K-Nearest Neighbors

Will load a customer dataset related to a telecommunication company, clean it, use **KNN (K-Nearest Neighbours)** to predict the category of customers, and evaluate the accuracy of your model.

Required libraries:

In [1]:

```
import os
import itertools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import numpy as np
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
```

About the dataset
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

Objective is to build a **classifier**, to predict the class of unknown cases.

Load Data From CSV File

In [2]:

```
df = pd.read_csv('teleCust1000t.csv')
df.head()
```

Out[2]:

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

Data Visualization and Analysis

Count of each class is in our data set

In [3]:

```
df['custcat'].value_counts()
```

Out[3]:

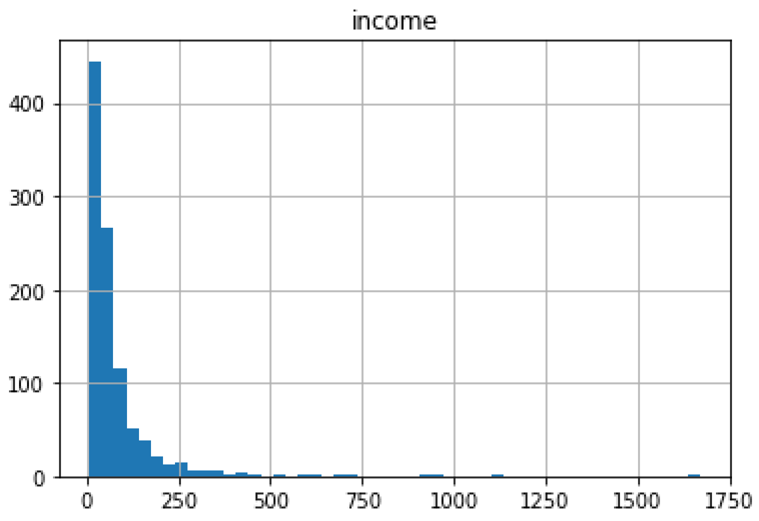
```
3    281
1    266
4    236
2    217
Name: custcat, dtype: int64
```

In [4]:

```
df.hist(column='income', bins=50)
```

Out[4]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000002B646D8612
8>]],
      dtype=object)
```



To use scikit-learn library, we have to convert the Pandas data frame to a Numpy array:

In [5]:

```
X = df.values
X[0:5]
```

Out[5]:

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

In [6]:

```
y = df['custcat'].values
y[0:5]
```

Out[6]:

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

Normalize Data

Data Standardization give data zero mean and unit variance

In [7]:

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

Out[7]:

```
array([[-0.02696767, -1.055125  ,  0.18450456,  1.0100505 , -0.25303431,
        -0.12650641,  1.0877526 , -0.5941226 , -0.22207644, -1.03459817,
        -0.23065004, -1.32797979],
       [ 1.19883553, -1.14880563, -0.69181243,  1.0100505 , -0.4514148 ,
         0.54644972,  1.9062271 , -0.5941226 , -0.22207644, -1.03459817,
         2.55666158,  1.35119935],
       [ 1.19883553,  1.52109247,  0.82182601,  1.0100505 ,  1.23481934,
         0.35951747, -1.36767088,  1.78752803, -0.22207644,  0.96655883,
        -0.23065004,  0.45813963],
       [-0.02696767, -0.11831864, -0.69181243, -0.9900495 ,  0.04453642,
        -0.41625141, -0.54919639, -1.09029981, -0.22207644,  0.96655883,
        -0.92747794, -1.32797979],
       [-0.02696767, -0.58672182, -0.93080797,  1.0100505 , -0.25303431,
        -0.44429125, -1.36767088, -0.89182893, -0.22207644, -1.03459817,
         1.16300577,  0.45813963]])
```

Train Test Split

In [8]:

```
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, 12) (800,)
Test set: (200, 12) (200,)
```

Classification

K nearest neighbor (KNN)

In [9]:

```
from sklearn.neighbors import KNeighborsClassifier
```

Training

In [10]:

```
k = 4

neighbor = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neighbor
```

Out[10]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=4, p=2,
           weights='uniform')
```

Predicting

In [11]:

```
yhat = neighbor.predict(X_test)
yhat[0:5]
```

Out[11]:

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

Accuracy evaluation

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

In [12]:

```
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neighbor.predict(X_train
)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.82875
Test set Accuracy:  0.735
```

We can calculate the accuracy of KNN for different Ks.

In [13]:

```
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
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
```

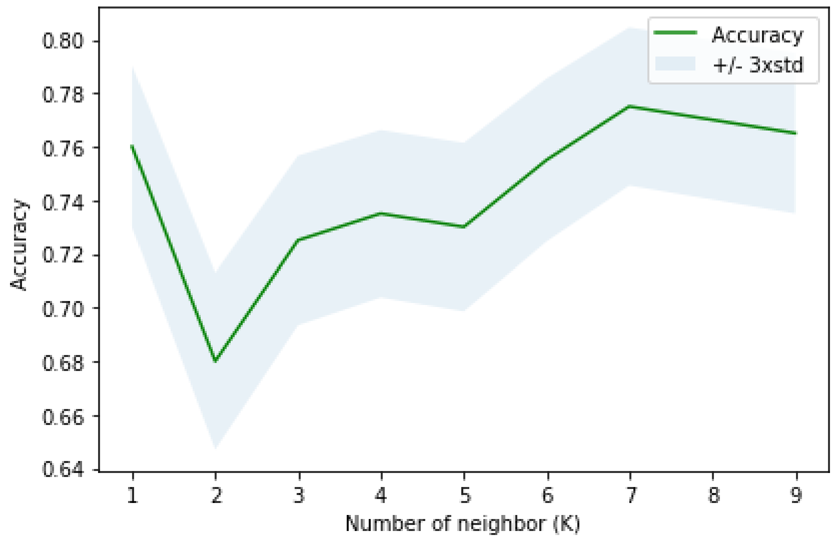
Out[13]:

```
array([0.76 , 0.68 , 0.725, 0.735, 0.73 , 0.755, 0.775, 0.77 , 0.765])
```

Plot model accuracy for Different number of Neighbors

In [14]:

```
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.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of neighbor (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 0.775 with k= 7
```

In [ ]:

```
```

In [ ]:

```
```