetic

NumPy arrays can be used directly in arithmetic.

```
# arithmetic
import numpy
myarray1 = numpy.array([2, 2, 2])
myarray2 = numpy.array([3, 3, 3])
print("Addition: %s") % (myarray1 + myarray2)
print("Multiplication: %s") % (myarray1 * myarray2)
```

Listing 3.29: Example of doing arithmetic with NumPy arrays.

Running the example prints:

```
Addition: [5 5 5]
Multiplication: [6 6 6]
```

Listing 3.30: Output of example of doing arithmetic with NumPy arrays.

There is a lot more to NumPy arrays but these examples give you a flavor of the efficiencies they provide when working with lots of numerical data. See Chapter 24 for resources to learn more about the NumPy API.

3.3 Matplotlib Crash Course

Matplotlib can be used for creating plots and charts. The library is generally used as follows:

- Call a plotting function with some data (e.g. `.plot()`).
- Call many functions to setup the properties of the plot (e.g. labels and colors).
- Make the plot visible (e.g. `.show()`).

3.3.1 Line Plot

The example below creates a simple line plot from one dimensional data.

```
# basic line plot
import matplotlib.pyplot as plt
import numpy
myarray = numpy.array([1, 2, 3])
plt.plot(myarray)
plt.xlabel('some x axis')
plt.ylabel('some y axis')
plt.show()
```

Listing 3.31: Example of creating a line plot with Matplotlib.

Running the example produces:

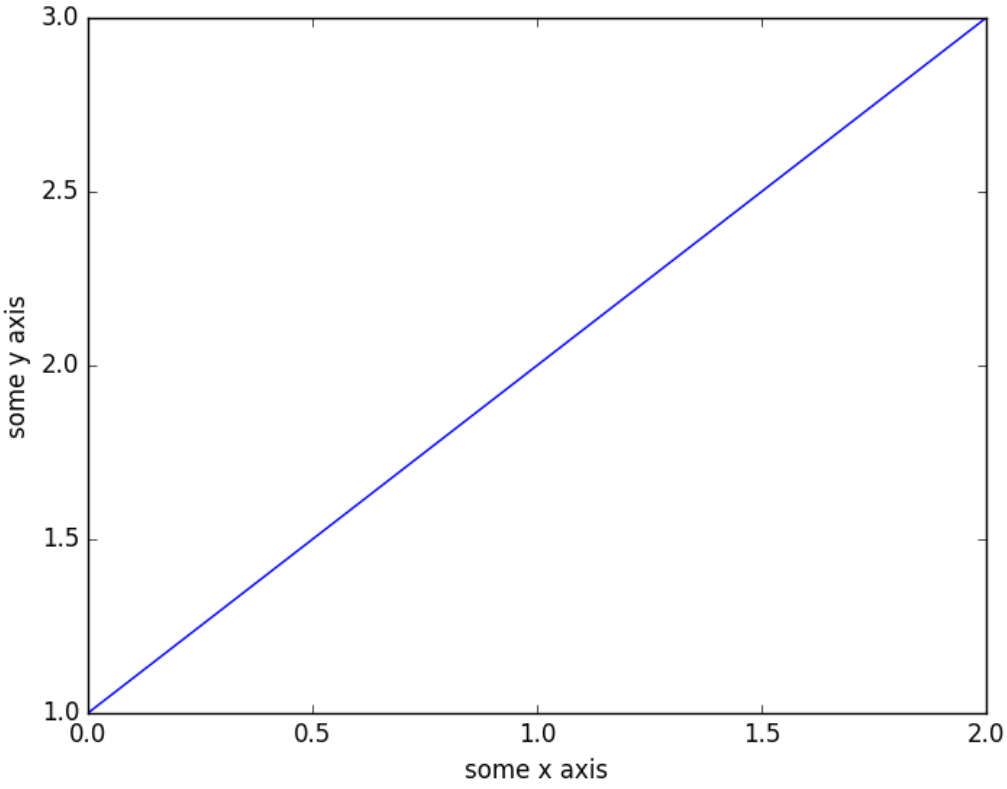

Figure 3.1: Line Plot with Matplotlib

3.3.2 Scatter Plot

Below is a simple example of creating a scatter plot from two dimensional data.

```
# basic scatter plot
import matplotlib.pyplot as plt
import numpy
x = numpy.array([1, 2, 3])
y = numpy.array([2, 4, 6])
plt.scatter(x,y)
plt.xlabel('some x axis')
plt.ylabel('some y axis')
plt.show()
```

Listing 3.32: Example of creating a line plot with Matplotlib.

Running the example produces:

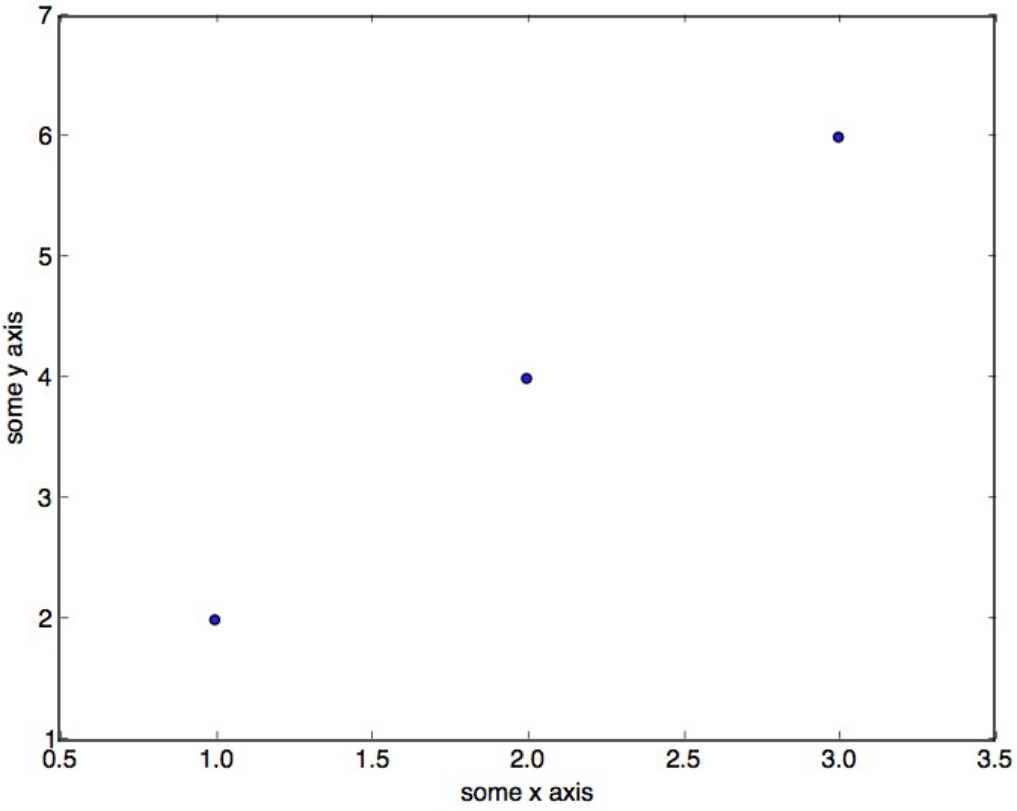

Figure 3.2: Scatter Plot with Matplotlib

There are many more plot types and many more properties that can be set on a plot to configure it. See Chapter 24 for resources to learn more about the Matplotlib API.

3.4 Pandas Crash Course

Pandas provides data structures and functionality to quickly manipulate and analyze data. The key to understanding Pandas for machine learning is understanding the Series and DataFrame data structures.

3.4.1 Series

A series is a one dimensional array where the rows and columns can be labeled.

```
# series
import numpy
import pandas
myarray = numpy.array([1, 2, 3])
rownames = ['a', 'b', 'c']
myseries = pandas.Series(myarray, index=rownames)
```

```
print(myseries)
```

Listing 3.33: Example of creating a Pandas Series.

Running the example prints:

```
a   1
b   2
c   3
```

Listing 3.34: Output of example of creating a Pandas Series.

