K-Nearest Neighbor

In [3]:
```
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in ./opt/anaconda3/lib/python3.9/site-packages (0.24.2)
Requirement already satisfied: joblib>=0.11 in ./opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (1.1.0)
Requirement already satisfied: scipy>=0.19.1 in ./opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (1.7.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in ./opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: numpy>=1.13.3 in ./opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (1.20.3)
```

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn import preprocessing
%matplotlib inline
```

In [5]:
```
# Customer classification by a telecommunication company bassed on four groups of customers:
#1- Basic Service 2- E-Service 3- Plus Service 4- Total Service

df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-
df.head(10)
```

Out[5]:

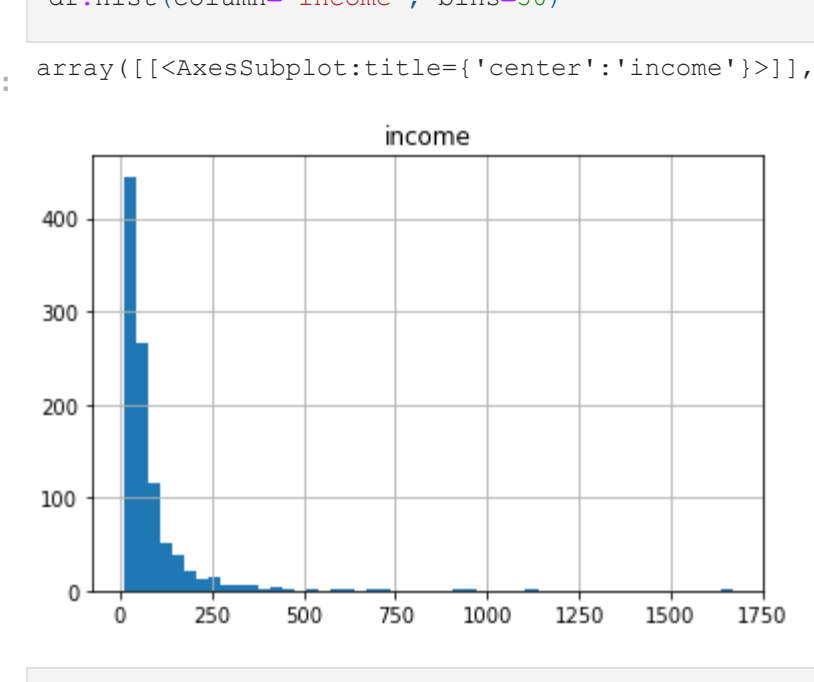
	region	tenure	age	marital	address	income	ed	employ	retire	gender	reside	custcat
0	2	13	44	1	9	64.0	4	5	0.0	0	2	1
1	3	11	33	1	7	136.0	5	5	0.0	0	6	4
2	3	68	52	1	24	116.0	1	29	0.0	1	2	3
3	2	33	33	0	12	33.0	2	0	0.0	1	1	1
4	2	23	30	1	9	30.0	1	2	0.0	0	4	3
5	2	41	39	0	17	78.0	2	16	0.0	1	1	3
6	3	45	22	1	2	19.0	2	4	0.0	1	5	2
7	2	38	35	0	5	76.0	2	10	0.0	0	3	4
8	3	45	59	1	7	166.0	4	31	0.0	0	5	3
9	1	68	41	1	21	72.0	1	22	0.0	0	3	2

In [6]:
```
#Visualizing and Analizing the Data

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

Out[6]:
```
3    281
1    266
4    236
2    217
Name: custcat, dtype: int64
```

In [7]:
```
df.hist(column='income', bins=50)
```

Out[7]:
```
array([[<AxesSubplot:title={'center':'income'}>]], dtype=object)
```



In [8]:
```
df.columns
```

Out[8]:
```
Index(['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',
       'employ', 'retire', 'gender', 'reside', 'custcat'],
      dtype='object')
```

In [9]:
```
X = df[['region', 'tenure','age', 'marital', 'address', 'income', 'ed', 'employ','retire', 'gender', 'reside']]
X[0:5]
```

Out[9]:
```
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 [10]:
```
y = df['custcat'].values
y[0:5]
```

Out[10]:
```
array([1, 4, 3, 1, 3])
```

In [11]:
```
#Normalizing the data

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

Out[11]:
```
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 [12]:
```
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,)
```

In [13]:
```
#Classification
#K nearest neighbor (KNN)

from sklearn.neighbors import KNeighborsClassifier
```

In [14]:
```
k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

Out[14]:
```
KNeighborsClassifier(n_neighbors=4)
```

In [15]:
```
#predicting
yhat = neigh.predict(X_test)
yhat[0:5]
```

Out[15]:
```
array([1, 1, 3, 2, 4])
```

In [16]:
```
#Accuracy evaluation
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 [17]:
```
#Building the algorithm with k = 6

k = 6
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

Out[17]:
```
KNeighborsClassifier(n_neighbors=6)
```

In [18]:
```
yhat = neigh.predict(X_test)
yhat[0:5]
```

Out[18]:
```
array([3, 3, 3, 4, 4])
```

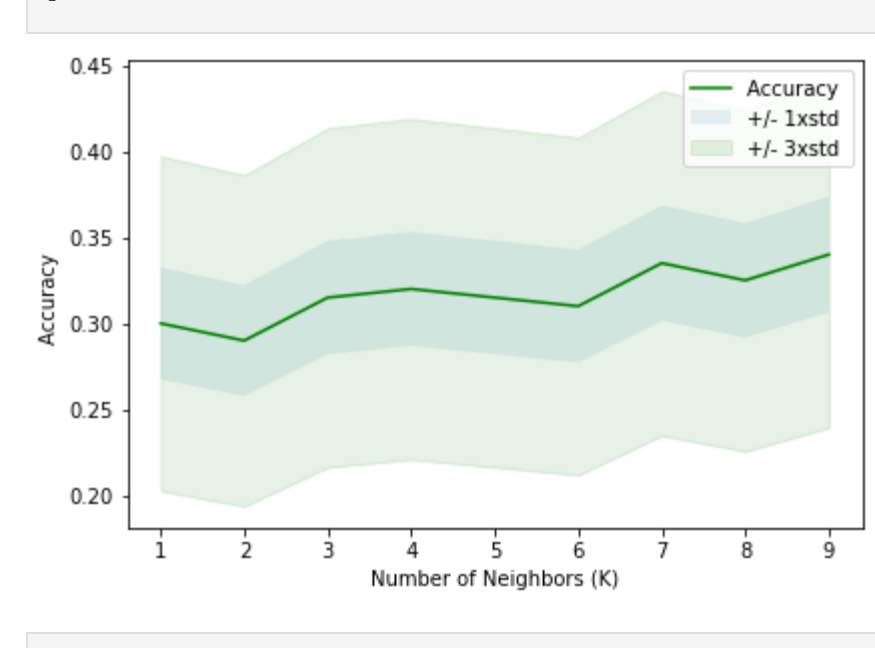
In [19]:
```
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

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

In [20]:
```
#For other values of k

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[20]:
```
array([0.3  , 0.29 , 0.315, 0.32 , 0.315, 0.31 , 0.335, 0.325, 0.34 ])
```

In [21]:
```
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 [22]:
```
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
```

Decision Tree

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
```

In [3]:
```
# The data: A set of patients, all of whom suffered from the same illness.
#During their course of treatment, each patient responded to one of 5 medications:
#Drug A, Drug B, Drug c, Drug x and y.

df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-
```

In [4]:
```
df.head(10)
```

Out[4]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY
5	22	F	NORMAL	HIGH	8.607	drugX
6	49	F	NORMAL	HIGH	16.275	drugY
7	41	M	LOW	HIGH	11.037	drugC
8	60	M	NORMAL	HIGH	15.171	drugY
9	43	M	LOW	NORMAL	19.368	drugY

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

Out[5]:
```
(200, 6)
```

In [7]:
```
""""Using df as the Drug.csv data read by pandas, declare the following variables:
X as the Feature Matrix (data of df)
y as the response vector (target)
Remove the column containing the target name since it doesn't contain numeric values."""

X = df[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].values
X[0:5]
```

Out[7]:
```
array([[23, 'F', 'HIGH', 'HIGH', 25.355],
       [47, 'M', 'LOW', 'HIGH', 13.093],
       [47, 'M', 'LOW', 'HIGH', 10.114],
       [28, 'F', 'NORMAL', 'HIGH', 7.798],
       [61, 'F', 'LOW', 'HIGH', 18.043]], dtype=object)
```

In [8]:
```
#using pandas.get_dummies() to convert the categorical variable into dummy/indicator variables.

from sklearn import preprocessing
le_sex = preprocessing.LabelEncoder()
le_sex.fit(['F','M'])
X[:,1] = le_sex.transform(X[:,1])


le_BP = preprocessing.LabelEncoder()
le_BP.fit([ 'LOW', 'NORMAL', 'HIGH'])
X[:,2] = le_BP.transform(X[:,2])


le_Chol = preprocessing.LabelEncoder()
le_Chol.fit([ 'NORMAL', 'HIGH'])
X[:,3] = le_Chol.transform(X[:,3])

X[0:5]
```

Out[8]:
```
array([[23, 0, 0, 0, 25.355],
       [47, 1, 1, 0, 13.093],
       [47, 1, 1, 0, 10.114],
       [28, 0, 2, 0, 7.798],
       [61, 0, 1, 0, 18.043]], dtype=object)
```

In [9]:
```
y = df["Drug"]
y[0:5]
```

Out[9]:
```
0    drugY
1    drugC
2    drugC
3    drugX
4    drugY
Name: Drug, dtype: object
```

In [10]:
```
#Setting up the decision tree
from sklearn.model_selection import train_test_split
```

In [11]:
```
"""train_test_split will return 4 different parameters. Name them:
X_trainset, X_testset, y_trainset, y_testset

The train_test_split will need the parameters:
X, y, test_size=0.3, and random_state=3.

The X and y are the arrays required before the split, the test_size represents the ratio of the testing dataset

X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.3, random_state=3)
```

In [12]:
```
print('Shape of X training set {}'.format(X_trainset.shape),'&',' Size of Y training set {}'.format(y_trainset.
```

```
Shape of X training set (140, 5) &  Size of Y training set (140,)
```

In [13]:
```
print('Shape of X training set {}'.format(X_testset.shape),'&',' Size of Y training set {}'.format(y_testset.sh
```

```
Shape of X training set (60, 5) &  Size of Y training set (60,)
```

In [15]:
```
#Modeling
#First create an instance of the DecisionTreeClassifier called drugTree.
#Inside of the classifier, specify criterion="entropy" so that the information gain of each node can be seen.

drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
drugTree # it shows the default parameters
```

Out[15]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In [16]:
```
#fit the data with the training feature matrix X_trainset and training response vector y_trainset

drugTree.fit(X_trainset,y_trainset)
```

Out[16]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In [18]:
```
#Prediction
predTree = drugTree.predict(X_testset)
```

In [19]:
```
print (predTree [0:5])
print (y_testset [0:5])
```

```
['drugY' 'drugX' 'drugX' 'drugX' 'drugX']
40     drugY
51     drugX
139    drugX
197    drugX
170    drugX
Name: Drug, dtype: object
```

In [20]:
```
#Evaluation

from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, predTree))
```

```
DecisionTrees's Accuracy:  0.9833333333333333
```

In [28]:
```
#Visualization

!conda install -c conda-forge python-graphviz -y
!conda install -c conda-forge pydotplus -y
```

In [29]:
```
from  io import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline
```

In [30]:
```
dot_data = StringIO()
filename = "drugtree.png"
featureNames = df.columns[0:5]
out=tree.export_graphviz(drugTree,feature_names=featureNames, out_file=dot_data, class_names= np.unique(y_train
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png(filename)
img = mpimg.imread(filename)
plt.figure(figsize=(100, 200))
plt.imshow(img,interpolation='nearest')
```

```
---------------------------------------------------------------------------
InvocationException                       Traceback (most recent call last)
/var/folders/cn/0zbpg_4n7556t3y1p21b9w1w0000gn/T/ipykernel_7694/3096224755.py in <module>
      4 out=tree.export_graphviz(drugTree,feature_names=featureNames, out_file=dot_data, class_names= np.unique
(y_trainset), filled=True,  special_characters=True,rotate=False)
      5 graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
----> 6 graph.write_png(filename)
      7 img = mpimg.imread(filename)
      8 plt.figure(figsize=(100, 200))

~/opt/anaconda3/lib/python3.9/site-packages/pydotplus/graphviz.py in <lambda>(path, f, prog)
   1808                 lambda path,
   1809                 f=frmt,
-> 1810                 prog=self.prog: self.write(path, format=f, prog=prog)
   1811             )
   1812

~/opt/anaconda3/lib/python3.9/site-packages/pydotplus/graphviz.py in write(self, path, prog, format)
   1916
   1917             else:
-> 1918                 fobj.write(self.create(prog, format))
   1919         finally:
   1920             if close:

~/opt/anaconda3/lib/python3.9/site-packages/pydotplus/graphviz.py in create(self, prog, format)
   2028
   2029         if status != 0:
-> 2030             raise InvocationException(
   2031                 'Program terminated with status: %d. stderr follows: %s' % (
   2032                     status, stderr_output))

InvocationException: Program terminated with status: -9. stderr follows: []
```

Logistic Regression

Logistic Regression to create a model for a telecommunication company, to predict when its customers will leave for a competitor

In [31]:
```
!pip install scikit-learn==0.23.1
```

In [32]:
```
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
%matplotlib inline
import matplotlib.pyplot as plt
```

In [34]:
```
"""The dataset includes information about:

Customers who left within the last month - the column is called Churn
Services that each customer has signed up for - phone, multiple lines, internet, online security, online backup
Customer account information - how long they had been a customer, contract, payment method, paperless billing,
Demographic info about customers - gender, age range, and if they have partners and dependents"""

churn_df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNe
churn_df.head(10)
```

Out[34]:

	tenure	age	address	income	ed	employ	equip	callcard	wireless	longmon	...	pager	internet	callwait	confer	ebill	loglon
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	4.40	...	1.0	0.0	1.0	1.0	0.0	1.48
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	9.45	...	0.0	0.0	0.0	0.0	0.0	2.24
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	6.30	...	0.0	0.0	0.0	1.0	0.0	1.84
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	6.05	...	1.0	1.0	1.0	1.0	1.0	1.80
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	7.10	...	0.0	0.0	1.0	1.0	0.0	1.96
5	68.0	52.0	17.0	120.0	1.0	24.0	0.0	1.0	0.0	20.70	...	0.0	0.0	0.0	0.0	0.0	3.03
6	42.0	40.0	7.0	37.0	2.0	8.0	1.0	1.0	1.0	8.25	...	0.0	1.0	1.0	1.0	1.0	2.11
7	9.0	21.0	1.0	17.0	2.0	2.0	0.0	0.0	0.0	2.90	...	0.0	0.0	0.0	0.0	0.0	1.06
8	35.0	50.0	26.0	140.0	2.0	21.0	0.0	1.0	0.0	6.50	...	0.0	0.0	1.0	1.0	0.0	1.87
9	49.0	51.0	27.0	63.0	4.0	19.0	0.0	1.0	0.0	12.85	...	0.0	1.0	1.0	0.0	1.0	2.55

10 rows × 28 columns

In [35]:
```
#Pre-processing and selecting the data
#selecting some features for the modeling and changing the target data type to integer, (a requirement by the s

churn_df = churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip',   'callcard', 'wireless','c
churn_df['churn'] = churn_df['churn'].astype('int')
churn_df.head()
```

Out[35]:

	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	1
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	1
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	0
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	0
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	0

In [36]:
```
churn_df.shape
```

Out[36]:
```
(200, 10)
```

In [37]:
```
X = np.asarray(churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip']])
X[0:5]
```

Out[37]:
```
array([[ 11.,  33.,   7., 136.,   5.,   5.,   0.],
       [ 33.,  33.,  12.,  33.,   2.,   0.,   0.],
       [ 23.,  30.,   9.,  30.,   1.,   2.,   0.],
       [ 38.,  35.,   5.,  76.,   2.,  10.,   1.],
       [  7.,  35.,  14.,  80.,   2.,  15.,   0.]])
```

In [38]:
```
y = np.asarray(churn_df['churn'])
y [0:5]
```

Out[38]:
```
array([1, 1, 0, 0, 0])
```

In [39]:
```
#Normalizing the data set

from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[39]:
```
array([[-1.13518441, -0.62595491, -0.4588971 ,  0.4751423 ,  1.6961288 ,
        -0.58477841, -0.85972695],
       [-0.11604313, -0.62595491,  0.03454064, -0.32886061, -0.6433592 ,
        -1.14437497, -0.85972695],
       [-0.57928917, -0.85594447, -0.261522  , -0.35227817, -1.42318853,
        -0.92053635, -0.85972695],
       [ 0.11557989, -0.47262854, -0.65627219,  0.00679109, -0.6433592 ,
        -0.02518185,  1.16316   ],
       [-1.32048283, -0.47262854,  0.23191574,  0.03801451, -0.6433592 ,
         0.53441472, -0.85972695]])
```

In [40]:
```
#spliting dataset into train and test set:

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: (160, 7) (160,)
Test set: (40, 7) (40,)
```

In [41]:
```
#Modeling

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
```

Out[41]:
```
LogisticRegression(C=0.01, solver='liblinear')
```

In [42]:
```
#Predicting using test set

yhat = LR.predict(X_test)
yhat
```

Out[42]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0])
```

In [43]:
```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

Out[43]:
```
array([[0.54132919, 0.45867081],
       [0.60593357, 0.39406643],
       [0.56277713, 0.43722287],
       [0.63432489, 0.36567511],
       [0.56431839, 0.43568161],
       [0.55386646, 0.44613354],
       [0.52237207, 0.47762793],
       [0.60514349, 0.39485651],
       [0.41069572, 0.58930428],
       [0.6333873 , 0.3666127 ],
       [0.58068791, 0.41931209],
       [0.62768628, 0.37231372],
       [0.47559883, 0.52440117],
       [0.4267593 , 0.5732407 ],
       [0.66172417, 0.33827583],
       [0.55092315, 0.44907685],
       [0.51749946, 0.48250054],
       [0.485743  , 0.514257  ],
       [0.49011451, 0.50988549],
       [0.52423349, 0.47576651],
       [0.61619519, 0.38380481],
       [0.52696302, 0.47303698],
       [0.63957168, 0.36042832],
       [0.52205164, 0.47794836],
       [0.50572852, 0.49427148],
       [0.70706202, 0.29293798],
       [0.55266286, 0.44733714],
       [0.52271594, 0.47728406],
       [0.51638863, 0.48361137],
       [0.71331391, 0.28668609],
       [0.67862111, 0.32137889],
       [0.50896403, 0.49103597],
       [0.42348082, 0.57651918],
       [0.71495838, 0.28504162],
       [0.59711064, 0.40288936],
       [0.63808839, 0.36191161],
       [0.39957895, 0.60042105],
       [0.52127638, 0.47872362],
       [0.65975464, 0.34024536],
       [0.5114172 , 0.4885828 ]])
```

In [44]:
```
# Evaluation, Jaccard index

from sklearn.metrics import jaccard_score
jaccard_score(y_test, yhat,pos_label=0)
```

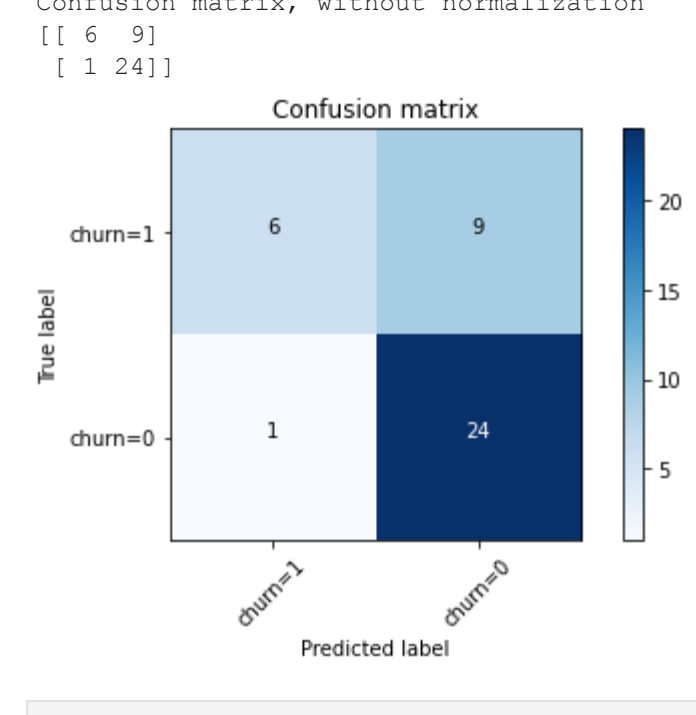
Out[44]:
```
0.7058823529411765
```

In [45]:
```
#Confusion matrics

from sklearn.metrics import classification_report, confusion_matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
print(confusion_matrix(y_test, yhat, labels=[1,0]))
```

```
[[ 6  9]
 [ 1 24]]
```

In [46]:
```
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['churn=1','churn=0'],normalize= False,  title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[ 6  9]
 [ 1 24]]
```



In [47]:
```
print (classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

           0       0.73      0.96      0.83        25
           1       0.86      0.40      0.55        15

    accuracy                           0.75        40
   macro avg       0.79      0.68      0.69        40
weighted avg       0.78      0.75      0.72        40
```

In [48]:
```
# log loss

from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob)
```

Out[48]:
```
0.6017092478101186
```

In [49]:
```
# Regression model with the same dataset, but this time, use different __solver__ and __regularization__ values
# new __logLoss__ value?

LR2 = LogisticRegression(C=0.01, solver='sag').fit(X_train,y_train)
yhat_prob2 = LR2.predict_proba(X_test)
print ("LogLoss: : %.2f" % log_loss(y_test, yhat_prob2))
```

```
LogLoss: : 0.61
```

Support Vector Machines (SVM)

In [50]:
```
# Using human cell records, and classifying cells to whether the samples are benign or malignant.

!pip install scikit-learn==0.23.1
```

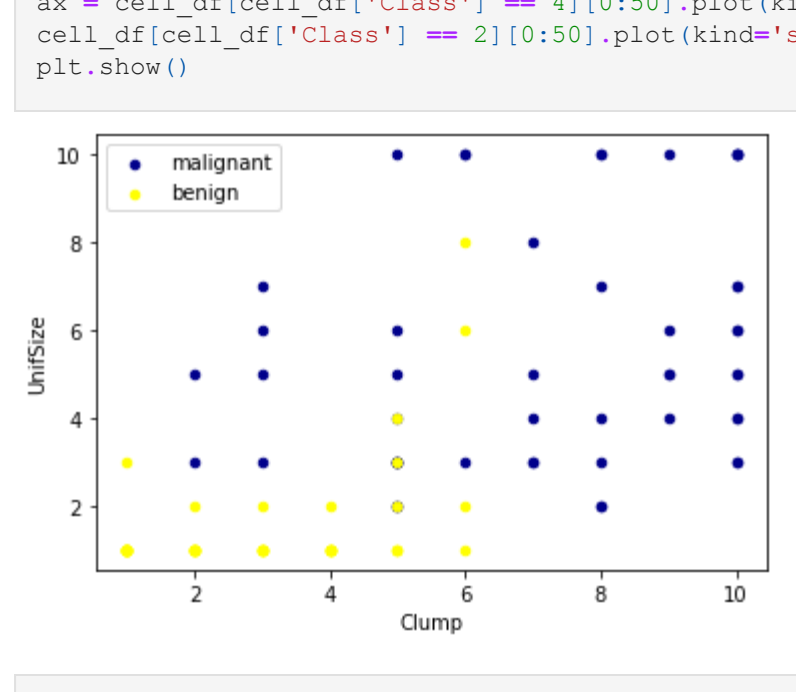
In [51]:
```
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
%matplotlib inline
import matplotlib.pyplot as plt
```

In [53]:
```
#dataset that is publicly available from the UCI Machine Learning Repository (Asuncion and Newman, 2007)
#he dataset consists of several hundred human cell sample records, each of which contains the values of a set o

cell_df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNet
cell_df.head(10)
```

Out[53]:

	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2
5	1017122	8	10	10	8	7	10	9	7	1	4
6	1018099	1	1	1	1	2	10	3	1	1	2
7	1018561	2	1	2	1	2	1	3	1	1	2
8	1033078	2	1	1	1	2	1	1	1	5	2
9	1033078	4	2	1	1	2	1	2	1	1	2

In [54]:
```
"""The ID field contains the patient identifiers. The characteristics of the cell samples from each patient are

The Class field contains the diagnosis, as confirmed by separate medical procedures, as to whether the samples

Distribution of the classes based on Clump thickness and Uniformity of cell size:"""

ax = cell_df[cell_df['Class'] == 4][0:50].plot(kind='scatter', x='Clump', y='UnifSize', color='DarkBlue', label
cell_df[cell_df['Class'] == 2][0:50].plot(kind='scatter', x='Clump', y='UnifSize', color='Yellow', label='benig
plt.show()
```



In [55]:
```
cell_df.dtypes
```

Out[55]:
```
ID             int64
Clump          int64
UnifSize       int64
UnifShape      int64
MargAdh        int64
SingEpiSize    int64
BareNuc       object
BlandChrom     int64
NormNucl       int64
Mit            int64
Class          int64
dtype: object
```

In [56]:
```
#BareNuc column includes some values that are not numerical. We can drop those rows:

cell_df = cell_df[pd.to_numeric(cell_df['BareNuc'], errors='coerce').notnull()]
cell_df['BareNuc'] = cell_df['BareNuc'].astype('int')
cell_df.dtypes
```

Out[56]:
```
ID             int64
Clump          int64
UnifSize       int64
UnifShape      int64
MargAdh        int64
SingEpiSize    int64
BareNuc        int64
BlandChrom     int64
NormNucl       int64
Mit            int64
Class          int64
dtype: object
```

In [57]:
```
feature_df = cell_df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpiSize', 'BareNuc', 'BlandChrom', 'Nor
X = np.asarray(feature_df)
X[0:5]
```

Out[57]:
```
array([[ 5,  1,  1,  1,  2,  1,  3,  1,  1],
       [ 5,  4,  4,  5,  7, 10,  3,  2,  1],
       [ 3,  1,  1,  1,  2,  2,  3,  1,  1],
       [ 6,  8,  8,  1,  3,  4,  3,  7,  1],
       [ 4,  1,  1,  3,  2,  1,  3,  1,  1]])
```

In [58]:
```
#Model to predict the value of Class (that is, benign (=2) or malignant (=4)).
#As this field can have only two possible values, we need to change its measurement level to reflect thi

cell_df['Class'] = cell_df['Class'].astype('int')
y = np.asarray(cell_df['Class'])
y [0:5]
```

Out[58]:
```
array([2, 2, 2, 2, 2])
```

In [59]:
```python
#Split dataset into train and test set

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: (546, 9) (546,)
Test set: (137, 9) (137,)
```

In [60]:
```python
"""Modeling (SVM with Scikit-learn)
The SVM algorithm offers a choice of kernel functions for performing its processing. Basically, mapping data ir
The mathematical function used for the transformation is known as the kernel function, and can be of different
1.Linear
2.Polynomial
3.Radial basis function (RBF)
4.Sigmoid
Each of these functions has its characteristics, its pros and cons, and its equation, but as there's no easy wa
We usually choose different functions in turn and compare the results.
Using the default, RBF (Radial Basis Function)."""

from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

Out[60]:
```
SVC()
```

In [61]:
```python
yhat = clf.predict(X_test)
yhat [0:5]
```

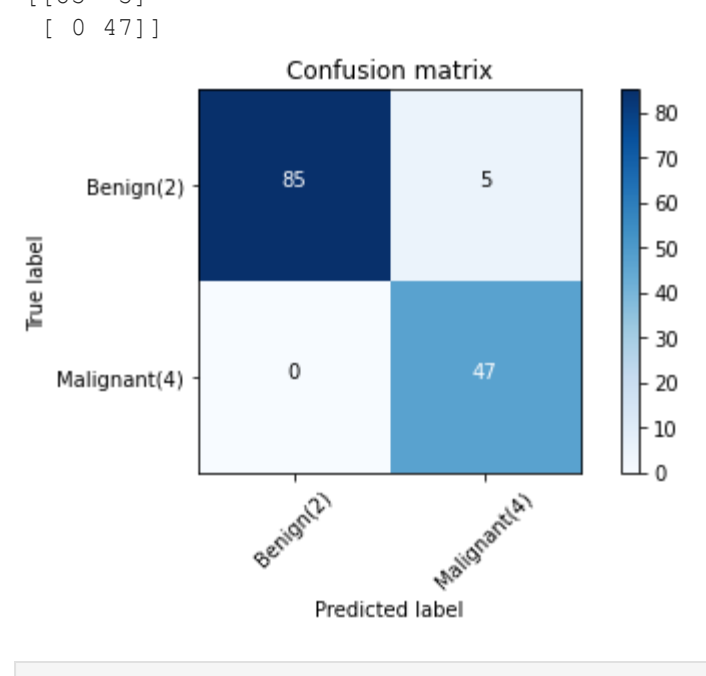
Out[61]:
```
array([2, 4, 2, 4, 2])
```

In [62]:
```python
#Evaluation

from sklearn.metrics import classification_report, confusion_matrix
import itertools
```

In [63]:
```python
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [64]:
```python
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=[2,4])
np.set_printoptions(precision=2)

print (classification_report(y_test, yhat))

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['Benign(2)','Malignant(4)'],normalize= False,  title='Confusion matr
```

```
              precision    recall  f1-score   support

           2       1.00      0.94      0.97        90
           4       0.90      1.00      0.95        47

    accuracy                           0.96       137
   macro avg       0.95      0.97      0.96       137
weighted avg       0.97      0.96      0.96       137

Confusion matrix, without normalization
[[85  5]
 [ 0 47]]
```



In [65]:
```python
# Using f1_score from sklearn

from sklearn.metrics import f1_score
f1_score(y_test, yhat, average='weighted')
```

Out[65]:
```
0.9639038982104676
```

In [66]:
```python
# jaccard index for accuracy

from sklearn.metrics import jaccard_score
jaccard_score(y_test, yhat,pos_label=2)
```

Out[66]:
```
0.9444444444444444
```

In [67]:
```python
#Rebuilding the model, with a _linear_ kernel.
# using __kernel='linear'__ option, when defining the svm. How does the accuracy change with the new kernel fur

clf2 = svm.SVC(kernel='linear')
clf2.fit(X_train, y_train)
yhat2 = clf2.predict(X_test)
print("Avg F1-score: %.4f" % f1_score(y_test, yhat2, average='weighted'))
print("Jaccard score: %.4f" % jaccard_score(y_test, yhat2,pos_label=2))
```

```
Avg F1-score: 0.9639
Jaccard score: 0.9444
```

In [ ]: