K-Nearest Neighbors

Objetivos

- Usar K Nearest Neighbors para clasificar datos

En este laboratorio cargará un dataset de clientes, ajustará los datos y utilizará K-NN para predecir un punto de datos. Pero, **qué es K-NN?**

K-NN es un algoritmo de aprendizaje supervisado. Los datos son "entrenados" con puntos de datos que tienen clarita clasificación. Cuando se va a predecir un punto, se toman en cuenta los K puntos más cercanos para determinar su clasificación.

Aquí una visualización del algoritmo

En este caso, tenemos puntos de las clases A y B. Queremos predecir qué es la estrella (el punto de test). Si consideramos K=3 (los 3 puntos más cercanos) la predicción dirá que la clase es B. Si K=6, la predicción dirá que es de la clase A.

En este sentido, es importante considerar el valor de K. No obstante, el diagrama brinda una buena idea de cómo se comporta el algoritmo, considera los K vecinos más cercanos para clasificar un punto de test.

Tabla de contenido

Cargaremos las librerías requeridas

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
```

Acerca del dataset

Imagine un proveedor de servicios de telecomunicaciones ha segmentado a sus clientes en base a sus patrones de uso de servicios y los ha categorizado en 4 grupos. Si los datos demográficos pueden ser utilizados para predecir el grupo, la compañía pued personalizar su oferta. Es un problema de clasificación, es decir, dado un dataset, con etiquetas predefinidas, necesitamos construir un modelo para predecir la clase de un caso desconocido.

El ejemplo se centra en la utilización de datos demográficos, tales como región, edad y estado marital para predecir los patrones de uso.

El campo objetivo, llamado **custcat** tiene 4 posibles valores que corresponden a los 4 grupos de clientes:

1- Basic Service 2- E-Service 3- Plus Service 4- Total Service

Nuestro objetivo es construir un clasificador. Utilizaremos K-NN.

Descarguemos el dataset.

In [2]:
```
#!wget -O teleCust1000t.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/teleCust1000t.csv
```

```
import urllib.request
url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%203/data/teleCust1000t.csv'
filename = 'teleCust1000t.csv'
urllib.request.urlretrieve(url, filename)
```

Out[2]:
```
('teleCust1000t.csv', <http.client.HTTPMessage at 0x1a3c5b08f70>)
```

Cargamos los datos desde el CSV

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

Visualización de datos y análisis

Veamos cuántos de cada clase hay en el dataset

In [4]:
```
df['custcat'].value_counts()
```

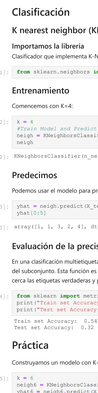
Out[4]:
```
3    281
1    266
4    236
2    217
Name: custcat, dtype: int64
```

281 Plus Service, 266 Basic-service, 236 Total Service, and 217 E-Service customers

Puede explorar los datos utilizando técnicas de visualización

In [5]:
```
df.hist(column='income', bins=50)
```

Out[5]:
```
array([[<AxesSubplot:title={'center':'income'}>]], dtype=object)
```



Conjunto de características

Definamos el conjunto de características X:

In [6]:
```
df.columns
```

Out[6]:
```
Index(['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',
       'employ', 'retire', 'gender', 'reside', 'custcat'],
      dtype='object')
```

Para poder usar la libreria scikit-learn debemos convertir el dataframe Pandas en un arreglo de NumPy:

In [7]:
```
X = df[['region', 'tenure','age', 'marital', 'address', 'income', 'ed', 'employ','retire', 'gender', 'reside']] .values  #.astype(float)
X[0:5]
```

Out[7]:
```
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.]])
```

Cuales son nuestras etiquetas?

In [8]:
```
y = df['custcat'].values
y[0:5]
```

Out[8]:
```
array([1, 4, 3, 1, 3], dtype=int64)
```

Normalizamos los datos

La estandarización de datos devuelve datos con media nula y varianza unitaria, lo que es una buena práctica, especialmente para algoritmos como K-NN que se basan en distancia:

In [9]:
```
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]
```

Out[9]:
```
array([[-0.0269767, -1.055125 ,  0.18545456,  1.0100505 , -0.25303431,
        -0.12650641,  1.0877526 , -0.5941226 , -0.22207644, -1.03459817,
        -0.23065004],
       [-0.02691712,  1.14880563, -0.69181243,  1.0100505 , -0.4514148 ,
         0.54644972,  1.9062271 , -0.5941226 , -0.22207644, -1.03459817,
         2.55666158],
       [-0.02691712,  1.52109247,  0.82182601,  1.0100505 ,  1.23481934,
         0.35951747, -1.36767088,  1.8957118 , -0.22207644,  0.96655883,
        -0.23065004],
       [-0.02691767, -0.11834864, -0.69181243, -0.9900495 ,  0.04453642,
        -0.41625141, -0.54919639, -1.0851738 , -0.22207644,  0.96655883,
        -0.92746794],
       [-0.02691767, -0.58672182, -0.93080797,  1.0100505 , -0.25303431,
        -0.44232525, -1.36767088, -0.89182893, -0.22207644, -1.03459817,
        -0.92746794]])
```

Train Test Split

La precisión fuera de muestra es el porcentaje de predicciones correctas que el modelo realiza en datos en los que NO ha sido entrenado. Entrenar y testear con el mismo dataset probablemente lleve a una baja precisión fuera de muestra, debido a la alta probabilidad de sobreajuste.

Es importante que nuestros modelos tengan una alta precisión fuera de muestra, ya que el propósito es realizar predicciones sobre datos desconocidos.

Cómo mejoramos la precisión fuera de muestra? Una forma es utilizar Train/Test Split: se divide el dataset en conjuntos de entrenamiento y test, que son mutuamente excluisvos. Luego de esto, se entrena con el conjunto de entrenamiento y se testea con el de test.

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=0.2, 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,)
```

Clasificación

K nearest neighbor (KNN)

Importamos la libreria

Clasificador que implementa K-NN

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

Entrenamiento

Comencemos con K=4:

In [12]:
```
k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

Out[12]:
```
KNeighborsClassifier(n_neighbors=4)
```

Predecimos

Podemos usar el modelo para predecir sobre el conjunto de test:

In [13]:
```
yhat = neigh.predict(X_test)
yhat[0:5]
```

Out[13]:
```
array([1, 1, 3, 2, 4], dtype=int64)
```

Evaluación de la precisión

En una clasificación multietiqueta, **accuracy classification score** es una función que computa la precisión del subconjunto. Esta función es igual al score de similaridad de Jaccard. Esencialmente calcula qué tan cerca las etiquetas verdaderas y predecidas coinciden en el conjunto de prueba.

In [14]:
```
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
```

Práctica

Construyamos un modelo con K=6:

In [15]:
```
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
```

Qué hay acerca de otros valores de K?

K en K-NN es el número de vecinos a examinar. Debe ser especificado por el usuario, entonces, cómo lo elegimos?

La solución general es reservar parte de los datos para testear la precisión del modelo. Luego se elige K=1, se usa la parte de entrenamiento para entrenar y se calcula la precisión usando todas las muestras en el conjunto de test. Se repite el proceso incrementando K y se selecciona el K que ajusta mejor.

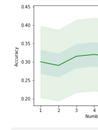
Podemos calcular la precisión de K-NN para diferentes valores de K:

In [16]:
```
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[16]:
```
array([0.3  , 0.29 , 0.315, 0.32 , 0.315, 0.31 , 0.335, 0.325, 0.34 ])
```

Imprimimos la precisión del modelo para diferentes números de vecinos

In [17]:
```
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,color="green")
plt.legend(('Accuracy ', '+/- 1xstd','+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



In [18]:
```
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
```