K Nearest Neighbors with Python

You've been given a classified data set from a company! They've hidden the feature column names but have given you the data and the target classes.

We'll try to use KNN to create a model that directly predicts a class for a new data point based off of the features.

Let's grab it and use it!

Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Get the Data

Set index_col=0 to use the first column as the index.

In [2]:
```
df=pd.read_csv('Classified Data',index_col=0)
```

In [3]:
```
df.head()
```

Out[3]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	
0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.23
1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.49
2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.28
3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.15
4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812	1.41

Standardize the Variables

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Any variables that are on a large scale will have a much larger effect on the distance between the observations, and hence on the KNN classifier, than variables that are on a small scale.

In [4]:
```
from sklearn.preprocessing import StandardScaler
```

In [5]:
```
scaler=StandardScaler()
```

In [6]:
```
scaler.fit(df.drop('TARGET CLASS',axis=1))
```

Out[6]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [7]:
```
scaled_features=scaler.transform(df.drop('TARGET CLASS',axis=1))
```

In [8]:
```
df_feat=pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

Out[8]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040772

Train Test Split

In [9]:
```
X=df_feat
y=df['TARGET CLASS']
```

In [10]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

Using KNN

Remember that we are trying to come up with a model to predict whether someone will TARGET CLASS or not. We'll start with k=1.

In [11]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [12]:
```
knn=KNeighborsClassifier()
```

In [13]:
```
knn.fit(X_train,y_train)
```

Out[13]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [14]:
```
pred=knn.predict(X_test)
```

Predictions and Evaluations

Let's evaluate our KNN model!

In [15]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [16]:
```
print(confusion_matrix(y_test,pred))
```

```
[[154   5]
 [ 12 129]]
```

In [17]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       159
           1       0.96      0.91      0.94       141

    accuracy                           0.94       300
   macro avg       0.95      0.94      0.94       300
weighted avg       0.94      0.94      0.94       300
```

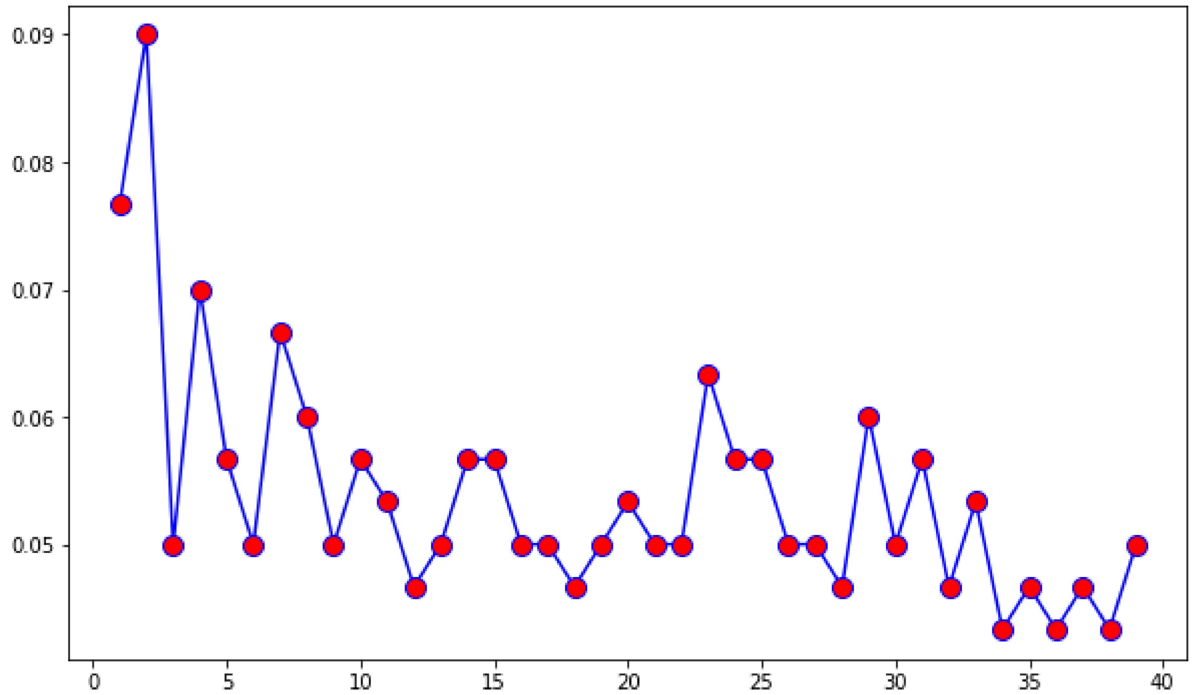
Choosing a K Value

Let's go ahead and use the elbow method to pick a good K Value:

In [20]:
```
error_rate=[]
for i in range(1,40):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i=knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [24]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue',marker='o',markersize=10,markerface
```

Out[24]:
```
[<matplotlib.lines.Line2D at 0x1ae0e9fc940>]
```



Here we can see that that after arouns K>23 the error rate just tends to hover around 0.06-0.05 Let's retrain the model with that and check the classification report!

In [35]:
```
#now lets choose diferent k value to compare ealier model

knn=KNeighborsClassifier(n_neighbors=17)
knn.fit(X_train,y_train)
pred=knn.predict(X_test)

print(classification_report(pred,y_test))

print('/n')

print(confusion_matrix(pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.96      0.94      0.95       162
           1       0.94      0.96      0.95       138

    accuracy                           0.95       300
   macro avg       0.95      0.95      0.95       300
weighted avg       0.95      0.95      0.95       300

/n
[[153   9]
 [  6 132]]
```

Great job!

#now our model is good in camparion of earler model in terms of accuracy

In [ ]: