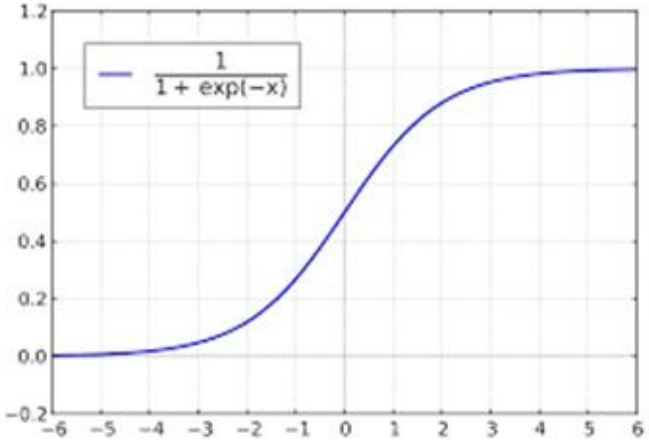
8.3 Logistic Regression

Logistic Regression

Logistic regression is kind of binary classification where response variable Y takes only two value e.g. either zero or one. You can also view it as a binary classification problem where there are only two classes to predict.

At the heart of logistic regression is the logistic or sigmoidal or logit function which is given by following equation and graph.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



If we look at the above graph we can observe that if

$$\sigma(x) \geq 0.5\, then\, we\, can\, predict\, that\, y = 1$$
$$\sigma(x) < 0.5\, then\, we\, can\, predict\, that\, y = 0$$

Here 1 and 0 are just class labels. It can be spam or non-spam or any binary value.

Hypothesis function: is given by following equation

$$h_W(x) = \sigma(W^T x) = \frac{1}{1 + e^{-W^T x}}$$

Note: compare it with the hypothesis function of linear regression.

Also, hypothesis function gives the probability of y=1 given x and W.

$$h_W(x) = P(y = 1|x; W)$$ if

$$h_W(x) \geq 0.5\, then\, we\, can\, predict\, that\, y = 1$$
$$h_W(x) < 0.5\, then\, we\, can\, predict\, that\, y = 0$$

Cost function: we cannot have same cost function as that of linear regression associated with gradient descent because sigmoidal hypothesis function makes it non-convex i.e. having many local maxima or minima. So we use following cost function to perform gradient descent to find optimal weights for logistic regression

$$Cost function$$
$$J(W) = \{-log(h_W(x))\, if\, y = 1\, and - log(1 - h_W(x))\, if\, y = 0$$
$$combining\, the\, above\, two\, we\, get\, following\, cost\, function$$
$$J(W) = -\frac{1}{m}[\sum_{i=1}^{i=m} y^i\, log(h_W(x^i)) + (1 - y^i) log(1 - h_W(x^i))]$$
$$again,\, our\, goal\, is\, to\, minimize\, J(W)$$
$$to\, find\, correct\, weights\, we\, do\, gradient\, descent$$

$$Repeat\, until\, cost\, function\, stop\, changing\, significantly$$
$$\left\{ w_j^{new} := w_j^{old} - \alpha \frac{\delta}{\delta w_j} J(W) \right\}$$

$$Solving\, above\, equation$$
$$\left\{ w_j^{new} := w_j^{old} - \alpha \sum_{i=1}^{i=m} (h_w(x^i) - y^i) x_j^i \right\}$$
$$where,$$

$$h_w(x) = \frac{1}{1 + e^{-W^T x}}$$

Problem Statement: We have been given 11 features of white wine and we have to predict if the wine is of good quality or poor quality. These features or predictor variables are as follows.

fixed acidity

volatile acidity

citric acid

residual sugar

chlorides

free sulfur dioxide

total sulfur dioxide

density

pH

). sulphates

1. alcohol

The data is taken from following link : **http://archive.ics.uci.edu/ml/datasets/Wine+Quality (http://archive.ics.uci.edu/ml/datasets/Wine+Quality)**

To solve this problem using logistic regression we are going to use scikit learn's **sklearn.linear_model.LogisticRegression** package . To use this package, we have to follow four steps

1. Import the package

```
from sklearn import linear_model
```

2. Create the model

```
logreg = linear_model.LogisticRegression(C=0.09,n_jobs=-1)
```

C = is inverse of penalty. We penalize our model for overfitting. Lower value of C means higher penalty

n_jobs = Number of CPU cores used during the cross-validation loop. If given a value of -1, all cores are used

3. Train the model on test data and label

```
logreg.fit(training, label)
```

First parameter is training data and second parameter is label associated with training data

4. Predict the label of test data by using trained model.

```
predictedLabel=logreg.predict(test)
```

In this problem we are going to use **2-Fold cross validation**. Where we will train our model on first half of the data and test it on the second half of the data. It is 2-fold validation because we have split our data

into two halves.

```
import pandas

  import os

  import numpy as np

  from sklearn import linear_model

relativePath=os.getcwd()

dataFilePath=relativePath+"/Resources/wine.csv"

  data = pandas.read_csv(dataFilePath)



  def Z_ScoreNormalization(data,targetColName): # we need to normalize the data so that each
 attribute is brought to same scale

    for i in data.columns:

        if i==targetColName: # do not modify target values

            continue

        mean=data[i].mean()

        std=data[i].std()

        data[i]=data[i].apply(lambda d : float(d-mean)/float(std)) #perform z-score normaliza
tion




  def dataFilter(data,targetColumn): # filter the data and split it into two parts test and t
rain

    columnName= [col  for col in data.columns if col not in targetColumn] # get all the predi
ctor variable names leaving target column name

    dataFrame= data[columnName]# get all the data i.e X

    labelFrame=data[[targetColumn]] # get all the target or labels i.e Y

    size=len(dataFrame)

    trainingData= dataFrame.loc[range(1,int(size/2))] # Take first half as training

    trainingLabel=labelFrame.loc[range(1,int(size/2))] #take first half as training label
```

```
    testData=dataFrame.loc[range(int(size/2),size)] # take second half as test

    testLabel=labelFrame.loc[range(int(size/2),size)]

    print trainingData.shape,trainingLabel.shape,testData.shape,testLabel.shape

    return trainingData,np.asarray(trainingLabel).flatten(),testData,np.asarray(testLabel).fl
atten() #convert all the labels into one dimensional ndarray. This is done to prevent warning
s getting generated by sklearn




  def calBaseLine(data): # calculate baseline : it is percentage of records belonging to majo
rity class

    classValues=np.unique(data) # from target values find out unique classes

    highest=0

    baseClass=""

    for label in classValues: # iterate over these classes to find number of records belongin
g to that class

        count=len(data[data==label])

        if count>highest:

            highest=count

            baseClass=label

    print  "Base Class :",baseClass

    print "base Line :",(float(highest)/len(data))*100



  def calAccuracy(testLabel,predictLable): # accuracy is percentage of correct prediction mad
e by classifier

    count=0

    for i in range(len(testLabel)):

        if testLabel[i] in predictLable[i]:

            count += 1

    print "Accuracy :", (float(count) / len(testLabel))*100
```

```
  Z_ScoreNormalization(data,"quality") # normalize the data



  training,label,test,testLabel=dataFilter(data,"quality") # filter the data to get test and
 training



  logreg = linear_model.LogisticRegression(C=0.09,n_jobs=-1) # create the model C=1.09 is inv
erse of penalization like we will see in ridge regression

# it is a penalty for overfitting the model

#n_jobs=-1 Number of CPU cores used during the cross-validation loop. If given a value of -1,
 all cores are used



  calBaseLine(testLabel)# calculate the base line



  logreg.fit(training, label) # train our model on training data

  predictedLabel=logreg.predict(test)# test our model pn test data .



  calAccuracy(testLabel,predictedLabel) #calculate accuracy
```