

K-Nearest Neighbors

K-Nearest Neighbors (KNN) is one of the simplest algorithms used in Machine Learning for regression and classification problem. KNN algorithms use data and classify new data points based on similarity measures (e.g. distance function). Classification is done by a majority vote to its neighbors .

Working of KNN Algorithm

K-nearest neighbors (KNN) algorithm uses 'feature similarity' to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps –

Step 1 – For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step 2 – Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3 – For each point in the test data do the following –

- **3.1** – Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
- **3.2** – Now, based on the distance value, sort them in ascending order.
- **3.3** – Next, it will choose the top K rows from the sorted array.
- **3.4** – Now, it will assign a class to the test point based on most frequent class of these rows.

Step 4 – End

K Nearest Neighbors with Python

Now we will apply KNN Theory to 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.

Import Libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```


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.

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train,y_train)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=1, p=2,
                    weights='uniform')

pred = knn.predict(X_test)
```

Predictions and Evaluations

```
from sklearn.metrics import classification_report,confusion_matrix

print(confusion_matrix(y_test,pred))

[[125  18]
 [ 13 144]]

print(classification_report(y_test,pred))
```

	precision	recall	f1-score	support
0	0.91	0.87	0.89	143
1	0.89	0.92	0.90	157
avg / total	0.90	0.90	0.90	300

Choosing a K Value

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

```
error_rate = []

# Will take some time
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))
```

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

<matplotlib.text.Text at 0x11ca82ba8>



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

```
# FIRST A QUICK COMPARISON TO OUR ORIGINAL K=1
knn = KNeighborsClassifier(n_neighbors=1)

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

print('WITH K=1')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

WITH K=1

```
[[125 18]
 [ 13 144]]
```

	precision	recall	f1-score	support
0	0.91	0.87	0.89	143
1	0.89	0.92	0.90	157
avg / total	0.90	0.90	0.90	300

```
# NOW WITH K=23
knn = KNeighborsClassifier(n_neighbors=23)

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

print('WITH K=23')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

WITH K=23

```
[[132  11]
 [  5 152]]
```

	precision	recall	f1-score	support
0	0.96	0.92	0.94	143
1	0.93	0.97	0.95	157
avg / total	0.95	0.95	0.95	300