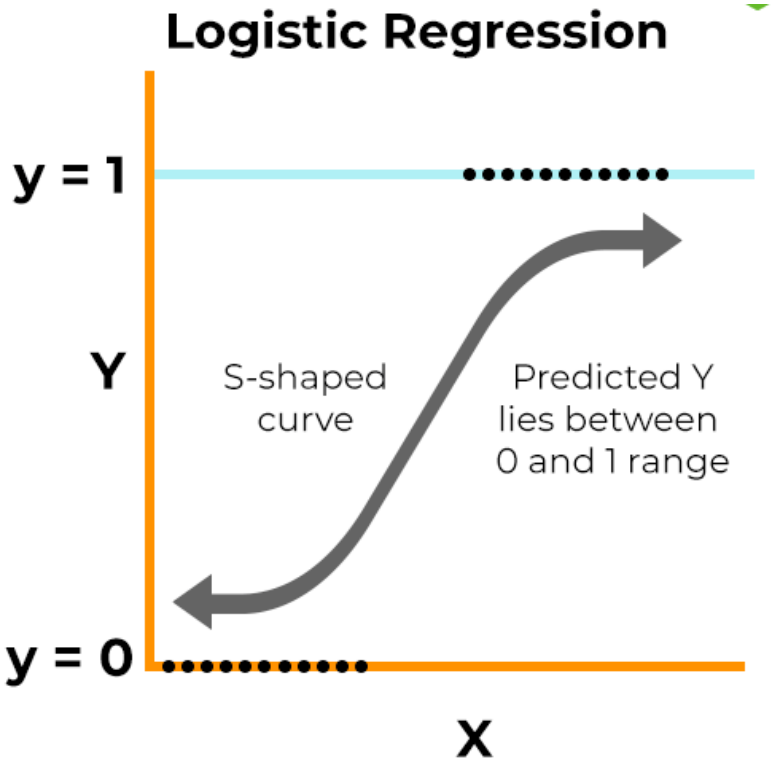


Logistic Regression with Python

Estimated time needed: **25** minutes

Objectives

After completing this lab you will be able to:

- Use scikit Logistic Regression to classify
- Understand confusion matrix

In this notebook, you will learn Logistic Regression, and then, you'll create a model for a telecommunication company, to predict when its customers will leave for a competitor, so

that they can take some action to retain the customers.

Table of contents

What is the difference between Linear and Logistic Regression?

While Linear Regression is suited for estimating continuous values (e.g. estimating house price), it is not the best tool for predicting the class of an observed data point. In order to estimate the class of a data point, we need some sort of guidance on what would be the **most probable class** for that data point. For this, we use **Logistic Regression**.

Recall linear regression:

As you know, **Linear regression** finds a function that relates a continuous dependent variable, **y**, to some predictors (independent variables x_1, x_2, etc.). For example, simple linear regression assumes a function of the form:

$$ y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots $$
and finds the values of parameters $\theta_0, \theta_1, \theta_2$, etc, where the term θ_0 is the "intercept". It can be generally shown as:

$$ $h_\theta(x)$ = \theta^TX $$

Logistic Regression is a variation of Linear Regression, used when the observed dependent variable, **y**, is categorical. It produces a formula that predicts the probability of the class label as a function of the independent variables.

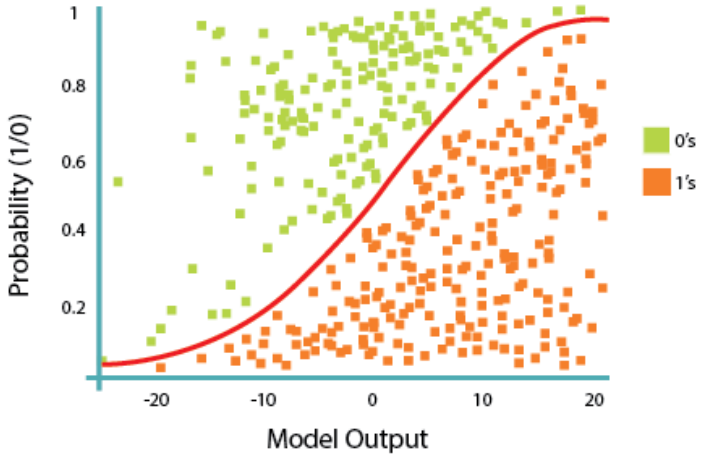
Logistic regression fits a special s-shaped curve by taking the linear regression function and transforming the numeric estimate into a probability with the following function, which is

called the sigmoid function σ:

$$ $h_\theta(x)$ = \sigma({\theta^TX}) = \frac {e^{(\theta_0 + \theta_1 x_1 + \theta_2 x_2 +...)}} {1 + e^{(\theta_0 + \theta_1 x_1 + \theta_2 x_2 +\cdots)}} $$
Or: $$ ProbabilityOfaClass_1 = P(Y=1|X) = \sigma({\theta^TX}) = \frac{e^{\theta^TX}} {1+e^{\theta^TX}} $$

In this equation, ${\theta^TX}$ is the regression result (the sum of the variables weighted by the coefficients), `exp` is the exponential function and $\sigma(\theta^TX)$ is the sigmoid or logistic function, also called logistic curve. It is a common "S" shape (sigmoid curve).

So, briefly, Logistic Regression passes the input through the logistic/sigmoid but then treats the result as a probability:



The objective of the **Logistic Regression** algorithm, is to find the best parameters θ, for $$h_\theta(x)$$ = $\sigma({\theta^TX})$, in such a way that the model best predicts the class of each case.

Customer churn with Logistic Regression

A telecommunications company is concerned about the number of customers leaving their land-line business for cable competitors. They need to understand who is leaving. Imagine that you are an analyst at this company and you have to find out who is leaving and why.

In [1]:
```
!pip install scikit-learn==0.23.1
```

```
Collecting scikit-learn==0.23.1
  Downloading scikit_learn-0.23.1-cp37-cp37m-manylinux1_x86_64.whl (6.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.8/6.8 MB 85.1 MB/s eta 0:00:00:00:01
00:01
Requirement already satisfied: numpy>=1.13.3 in /home/jupyterlab/conda/envs/python/l
ib/python3.7/site-packages (from scikit-learn==0.23.1) (1.21.6)
Requirement already satisfied: scipy>=0.19.1 in /home/jupyterlab/conda/envs/python/l
ib/python3.7/site-packages (from scikit-learn==0.23.1) (1.7.3)
Collecting joblib>=0.11 (from scikit-learn==0.23.1)
  Downloading joblib-1.3.2-py3-none-any.whl (302 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.2/302.2 kB 46.2 MB/s eta 0:00:00
Collecting threadpoolctl>=2.0.0 (from scikit-learn==0.23.1)
  Downloading threadpoolctl-3.1.0-py3-none-any.whl (14 kB)
Installing collected packages: threadpoolctl, joblib, scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 0.20.1
    Uninstalling scikit-learn-0.20.1:
      Successfully uninstalled scikit-learn-0.20.1
Successfully installed joblib-1.3.2 scikit-learn-0.23.1 threadpoolctl-3.1.0
```

Let's first import required libraries:

In [2]:
```
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
```

About the dataset

We will use a telecommunications dataset for predicting customer churn. This is a historical customer dataset where each row represents one customer. The data is relatively easy to understand, and you may uncover insights you can use immediately. Typically it is less expensive to keep customers than acquire new ones, so the focus of this analysis is to predict the customers who will stay with the company.
This data set provides information to help you predict what behavior will help you to retain customers. You can analyze all relevant customer data and develop focused customer retention programs.

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, device protection, tech support, and streaming TV and movies
- Customer account information – how long they had been a customer, contract, payment method, paperless billing, monthly charges, and total charges
- Demographic info about customers – gender, age range, and if they have partners and dependents

Load the Telco Churn data

Telco Churn is a hypothetical data file that concerns a telecommunications company's efforts to reduce turnover in its customer base. Each case corresponds to a separate customer and it records various demographic and service usage information. Before you can work with the data, you must use the URL to get the ChurnData.csv.

To download the data, we will use `!wget` to download it from IBM Object Storage.

In [3]:
```
#Click here and press Shift+Enter
!wget -O ChurnData.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain
```

```
--2024-06-14 09:22:28--  https://cf-courses-data.s3.us.cloud-object-storage.appdomai
n.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%203/data/ChurnD
ata.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)... 169.63.118.104, 169.63.118.104
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses
-data.s3.us.cloud-object-storage.appdomain.cloud)|169.63.118.104|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 35943 (35K) [text/csv]
Saving to: ‘ChurnData.csv’

ChurnData.csv       100%[===================>]  35.10K  --.-KB/s    in 0.002s

2024-06-14 09:22:28 (22.8 MB/s) - ‘ChurnData.csv’ saved [35943/35943]
```

Did you know? When it comes to Machine Learning, you will likely be working with large datasets. As a business, where can you host your data? IBM is offering a unique opportunity for businesses, with 10 Tb of IBM Cloud Object Storage: Sign up now for free

Load Data From CSV File

In [4]:
```
churn_df = pd.read_csv("ChurnData.csv")
churn_df.head()
```

Out[4]:

	tenure	age	address	income	ed	employ	equip	callcard	wireless	longmon	...	pa
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	4.40	...	
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	9.45	...	
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	6.30	...	
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	6.05	...	
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	7.10	...	

5 rows × 28 columns

Data pre-processing and selection

Let's select some features for the modeling. Also, we change the target data type to be an integer, as it is a requirement by the skitlearn algorithm:

In [5]:
```
churn_df = churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip',
churn_df['churn'] = churn_df['churn'].astype('int')
churn_df.head()
```

Out[5]:

	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

Practice

How many rows and columns are in this dataset in total? What are the names of columns?

In [6]:
```
# write your code here
```

▶ Click here for the solution

Let's define X, and y for our dataset:

In [7]:
```
X = np.asarray(churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equ
X[0:5]
```

Out[7]:
```
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 [8]:
```
y = np.asarray(churn_df['churn'])
y [0:5]
```

Out[8]:
```
array([1, 1, 0, 0, 0])
```

Also, we normalize the dataset:

In [9]:
```
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[9]:
```
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]])
```

Train/Test dataset

We split our dataset into train and test set:

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_st
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 (Logistic Regression with Scikit-learn)

Let's build our model using **LogisticRegression** from the Scikit-learn package. This function implements logistic regression and can use different numerical optimizers to find parameters, including 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga' solvers. You can find extensive information about the pros and cons of these optimizers if you search it in the internet.

The version of Logistic Regression in Scikit-learn, support regularization. Regularization is a technique used to solve the overfitting problem of machine learning models. **C** parameter

indicates **inverse of regularization strength** which must be a positive float. Smaller values specify stronger regularization. Now let's fit our model with train set:

In [11]:
```
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[11]:
```
LogisticRegression(C=0.01, solver='liblinear')
```

Now we can predict using our test set:

In [12]:
```
yhat = LR.predict(X_test)
yhat
```

Out[12]:
```
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 0, P(Y=0|X), and second column is probability of class 1, P(Y=1|X):

In [13]:
```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

Out[13]:
```
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 ]])
```

Evaluation

jaccard index

Let's try the jaccard index for accuracy evaluation. we can define jaccard as the size of the intersection divided by the size of the union of the two label sets. If the entire set of predicted labels for a sample strictly matches with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.

In [14]:
```
from sklearn.metrics import jaccard_score
jaccard_score(y_test, yhat,pos_label=0)
```

Out[14]: 0.7058823529411765

confusion matrix

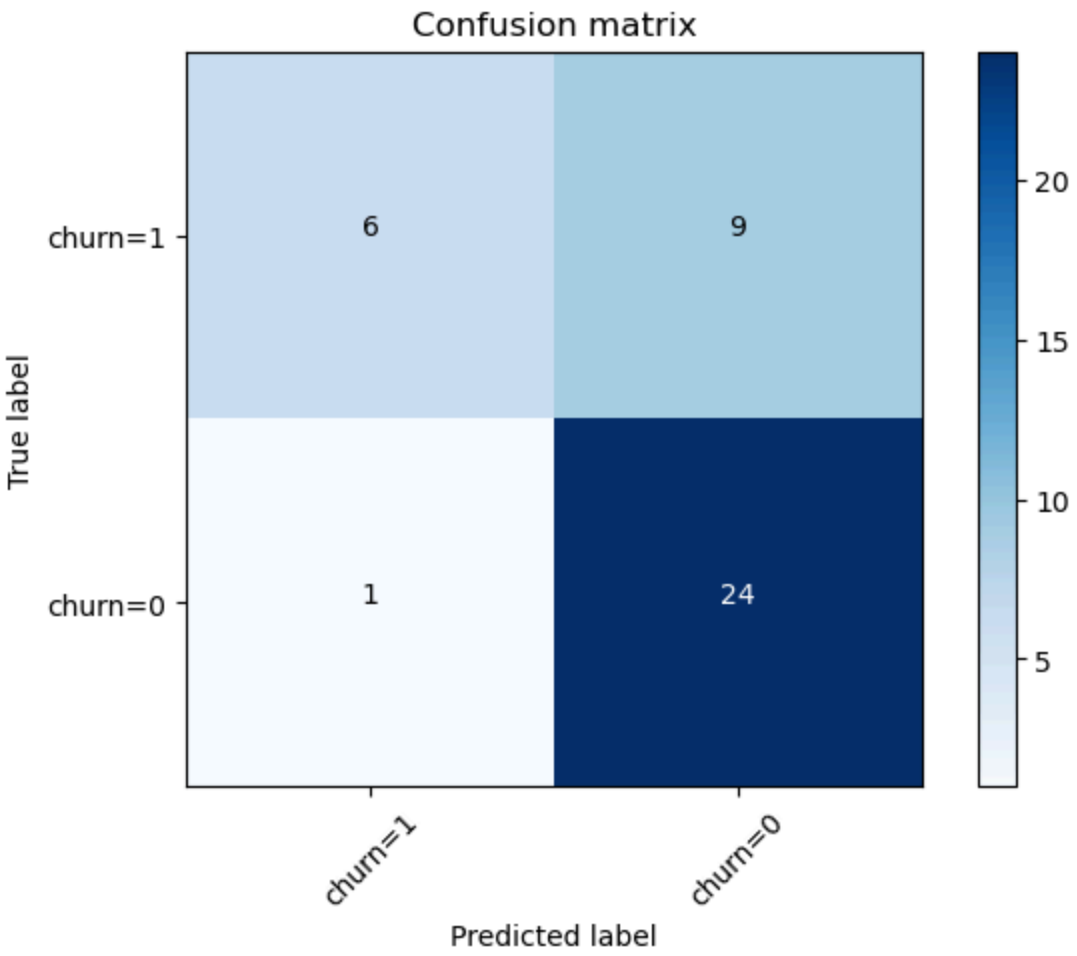
Another way of looking at the accuracy of the classifier is to look at **confusion matrix**.

In [15]:
```
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 [16]:
```
# 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'],normalize= False,
```

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



Let's look at first row. The first row is for customers whose actual churn value in the test set is 1. As you can calculate, out of 40 customers, the churn value of 15 of them is 1. Out of these 15 cases, the classifier correctly predicted 6 of them as 1, and 9 of them as 0.

This means, for 6 customers, the actual churn value was 1 in test set and classifier also correctly predicted those as 1. However, while the actual label of 9 customers was 1, the classifier predicted those as 0, which is not very good. We can consider it as the error of the model for first row.

What about the customers with churn value 0? Lets look at the second row. It looks like there were 25 customers whom their churn value were 0.

The classifier correctly predicted 24 of them as 0, and one of them wrongly as 1. So, it has done a good job in predicting the customers with churn value 0. A good thing about the confusion matrix is that it shows the model's ability to correctly predict or separate the classes. In a specific case of the binary classifier, such as this example, we can interpret these numbers as the count of true positives, false positives, true negatives, and false negatives.

In [17]:
```
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
```

Based on the count of each section, we can calculate precision and recall of each label:

- **Precision** is a measure of the accuracy provided that a class label has been predicted. It is defined by: precision = TP / (TP + FP)
- **Recall** is the true positive rate. It is defined as: Recall = TP / (TP + FN)

So, we can calculate the precision and recall of each class.

F1 score: Now we are in the position to calculate the F1 scores for each label based on the precision and recall of that label.

The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. It is a good way to show that a classifer has a good value for both recall and precision.

Finally, we can tell the average accuracy for this classifier is the average of the F1-score for both labels, which is 0.72 in our case.

log loss

Now, let's try **log loss** for evaluation. In logistic regression, the output can be the probability of customer churn is yes (or equals to 1). This probability is a value between 0 and 1. Log loss(Logarithmic loss) measures the performance of a classifier where the predicted output is a probability value between 0 and 1.

In [18]:
```
from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob)
```

Out[18]:
```
0.6017092478101185
```

Practice

Try to build Logistic Regression model again for the same dataset, but this time, use different __solver__ and __regularization__ values? What is new __logLoss__ value?

In [20]:
```
# write your code here


```

```
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
```

▶ Click here for the solution

Thank you for completing this lab!

Author

Dev Agnihotri

In [ ]: