Logistic Regression on Customer churn

Creating a logistic Model for a telecommunication comapany to predict whehter its customers will leave or not. Logistic regression uses sigmoid function to predict probability of a class. Logistic regression passes input through logistic or sigmoid function to get results as probability. Using Logistic Regression we have to predict who is leaving and why?

Working on DataSet from Kaggle and Using Logistic Regression to predict churn or not..

Target : churn (data set has customers who left last month) Features : Service customer signed for, contract, payement info ...

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

In [17]:

```
data = pd.read_csv("data/ChurnData.csv")
```

In [18]:

```
print(data.shape
```

```
(200, 28)
```

In [19]:

```
#Print no of integers, floats and strings
data.dtypes.value_counts()
```

Out[19]:

```
float64    28
dtype: int64
```

In [20]:

```
data.head()
```

Out[20]:

ure	age	address	income	ed	employ	equip	callcard	wireless	longmon	...
11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	4.40	...
33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	9.45	...
23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	6.30	...
38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	6.05	...
7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	7.10	...

× 28 columns

Preprocessing Steps

1. Select Features and convert target variable to int.
2. Extract X and target y and Normalize data
3. Split the data into train and test sets.

1. Select Features and Convert target variable to int.

In [21]:

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

Out[21]:

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

2. Extracting X and y and Normalize data

In [23]:

```
X = np.asarray(churn_df[['tenure', 'age', 'address', 'income', 'ed', 'e
y = np.asarray(churn_df['churn'])
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[23]:

```
array([[-1.13518441, -0.62595491, -0.4588971 ,  0.4751423
,  1.6961288 ,
        -0.58477841, -0.85972695],
       [-0.11604313, -0.62595491,  0.03454064, -0.3288606
1, -0.6433592 ,
        -1.14437497, -0.85972695],
       [-0.57928917, -0.85594447, -0.261522  , -0.3522781
7, -1.42318853,
        -0.92053635, -0.85972695],
       [ 0.11557989, -0.47262854, -0.65627219,  0.0067910
9, -0.6433592 ,
        -0.02518185,  1.16316   ],
       [-1.32048283, -0.47262854,  0.23191574,  0.0380145
1, -0.6433592 ,
         0.53441472, -0.85972695]])
```

3. Split Data to Train and Test sets

In [24]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.
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,)
```

Modeling with Logisitc Regression

In [25]:

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

In [27]:

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

Out[27]:

```
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, 0, 1, 0, 0, 0, 1, 0, 0,
0])
```

predict_proba returns estimates for all classes, ordered by the label of classes. So, the first column is the probability of class 1, P(Y=1|X), and second column is probability of class 0, P(Y=0|X) From yhat and yhat_prob we can get what will be the values of yhat predicted.

In [29]:

```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

Out[29]:

```
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 ]])
```

Confusion Matrix

In [30]:

```
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]))
```

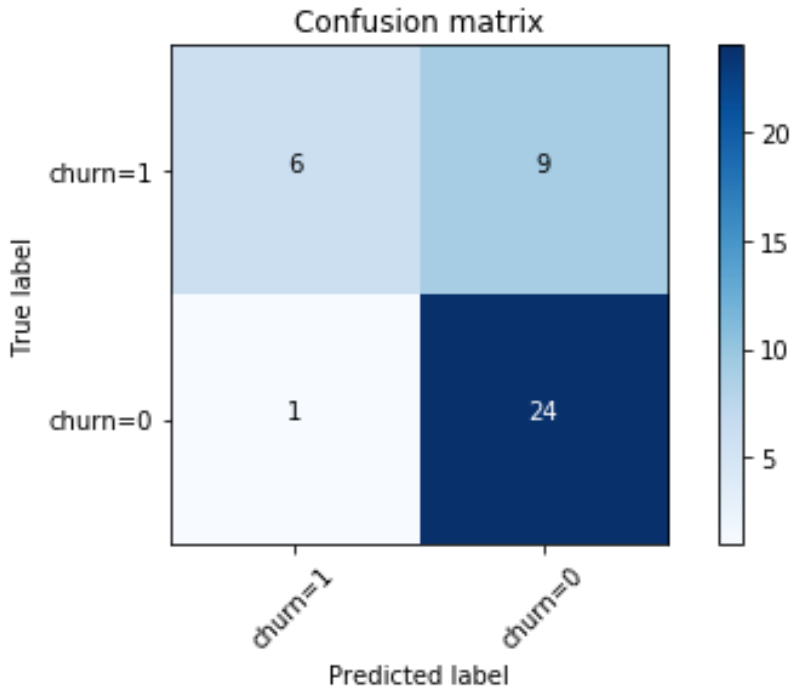
```
[[ 6  9]
 [ 1 24]]
```

In [31]:

```
# Compute confusion matrix
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'],normali
```

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



Confusion Matrix shows models ability to correctly predict or seperate classes .

```
  Note- Out of 15 customers whose actual churn value is 1 ,clas
sifier could predict only 6 correctly.
  1. For churn value 1 : Recall = 6/(6+9)= .4 , Precision = 6/
(6+1)= .86 and F1-score = .55
  2. For churn value 0 : Recall = 24/(24+1)= .96 , Precision =
 24/(24+9)= .73 and F1-score = .83
  Classifier is good for predicting customers with churn value
 0.
```

Precison /Recall and F1- score

In [32]:

```
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
```

Summary

Logistic Classifier could predict customer churn value 0 with f1-score .83 and churn value 1 with f1-score .55.

Classifier could predict churn value 0 properly but failed to give good results in case of churn value 1.

In [ ]: