# 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.
```

It is a popular language in general, consistently appearing in the top 10 programming languages in surveys on StackOverflow . It's a dynamic language and very suited to interactive 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 and the KDD Nuggets 2015 tool survey results . 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.

## 2.4 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 $Downloading\ Python$ in the $Python\ Beginners\ Guide$. Once installed you can confirm the installation was successful.

Open a command line and type:

[ ] ↳ *1 cell hidden*

### 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 $Installing\ the\ SciPy\ Stack$ . 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.

[ ] ↳ *1 cell hidden*

### 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 , 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.

[ ] *↳ 1 cell hidden*

### 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. 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.

*↳ 1 cell hidden*

## 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.

# 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.

[ ] *↳ 10 cells hidden*

## 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.

[ ] ↳ *6 cells hidden*

## 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.

```
# Listing 3.17: Example of working with a Tuple.
a = (1, 2, 3)
print(a)
```

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

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

```
# Listing 3.19: Example of working with a List.

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)
```

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

```
1
2
3
4
```

**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.

```
# Listing 3.21: Example of working with a 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])
```

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

**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.

```
# Listing 3.23: Example of working with a custom function
# Sum Funtion
```

```
def mysum(x, y):
  return x + y

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

```
4
```

---

**My Understanding**

In the above section:

- We learnt how one works with strings, numbers and booleans in Python
- We learnt how assignments work, and how multiple assignments work. Additionally learnt how no value operations work.
- We learnt how if-then-else, for loop and while loop in Python works.
- We learnt how tuple, list and dictionary works in Python. Also learnt about the associated operations with the same.
- Learnt how functions work in Python. Additionally, created a simple Sum function in Python.

---

## 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

[ ] *4 cells hidden*

### 3.2.2 Access Data

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

[ ] ↳ *1 cell hidden*

## 3.2.3 Arithmetic

NumPy arrays can be used directly in arithmetic.

```
# Listing 3.29: Example of doing arithmetic with NumPy arrays.
# 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,))
```

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

---

**My Understanding**

In the above section,

- Learnt how one can create an array using Numpy in Python.
- Learnt how we can perform different operations like accessing data in an array and performing of arithmetic opearions like addition and multiplication in numpy array.

---

# 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()).

[ ] ↳ *5 cells hidden*

## 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.

[ ] ↳ *10 cells hidden*

## 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.

# 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*.

*↳ 4 cells hidden*

## 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 .

## 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 download3 the Pima Indians dataset into your local directory with the filename `diabetes.csv`. All fields in this dataset are numeric and there is no header line.

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

```
# Listing 4.1: Example of loading a CSV file using the Python standard library.
# Load CSV Using Python Standard Library
import csv
import numpy
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
raw_data = open(filename, 'r')
reader = csv.reader(raw_data, delimiter=',', quoting=csv.QUOTE_NONE)
x = list(reader)
data = numpy.array(x).astype('float')
print(data.shape)
```

```
(768, 9)
```

---

**My Understanding**

In the above code,

- We are trying to load the CSV files using Python standard library.
- We have mounted the file to Google Drive and specified the path as the filename.
- We are opening file using Python's built-in library method open in read only mode.
- We need to ensure that the filename is correct and path is properly specified.
- We are using `csv.reader()` method to open CSV files. Since the file content is separated by comma as a delimeter we are specifying the same.
- We are converting data extracted from CSV into list and then into array.
- We are printing the shape of the file.

---

## 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 `diabetes.csv` is in your current working directory.

```
# Listing 4.3: Example of loading a CSV file using NumPy.
# Load CSV Files using NumPy
from numpy import loadtxt
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
raw_data = open(filename, 'rb')
data = loadtxt(raw_data, delimiter=",")
print(data.shape)
```

```
    (768, 9)
```

---

**My Understanding**

In the above code,

- We are importing CSV file using library NumPy
- For this, we are importing `loadtxt` method from `numpy`
- We are using Python's built in library to read the file
- We need to ensure that the filename is correct and path is properly specified.
- Next we are using `loadtxt` method to load the raw data which is separated by comma delimeter.
- Next, we are printing the shape of the CSV file.

---

## 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 that you can immediately start summarizing and plotting. The example below assumes that the `diabetes.csv` file is in the current working directory.

Note that in this example we explicitly specify the names of each attribute to the DataFrame.

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

```
from pandas import read_csv
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
print(data.shape)
```

```
    (768, 9)
```

---

**My Understanding**

In the above code:

- We are importing data from the file using using Pandas library.
- We are using `read_csv` method available in Pandas library. Addotonally, we are assigning names to all the columns by passing it as a param.
- We need to ensure that the filename is correct and path is properly specified.
- We are later printing the shame of the data.
- Based on how easy it was to import CSV files using Pandas, we would prefer using the same for future.

---

## 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.

# Chapter 5 : 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.

## 5.1 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.

```
# Listing 5.1: Example of reviewing the first few rows of data.
# View first 20 rows
from pandas import read_csv
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
peek = data.head(20)
print(peek)
```

```
    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
```

---

**My Understanding**

In the above code,

- We are importing data from `diabetes.csv` CSV file using read_csv method in Python.
- We are printing first 20 elements of the data just to get an idea of contents of the data and how it is like to get better understanding of the data and we are printing the data.

---

## 5.2 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.

The results are listed in rows then columns.

```
# Listing 5.3: Example of reviewing the shape of the data.
# Dimensions of your data
from pandas import read_csv
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
shape = data.shape
print(shape)
```

```
    (768, 9)
```

---

**My Understanding** In the above code:

- We are importing data from `diabetes.csv` CSV file using read_csv method in Python.
- We are printing the shape of file, i.e., number of rows and columns using shape method.

---

## 5.3 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.

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

```
# Listing 5.5: Example of reviewing the data types of the data.
# Data Types for Each Attribute
from pandas import read_csv
```

```
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
types = data.dtypes
print(types)
```

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

---

**My Understanding**

- We are importing data from `diabetes.csv` CSV file using `read_csv` method in Python
- We are printing the datatype of each row of the data imported from the CSV files.
- As we can see above, the data types of rows are `int64` and `float64`.

---

## 5.4 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.

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.

```
# Listing 5.7: Example of reviewing a statistical summary of the data.
# Statistical Summary
from pandas import read_csv
from pandas import set_option
filename = '/content/drive/MyDrive/Datasets/diabetes.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)
```

```
             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
```

---

**My Understanding**

From the above code,

- We are importing data from `diabetes.csv` CSV file using read_csv method in Python
- We will be setting `display.width` and `precision` parameters that will be used while describing data.
- Setting width to a max 100 ensures that all the row is viewed. Setting precision to 3 ensures that we will have 3 digits post decimal.
- `describe()` method is used to describe properties like count, mean standard deviation, minimum value, 25 percentile, 50 percentile(median), 75th percentile and maximum value.

---

## 5.5 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.

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).

```
# Listing 5.9: Example of reviewing a class breakdown of the data.
# Class Distribution
from pandas import read_csv
filename = '/content/drive/MyDrive/Datasets/diabetes.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)
```

```
class
0    500
1    268
dtype: int64
```

---

**My Understanding**

- We are importing data from `diabetes.csv` CSV file using `read_csv` method in Python
- We are using `groupby` method to group the data based on the row named `class` and printing the value.
- We can observer from the output that there are 500 values with class 0 and 268 values with class value 1.

---

## 5.6 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.

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.

```
# Listing 5.11: Example of reviewing correlations of attributes in the data.
# Pairwise Pearson Correlations
from pandas import read_csv
from pandas import set_option
filename = '/content/drive/MyDrive/Datasets/diabetes.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)
```

```
            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
```

---

**My Understanding**

From the above code, we can understand that:

- We are importing data from `diabetes.csv` CSV file using `read_csv` method in Python
- We are setting width and precision to 100 and 3 respectivily as per our requirements.
- We are trying to find correlation of size m x m between the data available. Correlation is relation between two varibles and how they may or may not change between the operation. We will use `cors()` method to find the correlation.

---

## 5.7 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.

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

```
# Listing 5.13: Example of reviewing skew of attribute distributions in the data.
# Skew for each attribute
from pandas import read_csv
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
```

```
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
skew = data.skew()
print(skew)
```

```
    preg     0.902
    plas     0.174
    pres    -1.844
    skin     0.109
    test     2.272
    mass    -0.429
    pedi     1.920
    age      1.130
    class    0.635
    dtype: float64
```

---

**My Understanding**

From the above code, we can understand that:

- We are importing data from `diabetes.csv` CSV file using `read_csv` method in Python
- Skew refers to the distribution that is assumed Guassian that is shifted in one direction or another.
- We will used `skew()` method for the same.
- The result is either positve or negative showing where is skew is tiled. The value closer to zero means the skew is less.

---

## 5.8 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.

### 5.9 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.

## Chapter 6 : Understand Your Data With Visualization

You must understand your data in order to get the best results from machine learning algorithms. The fastest way to learn more about your data is to use data visualization. In this chapter you will discover exactly how you can visualize your machine learning data in Python using Pandas. Recipes in this chapter use the Pima Indians onset of diabetes dataset introduced in Chapter 4.

Let's get started.

### 6.1 Univariate Plots

In this section we will look at three techniques that you can use to understand each attribute of your dataset independently.

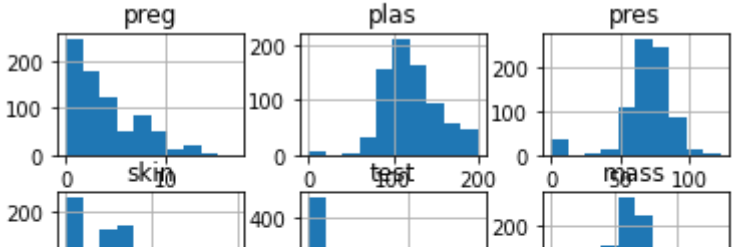
- Histograms.
- Density Plots.
- Box and Whisker Plots.

## 6.1.1 Histograms

A fast way to get an idea of the distribution of each attribute is to look at histograms. Histograms group data into bins and provide you a count of the number of observations in each bin. From the shape of the bins you can quickly get a feeling for whether an attribute is Gaussian, skewed or even has an exponential distribution. It can also help you see possible outliers.

We can see that perhaps the attributes age, pedi and test may have an exponential distribution. We can also see that perhaps the mass and pres and plas attributes may have a Gaussian or nearly Gaussian distribution. This is interesting because many machine learning techniques assume a Gaussian univariate distribution on the input variables.

```
# Listing 6.1: Example of creating histogram plots.
# Univariate Histograms
from matplotlib import pyplot
from pandas import read_csv
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
data.hist()
pyplot.show()
```




---

**My Understanding**

From the above code, we can understand that:

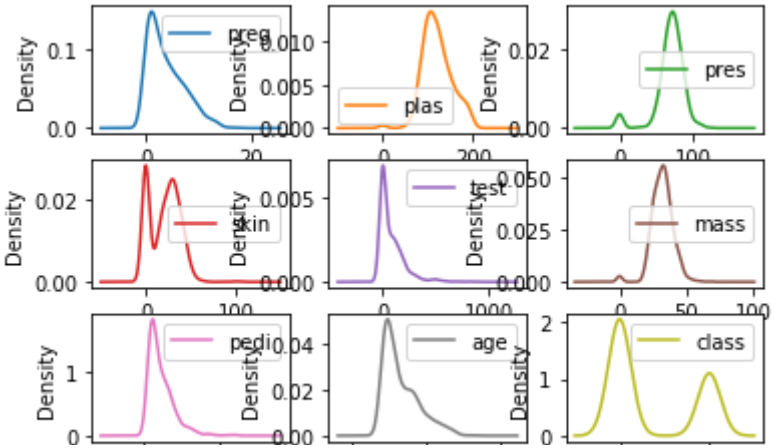
- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Python
- We are importing pyplot library to use graphs.
- We are using `hist()` funtion to display historgram representation of the data where we are showing count of the number of obervations in each bin.

---

## 6.1.2 Density Plots

Density plots are another way of getting a quick idea of the distribution of each attribute. The plots look like an abstracted histogram with a smooth curve drawn through the top of each bin, much like your eye tried to do with the histograms.

```
# Listing 6.2: Example of creating density plots.
# Univariate Density Plots
from matplotlib import pyplot
from pandas import read_csv
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
data.plot(kind="density", subplots=True, layout=(3,3), sharex=False)
pyplot.show()
```




**My Understanding**

From the above code, we can understand that:

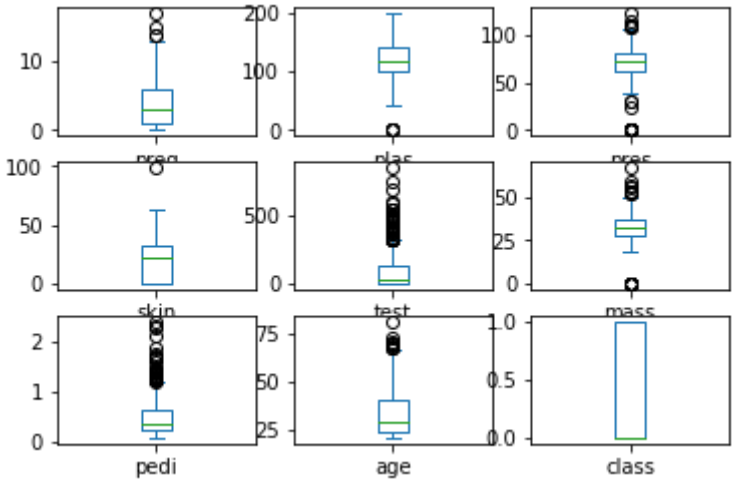
- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Python
- We are importing pyplot library to use graphs.
- We are showing the line graph showing the density of the count of the data available in the bin

## 6.1.3 Box and Whisker Plots

Another useful way to review the distribution of each attribute is to use Box and Whisker Plots or boxplots for short. Boxplots summarize the distribution of each attribute, drawing a line for the median (middle value) and a box around the 25th and 75th percentiles (the middle 50% of the data). The whiskers give an idea of the spread of the data and dots outside of the whiskers show candidate outlier values (values that are 1.5 times greater than the size of spread of the middle 50% of the data).

```
# Listing 6.3: Example of creating box and whisker plots.
# Box and Whisker Plots
from matplotlib import pyplot
from pandas import read_csv
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
```

```
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
data.plot(kind="box", subplots=True, layout=(3,3), sharex=False, sharey=False)
pyplot.show()
```




We can see that the spread of attributes is quite different. Some like age, test and skin appear quite skewed towards smaller values.

---

**My Understanding**

From the above code, we can understand that:

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Python
- We are importing pyplot library to use graphs.
- We are using `data.plot()` method and passing `box` as the kind of argument to print the box and whisker graph.
- The box in box and whisker shows how the data is distributed majorly in terms of values.
- The whiskers shows how the data is distributed in the each bin. The dots here reprents the data that is too far away from the median values.

---

## 6.2 Multivariate Plots

This section provides examples of two plots that show the interactions between multiple variables in your dataset.

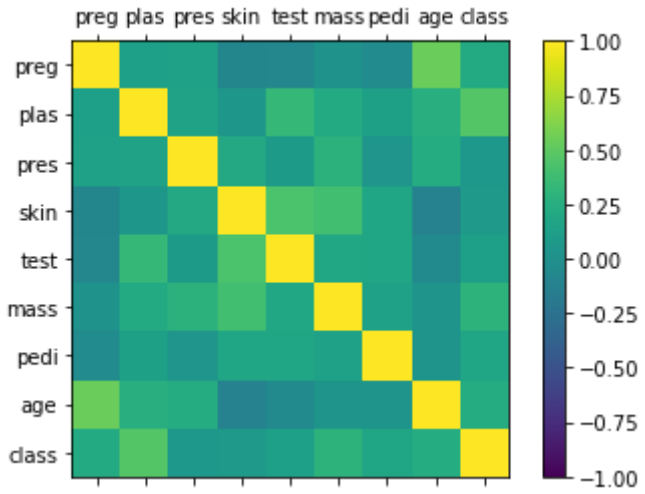
- Correlation Matrix Plot.
- Scatter Plot Matrix.

### 6.2.1 Correlation Matrix Plot

Correlation gives an indication of how related the changes are between two variables. If two variables change in the same direction they are positively correlated. If they change in opposite directions together (one goes up, one goes down), then they are negatively correlated. You can calculate the correlation between each pair of attributes. This is called a correlation matrix. You can then plot the correlation matrix and get an idea of which variables have a high correlation with each other. This is useful to know, because some machine learning algorithms like linear and logistic regression can have poor performance if there are highly correlated input variables in your data.

```
#Listing 6.4: Example of creating a correlation matrix plot.
# Correlation Matrix Plot
from matplotlib import pyplot
from pandas import read_csv
import numpy
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
correlations = data.corr()
# plot correlation matrix
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = numpy.arange(0, 9, 1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
```

```
ax.set_xticklabels(names)
ax.set_yticklabels(names)
pyplot.show()
```




---

**My Understanding**

From the above code, we can understand that:

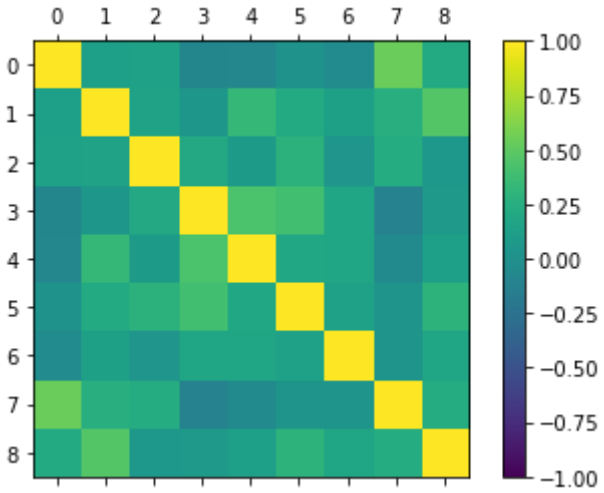
- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Python
- We are importing pyplot library to use graphs.
- We are using `figure()` method to create a correlation matrix. A correlation matrix shows how two data are related with each other.
- Correlation can either be negative or positive depending on how data is related to each other. A positive correlation means that the data is directly proportional whereas a negative correlation means the data is indirectly proportional.
- We are adding colorbar to show the data with colors, and additionally showing the data in matrix representation.
- We are also adding labels to the x and y axis and thus presenting the data using `plot()` method.

---

The example is not generic in that it specifies the names for the attributes along the axes as well as the number of ticks. This recipe cam be made more generic by removing these aspects as follows:

Generating the plot, you can see that it gives the same information although making it a little harder to see what attributes are correlated by name. Use this generic plot as a first cut to understand the correlations in your dataset and customize it like the first example in order to read off more specific data if needed.

```
# Listing 6.5: Example of creating a generic correlation matrix plot.
# Correction Matrix Plot (generic)
from matplotlib import pyplot
from pandas import read_csv
import numpy
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
correlations = data.corr()
# plot correlation matrix
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
pyplot.show()
```



---

**My Understanding**

From the above code, we can understand that:

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Python
- We are importing pyplot library to use graphs.
- We are using `figure()` method to create a correlation matrix. A correlation matrix shows how two data are related with each other.
- Correlation can either be negative or positive depending on how data is related to each other. A positive correlation means that the data is directly proportional whereas a negative correlation means the data is indirectly proportional.
- We are adding colorbar to show the data with colors, and additionally showing the data in matrix representation.
- We are pesenting the data using `plot()` method which shows the colored graph along with the generic labels for the corresponding axises.

---

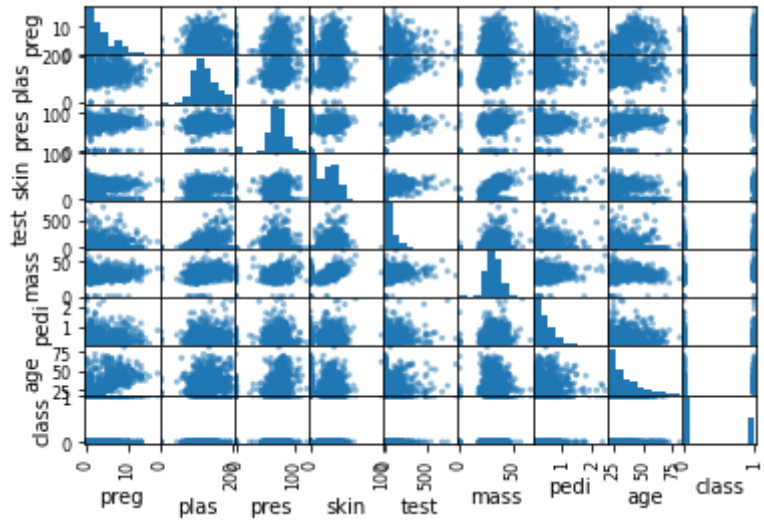
## 6.2.2 Scatter Plot Matrix

A scatter plot shows the relationship between two variables as dots in two dimensions, one axis for each attribute. You can create a scatter plot for each pair of attributes in your data. Drawing all these scatter plots together is called a scatter plot matrix. Scatter plots are useful for spotting structured relationships between variables, like whether you could summarize the relationship between two variables with a line. Attributes with structured relationships may also be correlated and good candidates for removal from your dataset.

Like the Correlation Matrix Plot above, the scatter plot matrix is symmetrical. This is useful to look at the pairwise relationships from different perspectives. Because there is little point of drawing a scatter plot of each variable with itself, the diagonal shows histograms of each attribute.

```
# Listing 6.6: Example of creating a scatter plot matrix.
# Scatterplot Matrix
from matplotlib import pyplot
from pandas import read_csv
from pandas.plotting import scatter_matrix
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
```

```
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
scatter_matrix(data)
pyplot.show()
```




---

**My Understanding**

From the above code, we can understand that:

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Python
- We are importing pyplot library to use graphs.
- We are importing `scatter_matrix` from `pandas.plotting` to present the data in scatter plot matrix.
- Scatter plot describes how two variables are related using dots in 2D. Scatter plot matrix too is symmetrical. The closer the dots are together to form the bar, the more closely related the data are rather than being scattered.

---

## 6.3 Summary

In this chapter you discovered a number of ways that you can better understand your machine learning data in Python using Pandas. Specifically, you learned how to plot your data using:

- Histograms.
- Density Plots.
- Box and Whisker Plots.
- Correlation Matrix Plot.
- Scatter Plot Matrix.

# Chapter 7 : 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 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.

## 7.1 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

## 7.2 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.

## 7.3 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

```
# Listing 7.1: Example of rescaling data.
# Rescale Data (between 0 and 1)
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import MinMaxScaler
filename = '/content/drive/MyDrive/Datasets/diabetes.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 transfromed data
set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

```
    [[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  ]]
```

---

**My Understanding**

From the above code, we can understand that:

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Python
- We are also importing MinMaxScaler to rescale the existing data.
- We are rescaling the data from 0 to 1. We can rescale the data from $n$ to $m$ as well depending on our requirement by passing the range as a tuple as `feature_range` param. We are calling `fit_transform` method to finish transformation and execute the rescaling and finally print the value.

- We are printing the rescaled values of the data in a 2D matix data to better understand how the data is ranged between.

## 7.4 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 .

```
# Listing 7.3: Example of standardizing data.
# Standardize date (0 mean, 1 stdev)
from sklearn.preprocessing import StandardScaler
from pandas import read_csv
from numpy import set_printoptions
filename = '/content/drive/MyDrive/Datasets/diabetes.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 componenets
X = array[:, 0:8]
Y = array[:, 8]
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)
# summarized transform data
set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

```
[[ 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 ]]
```

---

**My Understanding**

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Python
- We are also importing StandardScaler from `sklearn.preprocessing` that is used to standardize scaler data.
- StandardScaler standardizes features by removing the mean and scaling to unit variance.
- We are then using transform to tranform the data into 2D array and print the corresponding values.

---

## 7.5 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 .

```
# Listing 7.5: Example of normalizing data.
# Normalize data (length of 1)
from sklearn.preprocessing import Normalizer
from pandas import read_csv
from numpy import set_printoptions
filename = '/content/drive/MyDrive/Datasets/diabetes.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 transfromed data
set_printoptions(precision=3)
print(normalizedX[0:5,:])
```

```
    [[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]]
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Python
- We are also importing `Normalizer` from `sklearn.preprocessing` that is used to standardize scaler data.
- Normalizer function normalizes sample individually to unit norm.
- We are further using transform method to get the data in 2D array format.

---

## 7.6 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.

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

```
# Listing 7.7: Example of binarizing data.
# binarization
from sklearn.preprocessing import Binarizer
from pandas import read_csv
from numpy import set_printoptions
filename = '/content/drive/MyDrive/Datasets/diabetes.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,:])
```

```
    [[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.]]
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Python
- We are also importing `Binarizer` from `sklearn.preprocessing` that is used to standardize scaler data.
- Binarizer funtion is used to set the data in either of the binary numbers, i.e., 0 or 1.
- We are further displaying the data in 2D matrix, and thus avoiding decimal values.

---

## 7.7 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.

# Chapter 8 : Feature Selection for Machine Learning

The data features that you use to train your machine learning models have a huge influence on the performance you can achieve. Irrelevant or partially relevant features can negatively impact model performance. In this chapter you will discover automatic feature selection techniques that you can use to prepare your machine learning data in Python with scikit-learn. After completing this lesson you will know how to use:

1. Univariate Selection.
2. Recursive Feature Elimination.
3. Principle Component Analysis.
4. Feature Importance.

Let's get started.

## 8.1 Feature Selection

Feature selection is a process where you automatically select those features in your data that contribute most to the prediction variable or output in which you are interested. Having irrelevant features in your data can decrease the accuracy of many models, especially linear algorithms like linear and logistic regression. Three benefits of performing feature selection before modeling your data are:

- **Reduces Overfitting**: Less redundant data means less opportunity to make decisions based on noise.
- **Improves Accuracy**: Less misleading data means modeling accuracy improves.
- **Reduces Training Time**: Less data means that algorithms train faster.

You can learn more about feature selection with scikit-learn in the article Feature selection . Each feature selection recipes will use the Pima Indians onset of diabetes dataset.

# 8.2 Univariate Selection

Statistical tests can be used to select those features that have the strongest relationship with the output variable. The scikit-learn library provides the SelectKBest class2 that can be used with a suite of different statistical tests to select a specific number of features. The example below uses the chi-squared ($chi^2$ ) statistical test for non-negative features to select 4 of the best features from the Pima Indians onset of diabetes dataset.

```
# Listing 8.1: Example of univariate feature selection.
# Feature Extraction with Unvariate Statistical Tests (Chi-squared and classification)
from pandas import read_csv
from numpy import set_printoptions
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# load data
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, Y)
# summarize scores
set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
# summarize selected features
print(features[0:5,:])
```

```
     [ 111.52  1411.887   17.605   53.108 2175.565  127.669    5.393  181.304]
     [[148.    0.   33.6  50. ]
      [ 85.    0.   26.6  31. ]
      [183.    0.   23.3  32. ]
      [ 89.   94.   28.1  21. ]
      [137.  168.   43.1  33. ]]
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are importing `SelectKBest` and `chi2` methods from `sklearn.feature_selection`
- We are using `SelectKBest` to extract best features from the available data. Here, we are passing k=4 to select 4 best feature in the data. We are passing `chi2` as score funtion to the same method. We are storing the same in the test variable.
- We are further printing the fit scores of the same as 1D array with three decimal precision.
- We are printing features array to print the features most that have strongest relationship with the output variable.

---

## 8.3 Recursive Feature Elimination

The Recursive Feature Elimination (or RFE) works by recursively removing attributes and building a model on those attributes that remain. It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute. You can learn more about the RFE class3 in the scikit-learn documentation. The example below uses RFE with the logistic regression algorithm to select the top 3 features. The choice of algorithm does not matter too much as long as it is skillful and consistent.

```
# Listing 8.3: Example of RFE feature selection.
# Feature Extraction with RFE
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load data
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
```

```
model = LogisticRegression(solver='lbfgs', max_iter=3000)
rfe = RFE(model, n_features_to_select=3)
fit = rfe.fit(X, Y)
print("Num Features: %d" % (fit.n_features_,))
print("Selected Features: %s" % (fit.support_,))
print("Feature Ranking: %s" % (fit.ranking_,))
```

```
Num Features: 3
Selected Features: [ True False False False False  True  True False]
Feature Ranking: [1 2 4 6 5 1 1 3]
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are importing `RFE` and `LogisticRegression` from `feature_selection` and `linear_model`
- Since we are using LogisticRegresson model, we are using that model to pass to RFE method and passing it to RFE method.
- We are selecting three features to find the recursive feature elimination and printing the the final features that were selected. The features that were selected are indicated by `True` as the variable returned from the array. We are also printing the ranking of features that were returned to identify which of them is the best fit.

---

## 8.4 Principal Component Analysis

Principal Component Analysis (or PCA) uses linear algebra to transform the dataset into a compressed form. Generally this is called a data reduction technique. A property of PCA is that you can choose the number of dimensions or principal components in the transformed result. In the example below, we use PCA and select 3 principal components. Learn more about the PCA class in scikit-learn by reviewing the API.

```
# Feature Extraction with PCA
from pandas import read_csv
from sklearn.decomposition import PCA
```

```
# load data
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(X)
# summarize components
print("Explained Variance: %s" % (fit.explained_variance_ratio_,))
print(fit.components_)
```

```
Explained Variance: [0.88854663 0.06159078 0.02579012]
[[-2.02176587e-03  9.78115765e-02  1.60930503e-02  6.07566861e-02
   9.93110844e-01  1.40108085e-02  5.37167919e-04 -3.56474430e-03]
 [-2.26488861e-02 -9.72210040e-01 -1.41909330e-01  5.78614699e-02
   9.46266913e-02 -4.69729766e-02 -8.16804621e-04 -1.40168181e-01]
 [-2.24649003e-02  1.43428710e-01 -9.22467192e-01 -3.07013055e-01
   2.09773019e-02 -1.32444542e-01 -6.39983017e-04 -1.25454310e-01]]
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are importing `PCA` from `sklearn.decomposition` to calculate Principal Component Analysis.
- Since we have to select 3 best features, that is why we are passing `n_components=3` as a feature
- We are further printing explained variance ratio which is percentage of variance explained by each of the components that were selected previously.
- In the ending, we are printing principle axes in feature space which represents the direction of maximum variance of the data.

---

## 8.5 Feature Importance

Bagged decision trees like Random Forest and Extra Trees can be used to estimate the importance of features. In the example below we construct a `ExtraTreesClassifier` classifier for the Pima Indians onset of diabetes dataset. You can learn more about the `ExtraTreesClassifier` class in the scikit-learn API. You can see that we are given an importance score for each attribute where the larger the score, the more important the attribute. The scores suggest at the importance of plas, age and mass.

```
# Listing 8.7: Example of feature importance.
# Feature Extraction with Extra Trees Classifier
from pandas import read_csv
from sklearn.ensemble import ExtraTreesClassifier
# load data
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = ExtraTreesClassifier()
model.fit(X, Y)
print(model.feature_importances_)
```

```
[0.11177511 0.22847196 0.10089279 0.0797169  0.07834494 0.14149801
 0.11664748 0.14265281]
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas2

- We are also importing `ExtraTreesClassifier` from `sklearn.ensemble`. This `ExtraTreesClassifier` can be used to understand the importance of features.
- In the end, we are printing how important a certain feature is. The bigger the value of the feature, more is the importance of the feature.

### 8.6 Summary

In this chapter you discovered feature selection for preparing machine learning data in Python with scikit-learn. You learned about 4 different automatic feature selection techniques:

- Univariate Selection.
- Recursive Feature Elimination.
- Principle Component Analysis.
- Feature Importance.

## Chapter 9 : Evaluate the Performance of Machine Learning Algorithms with Resampling

You need to know how well your algorithms perform on unseen data. The best way to evaluate the performance of an algorithm would be to make predictions for new data to which you already know the answers. The second best way is to use clever techniques from statistics called resampling methods that allow you to make accurate estimates for how well your algorithm will perform on new data. In this chapter you will discover how you can estimate the accuracy of your machine learning algorithms using resampling methods in Python and scikit-learn on the Pima Indians dataset. Let's get started.

### 9.1 Evaluate Machine Learning Algorithms

Why can't you train your machine learning algorithm on your dataset and use predictions from this same dataset to evaluate machine learning algorithms? The simple answer is overfitting.

Imagine an algorithm that remembers every observation it is shown during training. If you evaluated your machine learning algorithm on the same dataset used to train the algorithm, then an algorithm like this would have a perfect score on the training dataset. But the predictions it

made on new data would be terrible. We must evaluate our machine learning algorithms on data that is not used to train the algorithm.

The evaluation is an estimate that we can use to talk about how well we think the algorithm may actually do in practice. It is not a guarantee of performance. Once we estimate the performance of our algorithm, we can then re-train the final algorithm on the entire training dataset and get it ready for operational use. Next up we are going to look at four different techniques that we can use to split up our training dataset and create useful estimates of performance for our machine learning algorithms:

- Train and Test Sets.
- k-fold Cross Validation.
- Leave One Out Cross Validation.
- Repeated Random Test-Train Splits.

## 9.2 Split into Train and Test Sets

The simplest method that we can use to evaluate the performance of a machine learning algorithm is to use different training and testing datasets. We can take our original dataset and split it into two parts. Train the algorithm on the first part, make predictions on the second part and evaluate the predictions against the expected results. The size of the split can depend on the size and specifics of your dataset, although it is common to use 67% of the data for training and the remaining 33% for testing.

This algorithm evaluation technique is very fast. It is ideal for large datasets (millions of records) where there is strong evidence that both splits of the data are representative of the underlying problem. Because of the speed, it is useful to use this approach when the algorithm you are investigating is slow to train. A downside of this technique is that it can have a high variance. This means that differences in the training and test dataset can result in meaningful differences in the estimate of accuracy. In the example below we split the Pima Indians dataset into 67%/33% splits for training and test and evaluate the accuracy of a Logistic Regression model.

```
# Listing 9.1: Example of evaluating an algorithm with a train and test set.
# Evaluate using Train and a Test Set
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
model = LogisticRegression(solver='lbfgs', max_iter=3000)
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (result*100.0,))
```

```
Accuracy: 78.740%
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas2
- We are also importing `train_test_split` method from `sklearn.model_selection`
- We are assigning the `test_size` to be 0.33, i.e, we are keeping 33% for the data for test purpose and remaining 67% for training purpose.
- We are then using Logistic Regression to generate a model and passing test values to our trained model.
- Using that we are generating accuracy of our trained model.

---

We can see that the estimated accuracy for the model was approximately 75%. Note that in addition to specifying the size of the split, we also specify the random seed. Because the split of the data is random, we want to ensure that the results are reproducible. By specifying the random seed we ensure that we get the same random numbers each time we run the code and in turn the same split of data. This is important

if we want to compare this result to the estimated accuracy of another machine learning algorithm or the same algorithm with a different

## 9.3 K-fold Cross Validation

Cross validation is an approach that you can use to estimate the performance of a machine learning algorithm with less variance than a single train-test set split. It works by splitting the dataset into k-parts (e.g. k = 5 or k = 10). Each split of the data is called a fold. The algorithm is trained on k − 1 folds with one held back and tested on the held back fold. This is repeated so that each fold of the dataset is given a chance to be the held back test set. After running cross validation you end up with k different performance scores that you can summarize using a mean and a standard deviation.

The result is a more reliable estimate of the performance of the algorithm on new data. It is more accurate because the algorithm is trained and evaluated multiple times on different data. The choice of k must allow the size of each test partition to be large enough to be a reasonable sample of the problem, whilst allowing enough repetitions of the train-test evaluation of the algorithm to provide a fair estimate of the algorithms performance on unseen data. For modest sized datasets in the thousands or tens of thousands of records, k values of 3, 5 and 10 are common. In the example below we use 10-fold cross validation.

```
# Listing 9.3: Example of evaluating an algorithm with k-fold Cross Validation.
# Evaluate using Cross Validation
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
num_folds = 10
kfold = KFold(n_splits=num_folds, shuffle=True)
model = LogisticRegression(solver='lbfgs', max_iter=3000)
results = cross_val_score(model, X, Y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100, ))
```

```
Accuracy: 77.483% (4.027%)
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas.
- We are also importing `KFold` and `cross_val_sccore` from sklearn.model_selection
- We are then splitting the data available in to 10 different folds ( `n_splits=10` ) and apply kfold method to find the value.
- We are then finding accuracy using `cross_val_score` to calculate the results

---

You can see that we report both the mean and the standard deviation of the performance measure. When summarizing performance measures, it is a good practice to summarize the distribution of the measures, in this case assuming a Gaussian distribution of performance (a very reasonable assumption) and recording the mean and standard deviation.

## 9.4 Leave One Out Cross Validation

You can configure cross validation so that the size of the fold is 1 (k is set to the number of observations in your dataset). This variation of cross validation is called leave-one-out cross validation. The result is a large number of performance measures that can be summarized in an effort to give a more reasonable estimate of the accuracy of your model on unseen data. A downside is that it can be a computationally more expensive procedure than k-fold cross validation. In the example below we use leave-one-out cross validation.

```
# Listing 9.5: Example of evaluating an algorithm with Leave One Out Cross Validation.
# Evaluate using Leave One Out Cross Validation
from pandas import read_csv
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
```

```
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
loocv = LeaveOneOut()
model = LogisticRegression(solver='lbfgs', max_iter=3000)
results = cross_val_score(model, X, Y, cv=loocv)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100, ))
```

```
    Accuracy: 77.604% (41.689%)
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `LeaveOneOut` and `cross_val_score` method from `sklearn.model_selection`
- We are using Logistic Regression as a model and passing data as parameter and calculating accuracy of the model.

---

You can see in the standard deviation that the score has more variance than the k-fold cross validation results described above.

## 9.5 Repeated Random Test-Train Splits

Another variation on k-fold cross validation is to create a random split of the data like the train/test split described above, but repeat the process of splitting and evaluation of the algorithm multiple times, like cross validation. This has the speed of using a train/test split and the reduction in variance in the estimated performance of k-fold cross validation. You can also repeat the process many more times as needed to improve the accuracy. A down side is that repetitions may include much of the same data in the train or the test split from run to run, introducing redundancy into the evaluation. The example below splits the data into a 67%/33% train/test split and repeats the process 10 times.

```
# Listing 9.7: Example of evaluating an algorithm with Shuffle Split Cross Validation.
# Evaluate using Shuffle Split Cross Validation
from pandas import read_csv
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
n_splits = 10
test_size = 0.33
seed = 7
kfold = ShuffleSplit(n_splits=n_splits, test_size=test_size, random_state=seed)
model = LogisticRegression(solver='lbfgs', max_iter=3000)
results = cross_val_score(model, X, Y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100, ))
```

```
    Accuracy: 76.535% (2.235%)
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `ShuffleSplit` and `cross_val_score` method from `sklearn.model_selection`
- We are spliting the data into two parts with the test data consisting of 33% of the total data and split data consisting of 67% of the total data.
- We are using Logistic Regression model as a training model and using shuffle split method to split the data into training and test data size.
- We are then calculating cross_val_score to calculate accuaracy of the data and thus sharing the mean and standard deviation.

---

We can see that in this case the distribution of the performance measure is on par with k-fold cross validation above.

## 9.6 What Techniques to Use When

This section lists some tips to consider what resampling technique to use in different circum  stances.

- Generally k-fold cross validation is the gold standard for evaluating the performance of a machine learning algorithm on unseen data with k set to 3, 5, or 10.
- Using a train/test split is good for speed when using a slow algorithm and produces performance estimates with lower bias when using large datasets.
- Techniques like leave-one-out cross validation and repeated random splits can be useful intermediates when trying to balance variance in the estimated performance, model training speed and dataset size.

The best advice is to experiment and find a technique for your problem that is fast and produces reasonable estimates of performance that you can use to make decisions. If in doubt, use 10-fold cross validation.

## 9.7 Summary

In this chapter you discovered statistical techniques that you can use to estimate the performance of your machine learning algorithms, called resampling. Specifically, you learned about:

- Train and Test Sets.
- Cross Validation.
- Leave One Out Cross Validation.
- Repeated Random Test-Train Splits.

# Chapter 10 : Machine Learning Algorithm Performance Metrics

The metrics that you choose to evaluate your machine learning algorithms are very important. Choice of metrics influences how the performance of machine learning algorithms is measured and compared. They influence how you weight the importance of different characteristics in the results and your ultimate choice of which algorithm to choose. In this chapter you will discover how to select and use different machine learning performance metrics in Python with scikit-learn. Let's get started.

## 10.1 Algorithm Evaluation Metrics

In this lesson, various different algorithm evaluation metrics are demonstrated for both classifi cation and regression type machine learning problems. In each recipe, the dataset is downloaded directly from the UCI Machine Learning repository.

- For classification metrics, the Pima Indians onset of diabetes dataset is used as demon stration. This is a binary classification problem where all of the input variables are numeric.
- For regression metrics, the Boston House Price dataset is used as demonstration. this is a regression problem where all of the input variables are also numeric.

All recipes evaluate the same algorithms, Logistic Regression for classification and Linear Regression for the regression problems. A 10-fold cross validation test harness is used to demonstrate each metric, because this is the most likely scenario you will use when employing different algorithm evaluation metrics.

A caveat in these recipes is the cross validation.cross val score function1 used to report the performance in each recipe. It does allow the use of different scoring metrics that will be discussed, but all scores are reported so that they can be sorted in ascending order (largest score is best). Some evaluation metrics (like mean squared error) are naturally descending scores (the smallest score is best) and as such are reported as negative by the `cross_validation.cross_val_score()` function. This is important to note, because some scores will be reported as negative that by definition can never be negative. I will remind you about this caveat as we work through the lesson.

## 10.2 Classification Metrics

Classification problems are perhaps the most common type of machine learning problem and as such there are a myriad of metrics that can be used to evaluate predictions for these problems. In this section we will review how to use the following metrics:

- Classification Accuracy.
- Logarithmic Loss.
- Area Under ROC Curve.
- Confusion Matrix.

## 10.2.1 Classification Accuracy

Classification accuracy is the number of correct predictions made as a ratio of all predictions made. This is the most common evaluation metric for classification problems, it is also the most misused. It is really only suitable when there are an equal number of observations in each class (which is rarely the case) and that all predictions and prediction errors are equally important, which is often not the case. Below is an example of calculating classification accuracy.

```
# Listing 10.1: Example of evaluating an algorithm by classification accuracy.
# Cross Validation Classification Accuracy
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, shuffle=True)
model = LogisticRegression(solver='lbfgs', max_iter=3000)
scoring = 'accuracy'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std(), ))
```

```
Accuracy: 0.773 (0.031)
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are then splitting the data available in to 10 different folds (`n_splits=10`) and apply kfold method to find the value.
- We are then finding accuracy using `cross_val_score` to calculate the results

---

You can see that the ratio is reported. This can be converted into a percentage by multiplying the value by 100, giving an accuracy score of approximately 77% accurate.

## 10.2.2 Logarithmic Loss

Logarithmic loss (or logloss) is a performance metric for evaluating the predictions of probabilities of membership to a given class. The scalar probability between 0 and 1 can be seen as a measure of confidence for a prediction by an algorithm. Predictions that are correct or incorrect are rewarded or punished proportionally to the confidence of the prediction. Below is an example of calculating logloss for Logistic regression predictions on the Pima Indians onset of diabetes dataset.

```
# Listing 10.3: Example of evaluating an algorithm by logloss.
# Cross Validation Classification LogLoss
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
```

```
Y = array[:,8]
kfold = KFold(n_splits=10, shuffle=True)
model = LogisticRegression(solver='lbfgs', max_iter=3000)
scoring = 'neg_log_loss'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("Logloss: %.3f (%.3f)" % (results.mean(), results.std(), ))
```

```
    Logloss: -0.483 (0.060)
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are creating a training model using LogisticRegression.
- We are further using this model and calculating Logarithmic loss by passing `scoring="neg_log_loss"` as a parameter.
- In the end, we are calculating and printing mean and standard deviation of the log loss thus calculated.

---

Smaller logloss is better with 0 representing a perfect logloss. As mentioned above, the measure is inverted to be ascending when using the `cross_val_score()` function.

## 10.2.3 Area Under ROC Curve

Area under ROC Curve (or AUC for short) is a performance metric for binary classification problems. The AUC represents a model's ability to discriminate between positive and negative classes. An area of 1.0 represents a model that made all predictions perfectly. An area of 0.5 represents a model that is as good as random. ROC can be broken down into sensitivity and specificity. A binary classification problem is really a trade-off between sensitivity and specificity.

- Sensitivity is the true positive rate also called the recall. It is the number of instances from the positive (first) class that actually predicted correctly.
- Specificity is also called the true negative rate. Is the number of instances from the negative (second) class that were actually predicted correctly.

```
# Listing 10.5: Example of evaluating an algorithm by AUC.
# Cross Validation Classification ROC AUC
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, shuffle=True)
model = LogisticRegression(solver='lbfgs', max_iter=3000)
scoring = 'roc_auc'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("AUC: %.3f (%.3f)" % (results.mean(), results.std(), ))
```

```
AUC: 0.828 (0.051)
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are creating a training model using LogisticRegression.
- We are further using this model and calculating Area under RoC by passing `scoring="roc_auc"` as a parameter.

- In the end, we are calculating and printing mean and standard deviation of the log loss thus calculated.

---

You can see the AUC is relatively close to 1 and greater than 0.5, suggesting some skill in the predictions.

### 10.2.4 Confusion Matrix

The confusion matrix is a handy presentation of the accuracy of a model with two or more classes. The table presents predictions on the x-axis and accuracy outcomes on the y-axis. The cells of the table are the number of predictions made by a machine learning algorithm. For example, a machine learning algorithm can predict 0 or 1 and each prediction may actually have been a 0 or 1. Predictions for 0 that were actually 0 appear in the cell for $prediction = 0$ and $actual = 0$, whereas predictions for 0 that were actually 1 appear in the cell for $prediction = 0$ and $actual = 1$. And so on. Below is an example of calculating a confusion matrix for a set of predictions by a Logistic Regression on the Pima Indians onset of diabetes dataset.

```
# Listing 10.7: Example of evaluating an algorithm by confusion matrix.
# Cross Validation Confusion Matrix
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
model = LogisticRegression(solver='lbfgs', max_iter=3000)
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
matrix = confusion_matrix(Y_test, predicted)
print(matrix)
```

```
[[142  20]
 [ 34  58]]
```

---

**My Understanding**

In the above code

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `confusion_matrix` from `sklearn.metrics`
- We are splitting the data into train and test where training test data size is 33% and split test data size is 67%.
- We are creating a training model using LogisticRegression.
- We are further using this model to predict the data by passing test data.
- In the end, we are creating a confusion matrix to find y_test and predicted data to calculate the matrix.

---

## 10.2.5 Classification Report

The scikit-learn library provides a convenience report when working on classification prob lems to give you a quick idea of the accuracy of a model using a number of measures. The `classification_report()` function displays the precision, recall, F1-score and support for each class. The example below demonstrates the report on the binary classification problem.

```
# Listing 10.9: Example of evaluating an algorithm by classification report.
# Cross Validation Classification Report
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
```

```
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
model = LogisticRegression(solver='lbfgs', max_iter=3000)
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
report = classification_report(Y_test, predicted)
print(report)
```

```
                   precision    recall  f1-score   support

              0.0       0.81      0.88      0.84       162
              1.0       0.74      0.63      0.68        92

         accuracy                           0.79       254
        macro avg       0.78      0.75      0.76       254
     weighted avg       0.78      0.79      0.78       254
```

---

**My Understanding**

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `confusion_matrix` from `sklearn.metrics`
- We are splitting the data into train and test where training test data size is 33% and split test data size is 67%.
- We are creating a training model using LogisticRegression.
- We are further using this model to predict the data by passing test data.
- In the end, we are using a `classification_report` to generate a report for the same.

---

## 10.3 Regression Metrics

In this section will review 3 of the most common metrics for evaluating predictions on regression machine learning problems:

- Mean Absolute Error.
- Mean Squared Error.
- $R^2$.

## 10.3.1 Mean Absolute Error

The Mean Absolute Error (or MAE) is the sum of the absolute differences between predictions and actual values. It gives an idea of how wrong the predictions were. The measure gives an idea of the magnitude of the error, but no idea of the direction (e.g. over or under predicting). The example below demonstrates calculating mean absolute error on the Boston house price dataset.

```
# Listing 10.11: Example of evaluating an algorithm by Mean Absolute Error.
# Cross Validation Regression MAE
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
filename = '/content/drive/MyDrive/Datasets/boston.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = read_csv(filename, delim_whitespace=True, names=names)
array = dataframe.values
X = array[:, 0:13]
Y = array[:, 13]
kfold = KFold(n_splits=10, shuffle=True)
model = LinearRegression()
scoring = 'neg_mean_absolute_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("MAE: %.3f (%.3f)" % (results.mean(), results.std(), ))
```

```
    MAE: -3.375 (0.487)
```

---

**My Understanding**

- We are importing data from `boston.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are using KFold method to split teh data into 10 different folds.
- We are creating a training model using LogisticRegression.
- We are using `cross_val_score` method to calcuate the required results by passing `neg_mean_absolute_error` as a `scoring` parameter.
- In the end, we are printing the mean and standard deviation by calculating the accuracy.

---

## 10.3.2 Mean Squared Error

The Mean Squared Error (or MSE) is much like the mean absolute error in that it provides a gross idea of the magnitude of error. Taking the square root of the mean squared error converts the units back to the original units of the output variable and can be meaningful for description and presentation. This is called the Root Mean Squared Error (or RMSE). The example below provides a demonstration of calculating mean squared error.

```
# Listing 10.13: Example of evaluating an algorithm by Mean Squared Error.
# Cross Validation Regression MSE
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
filename = '/content/drive/MyDrive/Datasets/boston.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = read_csv(filename, delim_whitespace=True, names=names)
array = dataframe.values
X = array[:, 0:13]
Y = array[:, 13]
kfold = KFold(n_splits=10, shuffle=True)
model = LinearRegression()
```

```
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("MSE: %.3f (%.3f)" % (results.mean(), results.std(), ))
```

```
    MSE: -23.962 (8.274)
```

---

**My Understanding**

- We are importing data from `boston.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are using KFold method to split teh data into 10 different folds.
- We are creating a training model using LogisticRegression.
- We are using `cross_val_score` method to calcuate the required results by passing `neg_mean_asquared_error` as a `scoring` parameter.
- In the end, we are printing the mean and standard deviation by calculating the accuracy.

---

This metric too is inverted so that the results are increasing. Remember to take the absolute value before taking the square root if you are interested in calculating the RMSE.

### 10.3.3 $R^2$ Metric

The $R^2$ (or R Squared) metric provides an indication of the goodness of fit of a set of predictions to the actual values. In statistical literature this measure is called the coefficient of determination. This is a value between 0 and 1 for no-fit and perfect fit respectively. The example below provides a demonstration of calculating the mean $R^2$ for a set of predictions.

```
# Listing 10.15: Example of evaluating an algorithm by R Squared.
# Cross Validation Regression R^2
from pandas import read_csv
from sklearn.model_selection import KFold
```

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
filename = '/content/drive/MyDrive/Datasets/boston.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = read_csv(filename, delim_whitespace=True, names=names)
array = dataframe.values
X = array[:, 0:13]
Y = array[:, 13]
kfold = KFold(n_splits=10, shuffle=True)
model = LinearRegression()
scoring = 'r2'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("R^2: %.3f (%.3f)" % (results.mean(), results.std(), ))
```

```
    R^2: 0.693 (0.078)
```

---

**My Understanding**

In the above code,

- We are importing data from `boston.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are using KFold method to split teh data into 10 different folds.
- We are creating a training model using LogisticRegression.
- We are using `cross_val_score` method to calcuate the required results by passing `r2` as a `scoring` parameter.
- In the end, we are printing the mean and standard deviation by calculating the accuracy.

---

## 10.4 Summary

In this chapter you discovered metrics that you can use to evaluate your machine learning algorithms. You learned about three classification metrics: Accuracy, Logarithmic Loss and Area Under ROC Curve. You also learned about two convenience methods for classification

prediction results: the Confusion Matrix and the Classification Report. Finally, you also learned about three metrics for regression problems:

# Chapter 11 : Spot-Check Classification Algorithms

Spot-checking is a way of discovering which algorithms perform well on your machine learning problem. You cannot know which algorithms are best suited to your problem beforehand. You must trial a number of methods and focus attention on those that prove themselves the most promising. In this chapter you will discover six machine learning algorithms that you can use when spot-checking your classification problem in Python with scikit-learn. After completing this lesson you will know:

1. How to spot-check machine learning algorithms on a classification problem.
2. How to spot-check two linear classification algorithms.
3. How to spot-check four nonlinear classification algorithms.

Let's get started.

## 11.1 Algorithm Spot-Checking

You cannot know which algorithm will work best on your dataset beforehand. You must use trial and error to discover a shortlist of algorithms that do well on your problem that you can then double down on and tune further. I call this process spot-checking.

The question is not: *What algorithm should I use on my dataset?* Instead it is: *What algorithms should I spot-check on my dataset?* You can guess at what algorithms might do well on your dataset, and this can be a good starting point. I recommend trying a mixture of algorithms and see what is good at picking out the structure in your data. Below are some suggestions when spot-checking algorithms on your dataset:

- Try a mixture of algorithm representations (e.g. instances and trees).
- Try a mixture of learning algorithms (e.g. different algorithms for learning the same type of representation).
- Try a mixture of modeling types (e.g. linear and nonlinear functions or parametric and nonparametric).

Let's get specific. In the next section, we will look at algorithms that you can use to spot-check on your next classification machine learning project in Python.

## 11.2 Algorithms Overview

We are going to take a look at six classification algorithms that you can spot-check on your dataset. Starting with two linear machine learning algorithms:

- Logistic Regression.
- Linear Discriminant Analysis.

Then looking at four nonlinear machine learning algorithms:

- $k$-Nearest Neighbors.
- Naive Bayes.
- Classification and Regression Trees.
- Support Vector Machines. Each recipe is demonstrated on the Pima Indians onset of Diabetes dataset. A test harness using 10-fold cross validation is used to demonstrate how to spot-check each machine learning algorithm and mean accuracy measures are used to indicate algorithm performance. The recipes assume that you know about each machine learning algorithm and how to use them. We will not go into the API or parameterization of each algorithm.

## 11.3 Linear Machine Learning Algorithms

This section demonstrates minimal recipes for how to use two linear machine learning algorithms: logistic regression and linear discriminant analysis.

### 11.3.1 Logistic Regression

Logistic regression assumes a Gaussian distribution for the numeric input variables and can model binary classification problems. You can construct a logistic regression model using the `LogisticRegression` class .

```
# Listing 11.1: Example of the logistic regression algorithm.
# Logistic Regression Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, shuffle=True)
model = LogisticRegression(solver='lbfgs', max_iter=3000)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
    0.7761107313738893
```

---

**My Understanding**

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- Additionally, we are importing `LogisticRegression` as a linear model
- We are using `cross_val_score` method to calcuate the required results
- In the end, we are printing the mean by calculating the accuracy.

---

## 11.3.2 Linear Discriminant Analysis

Linear Discriminant Analysis or LDA is a statistical technique for binary and multiclass classification. It too assumes a Gaussian distribution for the numerical input variables. You can construct an LDA model using the `LinearDiscriminantAnalysis` class.

```
# Listing 11.3: Example of the LDA algorithm.
# LDA Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, shuffle=True)
model = LinearDiscriminantAnalysis()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
    0.7603383458646616
```

---

**My Understanding**

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are using KFold method to split the data into 10 different folds.
- We are creating a training model using `LinearDiscriminantAnalysis`.
- In the end, we are printing the mean by passing cross_val_score in the model and kfold.

---

## 11.4 Nonlinear Machine Learning Algorithms

This section demonstrates minimal recipes for how to use 4 nonlinear machine learning algorithms.

## 11.4.1 k-Nearest Neighbors

The $k$-Nearest Neighbors algorithm (or KNN) uses a distance metric to find the k most similar instances in the training data for a new instance and takes the mean outcome of the neighbors as the prediction. You can construct a KNN model using the `KNeighborsClassifier` class.

```
# Listing 11.5: Example of the KNN algorithm.
# KNN Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, shuffle=True)
model = KNeighborsClassifier()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
    0.722676008202324
```

---

**My Understanding**

From the above code, we can understand that:

- We are importing data from `dibetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are using KFold method to split th data into 10 different folds.
- We are creating a training model using `KNeighborsClassifier`.
- We are using `cross_val_score` method to calcuate the required results by passing data obtained by splitting using kfolds.
- In the end, we are printing the mean by calculating the final results.

---

## 11.4.2 Naive Bayes

Naive Bayes calculates the probability of each class and the conditional probability of each class given each input value. These probabilities are estimated for new data and multiplied together, assuming that they are all independent (a simple or naive assumption). When working with real-valued data, a Gaussian distribution is assumed to easily estimate the probabilities for input variables using the Gaussian Probability Density Function. You can construct a Naive Bayes model using the `GaussianNB` class.

```
# Listing 11.7: Example of the Naive Bayes algorithm.
# Guassiain Naive Bayes Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, shuffle=True)
model = GaussianNB()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
    0.7564593301435407
```

---

**My Understanding**

From the above code, we can understand that:

- We are importing data from `dibetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are using KFold method to split th data into 10 different folds.

- We are creating a training model using `GuassianNB`.
- We are using `cross_val_score` method to calcuate the required results by passing data obtained by splitting using kfolds.
- In the end, we are printing the mean by calculating the final results.

---

## 11.4.3 Classification and Regression Trees

Classification and Regression Trees (CART or just decision trees) construct a binary tree from the training data. Split points are chosen greedily by evaluating each attribute and each value of each attribute in the training data in order to minimize a cost function (like the Gini index). You can construct a CART model using the `DecisionTreeClassifier` class.

```
# Listing 11.9: Example of the CART algorithm.
# CART Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, shuffle=True)
model = DecisionTreeClassifier()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
    0.693984962406015
```

---

**My Understanding**

From the above code, we can understand that:

- We are importing data from `dibetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are using KFold method to split th data into 10 different folds.
- We are creating a training model using `DecisionTreeClassifier`.
- We are using `cross_val_score` method to calcuate the required results by passing data obtained by splitting using kfolds.
- In the end, we are printing the mean by calculating the final results.

---

## 11.4.4 Support Vector Machines

Support Vector Machines (or SVM) seek a line that best separates two classes. Those data instances that are closest to the line that best separates the classes are called support vectors and influence where the line is placed. SVM has been extended to support multiple classes. Of particular importance is the use of different kernel functions via the kernel parameter. A powerful Radial Basis Function is used by default. You can construct an SVM model using the `SVC` class.

```
# Listing 11.11: Example of the SVM algorithm.
# SVM Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, shuffle=True)
model = SVC()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.7592617908407382
```

---

**My Understanding**

From the above code, we can understand that:

- We are importing data from `dibetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are using KFold method to split th data into 10 different folds.
- We are creating a training model using `SVC`.
- We are using `cross_val_score` method to calcuate the required results by passing data obtained by splitting using kfolds.
- In the end, we are printing the mean by calculating the final results.

---

## 11.5 Summary

In this chapter you discovered 6 machine learning algorithms that you can use to spot-check on your classification problem in Python using scikit-learn. Specifically, you learned how to spot-check two linear machine learning algorithms: Logistic Regression and Linear Discriminant Analysis. You also learned how to spot-check four nonlinear algorithms: $k$-Nearest Neighbors, Naive Bayes, Classification and Regression Trees and Support Vector Machines

# Chapter 12 : Spot-Check Regression Algorithms

Spot-checking is a way of discovering which algorithms perform well on your machine learning problem. You cannot know which algorithms are best suited to your problem beforehand. You must trial a number of methods and focus attention on those that prove themselves the most promising. In this chapter you will discover six machine learning algorithms that you can use when spot-checking your regression problem in Python with scikit-learn. After completing this lesson you will know:

1. How to spot-check machine learning algorithms on a regression problem.

2. How to spot-check four linear regression algorithms.
3. How to spot-check three nonlinear regression algorithms.

Let's get started

## 12.1 Algorithms Overview

In this lesson we are going to take a look at seven regression algorithms that you can spot-check on your dataset. Starting with four linear machine learning algorithms:

- Linear Regression.
- Ridge Regression.
- LASSO Linear Regression.
- Elastic Net Regression.

Then looking at three nonlinear machine learning algorithms:

- $k$-Nearest Neighbors.
- Classification and Regression Trees.
- Support Vector Machines

Each recipe is demonstrated on the Boston House Price dataset. This is a regression problem where all attributes are numeric. A test harness with 10-fold cross validation is used to demonstrate how to spot-check each machine learning algorithm and mean squared error measures are used to indicate algorithm performance. Note that mean squared error values are inverted (negative). This is a quirk of the cross val score() function used that requires all algorithm metrics to be sorted in ascending order (larger value is better). The recipes assume that you know about each machine learning algorithm and how to use them. We will not go into the API or parameterization of each algorithm.

## 12.2 Linear Machine Learning Algorithms

This section provides examples of how to use four different linear machine learning algorithms for regression in Python with scikit-learn.

## 12.2.1 Linear Regression

Linear regression assumes that the input variables have a Gaussian distribution. It is also assumed that input variables are relevant to the output variable and that they are not highly correlated with each other (a problem called collinearity). You can construct a linear regression model using the `LinearRegression` class.

```
# Listing 12.1: Example of the linear regression algorithm.
# Linear Regression
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
filename = '/content/drive/MyDrive/Datasets/boston.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = read_csv(filename,delim_whitespace=True, names=names)
array = dataframe.values
X = array[:, 0:13]
Y = array[:, 13]
kfold = KFold(n_splits=10, shuffle=True)
model = LinearRegression()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print(results.mean())
```

```
    -24.307160844925225
```

---

**My Understanding**

In the above code,

- We are importing data from `boston.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `LinearRegression` from `sklearn.linear_model`

- We are using KFold method to split the data into 10 different folds.
- We are using `cross_val_score` method to calcuate the required results by passing `neg_mean_squared_error` as a `scoring` parameter.
- In the end, we are printing the mean and standard deviation by calculating the accuracy.

---

## 12.2.2 Ridge Regression

Ridge regression is an extension of linear regression where the loss function is modified to minimize the complexity of the model measured as the sum squared value of the coefficient values (also called the L2-norm). You can construct a ridge regression model by using the `Ridge` class.

```
# Listing 12.3: Example of the ridge regression algorithm.
# Ridge Regression
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Ridge
filename = '/content/drive/MyDrive/Datasets/boston.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = read_csv(filename,delim_whitespace=True, names=names)
array = dataframe.values
X = array[:, 0:13]
Y = array[:, 13]
kfold = KFold(n_splits=10, shuffle=True)
model = Ridge()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print(results.mean())
```

```
    -24.230183609911442
```

---

**My Understanding**

In the above code,

- We are importing data from `boston.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `Ridge` from `sklearn.linear_model`
- We are using KFold method to split the data into 10 different folds.
- We are using `cross_val_score` method to calcuate the required results by passing `neg_mean_squared_error` as a `scoring` parameter.
- In the end, we are printing the mean and standard deviation by calculating the accuracy.

---

## 12.2.3 LASSO Regression

The Least Absolute Shrinkage and Selection Operator (or LASSO for short) is a modification of linear regression, like ridge regression, where the loss function is modified to minimize the complexity of the model measured as the sum absolute value of the coefficient values (also called the L1-norm). You can construct a LASSO model by using the `Lasso` class .

```
# Listing 12.5: Example of the LASSO regression algorithm.
# LASSO Regression
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Lasso
filename = '/content/drive/MyDrive/Datasets/boston.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = read_csv(filename,delim_whitespace=True, names=names)
array = dataframe.values
X = array[:, 0:13]
Y = array[:, 13]
kfold = KFold(n_splits=10, shuffle=True)
```

```
model = Lasso()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print(results.mean())
```

```
    -28.913994871329937
```

---

**My Understanding**

In the above code,

- We are importing data from `boston.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `Lasso` from `sklearn.linear_model`
- We are using KFold method to split the data into 10 different folds.
- We are using `cross_val_score` method to calcuate the required results by passing `neg_mean_squared_error` as a `scoring` parameter.
- In the end, we are printing the mean and standard deviation by calculating the accuracy.

---

## 12.2.4 ElasticNet Regression

ElasticNet is a form of regularization regression that combines the properties of both Ridge Regression and LASSO regression. It seeks to minimize the complexity of the regression model (magnitude and number of regression coefficients) by penalizing the model using both the L2-norm (sum squared coefficient values) and the L1-norm (sum absolute coefficient values). You can construct an ElasticNet model using the `ElasticNet` class.

```
# Listing 12.7: Example of the ElasticNet regression algorithm.
# ElasticNet Regression
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

```
from sklearn.linear_model import ElasticNet
filename = '/content/drive/MyDrive/Datasets/boston.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = read_csv(filename,delim_whitespace=True, names=names)
array = dataframe.values
X = array[:, 0:13]
Y = array[:, 13]
kfold = KFold(n_splits=10, shuffle=True)
model = ElasticNet()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print(results.mean())
```

```
    -28.214395256080856
```

---

**My Understanding**

In the above code,

- We are importing data from `boston.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `ElasticNet` from `sklearn.linear_model`
- We are using KFold method to split the data into 10 different folds.
- We are using `cross_val_score` method to calcuate the required results by passing `neg_mean_squared_error` as a `scoring` parameter.
- In the end, we are printing the mean and standard deviation by calculating the accuracy.

---

## 12.3 Nonlinear Machine Learning Algorithms

This section provides examples of how to use three different nonlinear machine learning algorithms for regression in Python with scikit-learn.

## 12.3.1 K-Nearest Neighbors

The $k$-Nearest Neighbors algorithm (or KNN) locates the k most similar instances in the training dataset for a new data instance. From the k neighbors, a mean or median output variable is taken as the prediction. Of note is the distance metric used (the `metric` argument). The Minkowski distance is used by default, which is a generalization of both the Euclidean distance (used when all inputs have the same scale) and Manhattan distance (for when the scales of the input variables differ). You can construct a KNN model for regression using the `KNeighborsRegressor` class .

```
# Listing 12.9: Example of the KNN regression algorithm.
# KNN Regression
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsRegressor
filename = '/content/drive/MyDrive/Datasets/boston.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = read_csv(filename,delim_whitespace=True, names=names)
array = dataframe.values
X = array[:, 0:13]
Y = array[:, 13]
kfold = KFold(n_splits=10, shuffle=True)
model = KNeighborsRegressor()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print(results.mean())
```

```
    -38.87997576470588
```

---

**My Understanding**

In the above code,

- We are importing data from `boston.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`

- We are also importing `KNeighborsRegressor` from `sklearn.neighbors`
- We are using KFold method to split the data into 10 different folds.
- We are using `cross_val_score` method to calcuate the required results by passing `neg_mean_squared_error` as a `scoring` parameter.
- In the end, we are printing the mean and standard deviation by calculating the accuracy.

---

## 12.3.2 Classification and Regression Trees

Decision trees or the Classification and Regression Trees (CART as they are known) use the train ing data to select the best points to split the data in order to minimize a cost metric. The default cost metric for regression decision trees is the mean squared error, specified in the criterion parameter. You can create a CART model for regression using the `DecisionTreeRegressor` class .

```
# Listing 12.11: Example of the CART regression algorithm.
# Decision Tree Regression
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
filename = '/content/drive/MyDrive/Datasets/boston.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = read_csv(filename,delim_whitespace=True, names=names)
array = dataframe.values
X = array[:, 0:13]
Y = array[:, 13]
kfold = KFold(n_splits=10, shuffle=True)
model = DecisionTreeRegressor()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print(results.mean())
```

```
    -17.574881568627454
```

---

**My Understanding**

In the above code,

- We are importing data from `boston.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `DecisionTreeRegressor` from `sklearn.tree`
- We are using KFold method to split the data into 10 different folds.
- We are using `cross_val_score` method to calcuate the required results by passing `neg_mean_squared_error` as a `scoring` parameter.
- In the end, we are printing the mean and standard deviation by calculating the accuracy.

---

## 12.3.3 Support Vector Machines

Support Vector Machines (SVM) were developed for binary classification. The technique has been extended for the prediction real-valued problems called Support Vector Regression (SVR). Like the classification example, SVR is built upon the LIBSVM library. You can create an SVM model for regression using the `SVR` class.

```
# Listing 12.13: Example of the SVM regression algorithm.
# SVM Regression
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVR
filename = '/content/drive/MyDrive/Datasets/boston.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = read_csv(filename, delim_whitespace=True, names=names)
array = dataframe.values
X = array[:, 0:13]
Y = array[:, 13]
kfold = KFold(n_splits=10, shuffle=True)
```

```
model = SVR()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print(results.mean())
```

```
    -67.41770451628356
```

---

**My Understanding**

In the above code,

- We are importing data from `boston.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `SVR` from `sklearn.svm`
- We are using KFold method to split the data into 10 different folds.
- We are using `cross_val_score` method to calcuate the required results by passing `neg_mean_squared_error` as a `scoring` parameter.
- In the end, we are printing the mean and standard deviation by calculating the accuracy.

---

## 12.4 Summary

In this chapter you discovered how to spot-check machine learning algorithms for regression problems in Python using scikit-learn. Specifically, you learned about four linear machine learning algorithms: Linear Regression, Ridge Regression, LASSO Linear Regression and Elastic Net Regression. You also learned about three nonlinear algorithms: k-Nearest Neighbors, Classification and Regression Trees and Support Vector Machines.

# Chapter 13 : Compare Machine Learning Algorithms

It is important to compare the performance of multiple different machine learning algorithms consistently. In this chapter you will discover how you can create a test harness to compare multiple different machine learning algorithms in Python with scikit-learn. You can use this test harness as a template on your own machine learning problems and add more and different algorithms to compare. After completing this lesson you will know:

1. How to formulate an experiment to directly compare machine learning algorithms.
2. A reusable template for evaluating the performance of multiple algorithms on one dataset.
3. How to report and visualize the results when comparing algorithm performance.

Let's get started

## 13.1 Choose The Best Machine Learning Model

When you work on a machine learning project, you often end up with multiple good models to choose from. Each model will have different performance characteristics. Using resampling methods like cross validation, you can get an estimate for how accurate each model may be on unseen data. You need to be able to use these estimates to choose one or two best models from the suite of models that you have created.

When you have a new dataset, it is a good idea to visualize the data using different techniques in order to look at the data from different perspectives. The same idea applies to model selection. You should use a number of different ways of looking at the estimated accuracy of your machine learning algorithms in order to choose the one or two algorithm to finalize. A way to do this is to use visualization methods to show the average accuracy, variance and other properties of the distribution of model accuracies. In the next section you will discover exactly how you can do that in Python with scikit-learn.

## 13.2 Compare Machine Learning Algorithms Consistently

The key to a fair comparison of machine learning algorithms is ensuring that each algorithm is evaluated in the same way on the same data. You can achieve this by forcing each algorithm to be evaluated on a consistent test harness. In the example below six different classification algorithms are compared on a single dataset:

- Logistic Regression.
- Linear Discriminant Analysis.

- k-Nearest Neighbors.
- Classification and Regression Trees.
- Naive Bayes.
- Support Vector Machines.

The dataset is the Pima Indians onset of diabetes problem. The problem has two classes and eight numeric input variables of varying scales. The 10-fold cross validation procedure is used to evaluate each algorithm, importantly configured with the same random seed to ensure that the same splits to the training data are performed and that each algorithm is evaluated in precisely the same way. Each algorithm is given a short name, useful for summarizing results afterward.

```
# Listing 13.1: Example of comparing multiple algorithms.
# Compare Algorithms
from pandas import read_csv
from matplotlib import pyplot
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
# load dataset
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# prepare models
models = []
models.append(('LR', LogisticRegression(solver='lbfgs', max_iter=3000)))
```

```
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
# Evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
  kfold = KFold(n_splits=10, shuffle=True)
  cv_results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  print("%s: %f (%f)" % (name, cv_results.mean(), cv_results.std(), ))
# boxplot algorithm comparison
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```

```
LR: 0.773274 (0.066818)
LDA: 0.773342 (0.035625)
KNN: 0.716148 (0.048251)
CART: 0.706904 (0.044523)
NB: 0.751299 (0.034102)
SVM: 0.759057 (0.039958)
```

---

**My Understanding**

In the above code,

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `LogisticRegression`, `DecisionTreeClassifier`, `KNeighborsClassifier`, `LinearDiscriminantAnalysis`, `GaussianNB` and `SVC` from `sklearn`
- We are also importing `pyplot` from `matplotlib`
- We are iterating through the loop to calculate the accuracy of each model and plot the same resutl on the a boxplot
- We are using KFold method to split the data into 10 different folds.
- We are using `accuracy` as a scoring param to calcuate the required results by passing it as a `scoring` parameter.
- In the end, we are comparing all the algorithms accuracy and thus printing the result and corresponding graph to understand the same.

---

Running the example provides a list of each algorithm short name, the mean accuracy and the standard deviation accuracy.

The example also provides a box and whisker plot showing the spread of the accuracy scores across each cross validation fold for each algorithm.

From these results, it would suggest that both logistic regression and linear discriminant analysis are perhaps worthy of further study on this problem.

## 13.3 Summary

In this chapter you discovered how to evaluate multiple different machine learning algorithms on a dataset in Python with scikit-learn. You learned how to both use the same test harness to evaluate the algorithms and how to summarize the results both numerically and using a box and whisker plot. You can use this recipe as a template for evaluating multiple algorithms on your own problems.

# Chapter 14 : Automate Machine Learning Workflows with Pipelines

There are standard workflows in a machine learning project that can be automated. In Python scikit-learn, Pipelines help to clearly define and automate these workflows. In this chapter you will discover Pipelines in scikit-learn and how you can automate common machine learning workflows. After completing this lesson you will know:

1. How to use pipelines to minimize data leakage.
2. How to construct a data preparation and modeling pipeline.
3. How to construct a feature extraction and modeling pipeline.

Let's get started.

## 14.1 Automating Machine Learning Workflows

There are standard workflows in applied machine learning. Standard because they overcome common problems like data leakage in your test harness. Python scikit-learn provides a Pipeline utility to help automate machine learning workflows. Pipelines work by allowing for a linear sequence of data transforms to be chained together culminating in a modeling process that can be evaluated.

The goal is to ensure that all of the steps in the pipeline are constrained to the data available for the evaluation, such as the training dataset or each fold of the cross validation procedure. You can learn more about Pipelines in scikit-learn by reading the Pipeline section of the user guide. You can also review the API documentation for the Pipeline and FeatureUnion classes and the `pipeline` module .

## 14.2 Data Preparation and Modeling Pipeline

An easy trap to fall into in applied machine learning is leaking data from your training dataset to your test dataset. To avoid this trap you need a robust test harness with strong separation of training and testing. This includes data preparation. Data preparation is one easy way to leak knowledge of the whole training dataset to the algorithm. For example, preparing your data using normalization or standardization on the entire training dataset before learning would not be a valid test because the training dataset would have been influenced by the scale of the data in the test set.

Pipelines help you prevent data leakage in your test harness by ensuring that data preparation like standardization is constrained to each fold of your cross validation procedure. The example below demonstrates this important data preparation and model evaluation workflow on the Pima Indians onset of diabetes dataset. The pipeline is defined with two steps:

1. Standardize the data.
2. Learn a Linear Discriminant Analysis model.

The pipeline is then evaluated using 10-fold cross validation.

```
# Listing 14.1: Example of a Pipeline to standardize and model data.
# Create a pipeline that standardizes the data then creates a model
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# load data
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# create pipeline
estimators = []
estimators.append(('standardize', StandardScaler()))
```

```
estimators.append(('lda', LinearDiscriminantAnalysis()))
model = Pipeline(estimators)
# evaluate pipeline
kfold = KFold(n_splits=10, shuffle=True)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
    0.7722146274777855
```

---

**My Understanding**

In the above code,

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `StandardScaler` and `Pipeline` from `sklearn`
- We are also importing `LinearDiscriminantAnalysis` from `sklearn.discriminant_analysis`
- We are using `StandardScalers` and `LinearDiscriminantAnalysis` as our estimators and passing this estimators in a pipeline
- We are using KFold method to split the data into 10 different folds.
- We are passing kfold generated above as cross validation generator
- In the end, we are printing the mean of the results

---

Notice how we create a Python list of steps that are provided to the Pipeline for process the data. Also notice how the Pipeline itself is treated like an estimator and is evaluated in its entirety by the k-fold cross validation procedure. Running the example provides a summary of accuracy of the setup on the dataset.

## 14.3 Feature Extraction and Modeling Pipeline

Feature extraction is another procedure that is susceptible to data leakage. Like data preparation, feature extraction procedures must be restricted to the data in your training dataset. The pipeline provides a handy tool called the FeatureUnion which allows the results of multiple

feature selection and extraction procedures to be combined into a larger dataset on which a model can be trained. Importantly, all the feature extraction and the feature union occurs within each fold of the cross validation procedure. The example below demonstrates the pipeline defined with four steps:

1. Feature Extraction with Principal Component Analysis (3 features).
2. Feature Extraction with Statistical Selection (6 features).
3. Feature Union.
4. Learn a Logistic Regression Model.

The pipeline is then evaluated using 10-fold cross validation

```
# Listing 14.3: Example of a Pipeline extract and combine features before modeling.
# Create a pipeline that extracts features from the data then creates a model
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest
# load data
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# create feature union
features = []
features.append(('pca', PCA(n_components=3)))
features.append(('select_best', SelectKBest(k=6)))
feature_union = FeatureUnion(features)
# create pipeline
estimators = []
```

```
estimators.append(('feature_union', feature_union))
estimators.append(('logistic', LogisticRegression(solver='lbfgs', max_iter=3000)))
model = Pipeline(estimators)
# evaluate pipeline
kfold = KFold(n_splits=10, shuffle=True)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
    0.7774265208475735
```

---

**My Understanding**

In the above code,

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `FeatureUnion` and `Pipeline` from `sklearn.pipeline`
- We are also importing `LogisticRegression` from `sklearn.discriminant_analysis`
- Additonally importing `PCA` and `SelectKBest` methods
- We are passing `PCA` and `SelectKBest` to FeatureUnion to concatenate results of multiple transformer objects
- We are using `feature_union` and `LogisticRegression` as our estimators and passing this estimators in a Pipeline.
- We are using KFold method to split the data into 10 different folds.
- We are passing kfold generated above as cross validation generator
- In the end, we are printing the mean of the results

---

Notice how the FeatureUnion is it's own Pipeline that in turn is a single step in the final Pipeline used to feed Logistic Regression. This might get you thinking about how you can start embedding pipelines within pipelines. Running the example provides a summary of accuracy of the setup on the dataset

## 14.4 Summary

In this chapter you discovered the difficulties of data leakage in applied machine learning. You discovered the Pipeline utilities in Python scikit-learn and how they can be used to automate standard applied machine learning workflows. You learned how to use Pipelines in two important use cases:

- Data preparation and modeling constrained to each fold of the cross validation procedure.
- Feature extraction and feature union constrained to each fold of the cross validation procedure

# Chapter 15 : Improve Performance with Ensembles

Ensembles can give you a boost in accuracy on your dataset. In this chapter you will discover how you can create some of the most powerful types of ensembles in Python using scikit-learn. This lesson will step you through Boosting, Bagging and Majority Voting and show you how you can continue to ratchet up the accuracy of the models on your own datasets. After completing this lesson you will know:

1. How to use bagging ensemble methods such as bagged decision trees, random forest and extra trees.
2. How to use boosting ensemble methods such as AdaBoost and stochastic gradient boosting.
3. How to use voting ensemble methods to combine the predictions from multiple algorithms.

Let's get started.

## 15.1 Combine Models Into Ensemble Predictions

The three most popular methods for combining the predictions from different models are:

- **Bagging.** Building multiple models (typically of the same type) from different subsamples of the training dataset.
- **Boosting.** Building multiple models (typically of the same type) each of which learns to fix the prediction errors of a prior model in the sequence of models.
- **Voting.** Building multiple models (typically of differing types) and simple statistics (like calculating the mean) are used to combine predictions.

This assumes you are generally familiar with machine learning algorithms and ensemble methods and will not go into the details of how the algorithms work or their parameters. The Pima Indians onset of Diabetes dataset is used to demonstrate each algorithm. Each ensemble algorithm is demonstrated using 10-fold cross validation and the classification accuracy performance metric.

## 15.2 Bagging Algorithms

Bootstrap Aggregation (or Bagging) involves taking multiple samples from your training dataset (with replacement) and training a model for each sample. The final output prediction is averaged across the predictions of all of the sub-models. The three bagging models covered in this section are as follows:

- Bagged Decision Trees.
- Random Forest.
- Extra Trees.

### 15.2.1 Bagged Decision Trees

Bagging performs best with algorithms that have high variance. A popular example are decision trees, often constructed without pruning. In the example below is an example of using the BaggingClassifier with the Classification and Regression Trees algorithm (`DecisionTreeClassifier`). A total of 100 trees are created.

```
# Listing 15.1: Example of Bagged Decision Trees Ensemble Algorithm.
# Bagged Decision Trees for Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
```

```
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = KFold(n_splits=10, shuffle=True)
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_state=seed)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
    0.7657552973342447
```

---

**My Understanding**

In the above code,

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `BaggingClassifier` from `sklearn.ensemble`
- We are also importing `DecisionTreeClassifier` from `sklearn.tree`
- We are passing kfold generated above as cross validation generator
- We are using KFold method to split the data into 10 different folds.
- We are using `DecisionTreeClassifier()` as a model.
- Next we are passing the above model as a base estimator for `BaggingClassifier`
- Next we are passing the model generated from the `BaggingClassifier` to calculate the cross_validation score
- In the end, we are printing the mean as the results

---

Running the example, we get a robust estimate of model accuracy.

## 15.2.2 Random Forest

Random Forests is an extension of bagged decision trees. Samples of the training dataset are taken with replacement, but the trees are constructed in a way that reduces the correlation between individual classifiers. Specifically, rather than greedily choosing the best split point in the construction of each tree, only a random subset of features are considered for each split. You can construct a Random Forest model for classification using the `RandomForestClassifier` class. The example below demonstrates using Random Forest for classification with 100 trees and split points chosen from a random selection of 3 features.

```
# Listing 15.3: Example of Random Forest Ensemble Algorithm.
# Random Forest Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
num_trees = 100
max_features = 3
kfold = KFold(n_splits=10, shuffle=True)
model = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
    0.7668660287081339
```

---

**My Understanding**

In the above code,

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`

- We are also importing `RandomForestClassifier` from `sklearn.ensemble`
- We are passing kfold generated above as cross validation generator
- We are using KFold method to split the data into 10 different folds.
- Next we are using RandomForestClassifier as model and passing the kfold generated above as a model
- In the end, we are printing the mean as the results

---

Running the example provides a mean estimate of classification accuracy.

### 15.2.3 Extra Trees

Extra Trees are another modification of bagging where random trees are constructed from samples of the training dataset. You can construct an Extra Trees model for classification using the `ExtraTreesClassifier` class . The example below provides a demonstration of extra trees with the number of trees set to 100 and splits chosen from 7 random features.

```
# Listing 15.5: Example of Extra Trees Ensemble Algorithm.
# Extra Trees Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import ExtraTreesClassifier
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
num_trees = 100
max_features = 7
kfold = KFold(n_splits=10, shuffle=True)
model = ExtraTreesClassifier(n_estimators=num_trees, max_features=max_features)
```

```
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
    0.762952836637047
```

---

**My Understanding**

In the above code,

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `ExtraTreesClassifier` from `sklearn.ensemble`
- We are passing kfold generated above as cross validation generator
- We are using KFold method to split the data into 10 different folds.
- Next we are using `ExtraTreesClassifier` as model and passing the kfold generated above as a model
- In the end, we are printing the mean as the results

---

Running the example provides a mean estimate of classification accuracy.

## 15.3 Boosting Algorithms

Boosting ensemble algorithms creates a sequence of models that attempt to correct the mistakes of the models before them in the sequence. Once created, the models make predictions which may be weighted by their demonstrated accuracy and the results are combined to create a final output prediction. The two most common boosting ensemble machine learning algorithms are:

- AdaBoost.
- Stochastic Gradient Boosting.

### 15.3.1 AdaBoost

AdaBoost was perhaps the first successful boosting ensemble algorithm. It generally works by weighting instances in the dataset by how easy or difficult they are to classify, allowing the algorithm to pay or less attention to them in the construction of subsequent models. You can construct an AdaBoost model for classification using the `AdaBoostClassifier` class . The example below demonstrates the construction of 30 decision trees in sequence using the AdaBoost algorithm.

```
# Listing 15.7: Example of AdaBoost Ensemble Algorithm.
# AdaBoost Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import AdaBoostClassifier
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
num_trees = 30
seed=7
kfold = KFold(n_splits=10, shuffle=True)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
    0.7330314422419686
```

---

**My Understanding**

In the above code,

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `AdaBoostClassifier` from `sklearn.ensemble`

- We are passing kfold generated above as cross validation generator
- We are using KFold method to split the data into 10 different folds.
- Next we are using `AdaBoostClassifier` as model and passing the kfold generated above as a model
- In the end, we are printing the mean as the results

---

Running the example provides a mean estimate of classification accuracy.

## 15.3.2 Stochastic Gradient Boosting

Stochastic Gradient Boosting (also called Gradient Boosting Machines) are one of the most sophisticated ensemble techniques. It is also a technique that is proving to be perhaps one of the best techniques available for improving performance via ensembles. You can construct a Gradient Boosting model for classification using the `GradientBoostingClassifier` class . The example below demonstrates Stochastic Gradient Boosting for classification with 100 trees.

```
# Listing 15.9: Example of Stochastic Gradient Boosting Ensemble Algorithm.
# Stochastic Gradient Boosting Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import GradientBoostingClassifier
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 100
kfold = KFold(n_splits=10, shuffle=True)
model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
```

```
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
    0.7630041011619959
```

---

**My Understanding**

In the above code,

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `GradientBoostingClassifier` from `sklearn.ensemble`
- We are passing kfold generated above as cross validation generator
- We are using KFold method to split the data into 10 different folds.
- Next we are using `GradientBoostingClassifier` as model and passing the kfold generated above as a model
- In the end, we are printing the mean as the results

---

Running the example provides a mean estimate of classification accuracy.

## 15.4 Voting Ensemble

Voting is one of the simplest ways of combining the predictions from multiple machine learning algorithms. It works by first creating two or more standalone models from your training dataset. A Voting Classifier can then be used to wrap your models and average the predictions of the sub-models when asked to make predictions for new data. The predictions of the sub-models can be weighted, but specifying the weights for classifiers manually or even heuristically is difficult. More advanced methods can learn how to best weight the predictions from sub-models, but this is called stacking (stacked aggregation) and is currently not provided in scikit-learn.

You can create a voting ensemble model for classification using the `VotingClassifier` class . The code below provides an example of combining the predictions of logistic regression, classification and regression trees and support vector machines together for a classification problem.

```
# Listing 15.11: Example of the Voting Ensemble Algorithm.
# Voting Ensemble for Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, shuffle=True)
# create the sub models
estimators = []
model1 = LogisticRegression(solver='lbfgs', max_iter=3000)
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC()
estimators.append(('svm', model3))
# create the ensemble model
ensemble = VotingClassifier(estimators)
results = cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```

```
    0.7708646616541353
```

---

**My Understanding**

In the above code,

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `KFold` and `cross_val_score` method from `sklearn.model_selection`
- We are also importing `LogisticRegression` from `sklearn.linear_model`
- We are also importing `DecisionTreeClassifier` and `SVC` from `sklearn.tree` and `sklearn.svc` respectively
- We are also importing `VotingClassifier` from `sklearn.ensemble`
- We are passing kfold generated above as cross validation generator
- We are using KFold method to split the data into 10 different folds.
- We are creating three different models, `LogisticRegression`, `DecisionTreeClassifier` and `SVC` and storing them in a variable estimators
- We are passing these models in a VotingClassifier function and calculating `cross_val_score`.
- We are finally printing the results for the same.

---

## 15.5 Summary

In this chapter you discovered ensemble machine learning algorithms for improving the perfor mance of models on your problems. You learned about:

- Bagging Ensembles including Bagged Decision Trees, Random Forest and Extra Trees.
- Boosting Ensembles including AdaBoost and Stochastic Gradient Boosting.
- Voting Ensembles for averaging the predictions for any arbitrary models.

# Chapter 16 : Improve Performance with Algorithm Tuning

Machine learning models are parameterized so that their behavior can be tuned for a given problem. Models can have many parameters and finding the best combination of parameters can be treated as a search problem. In this chapter you will discover how to tune the parameters of machine learning algorithms in Python using the scikit-learn. After completing this lesson you will know:

1. The importance of algorithm parameter tuning to improve algorithm performance.
2. How to use a grid search algorithm tuning strategy.
3. How to use a random search algorithm tuning strategy.

Let's get started.

## 16.1 Machine Learning Algorithm Parameters

Algorithm tuning is a final step in the process of applied machine learning before finalizing your model. It is sometimes called hyperparameter optimization where the algorithm parameters are referred to as hyperparameters, whereas the coefficients found by the machine learning algorithm itself are referred to as parameters. Optimization suggests the search-nature of the problem. Phrased as a search problem, you can use different search strategies to find a good and robust parameter or set of parameters for an algorithm on a given problem. Python scikit-learn provides two simple methods for algorithm parameter tuning:

- Grid Search Parameter Tuning.
- Random Search Parameter Tuning.

## 16.2 Grid Search Parameter Tuning

Grid search is an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid. You can perform a grid search using the `GridSearchCV` class . The example below evaluates different alpha values for the Ridge Regression algorithm on the standard diabetes dataset. This is a one-dimensional grid search.

```
# Listing 16.1: Example of a grid search for algorithm parameters.
# Grid Search for Algorithm Tuning
import numpy
from pandas import read_csv
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
```

```
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
alphas = numpy.array([1,0.1,0.01,0.001,0.0001,0])
param_grid = dict(alpha=alphas)
model = Ridge()
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid.fit(X, Y)
print(grid.best_score_)
print(grid.best_estimator_.alpha)
```

```
0.27610844129292433
1.0
```

---

**My Understanding**

In the above code,

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `Ridge` from `sklearn.linear_model`
- We are also importing `GridSearchCV` from `sklearn.model_selection`
- We are passing this model in a `GridSearchCV` function and creating grid.
- From the results of the grid, we are finally printing best score and best estimator.

---

Running the example lists out the optimal score achieved and the set of parameters in the grid that achieved that score. In this case the alpha value of 1.0.

## 16.3 Random Search Parameter Tuning

Random search is an approach to parameter tuning that will sample algorithm parameters from a random distribution (i.e. uniform) for a fixed number of iterations. A model is constructed and evaluated for each combination of parameters chosen. You can perform a random search for algorithm parameters using the `RandomizedSearchCV` class . The example below evaluates different random `alpha` values between 0 and 1 for the Ridge Regression algorithm on the standard diabetes dataset. A total of 100 iterations are performed with uniformly random `alpha` values selected in the range between 0 and 1 (the range that `alpha` values can take).

```
# Listing 16.3: Example of a random search for algorithm parameters.
# Randomized for Algorithm Tuning
import numpy
from pandas import read_csv
from scipy.stats import uniform
from sklearn.linear_model import Ridge
from sklearn.model_selection import RandomizedSearchCV
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
param_grid = {'alpha': uniform()}
model = Ridge()
rsearch = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_iter=100, random_state=7)
rsearch.fit(X, Y)
print(rsearch.best_score_)
print(rsearch.best_estimator_.alpha)
```

```
0.2761075573402853
0.9779895119966027
```

---

**My Understanding**

In the above code,

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `Ridge` from `sklearn.linear_model`
- We are also importing `RandomizedSearchCV` from `sklearn.model_selection`
- We are passing this model in a `RandomizedSearchCV` function and creating generating values.
- From the result, we are finally printing best score and best estimator.

---

Running the example produces results much like those in the grid search example above. An optimal alpha value near 1.0 is discovered.

## 16.4 Summary

Algorithm parameter tuning is an important step for improving algorithm performance right before presenting results or preparing a system for production. In this chapter you discovered algorithm parameter tuning and two methods that you can use right now in Python and scikit-learn to improve your algorithm results:

- Grid Search Parameter Tuning
- Random Search Parameter Tuning

# Chapter 17 : Save and Load Machine Learning Models

Finding an accurate machine learning model is not the end of the project. In this chapter you will discover how to save and load your machine learning model in Python using scikit-learn. This allows you to save your model to file and load it later in order to make predictions. After completing this lesson you will know:

1. The importance of serializing models for reuse.
2. How to use `pickle` to serialize and deserialize machine learning models.
3. How to use `Joblib` to serialize and deserialize machine learning models.

Let's get started.

## 17.1 Finalize Your Model with pickle

Pickle is the standard way of serializing objects in Python. You can use the `pickle` operation to serialize your machine learning algorithms and save the serialized format to a file. Later you can load this file to deserialize your model and use it to make new predictions. The example below demonstrates how you can train a logistic regression model on the Pima Indians onset of diabetes dataset, save the model to file and load it to make predictions on the unseen test set.

```
# Listing 17.1: Example of using pickle to serialize and deserialize a model.
# Save Model Using Pickle
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from pickle import dump
from pickle import load
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=7)
# Fit the model on 33%
model = LogisticRegression(solver='lbfgs', max_iter=3000)
model.fit(X_train, Y_train)
# save the model to disk
filename = '/content/sample_data/finalized_model.sav'
dump(model, open(filename, 'wb'))
# some time later...
# load the model from disk
loaded_model = load(open(filename, 'rb'))
result = loaded_model.score(X_test, Y_test)
print(result)
```

```
    0.7874015748031497
```

---

**My Understanding**

In the above code,

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `train_test_split` from `sklearn.model_selection`
- We are also importing `LogisticRegression` from `sklearn.linear_model`
- We are also importing `load` and `dump` from pickle
- We are splitting the test and set data into two parts in the ratio 1:2 for test and train
- We are creating a model using LogisticRegression and creating saving the model and training it using the availabed train data
- We are further saving the model into a file name `finalized_model.sav`
- We are reopening the model saved in the file and testing our data using the model we already saved in out file
- From the result, we are finally printing the score obtained from our model

---

Running the example saves the model to `finalized_model.sav` in your local working directory. Load the saved model and evaluating it provides an estimate of accuracy of the model on unseen data.

## 17.2 Finalize Your Model with Joblib

The Joblib library is part of the SciPy ecosystem and provides utilities for pipelining Python jobs. It provides utilities for saving and loading Python objects that make use of NumPy data structures, efficiently . This can be useful for some machine learning algorithms that require a lot of parameters or store the entire dataset (e.g. $k$-Nearest Neighbors). The example below demonstrates how you can train a logistic regression model on the Pima Indians onset of diabetes dataset, save the model to file using Joblib and load it to make predictions on the unseen test set.

```
# Listing 17.3: Example of using pickle to serialize and deserialize a model.
# Save Model Using joblib
from pandas import read_csv
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import joblib

# from sklearn.externals.joblib import load
filename = 'pima-indians-diabetes.data.csv'
filename = '/content/drive/MyDrive/Datasets/diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=7)
# Fit the model on 33%
model = LogisticRegression(solver='lbfgs', max_iter=3000)
model.fit(X_train, Y_train)
filename = '/content/sample_datafinalized_model.sav'
joblib.dump(model, open(filename, 'wb'))
# some time later...
# load the model from disk
loaded_model = joblib.load(filename)
result = loaded_model.score(X_test, Y_test)
print(result)
```

```
0.7874015748031497
```

---

**My Understanding** In the above code,

- We are importing data from `diabetes.csv` CSV file using `read_csv()` method in Pandas
- We are also importing `train_test_split` from `sklearn.model_selection`
- We are also importing `LogisticRegression` from `sklearn.linear_model`
- We are importing `joblib` using which we can load and dump our model that we can load our model
- We are splitting the test and set data into two parts in the ratio 1:2 for test and train
- We are creating a model using LogisticRegression and creating saving the model and training it using the availabed train data

- We are further saving the model into a file name `finalized_model.sav`
- We are reopening the model saved in the file using `joblib` and testing our data using the model we already saved in out file
- From the result, we are finally printing the score obtained from our model

---

## 17.3 Tips for Finalizing Your Model

This section lists some important considerations when finalizing your machine learning models.

- **Python Version**. Take note of the Python version. You almost certainly require the same major (and maybe minor) version of Python used to serialize the model when you later load it and deserialize it.
- **Library Versions**. The version of all major libraries used in your machine learning project almost certainly need to be the same when deserializing a saved model. This is not limited to the version of NumPy and the version of scikit-learn.
- **Manual Serialization.** You might like to manually output the parameters of your learned model so that you can use them directly in scikit-learn or another platform in the future. Often the techniques used internally by machine learning algorithms to make predictions are a lot simpler than those used to learn the parameters can may be easy to implement in custom code that you have control over.

Take note of the version so that you can re-create the environment if for some reason you cannot reload your model on another machine or another platform at a later time.

## 17.4 Summary

In this chapter you discovered how to persist your machine learning algorithms in Python with scikit-learn. You learned two techniques that you can use:

- The `pickle` API for serializing standard Python objects.
- The `Joblib` API for efficiently serializing Python objects with NumPy arrays.

# Chapter 18 : Predictive Modeling Project Template

Applied machine learning is an empirical skill. You cannot get better at it by reading books and articles. You have to practice. In this lesson you will discover the simple six-step machine learning project template that you can use to jump-start your project in Python. After completing this lesson you will know:

1. How to structure an end-to-end predictive modeling project.
2. How to map the tasks you learned about in Part II onto a project.
3. How to best use the structured project template to ensure an accurate result for your dataset.

Let's get started.

# 18.1 Practice Machine Learning With Projects

Working through machine learning problems from end-to-end is critically important. You can read about machine learning. You can also try out small one-off recipes. But applied machine learning will not come alive for you until you work through a dataset from beginning to end.

Working through a project forces you to think about how the model will be used, to challenge your assumptions and to get good at all parts of a project, not just your favorite parts. The best way to practice predictive modeling machine learning projects is to use standardized datasets from the UCI Machine Learning Repository. Once you have a practice dataset and a bunch of Python recipes, how do you put it all together and work through the problem end-to-end?

## 18.1.1 Use A Structured Step-By-Step Process

Any predictive modeling machine learning project can be broken down into six common tasks:

1. Define Problem.
2. Summarize Data.
3. Prepare Data.
4. Evaluate Algorithms.
5. Improve Results.

6. Present Results.

Tasks can be combined or broken down further, but this is the general structure. To work through predictive modeling machine learning problems in Python, you need to map Python onto this process. The tasks may need to be adapted or renamed slightly to suit the Python way of doing things (e.g. Pandas for data loading and scikit-learn for modeling). The next section provides exactly this mapping and elaborates each task and the types of sub-tasks and libraries that you can use.

## 18.2 Machine Learning Project Template in Python

This section presents a project template that you can use to work through machine learning problems in Python end-to-end.

### 18.2.1 Template Summary

Below is the project template that you can use in your machine learning projects in Python.

```
# Listing 18.1: Predictive modeling machine learning project template.
# Python Project Template

# 1. Prepare Problem
# a) Load libraries
# b) Load dataset

# 2. Summarize Data
# a) Descriptive statistics
# b) Data visualizations

# 3. Prepare Data
# a) Data Cleaning
# b) Feature Selection
# c) Data Transforms

# 4. Evaluate Algorithms
# a) Split-out validation dataset
```

```
# b) Test options and evaluation metric
# c) Spot Check Algorithms
# d) Compare Algorithms

# 5. Improve Accuracy
# a) Algorithm Tuning
# b) Ensembles

# 6. Finalize Model
# a) Predictions on validation dataset
# b) Create standalone model on entire training dataset
# c) Save model for later use
```

### 18.2.2 How To Use The Project Template

1. Create a new file for your project (e.g. project name.py).
2. Copy the project template.
3. Paste it into your empty project file.
4. Start to fill it in, using recipes from this book and others.

## 18.3 Machine Learning Project Template Steps

This section gives you additional details on each of the steps of the template.

### 18.3.1 Prepare Problem

This step is about loading everything you need to start working on your problem. This includes:

- Python modules, classes and functions that you intend to use.
- Loading your dataset from CSV.

This is also the home of any global configuration you might need to do. It is also the place where you might need to make a reduced sample of your dataset if it is too large to work with. Ideally, your dataset should be small enough to build a model or create a visualization within a minute, ideally 30 seconds. You can always scale up well performing models later.

### 18.3.2 Summarize Data

This step is about better understanding the data that you have available. This includes understanding your data using:

- Descriptive statistics such as summaries.
- Data visualizations such as plots with Matplotlib, ideally using convenience functions from Pandas.

Take your time and use the results to prompt a lot of questions, assumptions and hypotheses that you can investigate later with specialized models.

### 18.3.3 Prepare Data

This step is about preparing the data in such a way that it best exposes the structure of the problem and the relationships between your input attributes with the output variable. This includes tasks such as:

- Cleaning data by removing duplicates, marking missing values and even imputing missing values.
- Feature selection where redundant features may be removed and new features developed.
- Data transforms where attributes are scaled or redistributed in order to best expose the structure of the problem later to learning algorithms.

Start simple. Revisit this step often and cycle with the next step until you converge on a subset of algorithms and a presentation of the data that results in accurate or accurate-enough models to proceed.

### 18.3.4 Evaluate Algorithms

This step is about finding a subset of machine learning algorithms that are good at exploiting the structure of your data (e.g. have better than average skill). This involves steps such as:

- Separating out a validation dataset to use for later confirmation of the skill of your developed model.
- Defining test options using scikit-learn such as cross validation and the evaluation metric to use.
- Spot-checking a suite of linear and nonlinear machine learning algorithms.
- Comparing the estimated accuracy of algorithms.

On a given problem you will likely spend most of your time on this and the previous step until you converge on a set of 3-to-5 well performing machine learning algorithms.

### 18.3.5 Improve Accuracy

Once you have a shortlist of machine learning algorithms, you need to get the most out of them. There are two different ways to improve the accuracy of your models:

- Search for a combination of parameters for each algorithm using scikit-learn that yields the best results.
- Combine the prediction of multiple models into an ensemble prediction using ensemble techniques.

The line between this and the previous step can blur when a project becomes concrete. There may be a little algorithm tuning in the previous step. And in the case of ensembles, you may bring more than a shortlist of algorithms forward to combine their predictions.

### 18.3.6 Finalize Model

Once you have found a model that you believe can make accurate predictions on unseen data, you are ready to finalize it. Finalizing a model may involve sub-tasks such as:

- Using an optimal model tuned by scikit-learn to make predictions on unseen data.
- Creating a standalone model using the parameters tuned by scikit-learn.

- Saving an optimal model to file for later use.

Once you make it this far you are ready to present results to stakeholders and/or deploy your model to start making predictions on unseen data.

## 18.4 Tips For Using The Template Well

This section lists tips that you can use to make the most of the machine learning project template in Python.

- **Fast First Pass**. Make a first-pass through the project steps as fast as possible. This will give you confidence that you have all the parts that you need and a baseline from which to improve.
- **Cycles**. The process in not linear but cyclic. You will loop between steps, and probably spend most of your time in tight loops between steps 3-4 or 3-4-5 until you achieve a level of accuracy that is sufficient or you run out of time.
- **Attempt Every Step**. It is easy to skip steps, especially if you are not confident or familiar with the tasks of that step. Try and do something at each step in the process, even if it does not improve accuracy. You can always build upon it later. Don't skip steps, just reduce their contribution.
- **Ratchet Accuracy**. The goal of the project is model accuracy. Every step contributes towards this goal. Treat changes that you make as experiments that increase accuracy as the golden path in the process and reorganize other steps around them. Accuracy is a ratchet that can only move in one direction (better, not worse).
- **Adapt As Needed**. Modify the steps as you need on a project, especially as you become more experienced with the template. Blur the edges of tasks, such as steps 4-5 to best serve model accuracy.

## 18.5 Summary

In this lesson you discovered a machine learning project template in Python. It laid out the steps of a predictive modeling machine learning project with the goal of maximizing model accuracy. You can copy-and-paste the template and use it to jump-start your current or next machine learning project in Python.

# Chapter 19 : Your First Machine Learning Project in Python Step-By-Step

You need to see how all of the pieces of a predictive modeling machine learning project actually fit together. In this lesson you will complete your first machine learning project using Python. In this step-by-step tutorial project you will:

- Download and install Python SciPy and get the most useful package for machine learning in Python.
- Load a dataset and understand it's structure using statistical summaries and data visualization.
- Create 6 machine learning models, pick the best and build confidence that the accuracy is reliable.

If you are a machine learning beginner and looking to finally get started using Python, this tutorial was designed for you. Let's get started!

## 19.1 The Hello World of Machine Learning

The best small project to start with on a new tool is the classification of iris flowers. This is a good dataset for your first project because it is so well understood.

- Attributes are numeric so you have to figure out how to load and handle data.
- It is a classification problem, allowing you to practice with an easier type of supervised learning algorithm.
- It is a multiclass classification problem (multi-nominal) that may require some specialized handling.
- It only has 4 attributes and 150 rows, meaning it is small and easily fits into memory (and a screen or single sheet of paper).
- All of the numeric attributes are in the same units and the same scale not requiring any special scaling or transforms to get started.

In this tutorial we are going to work through a small machine learning project end-to-end. Here is an overview of what we are going to cover:

1. Loading the dataset.
2. Summarizing the dataset.
3. Visualizing the dataset.
4. Evaluating some algorithms.

5. Making some predictions. Take your time and work through each step. Try to type in the commands yourself or copy-and-paste the commands to speed things up. Start your Python interactive environment and let's get started with your $hello\ world$ machine learning project in Python.

## 19.2. Load The Data

In this step we are going to load the libraries and the iris data CSV file from URL.

### 19.2.1 Import libraries

First, let's import all of the modules, functions and objects we are going to use in this tutorial.

```
# Listing 19.1: Load libraries.
# Load libraries
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

---

**My Understanding**

- We are importing `train_test_split`, `KFold` and `cross_val_score` from `sklearn.model_selection`
- We are importing `classification_report`, `confusion_matrix` and `accuracy _score`
- We are importing multiple machine learning models like `LogisticRegression`, `DecisionTreeClassifier`, `KNeighborsClassifier`, `LinearDiscriminantAnalysis`, `GuassianNB` and `SVC`

---

Everything should load without error. If you have an error, stop. You need a working SciPy environment before continuing. See the advice in Chapter 2 about setting up your environment.

## 19.2.2 Load Dataset

The iris dataset can be downloaded from the UCI Machine Learning repository1 . We are using Pandas to load the data. We will also use Pandas next to explore the data both with descriptive statistics and data visualization. Note that we are specifying the names of each column when loading the data. This will help later when we explore the data.

```
# Listing 19.2: Load the Iris dataset.
# Load dataset
filename = '/content/drive/MyDrive/Datasets/iris.csv'
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(filename, names=names)
```

---

**My Understanding**

- We are importing data from `iris.csv` CSV file using `read_csv()` method in Pandas

---

# 19.3 Summarize the Dataset

Now it is time to take a look at the data. In this step we are going to take a look at the data a few different ways:

- Dimensions of the dataset.
- Peek at the data itself.
- Statistical summary of all attributes.
- Breakdown of the data by the class variable.

Don’t worry, each look at the data is one command. These are useful commands that you can use again and again on future projects.

## 19.3.1 Dimensions of Dataset

We can get a quick idea of how many instances (rows) and how many attributes (columns) the data contains with the shape property.

```
# Listing 19.3: Print the shape of the dataset.
# shape
print(dataset.shape)
```

```
    (150, 5)
```

---

**My Understanding**

- We are printing the size of the iris dataset and it turns out to be 150 rows and 5 columns.

---

## 19.3.2 Peek at the Data

It is also always a good idea to actually eyeball your data.

```
# Listing 19.5: Print the first few rows of the dataset.
#head
print(dataset.head(20))
```

```
        sepal-length  sepal-width  petal-length  petal-width        class
```

```
    0            5.1          3.5           1.4          0.2  Iris-setosa
    1            4.9          3.0           1.4          0.2  Iris-setosa
    2            4.7          3.2           1.3          0.2  Iris-setosa
    3            4.6          3.1           1.5          0.2  Iris-setosa
    4            5.0          3.6           1.4          0.2  Iris-setosa
    5            5.4          3.9           1.7          0.4  Iris-setosa
    6            4.6          3.4           1.4          0.3  Iris-setosa
    7            5.0          3.4           1.5          0.2  Iris-setosa
    8            4.4          2.9           1.4          0.2  Iris-setosa
    9            4.9          3.1           1.5          0.1  Iris-setosa
    10           5.4          3.7           1.5          0.2  Iris-setosa
    11           4.8          3.4           1.6          0.2  Iris-setosa
    12           4.8          3.0           1.4          0.1  Iris-setosa
    13           4.3          3.0           1.1          0.1  Iris-setosa
    14           5.8          4.0           1.2          0.2  Iris-setosa
    15           5.7          4.4           1.5          0.4  Iris-setosa
    16           5.4          3.9           1.3          0.4  Iris-setosa
    17           5.1          3.5           1.4          0.3  Iris-setosa
    18           5.7          3.8           1.7          0.3  Iris-setosa
    19           5.1          3.8           1.5          0.3  Iris-setosa
```

---

**My Understanding**

- We are printing the first 20 tuples of the dataset that we have available with us.
- We get a general idea of the content of the dataset.

---

## 19.3.3 Statistical Summary

Now we can take a look at a summary of each attribute. This includes the count, mean, the min and max values as well as some percentiles.

```
# Listing 19.7: Print the statistical descriptions of the dataset.
# descriptions
print(dataset.describe())
```

```
            sepal-length  sepal-width  petal-length  petal-width
```

```
       sepal-length  sepal-width  petal-length  petal-width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

---

**My Understanding**

- We are printing the count, mean, standard deviation, minimum and maximum.
- Additionally printing 25%, 50% and 75% median of the data separated by multiple columns.

---

We can see that all of the numerical values have the same scale (centimeters) and similar ranges between 0 and 8 centimeters.

## 19.3.4 Class Distribution

Let's now take a look at the number of instances (rows) that belong to each class. We can view this as an absolute count.

```
# Listing 19.9: Print the class distribution in the dataset.
# class distribution
print(dataset.groupby('class').size())
```

```
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

---

**My Understanding**

- We are printing classnames and count of the data having these classes.

## 19.4 Data Visualization

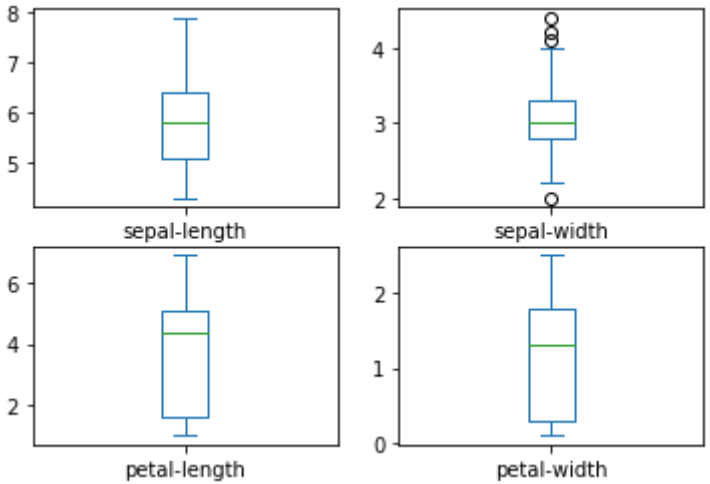
We now have a basic idea about the data. We need to extend this with some visualizations. We are going to look at two types of plots:

- Univariate plots to better understand each attribute.
- Multivariate plots to better understand the relationships between attributes.

### 19.4.1 Univariate Plots

We will start with some univariate plots, that is, plots of each individual variable. Given that the input variables are numeric, we can create box and whisker plots of each.

```
# Figure 19.1: Box and Whisker Plots of Each Attribute.
# box and whisker plots
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
pyplot.show()
```



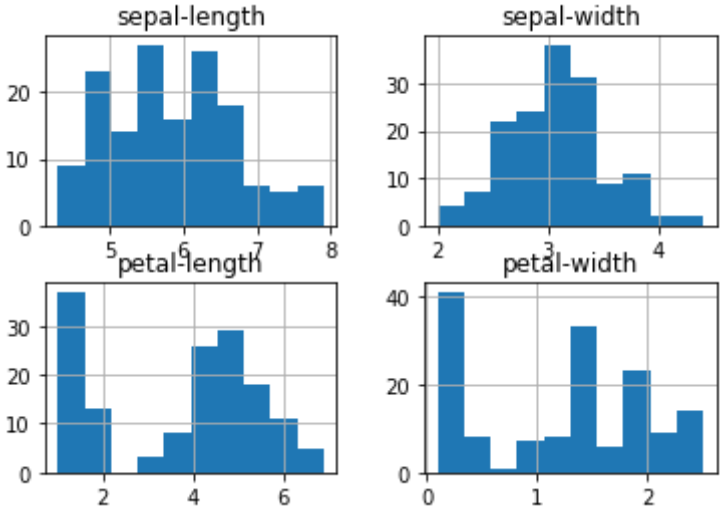
**My Understanding**

- We are plotting the existing data in a box and whisker plot to better understand the data.

---

We can also create a histogram of each input variable to get an idea of the distribution.

```
# Listing 19.12: Visualize the dataset using histogram plots.
# histograms
dataset.hist()
pyplot.show()
```




---

**My Understanding**

- We are printing the histogram of the dataset and understanding how the ranges of values are distribute for each column name

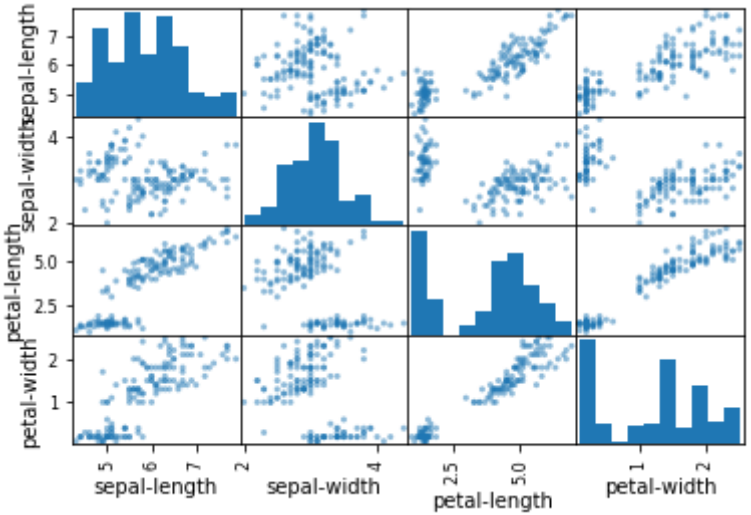
---

It looks like perhaps two of the input variables have a Gaussian distribution. This is useful to note as we can use algorithms that can exploit

## 19.4.2 Multivariate Plots

Now we can look at the interactions between the variables. Let's look at scatter plots of all pairs of attributes. This can be helpful to spot structured relationships between input variables.

```
# Listing 19.13: Visualize the dataset using scatter plots.
# scatter plot matrix
scatter_matrix(dataset)
pyplot.show()
```




---

**My Understanding**

From the above code:

- We are trying to understand how variables are related with each other.
- Scatter plot is used to show structured relations between two variables by creating a graph.

---

Note the diagonal grouping of some pairs of attributes. This suggests a high correlation and a predictable relationship.

## 19.5 Evaluate Some Algorithms

Now it is time to create some models of the data and estimate their accuracy on unseen data. Here is what we are going to cover in this step:

1. Separate out a validation dataset.
2. Setup the test harness to use 10-fold cross validation.
3. Build 5 different models to predict species from flower measurements
4. Select the best model.

### 19.5.1 Create a Validation Dataset

We need to know whether or not the model that we created is any good. Later, we will use statistical methods to estimate the accuracy of the models that we create on unseen data. We also want a more concrete estimate of the accuracy of the best model on unseen data by evaluating it on actual unseen data. That is, we are going to hold back some data that the algorithms will not get to see and we will use this data to get a second and independent idea of how accurate the best model might actually be. We will split the loaded dataset into two, 80% of which we will use to train our models and 20% that we will hold back as a validation dataset.

```
# Listing 19.14: Separate data into Train and Validation Datasets.
# Split-out validation dataset
array = dataset.values
X = array[:,0:4]
Y = array[:,4]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y, test_size=validation_size, random_state=seed)
```

---

**My Understanding**

In the above code:

- We are spliting the data into train and test part. The test part consists of 20% of test part and 80% of the training part.

---

You now have training data in the `X_train` and `Y_train` for preparing models and a `X_validation` and `Y_validation` sets that we can use later.

## 19.5.2 Test Harness

We will use 10-fold cross validation to estimate accuracy. This will split our dataset into 10 parts, train on 9 and test on 1 and repeat for all combinations of train-test splits. We are using the metric of `accuracy` to evaluate models. This is a ratio of the number of correctly predicted instances divided by the total number of instances in the dataset multiplied by 100 to give a percentage (e.g. 95% accurate). We will be using the scoring variable when we run build and evaluate each model next.

## 19.5.3 Build Models

We don't know which algorithms would be good on this problem or what configurations to use. We get an idea from the plots that some of the classes are partially linearly separable in some dimensions, so we are expecting generally good results. Let's evaluate six different algorithms:

- Logistic Regression (LR).
- Linear Discriminant Analysis (LDA).
- k-Nearest Neighbors (KNN).
- Classification and Regression Trees (CART).
- Gaussian Naive Bayes (NB).
- Support Vector Machines (SVM).

This list is a good mixture of simple linear (LR and LDA), nonlinear (KNN, CART, NB and SVM) algorithms. We reset the random number seed before each run to ensure that the evaluation of each algorithm is performed using exactly the same data splits. It ensures the results are directly comparable. Let's build and evaluate our five models:

```
# Listing 19.15: Evaluate a suite of algorithms on the dataset.
# Spot-Check Algorithms
models = []
models.append(('LR', LogisticRegression(solver='lbfgs', max_iter=3000)))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
# evaluate each model in turn
results = []
names = []
for name, model in models:
  kfold = KFold(n_splits=10, shuffle=True)
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
  results.append(cv_results)
  names.append(name)
  print("%s: %f (%f)" % (name, cv_results.mean(), cv_results.std(), ))
```

```
LR: 0.983333 (0.033333)
LDA: 0.975000 (0.038188)
KNN: 0.983333 (0.033333)
CART: 0.958333 (0.055902)
NB: 0.983333 (0.033333)
SVM: 0.975000 (0.053359)
```

---

**My Understanding** In the above code:

- We are creating a list of `LogisticRegression`, `LinearDiscriminantAnalysis`, `KNeighborsClassifier`, `DecisionTreeClassifier`, `GuassianNB` and `SVC` models
- We are iterating through that list, split the data using kfold method and specifying how many splits are needed there.
- We are calculating results from cross_val_score and appending the data.

- We are printing means and standard deviation of the results.

---

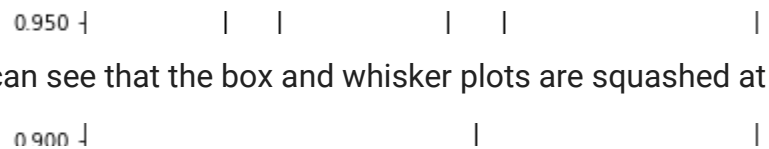
## 19.5.4 Select The Best Model

We now have 6 models and accuracy estimations for each. We need to compare the models to each other and select the most accurate. Running the example above, we get the above raw results. We can see that it looks like KNN has the largest estimated accuracy score. We can also create a plot of the model evaluation results and compare the spread and the mean accuracy of each model. There is a population of accuracy measures for each algorithm because each algorithm was evaluated 10 times (10 fold cross validation).

```
# Listing 19.17: Plot the distribution of scores for each algorithm.
# Compare Algorithms
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```

**My Understanding**

- We are printing the boxplot for all the algorithms to better understand the accuracy of all the algorithms



You can see that the box and whisker plots are squashed at the top of the range, with many samples achieving 100% accuracy.

## 19.6 Make Predictions

The KNN algorithm was the most accurate model that we tested. Now we want to get an idea of the accuracy of the model on our validation dataset. This will give us an independent final check on the accuracy of the best model. It is important to keep a validation set just in case you made a slip during training, such as overfitting to the training set or a data leak. Both will result in an overly optimistic result. We can run the KNN model directly on the validation set and summarize the results as a final accuracy score, a confusion matrix and a classification report.

```
# Listing 19.18: Make Predictions on the Validation Dataset.
# Make predictions on validation dataset
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
predictions = knn.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.9
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.85      0.92      0.88        12
 Iris-virginica       0.90      0.82      0.86        11
```

```
              accuracy                           0.90        30
             macro avg       0.92      0.91      0.91        30
          weighted avg       0.90      0.90      0.90        30
```

---

**My Understanding** From the previous code, we concluded that `KNeighborsClassifier` ks the best available classifier that is suitable for the iris data

- So, we using that classifier as a model to predict data and thus generate `accuracy_score`, `confusion_matrix` and `classification_report`
- We can observe that the accuracy is 0.9 or 90%
- Also, printed the confusion matrix and classification report for the same.

---

We can see that the accuracy is 0.9 or 90%. The confusion matrix provides an indication of the three errors made. Finally the classification report provides a breakdown of each class by precision, recall, f1-score and support showing excellent results (granted the validation dataset was small).

## 19.7 Summary

In this lesson you discovered step-by-step how to complete your first machine learning project in Python. You discovered that completing a small end-to-end project from loading the data to making predictions is the best way to get familiar with the platform.

# Chapter 20 : Regression Machine Learning Case Study Project

How do you work through a predictive modeling machine learning problem end-to-end? In this lesson you will work through a case study regression predictive modeling problem in Python including each step of the applied machine learning process. After completing this project, you will know:

- How to work through a regression predictive modeling problem end-to-end.
- How to use data transforms to improve model performance.
- How to use algorithm tuning to improve model performance.
- How to use ensemble methods and tuning of ensemble methods to improve model performance.

Let's get started.

## 20.1 Problem Definition

For this project we will investigate the Boston House Price dataset. Each record in the database describes a Boston suburb or town. The data was drawn from the Boston Standard Metropolitan Statistical Area (SMSA) in 1970. The attributes are defined as follows (taken from the UCI Machine Learning Repository): ):

1. `CRIM` : per capita crime rate by town
2. `ZN` : proportion of residential land zoned for lots over 25,000 sq.ft.
3. `INDUS` : proportion of non-retail business acres per town
4. `CHAS` : Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. `NOX` : nitric oxides concentration (parts per 10 million)
6. `RM` : average number of rooms per dwelling
7. `AGE` : proportion of owner-occupied units built prior to 1940
8. `DIS` : weighted distances to five Boston employment centers
9. `RAD` : index of accessibility to radial highways
10. `TAX` : full-value property-tax rate per $10,000
11. `PTRATIO` : pupil-teacher ratio by town
12. `B` : 1000$(Bk - 0.63)^2$ where Bk is the proportion of blacks by town
13. `LSTAT` : % lower status of the population
14. `MEDV` : Median value of owner-occupied homes in $1000s We can see that the input attributes have a mixture of units.

## 20.2 Load the Dataset

Let's start off by loading the libraries required for this project.

```
# Listing 20.1: Load libraries.
# Load libraries
import numpy
from numpy import arange
from matplotlib import pyplot
from pandas import read_csv
from pandas import set_option
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error
```

---

**My Understanding** In the above code,

- We are importing multiple libraries. Some notable libraries being `KFold`, `train_test_split`, `cross_val_score`, `GridSearchCV`.
- We are also importing multiple models like `LinearRegression`, `Lasso`, `ElasticNet`, `DecisionTreeREgressor`, `KNeightborsRegressor`, `RandomForestRegressor`, `GradientBoostingRegressor`, `ExtraTreesRegressor` and `AdaBoostRegressor`

---

We can now load the dataset that you can download from the UCI Machine Learning repository website.

```
# Listing 20.2: Load the dataset.
# Load dataset
filename = '/content/drive/MyDrive/Datasets/boston.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO',
'B', 'LSTAT', 'MEDV']
dataset = read_csv(filename, delim_whitespace=True, names=names)
```

---

**My Understanding**

In the above code:

- We are reading the `boston.csv` using Pandas and fetching the data from the file and storing it in our variable.

---

You can see that we are specifying the short names for each attribute so that we can reference them clearly later. You can also see that attributes are delimited by whitespace rather than commas in this file and we indicate this to `read_csv()` function via the `delim_whitespace` argument. We now have our data loaded.

## 20.3 Analyze Data

We can now take a closer look at our loaded data.

### 20.3.1 Descriptive Statistics

Let's start off by confirming the dimensions of the dataset, e.g. the number of rows and columns.

```
# Listing 20.3: Print the shape of the dataset.
# shapes
print(dataset.shape)
```

```
    (506, 14)
```

---

**My Understanding**

- We are printing the shape of dataset. We can observer that there are 506 rows and 14 columns.

---

We have 506 instances to work with and can confirm the data has 14 attributes including the output attribute `MEDV`.

Let's also look at the data types of each attribute.

```
# Listing 20.5: Print the data types of each attribute.
# types
print(dataset.dtypes)
```

```
    CRIM       float64
    ZN         float64
    INDUS      float64
    CHAS         int64
    NOX        float64
    RM         float64
    AGE        float64
    DIS        float64
    RAD          int64
    TAX        float64
    PTRATIO    float64
    B          float64
    LSTAT      float64
    MEDV       float64
    dtype: object
```

---

**My Understanding**

- We are printing the type of each column name in the dataset.

---

Let's now take a peek at the first 20 rows of the data.

```
# Listing 20.7: Print the first few rows of the dataset.
# head
print(dataset.head(20))
```

```
        CRIM    ZN  INDUS  CHAS    NOX     RM    AGE     DIS  RAD    TAX  \
0   0.00632  18.0   2.31     0  0.538  6.575   65.2  4.0900    1  296.0
1   0.02731   0.0   7.07     0  0.469  6.421   78.9  4.9671    2  242.0
2   0.02729   0.0   7.07     0  0.469  7.185   61.1  4.9671    2  242.0
3   0.03237   0.0   2.18     0  0.458  6.998   45.8  6.0622    3  222.0
4   0.06905   0.0   2.18     0  0.458  7.147   54.2  6.0622    3  222.0
5   0.02985   0.0   2.18     0  0.458  6.430   58.7  6.0622    3  222.0
6   0.08829  12.5   7.87     0  0.524  6.012   66.6  5.5605    5  311.0
7   0.14455  12.5   7.87     0  0.524  6.172   96.1  5.9505    5  311.0
8   0.21124  12.5   7.87     0  0.524  5.631  100.0  6.0821    5  311.0
9   0.17004  12.5   7.87     0  0.524  6.004   85.9  6.5921    5  311.0
10  0.22489  12.5   7.87     0  0.524  6.377   94.3  6.3467    5  311.0
11  0.11747  12.5   7.87     0  0.524  6.009   82.9  6.2267    5  311.0
12  0.09378  12.5   7.87     0  0.524  5.889   39.0  5.4509    5  311.0
13  0.62976   0.0   8.14     0  0.538  5.949   61.8  4.7075    4  307.0
14  0.63796   0.0   8.14     0  0.538  6.096   84.5  4.4619    4  307.0
15  0.62739   0.0   8.14     0  0.538  5.834   56.5  4.4986    4  307.0
16  1.05393   0.0   8.14     0  0.538  5.935   29.3  4.4986    4  307.0
17  0.78420   0.0   8.14     0  0.538  5.990   81.7  4.2579    4  307.0
18  0.80271   0.0   8.14     0  0.538  5.456   36.6  3.7965    4  307.0
19  0.72580   0.0   8.14     0  0.538  5.727   69.5  3.7965    4  307.0

    PTRATIO       B  LSTAT  MEDV
0      15.3  396.90   4.98  24.0
1      17.8  396.90   9.14  21.6
2      17.8  392.83   4.03  34.7
3      18.7  394.63   2.94  33.4
4      18.7  396.90   5.33  36.2
```

```
5      18.7  394.12   5.21  28.7
6      15.2  395.60  12.43  22.9
7      15.2  396.90  19.15  27.1
8      15.2  386.63  29.93  16.5
9      15.2  386.71  17.10  18.9
10     15.2  392.52  20.45  15.0
11     15.2  396.90  13.27  18.9
12     15.2  390.50  15.71  21.7
13     21.0  396.90   8.26  20.4
14     21.0  380.02  10.26  18.2
15     21.0  395.62   8.47  19.9
16     21.0  386.85   6.58  23.1
17     21.0  386.75  14.67  17.5
18     21.0  288.99  11.69  20.2
19     21.0  390.95  11.28  18.2
```

---

**My Understanding**

To understand the data further, we are printing first 20 tuples of the table to get the idea of what's inside the table.

---

We can confirm that the scales for the attributes are all over the place because of the differing units. We may benefit from some transforms later on.

Let’s summarize the distribution of each attribute.

```
# Listing 20.9: Print the statistical descriptions of the dataset.
# descriptions
print(dataset.describe())
```

```
              CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
```

```
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT        MEDV
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

---

**My Understanding**

From the above code:

- We are printing count, mean, standard deviation, minimum, maximum value. Also printing 25%, 50% and 75%.

---

We now have a better feeling for how different the attributes are. The min and max values as well are the means vary a lot. We are likely going to get better results by rescaling the data in some way.

Now, let's now take a look at the correlation between all of the numeric attributes.

```
# Listing 20.11: Print the correlations between the attributes.
# correlation
```

```
set_option('precision', 2)
print(dataset.corr(method='pearson'))
```

```
             CRIM    ZN  INDUS      CHAS   NOX    RM   AGE   DIS       RAD   TAX  \
CRIM     1.00 -0.20   0.41 -5.59e-02  0.42 -0.22  0.35 -0.38  6.26e-01  0.58
ZN      -0.20  1.00  -0.53 -4.27e-02 -0.52  0.31 -0.57  0.66 -3.12e-01 -0.31
INDUS    0.41 -0.53   1.00  6.29e-02  0.76 -0.39  0.64 -0.71  5.95e-01  0.72
CHAS    -0.06 -0.04   0.06  1.00e+00  0.09  0.09  0.09 -0.10 -7.37e-03 -0.04
NOX      0.42 -0.52   0.76  9.12e-02  1.00 -0.30  0.73 -0.77  6.11e-01  0.67
RM      -0.22  0.31  -0.39  9.13e-02 -0.30  1.00 -0.24  0.21 -2.10e-01 -0.29
AGE      0.35 -0.57   0.64  8.65e-02  0.73 -0.24  1.00 -0.75  4.56e-01  0.51
DIS     -0.38  0.66  -0.71 -9.92e-02 -0.77  0.21 -0.75  1.00 -4.95e-01 -0.53
RAD      0.63 -0.31   0.60 -7.37e-03  0.61 -0.21  0.46 -0.49  1.00e+00  0.91
TAX      0.58 -0.31   0.72 -3.56e-02  0.67 -0.29  0.51 -0.53  9.10e-01  1.00
PTRATIO  0.29 -0.39   0.38 -1.22e-01  0.19 -0.36  0.26 -0.23  4.65e-01  0.46
B       -0.39  0.18  -0.36  4.88e-02 -0.38  0.13 -0.27  0.29 -4.44e-01 -0.44
LSTAT    0.46 -0.41   0.60 -5.39e-02  0.59 -0.61  0.60 -0.50  4.89e-01  0.54
MEDV    -0.39  0.36  -0.48  1.75e-01 -0.43  0.70 -0.38  0.25 -3.82e-01 -0.47

             PTRATIO     B  LSTAT  MEDV
CRIM            0.29 -0.39   0.46 -0.39
ZN             -0.39  0.18  -0.41  0.36
INDUS           0.38 -0.36   0.60 -0.48
CHAS           -0.12  0.05  -0.05  0.18
NOX             0.19 -0.38   0.59 -0.43
RM             -0.36  0.13  -0.61  0.70
AGE             0.26 -0.27   0.60 -0.38
DIS            -0.23  0.29  -0.50  0.25
RAD             0.46 -0.44   0.49 -0.38
TAX             0.46 -0.44   0.54 -0.47
PTRATIO         1.00 -0.18   0.37 -0.51
B              -0.18  1.00  -0.37  0.33
LSTAT           0.37 -0.37   1.00 -0.74
MEDV           -0.51  0.33  -0.74  1.00
```

---

**My Understanding**

- We are now printing correlation in the dataset using the pearson method to understand how the data are related with each other.

This is interesting. We can see that many of the attributes have a strong correlation (e.g. > 0.70 or < −0.70). For example:

`NOX` and `INDUS` with 0.77.

`DIS` and `INDUS` with -0.71.

`TAX` and `INDUS` with 0.72.

`AGE` and `NOX` with 0.73.

`DIS` and `NOX` with -0.78.

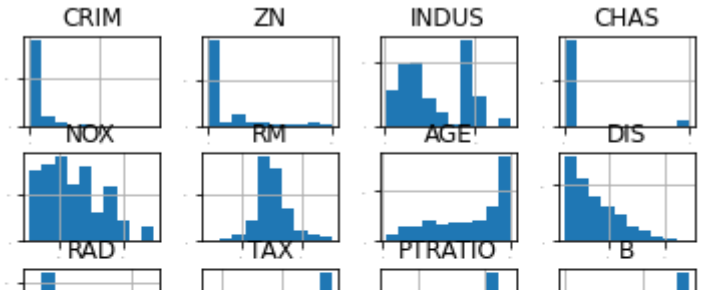
It also looks like `LSTAT` has a good negative correlation with the output variable `MEDV` with a value of -0.74.

## 20.4 Data Visualisations

### 20.4.1 Unimodal Data Visualizations

Let's look at visualizations of individual attributes. It is often useful to look at your data using multiple different visualizations in order to spark ideas. Let's look at histograms of each attribute to get a sense of the data distributions.

```
# Listing 20.13: Visualize the dataset using histogram plots.
# histograms
dataset.hist(sharex=False, sharey=False, xlabelsize=1, ylabelsize=1)
pyplot.show()
```




---

**My Understanding**

In the above code,

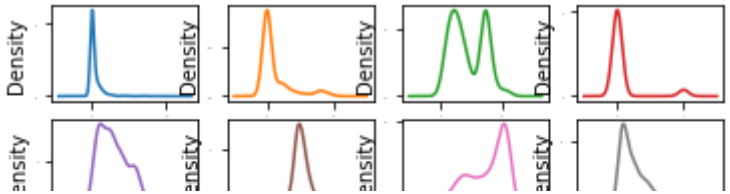
- We are printing the histogram of how the data is evaluated.
- From the graph, we can understand how the data is distributed over a range of data.

---

We can see that some attributes may have an exponential distribution, such as `CRIM`, `ZN`, `AGE` and `B`. We can see that others may have a bimodal distribution such as `RAD` and `TAX`.

Let's look at the same distributions using density plots that smooth them out a bit.

```
# Listing 20.14: Visualize the dataset using density plots.
# density
dataset.plot(kind='density', subplots=True, layout=(4,4), sharex=False, legend=False,
fontsize=1)
pyplot.show()
```




---

**My Understanding**

In the above code,

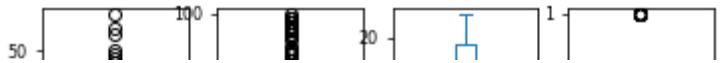
- We are printing the density plot for each row name of the table to understand the data and thus printing the graph as well.

---

This perhaps adds more evidence to our suspicion about possible exponential and bimodal distributions. It also looks like `NOX`, `RM` and `LSTAT` may be skewed Gaussian distributions, which might be helpful later with transforms.

Let's look at the data with box and whisker plots of each attribute.

```
# Listing 20.15: Visualize the dataset using box and whisker plots.
# box and whisker plots
dataset.plot(kind='box', subplots=True, layout=(4,4), sharex=False, sharey=False,
fontsize=8)
pyplot.show()
```




---

**My Understanding**

- In the above code, we are visualizing the data using box and whisker plots and showing the graph in the end.
- From the graph, we can see that the region with boxes are the one were the data is heavily concentrated, and the whiskers are the one where the data is deviating from the origin.

---

This helps point out the skew in many distributions so much so that data looks like outliers (e.g. beyond the whisker of the plots).

## 20.4.2 Multimodal Data Visualizations

Let's look at some visualizations of the interactions between variables. The best place to start is a scatter plot matrix.

```
# Listing 20.16: Visualize the dataset using scatter plots.
# scatter plot matrix
scatter_matrix(dataset)
pyplot.show()
```


---

**My Understanding**

- From the above graph, we can see the scatter plot that helps us to understand relation between two variables.

---

We can see that some of the higher correlated attributes do show good structure in their relationship. Not linear, but nice predictable curved relationships.

### 20.4.3 Summary of Ideas

There is a lot of structure in this dataset. We need to think about transforms that we could use later to better expose the structure which in turn may improve modeling accuracy. So far it would be worth trying:

- Feature selection and removing the most correlated attributes.
- Normalizing the dataset to reduce the effect of differing scales.
- Standardizing the dataset to reduce the effects of differing distributions.

With lots of additional time I would also explore the possibility of binning (discretization) of the data. This can often improve accuracy for decision tree algorithms.

## 20.5 Validation Dataset

It is a good idea to use a validation hold-out set. This is a sample of the data that we hold back from our analysis and modeling. We use it right at the end of our project to confirm the accuracy of our final model. It is a smoke test that we can use to see if we messed up and to give us confidence on our estimates of accuracy on unseen data. We will use 80% of the dataset for modeling and hold back 20% for validation.

```
# Listing 20.18: Separate Data into a Training and Validation Datasets.
# Split-out validation dataset
```

```
array = dataset.values
X = array[:,0:13]
Y = array[:,13]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y, test_size=validation_size, random_state=seed)
```

---

**My Understanding**

- Here, we are using `train_test_split` method to split the data into training and testing.
- The test data is 0.20 and the validation data is 0.80.

---

## 20.6 Evaluate Algorithms: Baseline

We have no idea what algorithms will do well on this problem. Gut feel suggests regression algorithms like Linear Regression and ElasticNet may do well. It is also possible that decision trees and even SVM may do well. I have no idea. Let's design our test harness. We will use 10-fold cross validation. The dataset is not too small and this is a good standard test harness configuration. We will evaluate algorithms using the Mean Squared Error (MSE) metric. MSE will give a gross idea of how wrong all predictions are (0 is perfect).

```
# Listing 20.19: Configure Algorithm Evaluation Test Harness.
# Test options and evaluation metric
num_folds = 10
seed = 7
scoring = 'neg_mean_squared_error'
```

---

**My Understanding**

- We are confguring and prepping variablea for training purposes.
- We are be using using `neg_mean_squared_error` as a scoring method.

---

Let's create a baseline of performance on this problem and spot-check a number of different algorithms. We will select a suite of different algorithms capable of working on this regression problem. The six algorithms selected include:

- **Linear Algorithms**: Linear Regression (LR), Lasso Regression (LASSO) and ElasticNet (EN).
- **Nonlinear Algorithms**: Classification and Regression Trees (CART), Support Vector Regression (SVR) and $k$-Nearest Neighbors (KNN).

```
# Listing 20.20: Create the List of Algorithms to Evaluate.
# Spot-Check Algorithms
models = []
models.append(('LR', LinearRegression()))
models.append(('LASSO', Lasso()))
models.append(('EN', ElasticNet()))
models.append(('KNN', KNeighborsRegressor()))
models.append(('CART', DecisionTreeRegressor()))
models.append(('SVR', SVR()))
```

---

**My Understanding**

In the above code,

- We are appending `LinearRegression`, `Lasso`, `ElasticNet`, `KneighborsRegressor`, `DecisionTreeRegressor` and `SVR` into models list
- These are the list of algorithms we will be using to evaluate the algorithm

---

The algorithms all use default tuning parameters. Let's compare the algorithms. We will display the mean and standard deviation of MSE for each algorithm as we calculate it and collect the results for use later.

```
# Listing 20.21: Evaluate the List of Algorithms.
```

```
# evaluate each model in turn
results = []
names = []
for name, model in models:
  kfold = KFold(n_splits=num_folds, shuffle=True)
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  print("%s: %f (%f)" % (name, cv_results.mean(), cv_results.std(), ))
```

```
LR: -21.427217 (4.348821)
LASSO: -26.723625 (7.394702)
EN: -27.636943 (11.101444)
KNN: -42.966289 (11.840376)
CART: -25.211651 (13.047387)
SVR: -67.559513 (34.550014)
```

---

**My Understanding**

In the above code,

- We are iterating through the models and using `cross_val_score` and storing the results
- We are appending the results and printing its mean and standard deviation

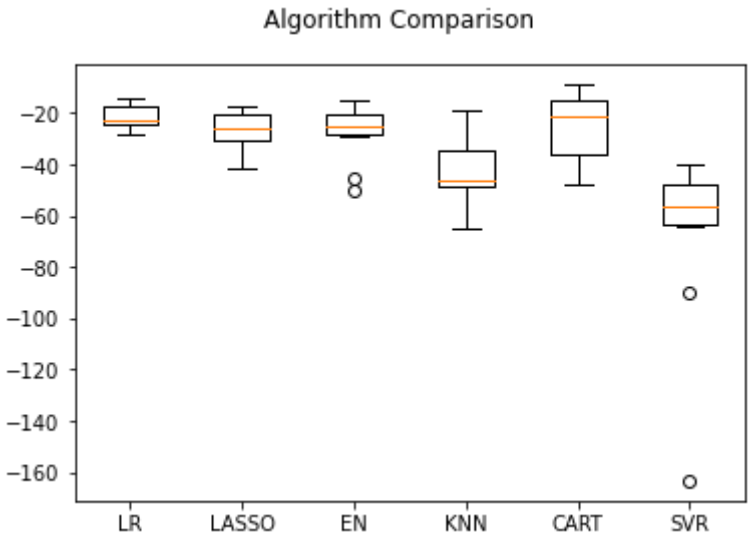
---

It looks like LR has the lowest MSE, followed closely by CART.

Let's take a look at the distribution of scores across all cross validation folds by algorithm.

```
# Listing 20.23: Visualize the Differences in Algorithm Performance.
# Compare Algorithms
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
```

```
ax.set_xticklabels(names)
pyplot.show()
```




---

**My Understanding**

- In the above code, we are comparing and printing the algorithm using boxplot to print the final results by comparing the algorithm.

---

We can see similar distributions for the regression algorithms and perhaps a tighter distribu  tion of scores for CART.

The differing scales of the data is probably hurting the skill of all of the algorithms and perhaps more so for SVR and KNN. In the next section we will look at running the same algorithms using a standardized copy of the data.

## 20.7 Evaluate Algorithms: Standardization

We suspect that the differing scales of the raw data may be negatively impacting the skill of some of the algorithms. Let's evaluate the same algorithms with a standardized copy of the dataset. This is where the data is transformed such that each attribute has a mean value of zero and a standard deviation of 1. We also need to avoid data leakage when we transform the data. A good way to avoid leakage is to use

pipelines that standardize the data and build the model for each fold in the cross validation test harness. That way we can get a fair estimation of how each model with standardized data might perform on unseen data.

```
# Listing 20.24: Evaluate Algorithms On Standardized Dataset.
# Standardize the dataset
pipelines = []
pipelines.append(('ScaledLR', Pipeline([('Scaler', StandardScaler()),('LR', LinearRegression())])))
pipelines.append(('ScaledLASSO', Pipeline([('Scaler', StandardScaler()),('LASSO', Lasso())])))
pipelines.append(('ScaledEN', Pipeline([('Scaler', StandardScaler()),('EN', ElasticNet())])))
pipelines.append(('ScaledKNN', Pipeline([('Scaler', StandardScaler()),('KNN', KNeighborsRegressor())])))
pipelines.append(('ScaledCART', Pipeline([('Scaler', StandardScaler()),('CART', DecisionTreeRegressor())])))
pipelines.append(('ScaledSVR', Pipeline([('Scaler', StandardScaler()),('SVR', SVR())])))
results = []
names = []
for name, model in pipelines:
  kfold = KFold(n_splits=num_folds, shuffle=True)
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  print("%s: %f (%f)" % (name, cv_results.mean(), cv_results.std(), ))
```

```
ScaledLR: -21.723520 (9.871308)
ScaledLASSO: -27.345069 (10.212967)
ScaledEN: -27.782660 (11.039111)
ScaledKNN: -21.426270 (6.467215)
ScaledCART: -29.441998 (17.673731)
ScaledSVR: -30.053934 (18.152283)
```

---

**My Understanding**

In the above code,

- We are running all the code in together in a same pipeline.
- We are standardizing by scaling all the values on the same scale.

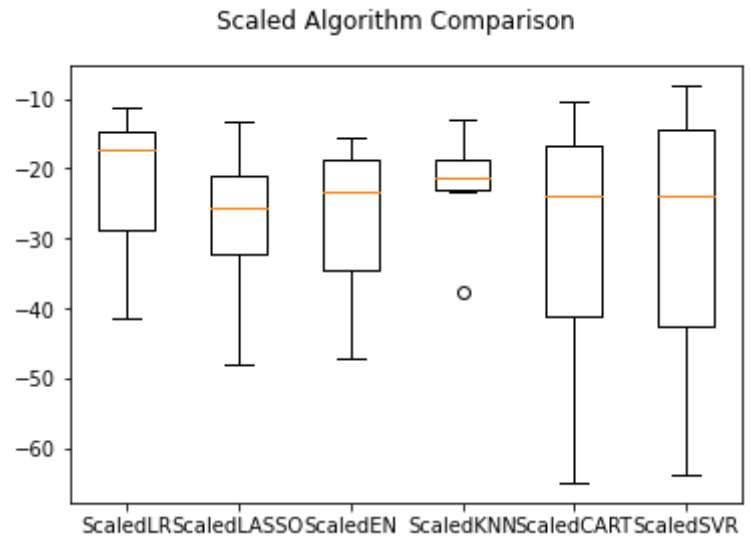
- Using this we can standardized data from the unseen variable.

---

Running the example provides a list of mean squared errors. We can see that scaling did have an effect on KNN, driving the error lower than the other models.

Let's take a look at the distribution of the scores across the cross validation folds.

```
# Listing 20.26: Visualize the Differences in Algorithm Performance on Standardized Dataset.
# Compare Algorithms
fig = pyplot.figure()
fig.suptitle('Scaled Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```




---

**My Understanding**

- We are creating a boxplot graph to understand how different algorithms that are scaled and performed corresponding to a value.

- We can observe that KNN performs the best among all the observations.

---

We can see that KNN has both a tight distribution of error and has the lowest score.

## 20.8 Improve Results With Tuning

We know from the results in the previous section that KNN achieves good results on a scaled version of the dataset. But can it do better. The default value for the number of neighbors in KNN is 7. We can use a grid search to try a set of different numbers of neighbors and see if we can improve the score. The below example tries odd k values from 1 to 21, an arbitrary range covering a known good value of 7. Each $k$ value (n neighbors) is evaluated using 10-fold cross validation on a standardized copy of the training dataset.

```
# Listing 20.27: Tune the Parameters of the KNN Algorithm on the Standardized Dataset.
# KNN Algorithm tuning
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
k_values = numpy.array([1,3,5,7,9,11,13,15,17,19,21])
param_grid = dict(n_neighbors=k_values)
model = KNeighborsRegressor()
kfold = KFold(n_splits=num_folds, shuffle=True)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(rescaledX, Y_train)
```

---

**My Understanding**

In the above code,

- We are trying to improve the dataset accuracy by changing the kFold of the KNN by changing the the k value.
- We are using GridSearchCV to calculate the model and save the results in the grid_result

---

We can display the mean and standard deviation scores as well as the best performing value for k below.

```
# Listing 20.28: Print Output From Tuning the KNN Algorithm.
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print("%f (%f) with: %r" % (mean, stdev, param, ))
```

```
Best: -19.208620 using {'n_neighbors': 3}
-20.924379 (10.616315) with: {'n_neighbors': 1}
-19.208620 (8.429471) with: {'n_neighbors': 3}
-20.017116 (9.631876) with: {'n_neighbors': 5}
-20.976419 (10.123381) with: {'n_neighbors': 7}
-20.770178 (10.114073) with: {'n_neighbors': 9}
-21.115689 (9.888887) with: {'n_neighbors': 11}
-20.889234 (9.951394) with: {'n_neighbors': 13}
-21.456481 (9.939184) with: {'n_neighbors': 15}
-22.370550 (10.448720) with: {'n_neighbors': 17}
-23.129173 (10.508250) with: {'n_neighbors': 19}
-23.812247 (11.307987) with: {'n_neighbors': 21}
```

---

**My Understanding**

In the above code, _ We are iterating through the results stored in the previous program, and printing `mean_test_score`, `std_test_score` and `params` to print the mean and standard deviation for different values of k in KNN algorithm

---

You can see that the best for k (n neighbors) is 3 providing a mean squared error of -18.172137, the best so far.

## 20.9 Ensemble Methods

Another way that we can improve the performance of algorithms on this problem is by using ensemble methods. In this section we will evaluate four different ensemble machine learning algorithms, two boosting and two bagging methods:

- **Boosting Methods**: AdaBoost (AB) and Gradient Boosting (GBM).
- **Bagging Methods**: Random Forests (RF) and Extra Trees (ET).

We will use the same test harness as before, 10-fold cross validation and pipelines that standardize the training data for each fold.

```
# Listing 20.30: Evaluate Ensemble Algorithms on the Standardized Dataset.
# ensembles
ensembles = []
ensembles.append(('ScaledAB', Pipeline([('Scaler', StandardScaler()),('AB', AdaBoostRegressor())])))
ensembles.append(('ScaledGBM', Pipeline([('Scaler', StandardScaler()),('GBM', GradientBoostingRegressor())])))
ensembles.append(('ScaledRF', Pipeline([('Scaler', StandardScaler()),('RF', RandomForestRegressor())])))
ensembles.append(('ScaledET', Pipeline([('Scaler', StandardScaler()),('ET',ExtraTreesRegressor())])))
results = []
names = []
for name, model in ensembles:
  kfold = KFold(n_splits=num_folds, shuffle=True)
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)
```

```
ScaledAB: -15.605923 (7.848111)
ScaledGBM: -10.214682 (4.822992)
ScaledRF: -12.011066 (7.272947)
ScaledET: -9.356451 (4.990799)
```

---

**My Understanding**

In the above program,

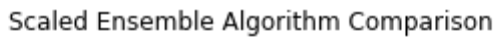
- We are appending `AdaBoostRegressor`, `GradientBoostingRegressor`, `RandomForestRegressor` and `ExtraTreesRegressor` to ensembles list
- We are iterating through these ensembles and and calculating the data in cv_results.
- We are printing mean and standard deviation of the results as well.

---

Running the example calculates the mean squared error for each method using the default parameters. We can see that we're generally getting better scores than our linear and nonlinear algorithms in previous sections.

We can also plot the distribution of scores across the cross validation folds.

```
# Listing 20.32: Visualzie the Differences in Ensemble Algorithm Performance on Standardized Dataset.
# Compare Algorithms
fig = pyplot.figure()
fig.suptitle('Scaled Ensemble Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```




**My Understanding**

In the above code,

- We are visualizing different ensemble algorithm comparisons in box plot graph that we just calculated from the previous graph.

It looks like Gradient Boosting has a better mean score, it also looks like Extra Trees has a similar distribution and perhaps a better median score.

We can probably do better, given that the ensemble techniques used the default parameters. In the next section we will look at tuning the Gradient Boosting to further lift the performance.

## 20.10 Tune Ensemble Methods

The default number of boosting stages to perform (n estimators) is 100. This is a good candidate parameter of Gradient Boosting to tune. Often, the larger the number of boosting stages, the better the performance but the longer the training time. In this section we will look at tuning the number of stages for gradient boosting. Below we define a parameter grid n estimators values from 50 to 400 in increments of 50. Each setting is evaluated using 10-fold cross validation.

```
# Listing 20.33: Tune GBM on Scaled Dataset.
# Tune scaled GBM
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = dict(n_estimators=numpy.array([50,100,150,200,250,300,350,400]))
model = GradientBoostingRegressor(random_state=seed)
kfold = KFold(n_splits=num_folds, shuffle=True)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(rescaledX, Y_train)
```

**My Understanding**

- In the above code,
- We are trying to improve the dataset accuracy by changing the Gradient accuracy the results in the grid_result and finally calculating the values and storing them in grid result variable which stores the values of them in the variable.

---

As before, we can summarize the best configuration and get an idea of how performance changed with each different configuration.

```
# Listing 20.34: Print Performance of Tuned GBM on Scaled Dataset.
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: -9.785200 using {'n_estimators': 400}
-11.100919 (5.237681) with: {'n_estimators': 50}
-10.415302 (4.922816) with: {'n_estimators': 100}
-10.131255 (4.734057) with: {'n_estimators': 150}
-9.931855 (4.647302) with: {'n_estimators': 200}
-9.869590 (4.555449) with: {'n_estimators': 250}
-9.809560 (4.496081) with: {'n_estimators': 300}
-9.804924 (4.461570) with: {'n_estimators': 350}
-9.785200 (4.438730) with: {'n_estimators': 400}
```

---

**My Understanding**

Here,

- we are iterating through the grid list and printing the `mean_test_score`, `std_test_score` and `params` of all the variable
- we are printing the mean and standard deviation corresponding to different types of `n_estimators`

---

We can see that the best configuration was n estimators=400 resulting in a mean squared error of -10.813227, about 0.65 units better than the untuned method.

Next we can finalize the model and prepare it for general use.

## 20.11 Finalize Model

In this section we will finalize the gradient boosting model and evaluate it on our hold out validation dataset. First we need to prepare the model and train it on the entire training dataset. This includes standardizing the training dataset before training.

```
# Listing 20.36: Construct the Finalized Model.
# prepare the model
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
model = GradientBoostingRegressor(random_state=seed, n_estimators=400)
model.fit(rescaledX, Y_train)
```

```
    GradientBoostingRegressor(n_estimators=400, random_state=7)
```

---

**My Understanding**

- We are using GradientBoostingRegressor to create a model and fitting training value datasets in the same.

---

We can then scale the inputs for the validation dataset and generate predictions.

```
# transform the validation dataset
rescaledValidationX = scaler.transform(X_validation)
predictions = model.predict(rescaledValidationX)
print(mean_squared_error(Y_validation, predictions))
```

```
11.902132586880027
```

---

**My Understanding**

- We are now transforming the validation dataset so that we can use it to generate the predict scale
- We are also calcualting `mean_squared_error` to calculate the value with the same.

---

We can see that the estimated mean squared error is 11.9, close to our estimate of -9.3.

## 20.12 Summary

In this chapter you worked through a regression predictive modeling machine learning problem from end-to-end using Python. Specifically, the steps covered were:

- Problem Definition (Boston house price data).
- Loading the Dataset.
- Analyze Data (some skewed distributions and correlated attributes).
- Evaluate Algorithms (Linear Regression looked good).
- Evaluate Algorithms with Standardization (KNN looked good).
- Algorithm Tuning (`K=3` for KNN was best).
- Ensemble Methods (Bagging and Boosting, Gradient Boosting looked good).
- Tuning Ensemble Methods (getting the most from Gradient Boosting).
- Finalize Model (use all training data and confirm using validation dataset).

Working through this case study showed you how the recipes for specific machine learning tasks can be pulled together into a complete project. Working through this case study is good practice at applied machine learning using Python and scikit-learn.

# Chapter 21 : Binary Classification Machine Learning Case Study Project

How do you work through a predictive modeling machine learning problem end-to-end? In this lesson you will work through a case study classification predictive modeling problem in Python including each step of the applied machine learning process. After completing this project, you will know:

- How to work through a classification predictive modeling problem end-to-end.
- How to use data transforms to improve model performance.
- How to use algorithm tuning to improve model performance.
- How to use ensemble methods and tuning of ensemble methods to improve model performance.

Let's get started.

## 21.1 Problem Definition

The focus of this project will be the Sonar Mines vs Rocks dataset1 . The problem is to predict metal or rock objects from sonar return data. Each pattern is a set of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular frequency band, integrated over a certain period of time. The label associated with each record contains the letter R if the object is a rock and M if it is a mine (metal cylinder). The numbers in the labels are in increasing order of aspect angle, but they do not encode the angle directly!

## 21.2 Load the Dataset

Let's start off by loading the libraries required for this project.

```
# Listing 21.1: Load libraries.
# Load libraries
import numpy
from matplotlib import pyplot
from pandas import read_csv
from pandas import set_option
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
```

---

**My Understanding**

- We are importing multiple libraries. Some notable libraries being `KFold`, `train_test_split`, `cross_val_score`, `GridSearchCV`.
- We are also importing multiple models like `LinearRegression`, `Lasso`, `ElasticNet`, `DecisionTreeREgressor`, `KNeightborsRegressor`, `RandomForestRegressor`, `GradientBoostingRegressor`, `ExtraTreesRegressor` and `AdaBoostRegressor`

---

You can download the dataset from the UCI Machine Learning repository website2 and save it in the local working directory with the filename `sonar.csv`.

```
# Listing 21.2: Load the dataset.
# Load dataset
url = '/content/drive/MyDrive/Datasets/sonar.csv'
dataset = read_csv(url, header=None)
```

---

**My Understanding**

- Using pandas, we are importing the sonar dataset using pandas library.

---

You can see that we are not specifying the names of the attributes this time. This is because other than the class attribute (the last column), the variables do not have meaningful names. We also indicate that there is no header information, this is to avoid file loading code taking the first record as the column names. Now that we have the dataset loaded we can take a look at it.

## 21.3 Analyze Data

Let's take a closer look at our loaded data.

### 21.3.1 Descriptive Statistics

We will start off by confirming the dimensions of the dataset, e.g. the number of rows and columns.

```
# Listing 21.3: Print the shape of the dataset.
# shape
print(dataset.shape)
```

```
    (208, 61)
```

---

**My Understanding**

- We are printing dimensions of the data. We can observe that there are 208 rowsn and 61 columns.

---

We have 208 instances to work with and can confirm the data has 61 attributes including the class attribute.

Let’s also look at the data types of each attribute.

```
# Listing 21.5: Print the data types of each attribute.
# types
set_option('display.max_rows', 500)
print(dataset.dtypes)
```

```
     4     float64
     5     float64
     6     float64
     7     float64
     8     float64
     9     float64
     10    float64
     11    float64
     12    float64
     13    float64
     14    float64
     15    float64
     16    float64
     17    float64
     18    float64
     19    float64
     20    float64
     21    float64
     22    float64
     23    float64
     24    float64
     25    float64
     26    float64
     27    float64
     28    float64
     29    float64
     30    float64
     31    float64
     32    float64

     33    float64
     34    float64
     35    float64
     36    float64
     37    float64
```

```
37    float64
38    float64
39    float64
40    float64
41    float64
42    float64
43    float64
44    float64
45    float64
46    float64
47    float64
48    float64
49    float64
50    float64
51    float64
52    float64
53    float64
54    float64
55    float64
56    float64
57    float64
58    float64
59    float64
60     object
dtype: object
```

---

**My Understanding**

- We are printing the data types of all the available columns python is interpretting the data into.

---

We can see that all of the attributes are numeric (float) and that the class value has been read in as an object.

Let's now take a peek at the first 20 rows of the data.

```
# Listing 21.7: Print the first few rows of the dataset.
# head
```

```
set_option('display.width', 100)
print(dataset.head(20))
```

```
          0        1        2        3     4     5     6        7     8     9   ...       51  \
0   2.00e-02  3.71e-02  4.28e-02  2.07e-02  0.10  0.10  0.15  1.60e-01  0.31  0.21  ...  2.70e-03
1   4.53e-02  5.23e-02  8.43e-02  6.89e-02  0.12  0.26  0.22  3.48e-01  0.33  0.29  ...  8.40e-03
2   2.62e-02  5.82e-02  1.10e-01  1.08e-01  0.10  0.23  0.24  3.77e-01  0.56  0.62  ...  2.32e-02
3   1.00e-02  1.71e-02  6.23e-02  2.05e-02  0.02  0.04  0.11  1.28e-01  0.06  0.13  ...  1.21e-02
4   7.62e-02  6.66e-02  4.81e-02  3.94e-02  0.06  0.06  0.12  2.47e-01  0.36  0.45  ...  3.10e-03
5   2.86e-02  4.53e-02  2.77e-02  1.74e-02  0.04  0.10  0.12  1.83e-01  0.21  0.30  ...  4.50e-03
6   3.17e-02  9.56e-02  1.32e-01  1.41e-01  0.17  0.17  0.07  1.40e-01  0.21  0.35  ...  2.01e-02
7   5.19e-02  5.48e-02  8.42e-02  3.19e-02  0.12  0.09  0.10  6.13e-02  0.15  0.28  ...  8.10e-03
8   2.23e-02  3.75e-02  4.84e-02  4.75e-02  0.06  0.06  0.08  9.80e-03  0.07  0.15  ...  1.45e-02
9   1.64e-02  1.73e-02  3.47e-02  7.00e-03  0.02  0.07  0.11  6.97e-02  0.10  0.03  ...  9.00e-03
10  3.90e-03  6.30e-03  1.52e-02  3.36e-02  0.03  0.03  0.04  2.72e-02  0.03  0.05  ...  6.20e-03
11  1.23e-02  3.09e-02  1.69e-02  3.13e-02  0.04  0.01  0.02  5.79e-02  0.11  0.08  ...  1.33e-02
12  7.90e-03  8.60e-03  5.50e-03  2.50e-02  0.03  0.05  0.05  9.58e-02  0.10  0.12  ...  1.76e-02
13  9.00e-03  6.20e-03  2.53e-02  4.89e-02  0.12  0.16  0.14  9.87e-02  0.10  0.19  ...  5.90e-03
14  1.24e-02  4.33e-02  6.04e-02  4.49e-02  0.06  0.04  0.05  3.43e-02  0.11  0.21  ...  8.30e-03
15  2.98e-02  6.15e-02  6.50e-02  9.21e-02  0.16  0.23  0.22  2.03e-01  0.15  0.09  ...  3.10e-03
16  3.52e-02  1.16e-02  1.91e-02  4.69e-02  0.07  0.12  0.17  1.54e-01  0.15  0.29  ...  3.46e-02
17  1.92e-02  6.07e-02  3.78e-02  7.74e-02  0.14  0.08  0.06  2.19e-02  0.10  0.12  ...  3.31e-02
18  2.70e-02  9.20e-03  1.45e-02  2.78e-02  0.04  0.08  0.10  1.14e-01  0.08  0.15  ...  8.40e-03
19  1.26e-02  1.49e-02  6.41e-02  1.73e-01  0.26  0.26  0.29  4.11e-01  0.50  0.59  ...  9.20e-03

          52        53        54        55        56        57        58        59  60
0   6.50e-03  1.59e-02  7.20e-03  1.67e-02  1.80e-02  8.40e-03  9.00e-03  3.20e-03   R
1   8.90e-03  4.80e-03  9.40e-03  1.91e-02  1.40e-02  4.90e-03  5.20e-03  4.40e-03   R
2   1.66e-02  9.50e-03  1.80e-02  2.44e-02  3.16e-02  1.64e-02  9.50e-03  7.80e-03   R
3   3.60e-03  1.50e-02  8.50e-03  7.30e-03  5.00e-03  4.40e-03  4.00e-03  1.17e-02   R
4   5.40e-03  1.05e-02  1.10e-02  1.50e-03  7.20e-03  4.80e-03  1.07e-02  9.40e-03   R
5   1.40e-03  3.80e-03  1.30e-03  8.90e-03  5.70e-03  2.70e-03  5.10e-03  6.20e-03   R
6   2.48e-02  1.31e-02  7.00e-03  1.38e-02  9.20e-03  1.43e-02  3.60e-03  1.03e-02   R
7   1.20e-02  4.50e-03  1.21e-02  9.70e-03  8.50e-03  4.70e-03  4.80e-03  5.30e-03   R
8   1.28e-02  1.45e-02  5.80e-03  4.90e-03  6.50e-03  9.30e-03  5.90e-03  2.20e-03   R
9   2.23e-02  1.79e-02  8.40e-03  6.80e-03  3.20e-03  3.50e-03  5.60e-03  4.00e-03   R
10  1.20e-02  5.20e-03  5.60e-03  9.30e-03  4.20e-03  3.00e-04  5.30e-03  3.60e-03   R
11  2.65e-02  2.24e-02  7.40e-03  1.18e-02  2.60e-03  9.20e-03  9.00e-04  4.40e-03   R
12  1.27e-02  8.80e-03  9.80e-03  1.90e-03  5.90e-03  5.80e-03  5.90e-03  3.20e-03   R
13  9.50e-03  1.94e-02  8.00e-03  1.52e-02  1.58e-02  5.30e-03  1.89e-02  1.02e-02   R
14  5.70e-03  1.74e-02  1.88e-02  5.40e-03  1.14e-02  1.96e-02  1.47e-02  6.20e-03   R
```

```
15  1.53e-02  7.10e-03  2.12e-02  7.60e-03  1.52e-02  4.90e-03  2.00e-02  7.30e-03   R
16  1.58e-02  1.54e-02  1.09e-02  4.80e-03  9.50e-03  1.50e-03  7.30e-03  6.70e-03   R
17  1.31e-02  1.20e-02  1.08e-02  2.40e-03  4.50e-03  3.70e-03  1.12e-02  7.50e-03   R
18  1.00e-03  1.80e-03  6.80e-03  3.90e-03  1.20e-02  1.32e-02  7.00e-03  8.80e-03   R
19  3.50e-03  9.80e-03  1.21e-02  6.00e-04  1.81e-02  9.40e-03  1.16e-02  6.30e-03   R

[20 rows x 61 columns]
```

---

**My Understanding**

- We are printing first 20 rows of the table so that we can undertand how the data is like and what values it has stored.

---

This does not show all of the columns, but we can see all of the data has the same scale. We can also see that the class attribute (60) has string values.

Let's summarize the distribution of each attribute.

```
# Listing 21.9: Print the statistical descriptions of the dataset.
# descriptions, change precision to 3 places
set_option('precision', 3)
print(dataset.describe())
```

```
              0          1         2         3         4         5         6         7         8         9   \
count   208.000  2.080e+02   208.000   208.000   208.000   208.000   208.000   208.000   208.000   208.000
mean      0.029  3.844e-02     0.044     0.054     0.075     0.105     0.122     0.135     0.178     0.208
std       0.023  3.296e-02     0.038     0.047     0.056     0.059     0.062     0.085     0.118     0.134
min       0.002  6.000e-04     0.002     0.006     0.007     0.010     0.003     0.005     0.007     0.011
25%       0.013  1.645e-02     0.019     0.024     0.038     0.067     0.081     0.080     0.097     0.111
50%       0.023  3.080e-02     0.034     0.044     0.062     0.092     0.107     0.112     0.152     0.182
75%       0.036  4.795e-02     0.058     0.065     0.100     0.134     0.154     0.170     0.233     0.269
max       0.137  2.339e-01     0.306     0.426     0.401     0.382     0.373     0.459     0.683     0.711

        ...         50          51          52        53          54          55          56          57  \
count   ...   208.000   2.080e+02   2.080e+02   208.000   2.080e+02   2.080e+02   2.080e+02   2.080e+02
mean    ...     0.016   1.342e-02   1.071e-02     0.011   9.290e-03   8.222e-03   7.820e-03   7.949e-03
std     ...     0.012   9.634e-03   7.060e-03     0.007   7.088e-03   5.736e-03   5.785e-03   6.470e-03
```

```
min    ...    0.000  8.000e-04  5.000e-04    0.001  6.000e-04  4.000e-04  3.000e-04  3.000e-04
25%    ...    0.008  7.275e-03  5.075e-03    0.005  4.150e-03  4.400e-03  3.700e-03  3.600e-03
50%    ...    0.014  1.140e-02  9.550e-03    0.009  7.500e-03  6.850e-03  5.950e-03  5.800e-03
75%    ...    0.021  1.673e-02  1.490e-02    0.015  1.210e-02  1.058e-02  1.043e-02  1.035e-02
max    ...    0.100  7.090e-02  3.900e-02    0.035  4.470e-02  3.940e-02  3.550e-02  4.400e-02

              58         59
count  2.080e+02  2.080e+02
mean   7.941e-03  6.507e-03
std    6.181e-03  5.031e-03
min    1.000e-04  6.000e-04
25%    3.675e-03  3.100e-03
50%    6.400e-03  5.300e-03
75%    1.033e-02  8.525e-03
max    3.640e-02  4.390e-02

[8 rows x 60 columns]
```

---

**My Understanding**

In the above code,

- we are calculating the count, mean and standard deviation, additionally, we are calculating min and maximum.
- We are also calculating 25%, 50% and 75% median of the above data.

---

Again, as we expect, the data has the same range, but interestingly differing mean values. There may be some benefit from standardizing the data.

Let’s take a quick look at the breakdown of class values.

```
# Listing 21.11: Print the class breakdown of the dataset.
# class distribution
print(dataset.groupby(60).size())
```

```
60
```

```
M    111
R     97
dtype: int64
```

---

**My Understanding**

In the above code

- we are grouping the data as per column number and printing corresponding count for the same.

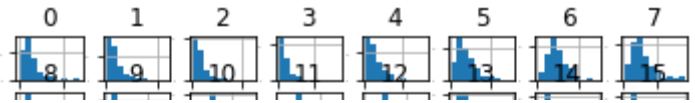
---

We can see that the classes are reasonably balanced between M (mines) and R (rocks).

## 21.3.2 Unimodal Data Visualizations

Let’s look at visualizations of individual attributes. It is often useful to look at your data using multiple different visualizations in order to spark ideas. Let’s look at histograms of each attribute to get a sense of the data distributions.

```
# Listing 21.13: Visualize the dataset with Histogram Plots.
# histograms
dataset.hist(sharex=False, sharey=False, xlabelsize=1, ylabelsize=1)
pyplot.show()
```



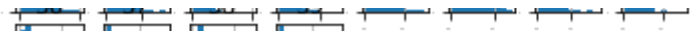

---

**My Understanding**

In the above code,

- We are printing the histogram of how the data is evaluated.
- From the graph, we can understand how the data is distributed over a range of data.

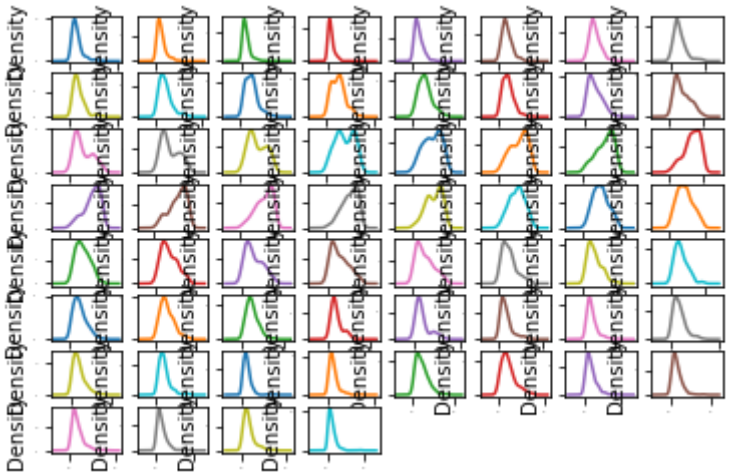
---



We can see that there are a lot of Gaussian-like distributions and perhaps some exponential like distributions for other attributes.

Let's take a look at the same perspective of the data using density plots.

```
# Listing 21.14: Visualize the dataset with Density Plots.
# density
dataset.plot(kind='density', subplots=True, layout=(8,8), sharex=False, legend=False, fontsize=1)
pyplot.show()
```



---

**My Understanding**

- From the above graph, we can see the density plot that helps us to understand relation between two variables.

---

This is useful, you can see that many of the attributes have a skewed distribution. A power transform like a Box-Cox transform that can correct for the skew in distributions might be useful.

It is always good to look at box and whisker plots of numeric attributes to get an idea of the spread of values.

```
# Listing 21.15: Visualize the dataset with Box and Whisker Plots.
# box and whisker plots
dataset.plot(kind='box', subplots=True, layout=(8,8), sharex=False, sharey=False, fontsize=1)
pyplot.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/pandas/core/series.py in __setitem__(self, key, value)
   1061         try:
-> 1062             self._set_with_engine(key, value)
   1063         except (KeyError, ValueError):

                                 8 frames
pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 0

During handling of the above exception, another exception occurred:

IndexError                                Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/pandas/core/series.py in __setitem__(self, key, value)
   1065             if is_integer(key) and self.index.inferred_type != "integer":
   1066                 # positional setter
-> 1067                 values[key] = value
```

**My Understanding**

- In the above code, we are visualizing the data using box and whisker plots and showing the graph in the end.
- From the graph, we can see that the region with boxes are the one were the data is heavily concentrated, and the whiskers are the one where the data is deviating from the origin.

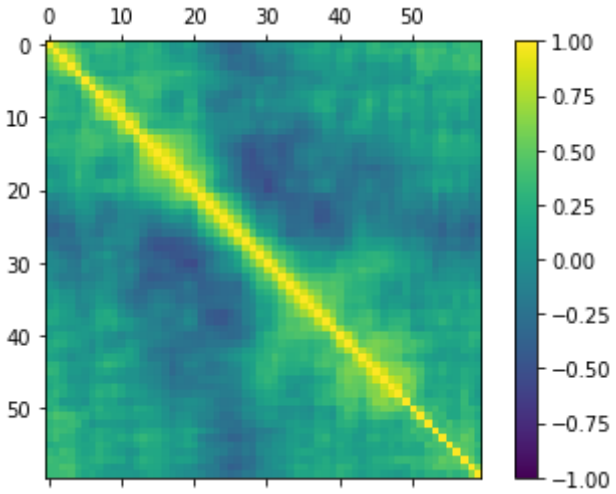


We can see that attributes do have quite different spreads. Given the scales are the same, it may suggest some benefit in standardizing the data for modeling to get all of the means lined up.

## 21.3.3 Multimodal Data Visualizations

Let's visualize the correlations between the attributes.

```
# Listing 21.16: Visualize the correlations between attributes.
# correlation matrix
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(dataset.corr(), vmin=-1, vmax=1, interpolation='none')
fig.colorbar(cax)
pyplot.show()
```




---

**My Understanding**

- In the above code, we are visualizing the the correlation between the data.
- We are plotting a colored matrix to understand how the data is related to each each other.

---

It looks like there is also some structure in the order of the attributes. The red around the diagonal suggests that attributes that are next to each other are generally more correlated with each other. The blue patches also suggest some moderate negative correlation the further attributes are away from each other in the ordering. This makes sense if the order of the attributes refers to the angle of sensors for the sonar chirp.

## 21.4 Validation Dataset

It is a good idea to use a validation hold-out set. This is a sample of the data that we hold back from our analysis and modeling. We use it right at the end of our project to confirm the accuracy of our final model. It is a smoke test that we can use to see if we messed up and to give us confidence on our estimates of accuracy on unseen data. We will use 80% of the dataset for modeling and hold back 20% for validation.

```
# Listing 21.17: Create Separate Training and Validation Datasets.
# Split-out validation dataset
array = dataset.values
X = array[:,0:60].astype(float)
Y = array[:,60]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y, test_size=validation_size, random_state=seed)
```

---

**My Understanding**

- Here, we are using `train_test_split` method to split the data into training and testing.
- The test data is 0.20 and the validation data is 0.80.

---

## 21.5 Evaluate Algorithms: Baseline

We don't know what algorithms will do well on this dataset. Gut feel suggests distance based algorithms like $k$-Nearest Neighbors and Support Vector Machines may do well. Let's design our test harness. We will use 10-fold cross validation. The dataset is not too small and this is a good standard test harness configuration. We will evaluate algorithms using the accuracy metric. This is a gross metric that will give a quick idea of how correct a given model is. More useful on binary classification problems like this one.

```
# Listing 21.18: Prepare the Test Harness for Evaluating Algorithms.
```

```
# Test options and evaluation metric
num_folds = 10
seed = 7
scoring = 'accuracy'
```

---

**My Understanding**

- We are confguring and prepping variablea for training purposes.
- We are be using using `accuracy` as a scoring method.

---

Let's create a baseline of performance on this problem and spot-check a number of different algorithms. We will select a suite of different algorithms capable of working on this classification problem. The six algorithms selected include:

- **Linear Algorithms**: Logistic Regression (LR) and Linear Discriminant Analysis (LDA).
- **Nonlinear Algorithms**: Classification and Regression Trees (CART), Support Vector Machines (SVM), Gaussian Naive Bayes (NB) and k-Nearest Neighbors (KNN).

```
# Listing 21.19: Prepare Algorithms to Evaluate.
# Spot-Check Algorithms
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
```

---

**My Understanding**

In the above code,

- We are appending `LinearRegression`, `LinearDiscriminantAnalysis`, `KneighborsClassfier`, `GyassianNB`, `DecisionTreeClassifier` and `SVC` into models list
- These are the list of algorithms we will be using to evaluate the algorithm

---

The algorithms all use default tuning parameters. Let's compare the algorithms. We will display the mean and standard deviation of accuracy for each algorithm as we calculate it and collect the results for use later.

```
# Listing 21.20: Evaluate Algorithms Using the Test Harness.
results = []
names = []
for name, model in models:
  kfold = KFold(n_splits=num_folds, shuffle=True)
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)
```

```
LR: 0.760294 (0.091865)
LDA: 0.769853 (0.100243)
KNN: 0.782721 (0.087132)
CART: 0.721691 (0.110417)
NB: 0.675368 (0.117556)
SVM: 0.783824 (0.078573)
```

---

**My Understanding**

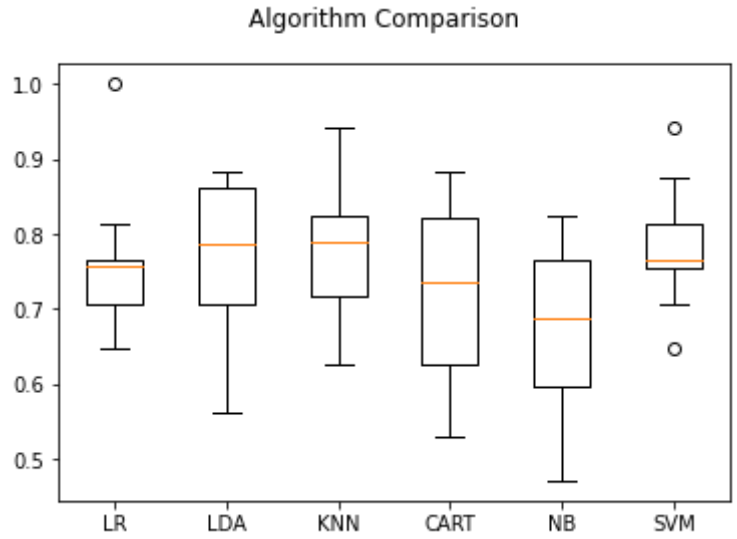
- We are iterating through the models and using `cross_val_score` and storing the results
- We are appending the results and printing mean and standard deviation for each algorithm

---

Running the example provides the output below. The results suggest That both Logistic Regression and k-Nearest Neighbors may be worth further study.

These are just mean accuracy values. It is always wise to look at the distribution of accuracy values calculated across cross validation folds. We can do that graphically using box and whisker plots.

```
# Listing 21.22: Visualization of the Distribution of Algorithm Performance.
# Compare Algorithms
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```




---

**My Understanding**

- In the above code, we are comparing and printing the algorithm using boxplot to print the final results by comparing the algorithm.

---

The results show a tight distribution for KNN which is encouraging, suggesting low variance. The poor results for SVM are surprising.

It is possible that the varied distribution of the attributes is having an effect on the accuracy of algorithms such as SVM. In the next section we will repeat this spot-check with a standardized copy of the training dataset.

## 21.6 Evaluate Algorithms: Standardize Data

We suspect that the differing distributions of the raw data may be negatively impacting the skill of some of the algorithms. Let's evaluate the same algorithms with a standardized copy of the dataset. This is where the data is transformed such that each attribute has a mean value of zero and a standard deviation of one. We also need to avoid data leakage when we transform the data. A good way to avoid leakage is to use pipelines that standardize the data and build the model for each fold in the cross validation test harness. That way we can get a fair estimation of how each model with standardized data might perform on unseen data.

```
# Listing 21.23: Evaluate Algorithms on a Scaled Dataset.
# Standardize the dataset
pipelines = []
pipelines.append(('ScaledLR', Pipeline([('Scaler', StandardScaler()),('LR', LogisticRegression())])))
pipelines.append(('ScaledLDA', Pipeline([('Scaler', StandardScaler()),('LDA', LinearDiscriminantAnalysis())])))
pipelines.append(('ScaledKNN', Pipeline([('Scaler', StandardScaler()),('KNN', KNeighborsClassifier())])))
pipelines.append(('ScaledCART', Pipeline([('Scaler', StandardScaler()),('CART', DecisionTreeClassifier())])))
pipelines.append(('ScaledNB', Pipeline([('Scaler', StandardScaler()),('NB', GaussianNB())])))
pipelines.append(('ScaledSVM', Pipeline([('Scaler', StandardScaler()),('SVM', SVC())])))
results = []
names = []
for name, model in pipelines:
  kfold = KFold(n_splits=num_folds, shuffle=True)
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)
```

```
ScaledLR: 0.741176 (0.088085)
ScaledLDA: 0.784559 (0.120902)
ScaledKNN: 0.783824 (0.101177)
ScaledCART: 0.740809 (0.062505)
ScaledNB: 0.655515 (0.101594)
ScaledSVM: 0.842279 (0.069376)
```

---

**My Understanding**

In the above code,

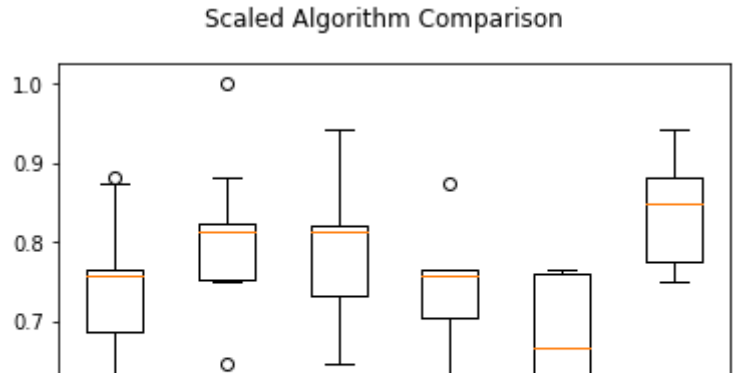
- We are running all the code in together in a same pipeline.
- We are standardizing by scaling all the values on the same scale.
- Using this we can standardized data from the unseen variable.

---

Running the example provides the results listed below. We can see that KNN is still doing well, even better than before. We can also see that the standardization of the data has lifted the skill of SVM to be the most accurate algorithm tested so far.

Again, we should plot the distribution of the accuracy scores using box and whisker plots.

```
# Listing 21.25: Visualization of the Distribution of Algorithm Performance on the Scaled Dataset.
# Compare Algorithms
fig = pyplot.figure()
fig.suptitle('Scaled Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```




---

**My Understanding**

- We are creating a boxplot graph to understand how different algorithms that are scaled and performed corresponding to a value.
- We can observe that KNN performs the best among all the observations.

---

The results suggest digging deeper into the SVM and KNN algorithms. It is very likely that configuration beyond the default may yield even more accurate models.

## 21.7 Algorithm Tuning

In this section we investigate tuning the parameters for two algorithms that show promise from the spot-checking in the previous section: KNN and SVM.

### 21.7.1 Tuning KNN

We can start off by tuning the number of neighbors for KNN. The default number of neighbors is 7. Below we try all odd values of k from 1 to 21, covering the default value of 7. Each k value is evaluated using 10-fold cross validation on the training standardized dataset.

```
# Listing 21.26: Tune the KNN Algorithm on the Scaled Dataset.
# Tune scaled KNN
```

```
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
neighbors = [1,3,5,7,9,11,13,15,17,19,21]
param_grid = dict(n_neighbors=neighbors)
model = KNeighborsClassifier()
kfold = KFold(n_splits=num_folds, shuffle=True)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(rescaledX, Y_train)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.843015 using {'n_neighbors': 1}
0.843015 (0.067151) with: {'n_neighbors': 1}
0.831618 (0.068777) with: {'n_neighbors': 3}
0.812500 (0.087451) with: {'n_neighbors': 5}
0.765441 (0.101869) with: {'n_neighbors': 7}
0.741912 (0.112368) with: {'n_neighbors': 9}
0.723162 (0.113864) with: {'n_neighbors': 11}
0.747426 (0.122411) with: {'n_neighbors': 13}
0.753676 (0.129085) with: {'n_neighbors': 15}
0.729412 (0.104326) with: {'n_neighbors': 17}
0.741544 (0.100349) with: {'n_neighbors': 19}
0.724265 (0.109704) with: {'n_neighbors': 21}
```

---

**My Understanding**

In the above code,

- We are trying to improve the dataset accuracy by changing the kFold of the KNN by changing the the k value.
- We are using GridSearchCV to calculate the model and save the results in the grid_result

---

We can print out configuration that resulted in the highest accuracy as well as the accuracy of all values tried. Running the example we see the results above.

We can see that the optimal configuration is `K=1`. This is interesting as the algorithm will make predictions using the most similar instance in the training dataset alone.

## 21.7.2 Tuning SVM

We can tune two key parameters of the SVM algorithm, the value of $C$ (how much to relax the margin) and the type of kernel. The default for SVM (the SVC class) is to use the Radial Basis Function (RBF) kernel with a $C$ value set to 1.0. Like with KNN, we will perform a grid search using 10-fold cross validation with a standardized copy of the training dataset. We will try a number of simpler kernel types and $C$ values with less bias and more bias (less than and more than 1.0 respectively).

```
# Listing 21.28: Tune the SVM Algorithm on the Scaled Dataset.
# Tune scaled SVM
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
c_values = [0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7, 2.0]
kernel_values = ['linear', 'poly', 'rbf', 'sigmoid']
param_grid = dict(C=c_values, kernel=kernel_values)
model = SVC()
kfold = KFold(n_splits=num_folds, shuffle=True)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(rescaledX, Y_train)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print("%f (%f) with: %r" % (mean, stdev, param))
```

```
    Best: 0.850368 using {'C': 1.5, 'kernel': 'poly'}
    0.771691 (0.132122) with: {'C': 0.1, 'kernel': 'linear'}
```

```
0.580147 (0.154943) with: {'C': 0.1, 'kernel': 'poly'}
0.650368 (0.133709) with: {'C': 0.1, 'kernel': 'rbf'}
0.730882 (0.113314) with: {'C': 0.1, 'kernel': 'sigmoid'}
0.741176 (0.120420) with: {'C': 0.3, 'kernel': 'linear'}
0.645956 (0.101926) with: {'C': 0.3, 'kernel': 'poly'}
0.729779 (0.112949) with: {'C': 0.3, 'kernel': 'rbf'}
0.765441 (0.100103) with: {'C': 0.3, 'kernel': 'sigmoid'}
0.729779 (0.148624) with: {'C': 0.5, 'kernel': 'linear'}
0.712500 (0.123595) with: {'C': 0.5, 'kernel': 'poly'}
0.802206 (0.092217) with: {'C': 0.5, 'kernel': 'rbf'}
0.777574 (0.136548) with: {'C': 0.5, 'kernel': 'sigmoid'}
0.741176 (0.154159) with: {'C': 0.7, 'kernel': 'linear'}
0.779412 (0.126494) with: {'C': 0.7, 'kernel': 'poly'}
0.832353 (0.095549) with: {'C': 0.7, 'kernel': 'rbf'}
0.753309 (0.142004) with: {'C': 0.7, 'kernel': 'sigmoid'}
0.734926 (0.154940) with: {'C': 0.9, 'kernel': 'linear'}
0.832353 (0.127040) with: {'C': 0.9, 'kernel': 'poly'}
0.838603 (0.091447) with: {'C': 0.9, 'kernel': 'rbf'}
0.752941 (0.140013) with: {'C': 0.9, 'kernel': 'sigmoid'}
0.734926 (0.154940) with: {'C': 1.0, 'kernel': 'linear'}
0.844118 (0.136240) with: {'C': 1.0, 'kernel': 'poly'}
0.844485 (0.092177) with: {'C': 1.0, 'kernel': 'rbf'}
0.746691 (0.122713) with: {'C': 1.0, 'kernel': 'sigmoid'}
0.746691 (0.117606) with: {'C': 1.3, 'kernel': 'linear'}
0.844118 (0.122887) with: {'C': 1.3, 'kernel': 'poly'}
0.837868 (0.092095) with: {'C': 1.3, 'kernel': 'rbf'}
0.758824 (0.093345) with: {'C': 1.3, 'kernel': 'sigmoid'}
0.746691 (0.117606) with: {'C': 1.5, 'kernel': 'linear'}
0.850368 (0.117314) with: {'C': 1.5, 'kernel': 'poly'}
0.837868 (0.092095) with: {'C': 1.5, 'kernel': 'rbf'}
0.752941 (0.090411) with: {'C': 1.5, 'kernel': 'sigmoid'}
0.746691 (0.117606) with: {'C': 1.7, 'kernel': 'linear'}
0.850368 (0.120227) with: {'C': 1.7, 'kernel': 'poly'}
0.843750 (0.096520) with: {'C': 1.7, 'kernel': 'rbf'}
0.746324 (0.114880) with: {'C': 1.7, 'kernel': 'sigmoid'}
0.746691 (0.117606) with: {'C': 2.0, 'kernel': 'linear'}
0.850000 (0.130043) with: {'C': 2.0, 'kernel': 'poly'}
0.837868 (0.099325) with: {'C': 2.0, 'kernel': 'rbf'}
0.740809 (0.092106) with: {'C': 2.0, 'kernel': 'sigmoid'}
```

**My Understanding**

In the above code,

- We are trying to improve the dataset accuracy by changing the param_grid values in the CV by assigning them the different c_values.
- We are using GridSearchCV to calculate the model and save the results in the grid_result
- We are also finding the best result, and printing all the possible results that exist out there.

---

Running the example prints out the best configuration, the accuracy as well as the accuracies for all configuration combinations.

We can see the most accurate configuration was SVM with an RBF kernel and a C value of 1.5. The accuracy 86.7470% is seemingly better than what KNN could achieve.

## 21.8 Ensemble Methods

Another way that we can improve the performance of algorithms on this problem is by using ensemble methods. In this section we will evaluate four different ensemble machine learning algorithms, two boosting and two bagging methods:

- Boosting Methods: AdaBoost (AB) and Gradient Boosting (GBM).
- Bagging Methods: Random Forests (RF) and Extra Trees (ET).

We will use the same test harness as before, 10-fold cross validation. No data standardization is used in this case because all four ensemble algorithms are based on decision trees that are less sensitive to data distributions.

```
# Listing 21.30: Evaluate Ensemble Algorithms.
# ensembles
ensembles = []
ensembles.append(('AB', AdaBoostClassifier()))
ensembles.append(('GBM', GradientBoostingClassifier()))
ensembles.append(('RF', RandomForestClassifier()))
ensembles.append(('ET', ExtraTreesClassifier()))
```

```
results = []
names = []
for name, model in ensembles:
  kfold = KFold(n_splits=num_folds, shuffle=True)
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)
```

```
AB: 0.758088 (0.135490)
GBM: 0.850368 (0.065727)
RF: 0.831618 (0.119797)
ET: 0.819485 (0.086510)
```

---

**My Understanding**

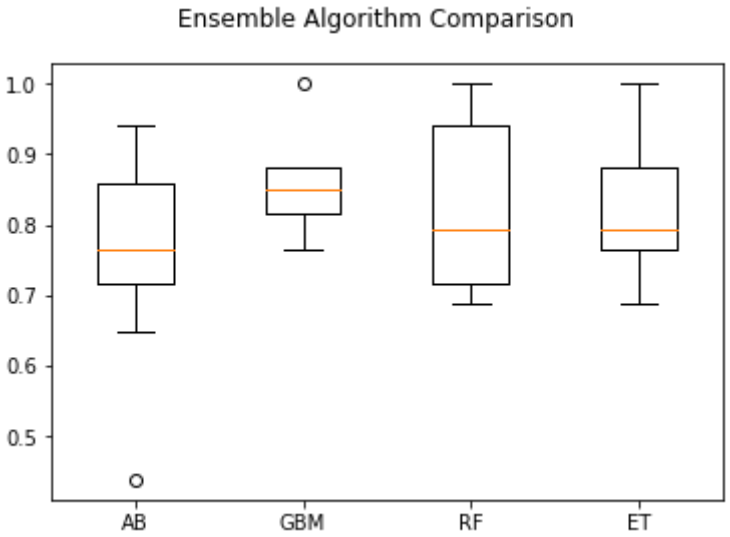
In the above program,

- We are appending `AdaBoostRegressor`, `GradientBoostingRegressor`, `RandomForestRegressor` and `ExtraTreesRegressor` to ensembles list
- We are iterating through these ensembles and and calculating the data in cv_results.
- We are printing mean and standard deviation of the results as well.

---

We can see that both boosting techniques provide strong accuracy scores in the low 80s (%) with default configurations. We can plot the distribution of accuracy scores across the cross validation folds.

```
# Compare Algorithms
fig = pyplot.figure()
fig.suptitle('Ensemble Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```




---

**My Understanding**

In the above code,

- We are preparing a graph to better understnad how the algorithms are compared.
- From the above graph, we can observe that GBM appaears to be best of them all and something we can work on more when it comes to sonar data

---

The results suggest GBM may be worthy of further study, with a strong mean and a spread that skews up towards high 90s (%) in accuracy.

## 21.9 Finalize Model

The SVM showed the most promise as a low complexity and stable model for this problem. In this section we will finalize the model by training it on the entire training dataset and make predictions for the hold-out validation dataset to confirm our findings. A part of the findings was that

SVM performs better when the dataset is standardized so that all attributes have a mean value of zero and a standard deviation of one. We

```
# Listing 21.33: Evaluate SVM on the Validation Dataset.
# prepare the model
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
model = SVC(C=1.5)
model.fit(rescaledX, Y_train)
# estimate accuracy on validation dataset
rescaledValidationX = scaler.transform(X_validation)
predictions = model.predict(rescaledValidationX)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.8571428571428571
[[23  4]
 [ 2 13]]
              precision    recall  f1-score   support

           M       0.92      0.85      0.88        27
           R       0.76      0.87      0.81        15

    accuracy                           0.86        42
   macro avg       0.84      0.86      0.85        42
weighted avg       0.86      0.86      0.86        42
```

---

**My Understanding**

- We are using SVC to create a model and fitting training value datasets in the same.
- We are now transforming the validation dataset so that we can use it to generate the predict scale
- Next we are calculating accuracy of the model for the data available and printing the accuracy of the data
- We are additionally printing confusion matrix and also printing classification report for the same.

---

We can see that we achieve an accuracy of nearly 86% on the held-out validation dataset. A score that matches closely to our expectations estimated above during the tuning of SVM.

## 21.10 Summary

In this chapter you worked through a classification predictive modeling machine learning problem from end-to-end using Python. Specifically, the steps covered were:

- Problem Definition (Sonar return data).
- Loading the Dataset.
- Analyze Data (same scale but different distributions of data).
- Evaluate Algorithms (KNN looked good).
- Evaluate Algorithms with Standardization (KNN and SVM looked good).
- Algorithm Tuning (K=1 for KNN was good, SVM with an RBF kernel and C=1.5 was best).
- Ensemble Methods (Bagging and Boosting, not quite as good as SVM).
- Finalize Model (use all training data and confirm using validation dataset).

Working through this case study showed you how the recipes for specific machine learning tasks can be pulled together into a complete project. Working through this case study is good practice at applied machine learning using Python.

Colab paid products - Cancel contracts here

✓ 0s completed at 2:45 AM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.