Python is an interpreted high-level programming language for general-purpose programming. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

The programming assignments will require coding in **Python**. The following books may be useful as a **quick introduction to Python**:

- *A Whirlwind Tour of Python* (and its companion repository of Jupyter Notebooks) by Jake VanderPlas is "…a fast-paced introduction to essential components of the Python language for researchers and developers who are already familiar with programming in another language" [author];
- *Python Data Science Handbook* by Jake VanderPlas (and its companion repository of Jupyter Notebooks) "introduces the core libraries essential for working with data in Python: particularly IPython, NumPy, Pandas, Matplotlib, Scikit-Learn, and related packages" [author].

The following books are also useful references if you want to learn Python from scratch:

- *Think Python: How to Think Like a Computer Scientist* by Allen B. Downey;
- *Automate the Boring Stuff with Python* by Al Sweigart.

Online Help

- ***https://www.python.org/about/help/***
- ***https://www.python-course.eu***

1. Output

```
print ('Hello, world!')
```

---

2. Input, assignment

```
name = input('What is your name?\n')
print ('Hi, %s.' % name)
print ('-'*100)
```

---

3. If Statement

```
if 5 > 2:
  print(“Five is greater than two!”)
```

4. For loop, built-in enumerate function, new style formatting

```
friends = ['john', 'pat', 'gary', 'michael']
for i, name in enumerate(friends):
    print ("iteration {iteration} is {name}".format(iteration=i,
name=name))
```

---

5. Fibonacci, tuple assignment

```
parents, babies = (1, 1)
while babies < 100:
    print ('This generation has {0} babies'.format(babies)
    parents, babies = (babies, parents + babies))
```

---

6. Functions

```
def greet(name):
    print ('Hello', name)
greet('Jack')
greet('Jill')
greet('Bob')
```

---

7. Import, regular expressions

```
import re
for test_string in ['555-1212', 'ILL-EGAL']:
    if re.match(r'^\d{3}-\d{4}$', test_string):
        print (test_string, 'is a valid US local phone number')
    else:
        print (test_string, 'rejected')
```

---

8. Dictionaries, generator expressions

```
prices = {'apple': 0.40, 'banana': 0.50}
my_purchase = {
    'apple': 1,
    'banana': 6}
grocery_bill = sum(prices[fruit] * my_purchase[fruit]
                   for fruit in my_purchase)
print ('I owe the grocer $%.2f' % grocery_bill)
```

9. Dictionary of Dictionaries

```
en_de = {"red" : "rot", "green" : "grün", "blue" : "blau",
"yellow":"gelb"}
de_fr = {"rot" : "rouge", "grün" : "vert", "blau" : "bleu",
"gelb":"jaune"}

dictionaries = {"en_de" : en_de, "de_fr" : de_fr }
print (dictionaries["de_fr"]["blau"])
```

10. Command line arguments, exception handling

```
# This program adds up integers in the command line
import sys
try:
    total = sum(int(arg) for arg in sys.argv[1:])
    print ('sum =', total)
except ValueError:
    print ('Please supply integer arguments')
```

11. Opening files

```
# indent your Python code to put into an email
import glob
# glob supports Unix style pathname extensions
python_files = glob.glob('*.py')
for file_name in sorted(python_files):
    print ('    ------' + file_name)
```

12. Time, conditionals, from..import, for..else

```
from time import localtime

activities = {8: 'Sleeping',
              9: 'Commuting',
              17: 'Working',
              18: 'Commuting',
              20: 'Eating',
              22: 'Resting' }

time_now = localtime()
hour = time_now.tm_hour

for activity_time in sorted(activities.keys()):
    if hour < activity_time:
        print (activities[activity_time])
        break
else:
    print ('Unknown, AFK or sleeping!')
```

---

13. Classes

```
class BankAccount(object):
    def __init__(self, initial_balance=0):
        self.balance = initial_balance
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def overdrawn(self):
        return self.balance < 0
my_account = BankAccount(15)
my_account.withdraw(5)
print (my_account.balance)
```

---

14. 8-Queens Problem (recursion)

```
BOARD_SIZE = 8

def under_attack(col, queens):
    left = right = col

    for r, c in reversed(queens):
        left, right = left - 1, right + 1

        if c in (left, col, right):
            return True
    return False

def solve(n):
    if n == 0:
        return [[]]

    smaller_solutions = solve(n - 1)

    return [solution+[(n,i+1)]
        for i in range(BOARD_SIZE)
            for solution in smaller_solutions
                if not under_attack(i+1, solution)]
for answer in solve(BOARD_SIZE):
    print (answer)
```

---

15. "Guess the Number" Game

```
import random

guesses_made = 0

name = input('Hello! What is your name?\n')

number = random.randint(1, 20)
print 'Well, {0}, I am thinking of a number between 1 and
20.'.format(name)

while guesses_made < 6:

    guess = int(input('Take a guess: '))

    guesses_made += 1

    if guess < number:
        print ('Your guess is too low.')

    if guess > number:
        print ('Your guess is too high.')

    if guess == number:
        break

if guess == number:
    print ('Good job, {0}! You guessed my number in {1}
guesses!'.format(name, guesses_made))
else:
    print ('Nope. The number I was thinking of was {0}'.format(number))
```

16. Example of Lambda Function in python

```
# Program to show the use of lambda functions

double = lambda x: x * 2

# Output: 10

print(double(5))
```

In the above program, `lambda x: x * 2` is the lambda function. Here `x` is the argument and `x * 2` is the expression that gets evaluated and returned.

This function has no name. It returns a function object which is assigned to the identifier `double`. We can now call it as a normal function. The statement

```
double = lambda x: x * 2
```

is nearly the same as

```
def double(x):

   return x * 2
```

17. Sorting Dictionary

```
temp = {'q':1, 'w':2, 'e':13, 'r':4, 't':5, 'y':6}

print (sorted(temp, key = lambda x:temp[x],reverse = True))
```

Download Anaconda

Choose Anaconda Download for Your Platform. Anaconda Python Installation Wizard should help do this. Installation is quick and painless. There should be no tricky questions or sticking points. The installation should take less than 10 minutes and take up a little more than 1 GB of space on your hard drive.

A simple Machine-Learning Exercise

We are going to use the iris flowers dataset. This dataset is famous because it is used as the "hello world" dataset in machine learning and statistics by pretty much everyone.

The dataset contains 150 observations of iris flowers. There are four columns of measurements of the flowers in centimeters. The fifth column is the species of the flower observed. All observed flowers belong to one of three species.

You can learn more about this dataset on Wikipedia.
In this step we are going to load the iris data from CSV file URL.

Import libraries

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

```
# Load libraries
import pandas
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn import model_selection
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
```

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

Load Dataset

We can load the data directly from the UCI Machine Learning repository.

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.

```
# Load dataset

url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',
'class']

dataset = pandas.read_csv(url, names=names)
```

The dataset should load without incident.

If you do have network problems, you can download the iris.data file into your working directory and load it using the same method, changing URL to the local file name.

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:

1. Dimensions of the dataset.
2. Peek at the data itself.
3. Statistical summary of all attributes.
4. 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.

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.

```
# shape
print(dataset.shape)
```

You should see 150 instances and 5 attributes:

```
(150, 5)
```

Peek at the Data

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

```
# head
print(dataset.head(20))
```

You should see the first 20 rows of the data:

```
    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
```

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.

```
# descriptions
print(dataset.describe())
```

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

```
       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
```

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.

```
# class distribution

print(dataset.groupby('class').size())
```

We can see that each class has the same number of instances (50 or 33% of the dataset).

```
class

Iris-setosa        50

Iris-versicolor    50

Iris-virginica     50
```

Data Visualization

We now have a basic idea about the data. We need to extend that with some visualizations.

We are going to look at two types of plots:

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

Univariate Plots

We start with some univariate plots, that is, plots of each individual variable.

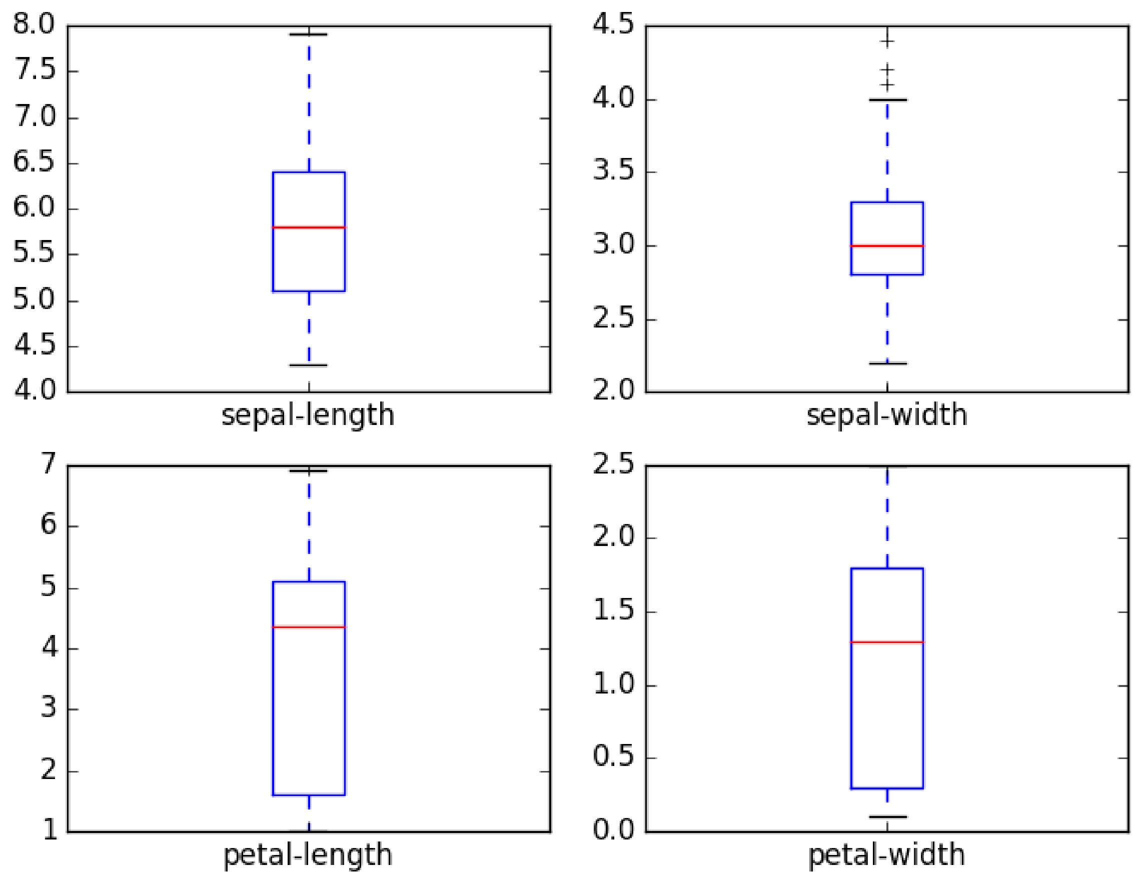
Given that the input variables are numeric, we can create box and whisker plots of each.

```
# box and whisker plots

dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)

plt.show()
```

This gives us a much clearer idea of the distribution of the input attributes:

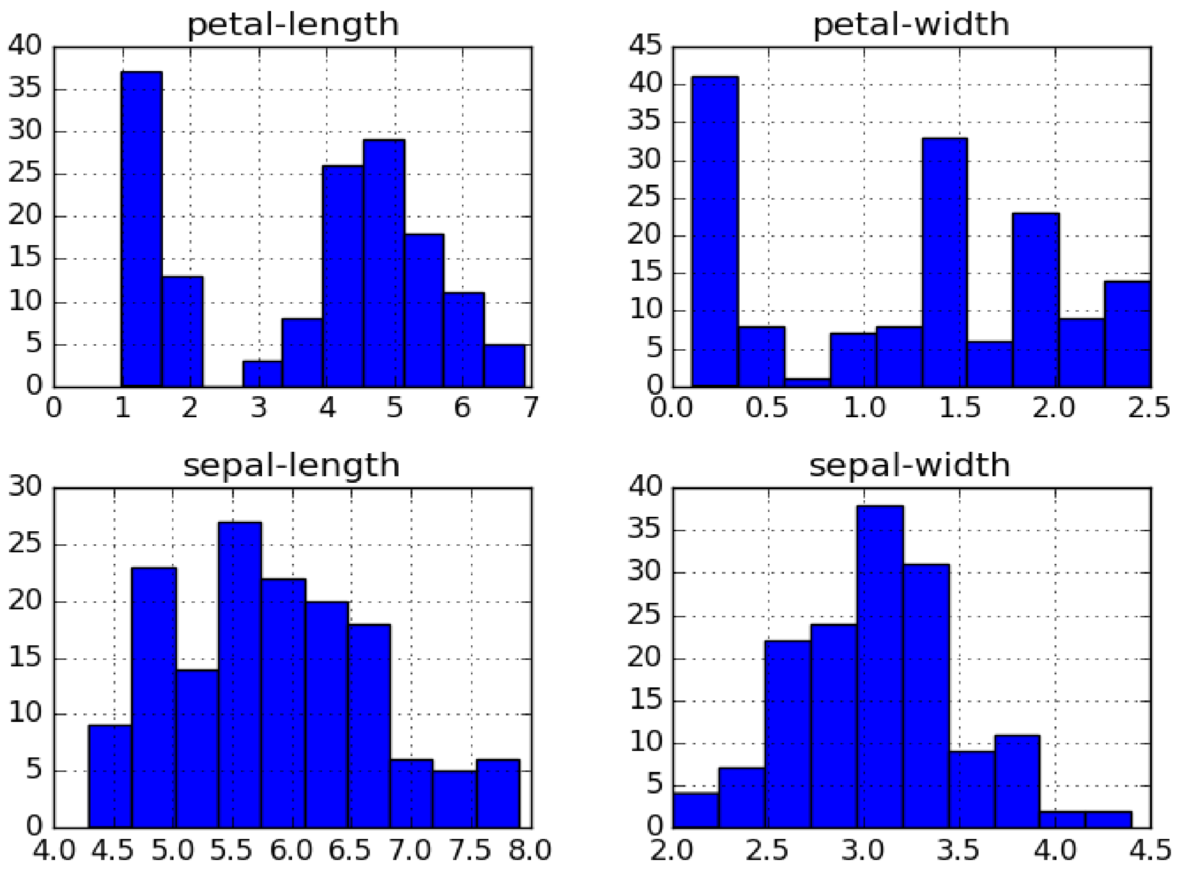


Box and Whisker Plots

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

```
# histograms
dataset.hist()
plt.show()
```

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 this assumption.



Histogram Plots

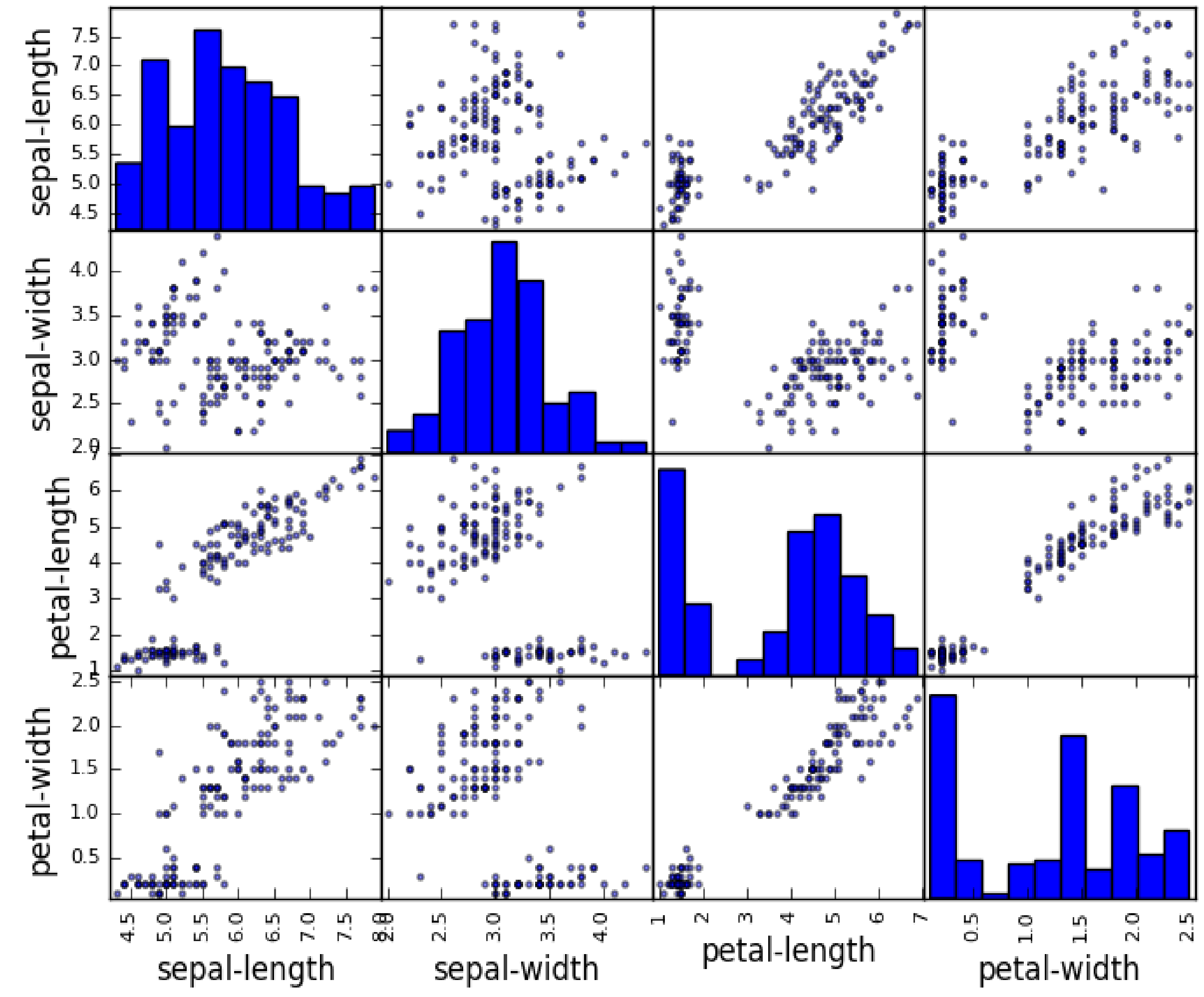
Multivariate Plots

Now we can look at the interactions between the variables.

First, let's look at scatterplots of all pairs of attributes. This can be helpful to spot structured relationships between input variables.

```
# scatter plot matrix
scatter_matrix(dataset)
plt.show()
```

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



Scatterplot Matrix

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. Set-up 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.

Create a Validation Dataset

We need to know that the model 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.

```
# 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 =
model_selection.train_test_split(X, Y, test_size=validation_size,
random_state=seed)
```

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.

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.

```
# Test options and evaluation metric

seed = 7
```

```
scoring = 'accuracy'
```

The specific random seed does not matter, learn more about pseudorandom number generators here:

We are using the metric of '*accuracy*' to evaluate models. This is a ratio of the number of correctly predicted instances in 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.

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 6 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 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:

```
# 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()))
# evaluate each model in turn
results = []
names = []
for name, model in models:
	kfold = model_selection.KFold(n_splits=10, random_state=seed)
	cv_results = model_selection.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)
```

Select 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 following raw results:

```
LR: 0.966667 (0.040825)
LDA: 0.975000 (0.038188)
KNN: 0.983333 (0.033333)
CART: 0.975000 (0.038188)
NB: 0.975000 (0.053359)
SVM: 0.981667 (0.025000)
```

Note, you're results may differ. For more on this see the post:

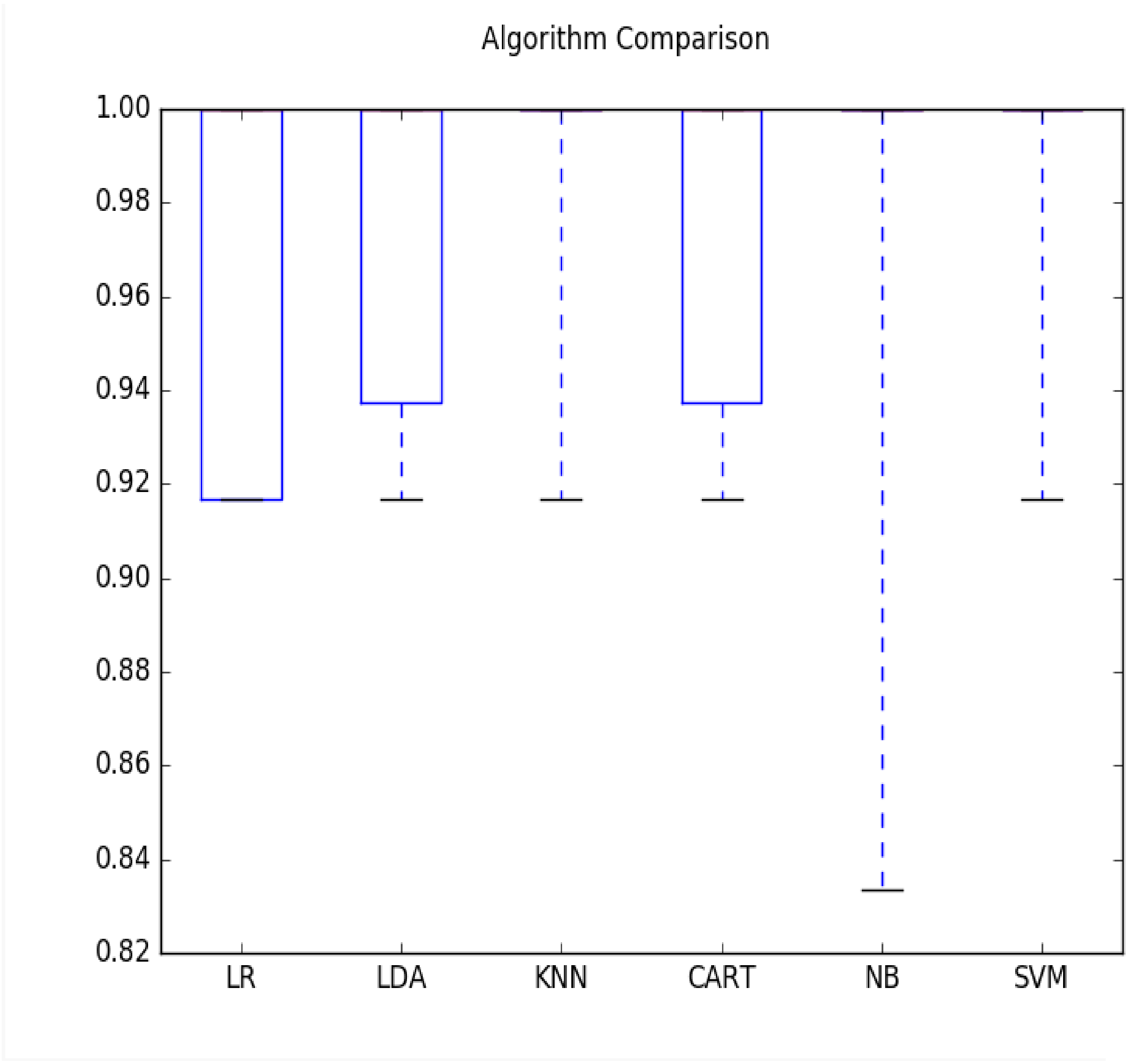
- Embrace Randomness in Machine Learning

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).

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

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



Compare Algorithm Accuracy

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 set.

This will give us an independent final check on the accuracy of the best model. It is valuable 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.

```
# 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))
```

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).

```
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

avg / total       0.90      0.90      0.90        30
```

Using Pandas to split dataset

```
import pandas as pd
import numpy as np
df = pandas.DataFrame(np.random.randn(10,4),columns=list('ABCD'))
print (df)
train = df.sample(frac=0.8)
print (train)
test = df.drop(train.index)
print(test)
```

Reading and Writing Image Files: Image to RGB array or RGB array to image

```
from matplotlib import pyplot as io
import numpy as np
from PIL import Image

# Image to array
img1 = io.imread(imagefilename)  #image is saved as rows * columns * 3 array
print (img1)




#Array to image file
array = np.zeros([10,20,3], dtype = np.uint8)

array[:,:10] = [255, 128, 0]    # Orange left side
array[:,10:]  = [0,0,255]   # Blue right side
print(array)
img2 = Image.fromarray(array)
img2.save('testrg.png')
```

Visualizing Two dimensional Scatter Plot

```
from matplotlib import pyplot as io
import numpy as np
import pandas as pd



df = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data', header = None)
df.tail()



# select setosa and versicolor
y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1, 1)

#extract sepal length and pedal length
X = df.iloc[0:100, [0,2]].values


#plot data
plt.scatter(X[:50, 0], X[:50, 1], color = 'red' , marker = 'o' , label =
'sectosa')
plt.scatter(X[50:100, 0], X[50:100, 1], color = 'blue' , marker = 'x' , label
= 'verisicolor')


plt.xlabel('sepal length [cm]')
plt.ylabel('patal length [cm]')
plt.legend(loc = 'upper left')
plt.show()
```