K Means Clustering in Python

stamfordresearch.com/k-means-clustering-in-python

K Means clustering is an unsupervised machine learning algorithm. An example of a supervised learning algorithm can be seen when looking at Neural Networks where the learning process involved both the inputs (x) and the outputs (y). During the learning process the error between the predicted outcome (predY) and actual outcome (y) is used to train the system. In an unsupervised method such as K Means clustering the outcome (y) variable is not used in the training process.

In this example we look at using the IRIS dataset and cover:

- Importing the sample IRIS dataset
- Converting the dataset to a Pandas Dataframe
- Visualising the classifications using scatter plots
- Simple performance metrics

Requirements: I am using Anaconda Python Distribution which has everything you need including Pandas, NumPy, Matplotlib and importantly SciKit-Learn. I am also using iPython Notebook but you can use whatever IDE you want.

Setup the environment and load the data

Bring in the libraries you need.

Python

```
importmatplotlib.pyplot asplt
fromsklearn importdatasets
fromsklearn.cluster importKMeans
importsklearn.metrics assm
importpandas aspd
importnumpy asnp
# Only needed if you want to display your plots inline if using Notebook
# change inline to auto if you have Spyder installed
%matplotlib inline


```

Next load the data. Scikit Learn has some sample datasets, by previously importing the datasets we can use the following

Python

```
# import some data to play with
iris=datasets.load_iris()
```

You can view the data running each line individually.

Python

```
iris.data
iris.feature_names
iris.target
iris.target_names
```

I like to work with Pandas Dataframes, so we will convert the data into that format. Note that we have separated out the inputs (x) and the outputs/labels (y).

Python

```
# Store the inputs as a Pandas Dataframe and set the column names
x=pd.DataFrame(iris.data)
x.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(iris.target)
y.columns=['Targets']

```

Visualise the data

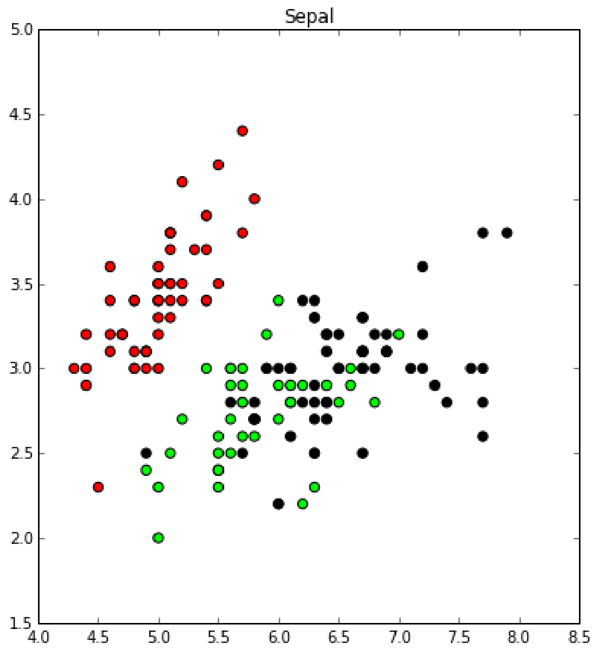
It is always important to have a look at the data. We will do this by plotting two scatter plots. One looking at the Sepal values and another looking at Petal. We will also set it to use some colours so it is clearer.

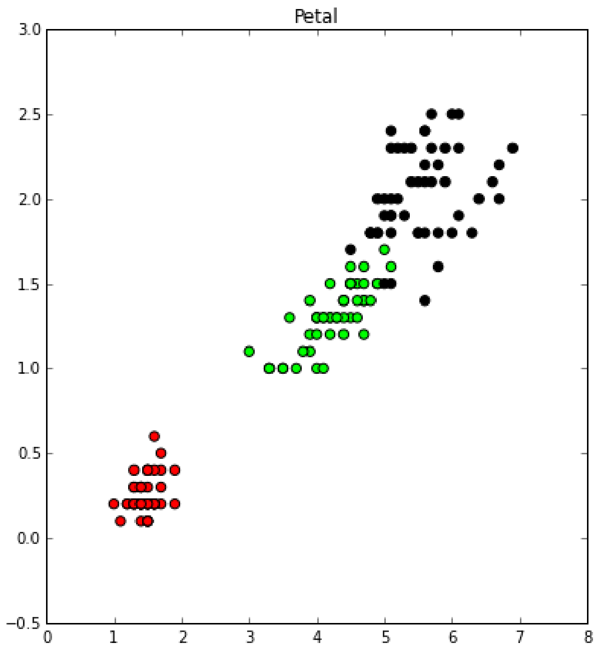
Python

```
# Set the size of the plot
plt.figure(figsize=(14,7))
# Create a colormap
colormap=np.array(['red','lime','black'])
# Plot Sepal
plt.subplot(1,2,1)
plt.scatter(x.Sepal_Length,x.Sepal_Width,c=colormap[y.Targets],s=40)
plt.title('Sepal')
plt.subplot(1,2,2)
plt.scatter(x.Petal_Length,x.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Petal')



```





Build the K Means Model

This is the easy part, providing you have the data in the correct format (which we do). Here we only need two lines. First we create the model and specify the number of clusters the model should find (n_clusters=3) next we fit the model to the data.

Python

```python
# K Means Cluster
model=KMeans(n_clusters=3)
model.fit(x)
```

Next we can view the results. This is the classes that the model decided, remember this is unsupervised and classified these purely based on the data.

Python

```python
# This is what KMeans thought
model.labels_
```

You should see something like this…

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 0, 0,
2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2,
0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0])
```

Visualise the classifier results

Lets plot the actual classes against the predicted classes from the K Means model.

Python

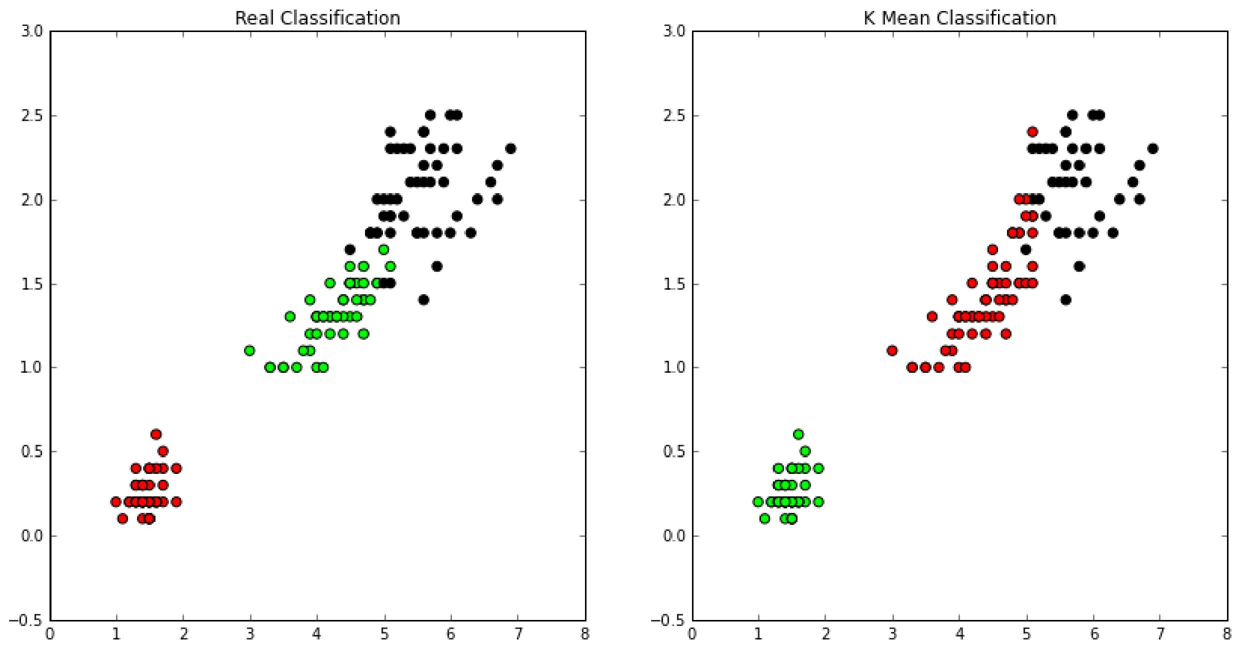
```
# View the results
# Set the size of the plot
plt.figure(figsize=(14,7))
# Create a colormap
colormap=np.array(['red','lime','black'])
# Plot the Original Classifications
plt.subplot(1,2,1)
plt.scatter(x.Petal_Length,x.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real Classification')
# Plot the Models Classifications
plt.subplot(1,2,2)
plt.scatter(x.Petal_Length,x.Petal_Width,c=colormap[model.labels_],s=40)
plt.title('K Mean Classification')



```

Here we are plotting the Petal Length and Width, however each plot changes the colors of the points using either *c=colormap[y.Targets]* for the original class and *c=colormap[model.labels_]* for the predicted classess.

The result is….



Ignore the colours (at the moment). Because the the model is unsupervised it did not know which label (class 0, 1 or 2) to assign to each class.

The Fix

Here we are going to change the class labels, we are not changing the any of the classification groups we are simply giving each group the correct number. We need to do this for measuring the performance.

Using this code below we using the *np.choose()* to assign new values, basically we are changing the 1's in the predicted values to 0's and the 0's to 1's. Class 2 matched so we can leave. By running the two print functions you can see that all we have done is swap the values.

NOTE: your results might be different to mine, if so you will have to figure out which class matches which and adjust the order of the values in the *np.choose()* function.

