Error-based classification using Linear Support Vector Machine

Data Analytics

CS40003

Project 71

Sarthak Maheshwari – 17HS20034 – _____

Naitik Goel – 17HS20019 – _____

Date of Submission: 6th December 2019

Support Vector Machine

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of attributes) that distinctly classifies the data points.

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e. the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Loss Function

The loss function that helps maximize the margin is hinge loss.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

$$c(x, y, f(x)) = (1 - y * f(x))_+$$

where c is the loss function, x is the sample, y is the true label, f(x) is the predicted label.

Objective Function

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

As you can see, our objective function of the SVM consists of two terms – the first term is a regularizer, the heart of the SVM, the second term is the loss. The

regularizer balances between margin maximization and loss. We want to find the decision surface that is maximally far away from any data points.

How do we minimize our loss/optimize for our objective (i.e. learn)? - Gradient Descent

We have to derive our objective function to get the gradients. As we have two terms, we will derive them separately using the sum rule in differentiation.

$$\frac{\delta}{\delta w_k}\lambda \parallel w \parallel^2 = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k}\left(1 - y_i\langle x_i, w\rangle\right)_+ = \begin{cases} 0, & \text{if } y_i\langle x_i, w\rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

This means, if we have a misclassified sample, we update the weight vector w using the gradients of both terms, else if classified correctly, we just update w by the gradient of the regularizer.

Misclassification condition -

$$y_i\langle x_i, w\rangle < 1$$

Update rule for our weights (misclassified) -

$$w = w + \eta(y_i x_i - 2\lambda w)$$

including the learning rate η and the regularizer λ. The learning rate is the length of the steps the algorithm makes down the gradient on the error curve.

The regularizer controls the trade-off between achieving a low training error and a low testing error that is the ability to generalize your classifier to unseen data. As a regularizing parameter we choose 1/epochs, so this parameter will decrease, as the number of epochs increases.

Update rule for our weights (correctly classified)

$$w = w + \eta(-2\lambda w)$$

K-Fold Cross-Validation

Cross-validation is a resampling procedure used to evaluate models on a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

Cross-validation is primarily used to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

The general procedure is as follows:

1) Shuffle the dataset randomly.
2) Split the dataset into k groups
3) For each unique group:
    a) Take the group as a hold out or test data set
    b) Take the remaining groups as a training data set
    c) Fit a model on the training set and evaluate it on the test set
    d) Retain the evaluation score and discard the model
4) Summarize the skill of the model.

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

Note - In addition to above theory what we have done is that we have stored all the test results in a single matrix and hence obtained a confusion matrix by

comparing it to the actual observations and similarly have obtained a ROC curve for the same.

Bootstrap Method

The bootstrap method is a statistical technique for estimating quantities about a population by averaging estimates from multiple small data samples.

Importantly, samples are constructed by drawing observations from a large data sample one at a time and returning them to the data sample after they have been chosen. This allows a given observation to be included in a given small sample more than once. This approach to sampling is called sampling with replacement.

The process for building one sample can be summarized as follows:

1. Choose the size of the sample.
2. Randomly select an observation from the dataset
3. Add it to the sample
4. Add another observation by replacement until we get the required number of obs.
5. Use this as the training sample
6. Remove the observations in training sample from the whole sample giving us the test sample
7. Test your model and record results

Confusion Matrix:

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us

insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

	Class 1 Predicted	*Class 2 Predicted*
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

Classification Rate/Accuracy:

Classification Rate or Accuracy is given by the relation:

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall:

Recall can be defined as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized (small number of FN).

Recall is given by the relation:

$$\textbf{\textit{Recall}} = \frac{TP}{TP + FN}$$

Precision:

To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labeled as positive is indeed positive (small number of FP). Precision is given by the relation:

$$Precision = \frac{TP}{TP + FP}$$

F-measure:

Since we have two measures (Precision and Recall) it helps to have a measurement that represents both of them. We calculate an F-measure which uses Harmonic Mean in place of Arithmetic Mean as it punishes the extreme values more.

The F-Measure will always be nearer to the smaller value of Precision or Recall.

$$F\text{-}measure = \frac{2*Recall*Precision}{Recall + Precision}$$

ROC Curve

A useful tool when predicting the probability of a binary outcome is the Receiver Operating Characteristic curve. It is a plot of the false positive rate (x-axis) versus the true positive rate (y-axis) for a number of different candidate threshold values.

The true positive rate is calculated as the number of true positives divided by the sum of the number of true positives and the number of false negatives. It describes how good the model is at predicting the positive class when the actual outcome is positive.

$$TPR = \frac{TP}{TP + FN}$$

The false positive rate is calculated as the number of false positives divided by the sum of the number of false positives and the number of true negatives.

$$FPR = \frac{FP}{FP + TN}$$

The shape of the curve contains a lot of information, including what we might care about most for a problem, the expected false positive rate, and the false negative rate.

Code in Jupyter – (also sent as an ipynb file)

In [189]:
```
import numpy as np
import pandas as pd                 #"""importing libraries"""
import random
import matplotlib.pyplot as plt
df = pd.read_csv("BCW.csv")         #"""read data in dataframe"""
df = df.interpolate()
Y = np.array(df['class'])
Yt = np.transpose(Y)
df = df.drop(['ID'], axis=1)
dfb = np.array(df)
df = df.drop(['class'], axis=1)
X = np.array(df)                    #"""convert dataframe into matrix followed by adding bias"""
b = np.full((len(Yt),1), -1)
X = np.append(X, b, axis=1)
```

In [190]:
```
def svm_train(X, Y):                #"""Training function for SVM"""
    w = np.zeros(X[0].size)
    eta = 1
    n = 10000

    for epoch in range(1,n):
        for i, x in enumerate(X):
            if (Y[i]*np.dot(X[i],w)) < 1:
                w = w + eta * ( (Y[i]*X[i]) + (-2  *(1/epoch)* w) )
            else:
                w = w + eta * (-2  *(1/epoch)* w)

    return w

def svm_test(X,w) :                  #"""Testing function returns predicted values """
    #w = svm_train(X,Y)
    Yp = np.dot(X,w)
    return Yp;
```

In [191]:
```
def K_fold(X,Y,k):                    #"""K-fold function where k represents number of groups"""
    Ypk = []
    for i in range(1,k+1) :
        tstX, trnX = fold_i_of_X(X, i, k)
        tstY, trnY = fold_i_of_X(Y, i, k)
        w = svm_train(trnX,trnY)
        Ypi = svm_test(tstX,w)
        Ypk.extend(Ypi)               #"""return predicted values for whole dataset"""
    return Ypk;

def fold_i_of_X(X, i, k):             #"""fold function return data in i-th fold and remaning data"""
    n = len(X)
    return X[n*(i-1)//k:n*i//k], np.delete(X,slice(n*(i-1)//k,n*i//k), 0) ;
```

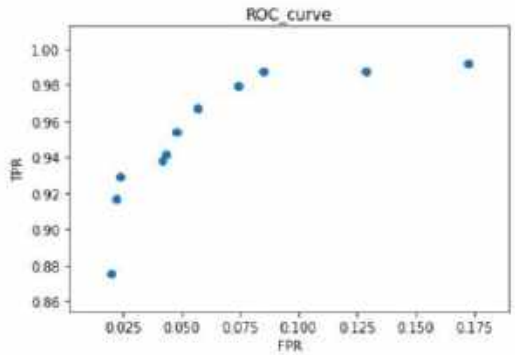
In [192]:
```
def boot_strap(df):                   #"""bootstrap function takes data with class value"""
    trnX,tstX = random_train_test_data(df)
    trnY = [row[len(X[0])-1] for row in trnX]
    tstY = [row[len(X[0])-1] for row in tstX]
    trnX = np.delete(trnX,np.s_[len(X[0])-1:] , axis=1)
    tstX = np.delete(tstX,np.s_[len(X[0])-1:] , axis=1)
    b1 = np.full((len(trnX),1), -1)
    trnX = np.append(trnX, b1, axis=1)
    b2 = np.full((len(tstX),1), -1)
    tstX = np.append(tstX, b2, axis=1)
    w = svm_train(trnX,trnY)
    Yp = svm_test(tstX,w)
    return Yp,tstY;                    #"""return predicted values and corresponding actual class value"""

def random_train_test_data(X) :       #"""It returns random data taken from original data with replacement"""
    randomRows = np.random.randint(len(X), size = len(X))
    trnx = []
    for i in randomRows:
        trnx.append(X[i,:])
    trnX  = np.array(trnx)
    trnx = np.unique(trnX, axis=0)
    tstX = np.delete(X, trnx, axis=0)
    return trnX,tstX;
```

In [193]:
```
def conf_matrx(Y,Yp,t):                #"""confusion matrix """
    TP=0
    FP=0
    TN=0
    FN=0
    for i in range(0,len(Y)):
        if((Y[i]<0)and(Yp[i]<t)):
            TN = TN+1
        elif((Y[i]>0)and(Yp[i]<t)):
            FN = FN+1
        elif((Y[i]<0)and(Yp[i]>t)):
            FP = FP+1
        elif((Y[i]>0)and(Yp[i]>t)):
            TP = TP+1
    #print("TN =",TN," FN =",FN," TP =", TP," FP =",FP)
    return TP,FP,FN,TN;

def Result(Y,Yp):                      #"""It gives different measures """
    TP,FP,FN,TN = conf_matrx(Y,Yp,0)
    PA = (TP + TN) / (TP + TN + FP + FN)     #"""predicted accuracy"""
    precision = TP / (TP + FP)
    recall = TP / (TP + FN)
    f = (2*recall*precision)/(recall+precision)          #"""f1-score"""
    #print("predictive accuracy is",PA)
    return "predictive accuracy is",PA,"f-measure is",f,"precision is",precision,"recall is",recall;
```

In [194]:
```
def fpr_tpr(Y,Yp,min_t,p,gap):                  #"""calculation of fpr and tpr for ROC curve """
    tp = []
    fp = []
    for i in range(0,p):
        TP,FP,FN,TN = conf_matrx(Y,Yp,min_t+i*gap)
        if (TN + FP)==0 or (TP + FN)== 0 :
            continue
        else :
            fpr = FP/(TN + FP)
            tpr = TP/(TP + FN)
        fp.append(fpr)
        tp.append(tpr)
    return fp,tp;

def ROC(Y,Yp,min_t,p,gap):                      #"""plotting ROC curve where p represents points on roc curve"""
    x,y = fpr_tpr(Y,Yp,min_t,p,gap)             #""" min_t represents minimum threshold and gap between points"""
    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.title('ROC_curve')
    plt.scatter(x,y)
    plt.show()
```

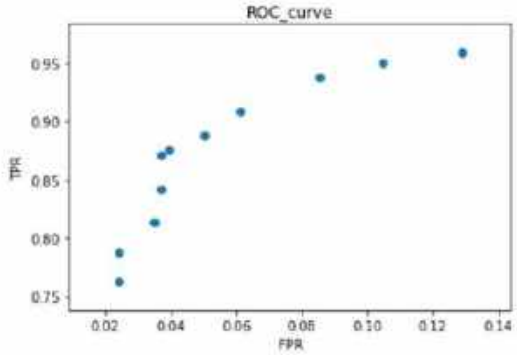
In [195]:
```
Yp = svm_test(X,svm_train(X,Y))                 #"""result of SVM model"""
print(Result(Y,Yp))
ROC(Y , Yp,-50,11,10)
```

```
('predictive accuracy is', 0.9527896995708155, 'f-measure is', 0.9330628803245437, 'precision is', 0.9126984126984127, 'recall is', 0.9543568464730291)
```



In [197]:
```
Ypk = K_fold(X,Y,5)                          #"""Result from cross validation with k-fold with 10 groups"""
print(Result(Y,Ypk))
ROC(Y,Ypk,-50,11,10)
```

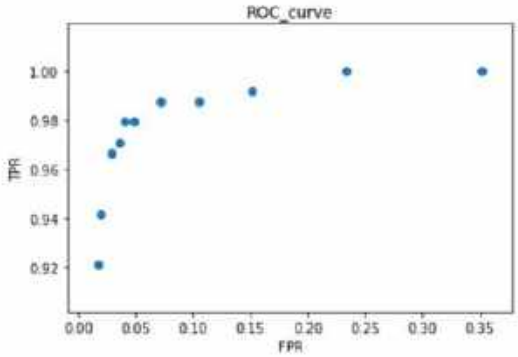
```
('predictive accuracy is', 0.9313304721030042, 'f-measure is', 0.897872340425532, 'precision is', 0.9213973799126638, 'recall i
s', 0.8755186721991701)
```



In [198]:
```
Ypb,Yb = boot_strap(dfb)                     #"""Result from cross validation with bootstrap method """
print(Result(Yb,Ypb))
ROC(Yb,Ypb,-50,11,10)
```

```
C:\Users\sarthak\Anaconda3\lib\site-packages\ipykernel_launcher.py:22: DeprecationWarning: using a non-integer array as obj in
delete will result in an error in the future
C:\Users\sarthak\Anaconda3\lib\site-packages\ipykernel_launcher.py:22: FutureWarning: in the future negative indices will not b
e ignored by `numpy.delete`.
```

```
('predictive accuracy is', 0.9608127721335269, 'f-measure is', 0.9456740442655935, 'precision is', 0.914396887159533, 'recall i
s', 0.9791666666666666)
```



In [ ]:

Results (for BCW data)

For Support Vector Machine –

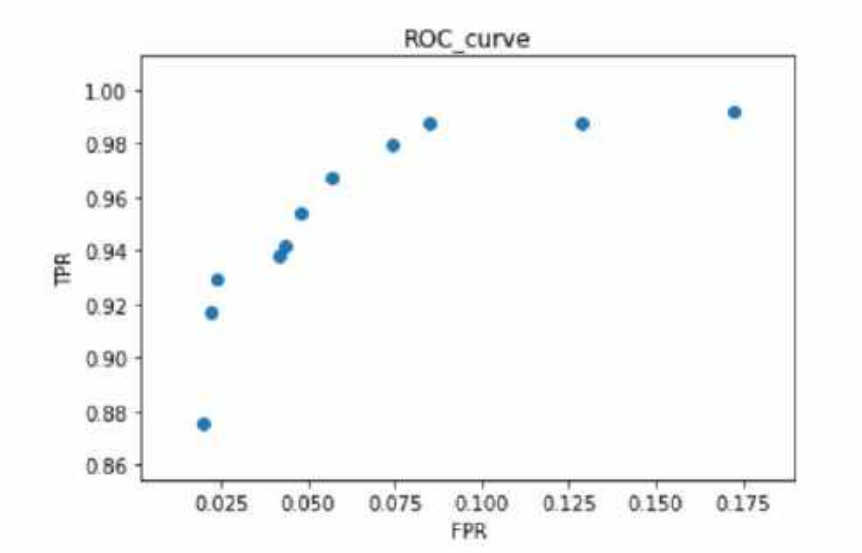
Predictive accuracy = 0.9528

F1-score = 0.9331

Precision = 0.9127

Recall = 0.9544

ROC curve –



For K-fold Cross Validation –

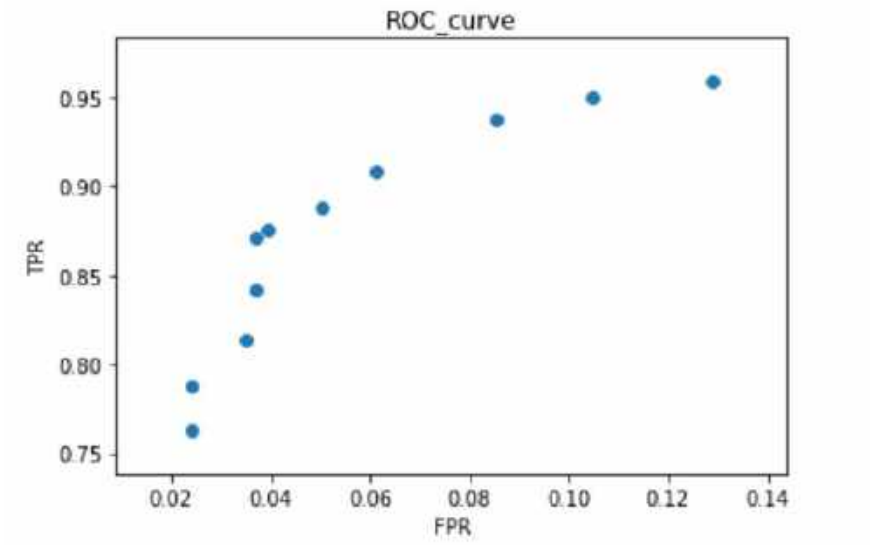
Predictive accuracy = 0.9313

F1-score = 0.8979

Precision = 0.9214

Recall = 0.8755

ROC curve –



For Bootstrap Method –

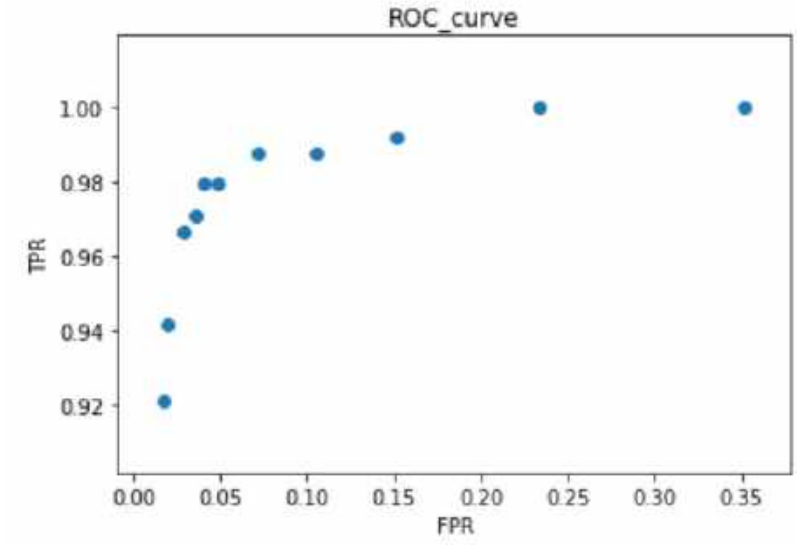
Predictive accuracy = 0.9608

F1-score = 0.9457

Precision = 0.9144

Recall = 0.9792

ROC curve –



Note: Results may change for Bootstrap method