You can access the data in a series like a NumPy array and like a dictionary, for example:

```
print(myseries[0])
print(myseries['a'])
```

Listing 3.35: Example of accessing data in a Pandas Series.

Running the example prints:

```
1
1
```

Listing 3.36: Output of example of accessing data in a Pandas Series.

3.4.2 DataFrame

A data frame is a multi-dimensional array where the rows and the columns can be labeled.

```
# dataframe
import numpy
import pandas
myarray = numpy.array([[1, 2, 3], [4, 5, 6]])
rownames = ['a', 'b']
colnames = ['one', 'two', 'three']
mydataframe = pandas.DataFrame(myarray, index=rownames, columns=colnames)
print(mydataframe)
```

Listing 3.37: Example of creating a Pandas DataFrame.

Running the example prints:

```
   one  two  three
a   1    2     3
b   4    5     6
```

Listing 3.38: Output of example of creating a Pandas DataFrame.

Data can be index using column names.

```
print("method 1:")
print("one column: %s") % mydataframe['one']
print("method 2:")
print("one column: %s") % mydataframe.one
```

Listing 3.39: Example of accessing data in a Pandas DataFrame.

Running the example prints:

```
method 1:
a    1
b    4
method 2:
a    1
b    4
```

Listing 3.40: Output of example of accessing data in a Pandas DataFrame.

Pandas is a very powerful tool for slicing and dicing you data. See Chapter 24 for resources to learn more about the Pandas API.

3.5 Summary

You have covered a lot of ground in this lesson. You discovered basic syntax and usage of Python and three key Python libraries used for machine learning:

- NumPy.
- Matplotlib.
- Pandas.

3.5.1 Next

You now know enough syntax and usage information to read and understand Python code for machine learning and to start creating your own scripts. In the next lesson you will discover how you can very quickly and easily load standard machine learning datasets in Python.

Chapter 4

How To Load Machine Learning Data

You must be able to load your data before you can start your machine learning project. The most common format for machine learning data is CSV files. There are a number of ways to load a CSV file in Python. In this lesson you will learn three ways that you can use to load your CSV data in Python:

1. Load CSV Files with the Python Standard Library.
2. Load CSV Files with NumPy.
3. Load CSV Files with Pandas.

Let's get started.

4.1 Considerations When Loading CSV Data

There are a number of considerations when loading your machine learning data from CSV files. For reference, you can learn a lot about the expectations for CSV files by reviewing the CSV request for comment titled *Common Format and MIME Type for Comma-Separated Values (CSV) Files*[1].

4.1.1 File Header

Does your data have a file header? If so this can help in automatically assigning names to each column of data. If not, you may need to name your attributes manually. Either way, you should explicitly specify whether or not your CSV file had a file header when loading your data.

4.1.2 Comments

Does your data have comments? Comments in a CSV file are indicated by a hash (#) at the start of a line. If you have comments in your file, depending on the method used to load your data, you may need to indicate whether or not to expect comments and the character to expect to signify a comment line.

[1] https://tools.ietf.org/html/rfc4180

4.1.3 Delimiter

The standard delimiter that separates values in fields is the comma (,) character. Your file could use a different delimiter like tab or white space in which case you must specify it explicitly.

4.1.4 Quotes

Sometimes field values can have spaces. In these CSV files the values are often quoted. The default quote character is the double quotation marks character. Other characters can be used, and you must specify the quote character used in your file.

4.2 Pima Indians Dataset

The Pima Indians dataset is used to demonstrate data loading in this lesson. It will also be used in many of the lessons to come. This dataset describes the medical records for Pima Indians and whether or not each patient will have an onset of diabetes within five years. As such it is a classification problem. It is a good dataset for demonstration because all of the input attributes are numeric and the output variable to be predicted is binary (0 or 1). The data is freely available from the UCI Machine Learning Repository[2].

4.3 Load CSV Files with the Python Standard Library

The Python API provides the module `CSV` and the function `reader()` that can be used to load CSV files. Once loaded, you can convert the CSV data to a NumPy array and use it for machine learning. For example, you can download[3] the Pima Indians dataset into your local directory with the filename `pima-indians-diabetes.data.csv`. All fields in this dataset are numeric and there is no header line.

```
# Load CSV Using Python Standard Library
import csv
import numpy
filename = 'pima-indians-diabetes.data.csv'
raw_data = open(filename, 'rb')
reader = csv.reader(raw_data, delimiter=',', quoting=csv.QUOTE_NONE)
x = list(reader)
data = numpy.array(x).astype('float')
print(data.shape)
```

Listing 4.1: Example of loading a CSV file using the Python standard library.

The example loads an object that can iterate over each row of the data and can easily be converted into a NumPy array. Running the example prints the shape of the array.

```
(768, 9)
```

Listing 4.2: Output of example loading a CSV file using the Python standard library.

[2] https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes
[3] https://goo.gl/vhm1eU

For more information on the `csv.reader()` function, see *CSV File Reading and Writing in the Python* API documentation[4].

4.4 Load CSV Files with NumPy

You can load your CSV data using NumPy and the `numpy.loadtxt()` function. This function assumes no header row and all data has the same format. The example below assumes that the file `pima-indians-diabetes.data.csv` is in your current working directory.

```
# Load CSV using NumPy
from numpy import loadtxt
filename = 'pima-indians-diabetes.data.csv'
raw_data = open(filename, 'rb')
data = loadtxt(raw_data, delimiter=",")
print(data.shape)
```

Listing 4.3: Example of loading a CSV file using NumPy.

Running the example will load the file as a `numpy.ndarray`[5] and print the shape of the data:

```
(768, 9)
```

Listing 4.4: Output of example loading a CSV file using NumPy.

This example can be modified to load the same dataset directly from a URL as follows:

```
# Load CSV from URL using NumPy
from numpy import loadtxt
from urllib import urlopen
url = 'https://goo.gl/vhm1eU'
raw_data = urlopen(url)
dataset = loadtxt(raw_data, delimiter=",")
print(dataset.shape)
```

Listing 4.5: Example of loading a CSV URL using NumPy.

Again, running the example produces the same resulting shape of the data.

```
(768, 9)
```

Listing 4.6: Output of example loading a CSV URL using NumPy.

For more information on the `numpy.loadtxt()`[6] function see the API documentation.

4.5 Load CSV Files with Pandas

You can load your CSV data using Pandas and the `pandas.read_csv()` function. This function is very flexible and is perhaps my recommended approach for loading your machine learning data. The function returns a pandas.DataFrame[7] that you can immediately start summarizing and plotting. The example below assumes that the `pima-indians-diabetes.data.csv` file is in the current working directory.

[4] https://docs.python.org/2/library/csv.html
[5] http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.ndarray.html
[6] http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.loadtxt.html
[7] http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html

```
# Load CSV using Pandas
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)
print(data.shape)
```

Listing 4.7: Example of loading a CSV file using Pandas.

Note that in this example we explicitly specify the names of each attribute to the DataFrame. Running the example displays the shape of the data:

```
(768, 9)
```

Listing 4.8: Output of example loading a CSV file using Pandas.

We can also modify this example to load CSV data directly from a URL.

```
# Load CSV using Pandas from URL
from pandas import read_csv
url = 'https://goo.gl/vhm1eU'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(url, names=names)
print(data.shape)
```

Listing 4.9: Example of loading a CSV URL using Pandas.

Again, running the example downloads the CSV file, parses it and displays the shape of the loaded DataFrame.

```
(768, 9)
```

Listing 4.10: Output of example loading a CSV URL using Pandas.

To learn more about the `pandas.read_csv()`[8] function you can refer to the API documentation.

4.6 Summary

In this chapter you discovered how to load your machine learning data in Python. You learned three specific techniques that you can use:

- Load CSV Files with the Python Standard Library.
- Load CSV Files with NumPy.
- Load CSV Files with Pandas.

Generally I recommend that you load your data with Pandas in practice and all subsequent examples in this book will use this method.

[8] http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

4.6.1 Next

Now that you know how to load your CSV data using Python it is time to start looking at it. In the next lesson you will discover how to use simple descriptive statistics to better understand your data.