```
# The fix, we convert all the 1s to 0s and 0s to 1s.
predY=np.choose(model.labels_,[1,0,2]).astype(np.int64)
print(model.labels_)
print(predY)
```

Re-plot

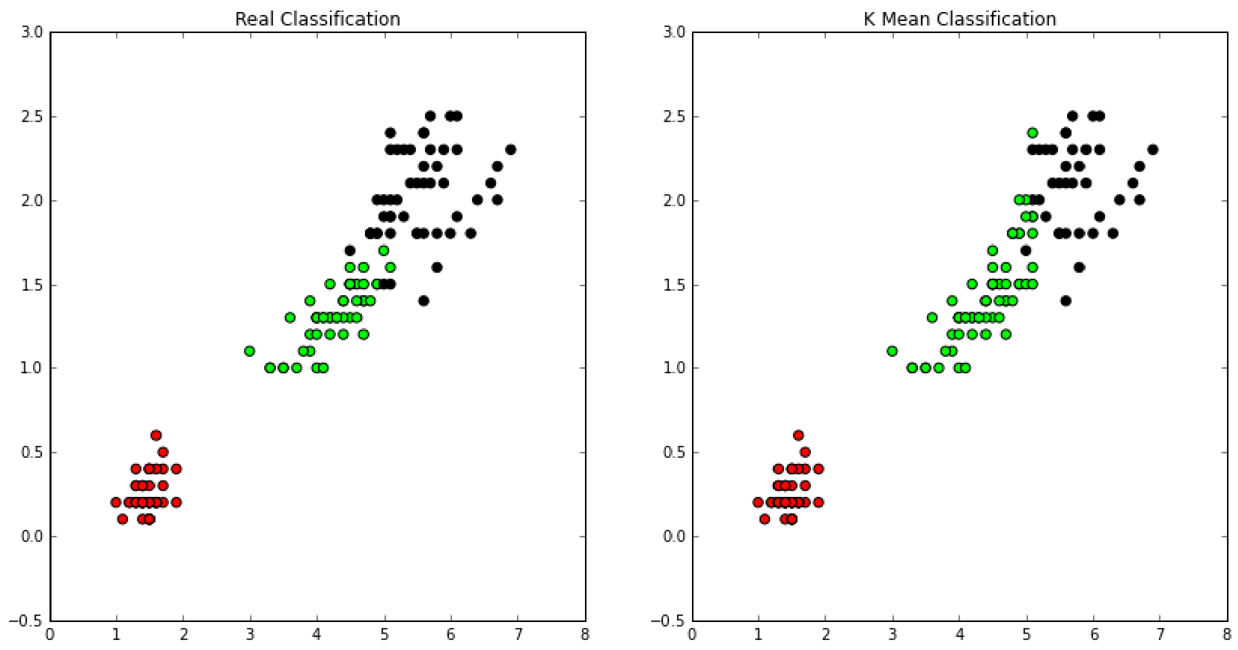
Now we can re plot the data as before but using *predY* instead of *model.labels_*.

Python

```
# View the results
# Set the size of the plot
plt.figure(figsize=(14,7))
# Create a colormap
colormap=np.array(['red','lime','black'])
# Plot Orginal
plt.subplot(1,2,1)
plt.scatter(x.Petal_Length,x.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real Classification')
# Plot Predicted with corrected values
plt.subplot(1,2,2)
plt.scatter(x.Petal_Length,x.Petal_Width,c=colormap[predY],s=40)
plt.title('K Mean Classification')



```



No we can see that the K Means classifier has identified one class correctly (red) but some blacks have been classed as greens and vice versa.

Performance Measures

There are a number of ways in which we can measure a classifiers performance. Here we will calculate the accuracy and also the confusion matrix.

We need to values *y* which is the true (original) values and *predY* which are the models values.

Accuracy

Python

```
# Performance Metrics
sm.accuracy_score(y,predY)
```

My result was 0.89333333333333331, so we can say that the model has an accuracy of 89.3%. Not base considering the model was unsupervised.

Confusion Matrix

```
# Confusion Matrix
sm.confusion_matrix(y,predY)
```

My results are…

```
array([[50, 0, 0],
[ 0, 48, 2],
[ 0, 14, 36]])
```

Hopefully the table below will render correctly, but we can summaries a the confusion matrix as shown below:

- correctly identifed all 0 classes as 0's
- correctly classified 48 class 1's but miss-classified 2 class 1's as class 2
- correctly classified 36 class 2's but miss-classified 14 class 2's as class 1

		Real Class		
		0	**1**	**2**
Predicted Class	**0**	50	0	0
	1	0	48	2
	2	0	14	36

The confusion matrix also allows for a wider range of performance metrics to be calculated, see here for more details: https://en.wikipedia.org/wiki/Confusion_matrix.