K-Nearest Neighbors

is a supervised learning algorithm. Where the data is 'trained' with data points corresponding to their classification. To predict the class of a given data point, it takes into account the classes of the 'K' nearest data points and chooses the class in which the majority of the 'K' nearest data points belong to as the predicted class.

About the dataset

Imagine a telecommunications provider has segmented its customer base by service usage patterns, categorizing the customers into four groups. If demographic data can be used to predict group membership, the company can customize offers for individual prospective customers. It is a classification problem. That is, given the dataset, with predefined labels, we need to build a model to be used to predict class of a new or unknown case.

The example focuses on using demographic data, such as region, age, and marital, to predict usage patterns.

The target field, called custcat, has four possible values that correspond to the four customer groups, as follows: 1- Basic Service 2- E-Service 3- Plus Service 4- Total Service

Our objective is to build a classifier, to predict the class of unknown cases. We will use a specific type of classification called K nearest neighbour.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn import preprocessing
%matplotlib inline
```

In [2]:
```
import os
os.chdir(r'C:\Users\HP\Downloads\Machine Learning with Python')
```

In [3]:
```
df = pd.read_csv('teleCust1000t.csv')
df.head()
```

Out[3]:

	region	tenure	age	marital	address	income	ed	employ	retire	gender	reside	custcat
0	2	13	44	1	9	64.0	4	5	0.0	0	2	1
1	3	11	33	1	7	136.0	5	5	0.0	0	6	4
2	3	68	52	1	24	116.0	1	29	0.0	1	2	3
3	2	33	33	0	12	33.0	2	0	0.0	1	1	1
4	2	23	30	1	9	30.0	1	2	0.0	0	4	3

Data Visualization and Analysis

```
In [4]: # Let's see how many of each class is in our data set
        df['custcat'].value_counts()
```

```
Out[4]: 3    281
        1    266
        4    236
        2    217
        Name: custcat, dtype: int64
```

```
In [5]: df.columns
```

```
Out[5]: Index(['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',
               'employ', 'retire', 'gender', 'reside', 'custcat'],
              dtype='object')
```

```
In [6]: # To use scikit-learn library, we have to convert the Pandas data frame to a Numpy array

        X = df[['region', 'tenure','age', 'marital', 'address', 'income', 'ed', 'employ',
                'retire', 'gender','reside']].to_numpy()
        X[0:5]
```

```
Out[6]: array([[  2.,  13.,  44.,   1.,   9.,  64.,   4.,   5.,   0.,   0.,   2.],
               [  3.,  11.,  33.,   1.,   7., 136.,   5.,   5.,   0.,   0.,   6.],
               [  3.,  68.,  52.,   1.,  24., 116.,   1.,  29.,   0.,   1.,   2.],
               [  2.,  33.,  33.,   0.,  12.,  33.,   2.,   0.,   0.,   1.,   1.],
               [  2.,  23.,  30.,   1.,   9.,  30.,   1.,   2.,   0.,   0.,   4.]])
```

```
In [7]: # Labels
        y = df['custcat'].values
        y[0:5]
```

```
Out[7]: array([1, 4, 3, 1, 3], dtype=int64)
```

Normalize Data

Data Standardization gives the data zero mean and unit variance, it is good practice, especially for algorithms such as KNN which is based on the distance of data points:

```
In [8]: X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
        X[0:5]
```

```
Out[8]: array([[-0.02696767, -1.055125  ,  0.18450456,  1.0100505 , -0.25303431,
                -0.12650641,  1.0877526 , -0.5941226 , -0.22207644, -1.03459817,
                -0.23065004],
               [ 1.19883553, -1.14880563, -0.69181243,  1.0100505 , -0.4514148 ,
                 0.54644972,  1.9062271 , -0.5941226 , -0.22207644, -1.03459817,
                 2.55666158],
               [ 1.19883553,  1.52109247,  0.82182601,  1.0100505 ,  1.23481934,
                 0.35951747, -1.36767088,  1.78752803, -0.22207644,  0.96655883,
                -0.23065004],
               [-0.02696767, -0.11831864, -0.69181243, -0.9900495 ,  0.04453642,
                -0.41625141, -0.54919639, -1.09029981, -0.22207644,  0.96655883,
                -0.92747794],
               [-0.02696767, -0.58672182, -0.93080797,  1.0100505 , -0.25303431,
                -0.44429125, -1.36767088, -0.89182893, -0.22207644, -1.03459817,
                 1.16300577]])
```

```
In [9]: # Train, Test Split
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4
        print ('Train set:', X_train.shape,  y_train.shape)
        print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (800, 11) (800,)
Test set: (200, 11) (200,)
```

Classification K nearest neighbor (KNN)

In [10]:
```python
from sklearn.neighbors import KNeighborsClassifier

k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat = neigh.predict(X_test)
yhat[0:5]
```

Out[10]:
```
array([1, 1, 3, 2, 4], dtype=int64)
```

Accuracy evaluation

accuracy classification score is a function that computes subset accuracy. This function is equal to the jaccard_score function. Essentially, it calculates how closely the actual labels and predicted labels are matched in the test set.

In [11]:
```python
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.5475
Test set Accuracy:  0.32
```

In [12]:
```python
# Changing K to 6
k = 6
neigh6 = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat6 = neigh6.predict(X_test)
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh6.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat6))
```

```
Train set Accuracy:  0.51625
Test set Accuracy:  0.31
```

how can we choose right value for K? The general solution is to reserve a part of your data for testing the accuracy of the model. Then choose k =1, use the training part for modeling, and calculate the accuracy of prediction using all samples in your test set. Repeat this process, increasing the k, and see which k is the best for your model.

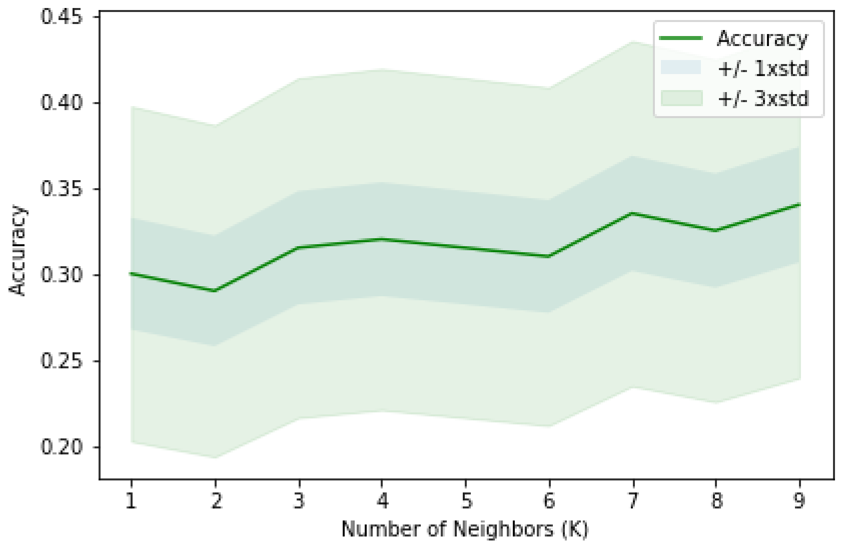
We can calculate the accuracy of KNN for different values of k.

In [13]:
```python
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))

for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])
```

```
mean_acc
```

Out[13]:
```
array([0.3  , 0.29 , 0.315, 0.32 , 0.315, 0.31 , 0.335, 0.325, 0.34 ])
```

In [14]:
```
# Plot the model accuracy for a different number of neighbors.

plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.fill_between(range(1,Ks),mean_acc - 3 * std_acc,mean_acc + 3 * std_acc, alpha=0.10,c
plt.legend(('Accuracy ', '+/- 1xstd','+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



In [15]:
```
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```

```
The best accuracy was with 0.34 with k= 9
```

In [ ]: