Machine Learning : Assignment 2

Nupur Kulkarni

Contents

1 Introduction

The perceptron is the linear algorithm,works on the feed-back principle.It linearly classifies data on the basis of linear combination of the attributes.If data is not linearly separable,its hard to classify it with perceptron.The attributes are features,provided as feature vector.The perceptron and its variants are explained in the sections ahead.Learning curve present accuracy of perceptron variants the Gisette data.The section ahead provide accuracy on scaled and normalized data.

2 Part 1:Variants of the Perceptron Algorithm

The variants of the perceptron are :

- Peceptron with Bias
- Perceptron without Bias

- Pocket Algorithm with Bias
- Adatrn with Bias

In this section,we will discuss variants of the perceptron algorithm and implementation of them along with demonstration on diff rent datasets.

2.1 What is Perceptron algorithm

The perceptron algorithm is supervised binary classifier which works on the principle of the linear classification.This algorithms adds features to the machine learning.It handles large training datasets with increased speed.Moreover it helps system to learn online continuously without referring to past results.
The algorithm iterates over the training samples and classifies feature input vector in to two classes.It update weight vector w so that sample xi will be correctly classified.Whenever algorithm detects misclassified sample ,it update weight vector to correct error.The following is the basic equation for the perceptron algorithm.

$$w' = w + \eta * y_i * x_i \tag{1}$$

where y_i are the labels associated with x_i.η is the learning rate between 0 to 1.
The perceptron algorithm converge when data is linearly separable.The weight vector adjusts its position several times during training.But id data is non separable ,it will not converge.Thus, this algorithms does not guarantee to converge and hence it is implemented in limited iterations.

2.2 Perceptron algorithm without Bias

Heres the given implementation code of the perceptron without Bias.

2.2.1 Implementation of Perceptron without Bias

Python Code :

```
import numpy as np
from matplotlib import pyplot as plt

class Perceptron :

    def __init__(self, max_iterations=100, learning_rate=0.2) :

        self.max_iterations = max_iterations
        self.learning_rate = learning_rate

    def fit(self, X, y) :

        self.w = np.zeros(len(X[0]))
        converged = False
        iterations = 0
        while (not converged and iterations < self.max_iterations) :
            converged = True
            for i in range(len(X)) :
                if y[i] * self.discriminant(X[i]) <= 0 :
                    self.w = self.w + y[i] * self.learning_rate * X[i]
                    converged = False
                    plot_data(X, y, self.w)
            iterations += 1
        self.converged = converged
        if converged :
```

```
            print ('converged in %d iterations ' % iterations)

    def discriminant(self, x) :
        return np.inner(self.w, x)

    def predict(self, X) :
        """
        make predictions using a trained linear classifier

        Parameters
        ----------

        X : ndarray, shape (num_examples, n_features)
        Training data.
        """

        scores = np.inner(self.w, X)
        return np.sign(scores)

def generate_separable_data(N) :
    w = np.random.uniform(-1, 1, 2)
    print (w,w.shape)
    X = np.random.uniform(-1, 1, [N, 2])
    print (X,X.shape)
    y = np.sign(np.dot(X, w))
    return X,y,w

def plot_data(X, y, w) :
    fig = plt.figure(figsize=(5,5))
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    a = -w[0]/w[1]
    pts = np.linspace(-1,1)
    plt.plot(pts, a*pts, 'k-')
    cols = {1: 'r', -1: 'b'}
    for i in range(len(X)):
        plt.plot(X[i][0], X[i][1], cols[y[i]]+'o')
    plt.show()

if __name__=='__main__' :
    X,y,w = generate_separable_data(40)
    p = Perceptron()
    p.fit(X,y)
```

The code given has four important funtions :

- fit :
  This function trains the model based on training sample.It updates weight vector according to misclassified data sample and keep the track of iteration required to classify data correctly.
- discriminant :
  This function computes dot product of weight vector for respective data sample.
- predict:to dot product predict function predict the sign of the result.
- generate$_s eparable_d ata$:
  $This function extract data and seprate it into train and testing datasets.$

2.3 Perceptron algorithm with Bias

The bias is used to improve the result of the perceptron algorithm.It adds the margine to the linearly separable data.With bias we can shift the activation function to the left or right.Thus,it helps in fitting the prediction of the data better.In short ,in the equation of the line ,it acts as constant term.

$$y = ax + b \tag{2}$$

In the perceptron without bias decision boundary always go through origin and thus gives less accuracy.

2.3.1 Implementation of Perceptron with Bias

```
#Fit function with bias defined as b
self.b=0
count=0
while (not converged and iterations < self.max_iterations) :
    converged = True
    for i in range(len(X)) :
        if y[i] * (self.discriminant(X[i]+self.b) <= 0 :
            self.w = self.w + y[i] * self.learning_rate * X[i]+ self.b
            self.b = self.b + y[i] * self.learning_rate
            converged = False

    iterations += 1
    count+=1
```

The bias term gets updated every time the samle gets misclassified.The given above is the code snippet of the fit function which iterates over training data.When we run the model on testing data we ave to consider the bias.So i have added it to the predict function as given below.

```
def predict(self, X) :
    scores = np.inner(self.w, X)
    return np.sign(scores+self.b)
```

2.4 Pocket Algorithm with Bias

In the pocket algorithm,we try to optimize current weight vector on the basis of In sample error(Ein).Whenever X_i gets misclassified weight vectors gets updated to its best possible value.This is done only when in sample error for current weight vector in pocket is lesser than in sample error for weight vector.

2.4.1 Implementation of Pocket Algorithm with Bias

This algorithm takes one more iteration that regular perceptron to selected optimised vector to keep in pocket.The following function calculates In sample error:

```
#In sample Error
def errorIn(self,X,y):
   eIn=0
   misCount=0
   N=len(X)
   for i in range(N):
       if(self.predict(X[i])!=y[i]):
           misCount+=1
   eIn=misCount/N
```

```
        return eIn
```

Calculating initial in sample error with errorIn function and defined list for saving optimized weights.

```
        Ewp=self.errorIn(X,y)
        w_pocket= list()
```

The following code snippet Updates and saves best weight vector based on in sample error:

```
        while (not converged and iterations < self.max_iterations) :
            converged = True
            for i in range(len(X)) :
                if y[i] * (self.discriminant(X[i])+self.b) <= 0 :
                    self.w = self.w + y[i] * self.learning_rate * X[i]+ self.b
                    self.b = self.b + y[i] * self.learning_rate
                    converged = False
                    #Current error
                    Ein_current=self.errorIn(X,y)

                    if(Ein_current < Ewp ):
                        w_pocket[:] = self.w
                        Ewp=Ein_current

            count +=1
            iterations += 1
        #finally wpocket to w to use for prediction on test data
        self.w=w_pocket
```

2.5 Adatron Algorithm with Bias

In adatron we calculate how much value each training vector contributes to the weight vector.This is calculated by given formula.

$$w = \sum_{i \in N} \alpha_i * X_i * y_i \tag{3}$$

$$Where$$

α is the positive number which describe ow much contributionX_i makes to w.
The adatron takes large amount of time to execute as its not depend on converges but on maximum iteration.It minimizes the misclassification error on large extent.In our case,no dataset converges on the implementation of adatron.

2.5.1 Implementation of Adatron Algorithm with Bias

The given below is the python code for adatron.

```
while (not converged and iterations < self.max_iterations) :
            converged = True
            for i in range(len(X)):
                gma=y[i] * (self.discriminant(X[i]+self.b))
                self.b=self.b+y[i]*self.learning_rate
                da = self.learning_rate * (1 - gma)

                if self.a[i]+da < 0:
                    self.a[i]=0
                else:
                    self.a[i]=self.a[i]+da
```

```
            if self.a[i]>2:
                self.a[i]=2

            wnew=np.zeros(len(X[0]))
            Ein=0
            for j in range(len(X)):
                wnew+=y[j]*self.a[j]*X[j]+self.b
                self.b=self.b+y[i]*self.learning_rate

            self.w=wnew


            for k in range(len(X)):
                if y[k] * (self.discriminant(X[k]+self.b)) <= 0 :
                    count=count+1

            converged = False


            Ein=count/len(X)

            if Ein == 0:
                 converged=True
                 #plot_data(X, y, self.w)
        iterations += 1

    self.converged = converged

    if converged:
        print ('converged in %d iterations ' % iterations)
    else:
        print("Data set is linearly inseparable")

    print("In sample Error for Pocket (Ein) :", float(count)/len(X))
```

2.6 Discussion about Datasets

I used three data sets here to implement all variants of perceptron.

- Heart Dataset :
  I used processed version that is provided. It has total 270 examples each with 13 features.It has two classes with labels as $+1$ and -1 for "presence of heart disease and "absence of heart disease" respectively.
- Gisette Data Set:
  It is handwritten digit reorganization data set with 600 samples and 5000 features for each sample.
- QSAR dataset :
  The QSAR is bio-degradation dataset having in total 1050 samples with 41 features.two classes have defined as discriminate ready and not ready biodegradable molecules.

2.7 Test Scenario

```
def test(self,X,y):
    count=0.0
    eOut=0.0
    #print("test weight vector",self.w)
```

```
for j in range(len(X)):
    if self.predict(X[j])!= y[j]:
        count=count+1
eOut=count/len(X)
print("Misclassified sample :",count)
print("length of sample :",len(X))
print("Eout :",eOut)
return eOut
```

Here X and y are testing feature vectors and labels associated.The function iterates over the testing sample.With the use of predict function it predicts the class of sample according to train model provided. Here, i have calculated out of sample error.

$$Eout = \frac{Misclassifiedsample}{Totaltestingsamples}$$

2.8 Results and Discussion

Here's the result of implementation of variants of perceptron algorithms.I have considered 2 datasets : QSAR and Gisette.Both data set are separated in training and testing examples in the ratio of 60:40 respectively.Implementation of "perceptron with Bias" ,"Pocket" and "Adatron" in the random splits of 10 gives the following results.

2.8.1 Comparison between Pocket Algorith and perceptron with Bias on Gisette

		Pocket On Gisette	Perceptron on Gisette
	Random Splits	EOUT	EOUT
	1	0.035	0.035
	2	0.033	0.035
	3	0.036	0.033
	4	0.035	0.036
	5	0.033	0.044
	6	0.0362	0.0301
	7	0.030	0.0320
	8	0.032	0.04
	9	0.0375	0.0275
	10	0.0375	0.0375
Average Error		0.0345	0.0621
Standard Deviation		0.0024	0.0084

Table 1: Comparision

observations From above table and Graph below:

- Out of sample errors n the above table,we could easily say that Pocket algorithm with bias performs better than Perceptron with bias on Gisette data. It gives lesser out of sample error and more accuracy.
- From standard deviation we could say that in case of pocket algorithm it gives normal distribution of data which is much better that distribution we get in case of perceptron.

The below graph show the comparison between performance of pocket and perceptron algorithm in term of "Average Out of sample error " and "Standard Deviation".

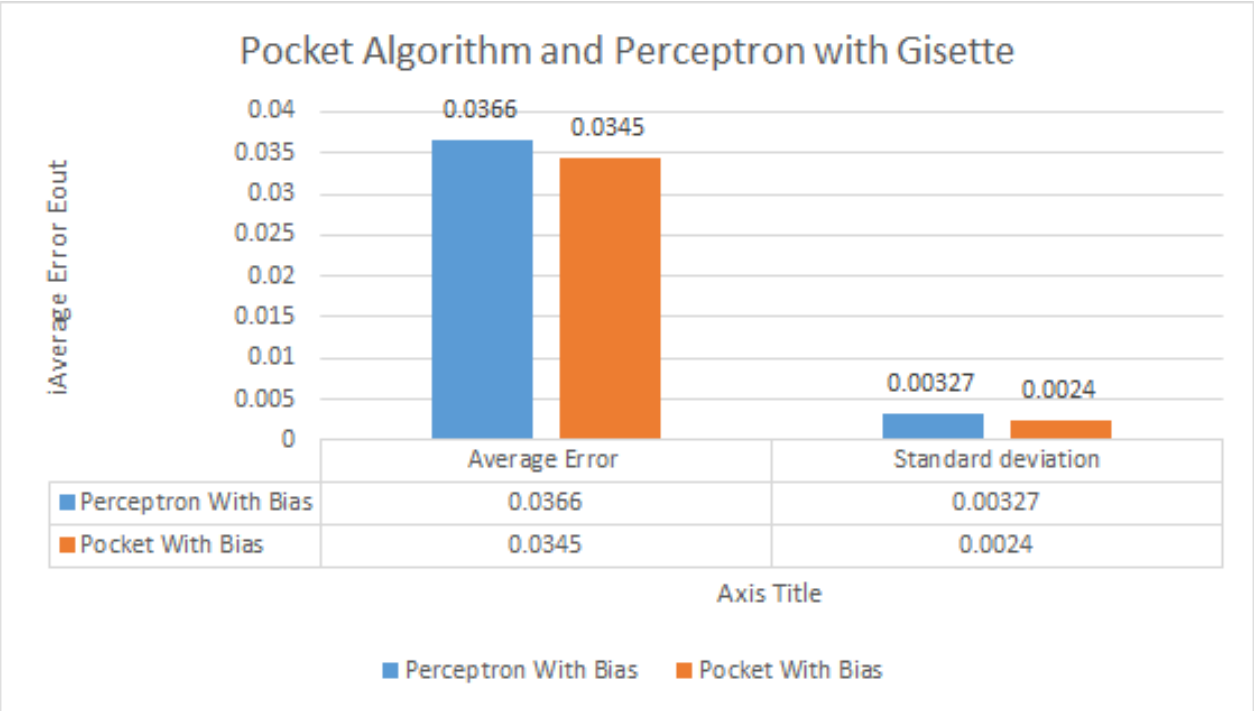

Figure 1: Comparison between Pocket and perceptron on Gisette

2.8.2 Comparison between Pocket Algorith and perceptron with Bias on QSAR

		Pocket on QSAR	Perceptron on QSAR
	Random Splits	EOUT	EOUT
	1	0.177	0.187
	2	0.135	0.3
	3	0.154	0.137
	4	0.170	0.277
	5	0.154	0.154
	6	0.175	0.239
	7	0.132	0.279
	8	0.175	0.215
	9	0.168	0.170
	10	0.170	0.199
Average Error		0.161	0.2157
Standard Deviation		0.01572	0.0563

Table 2: Comparision

observations From above table and Graph below:

- Out of sample errors n the above table,we could easily say that Pocket algorithm with bias performs better than Perceptron with bias on QSAR data. It gives lesser out of sample error and more accuracy.
- From standard deviation we could say that in case of pocket algorithm it gives normal distribution of data which is much better that distribution we get in case of perceptron.

The below graph show the comparison between performance of pocket and perceptron algorithm in term of "Average Out of sample error " and "Standard Deviation".

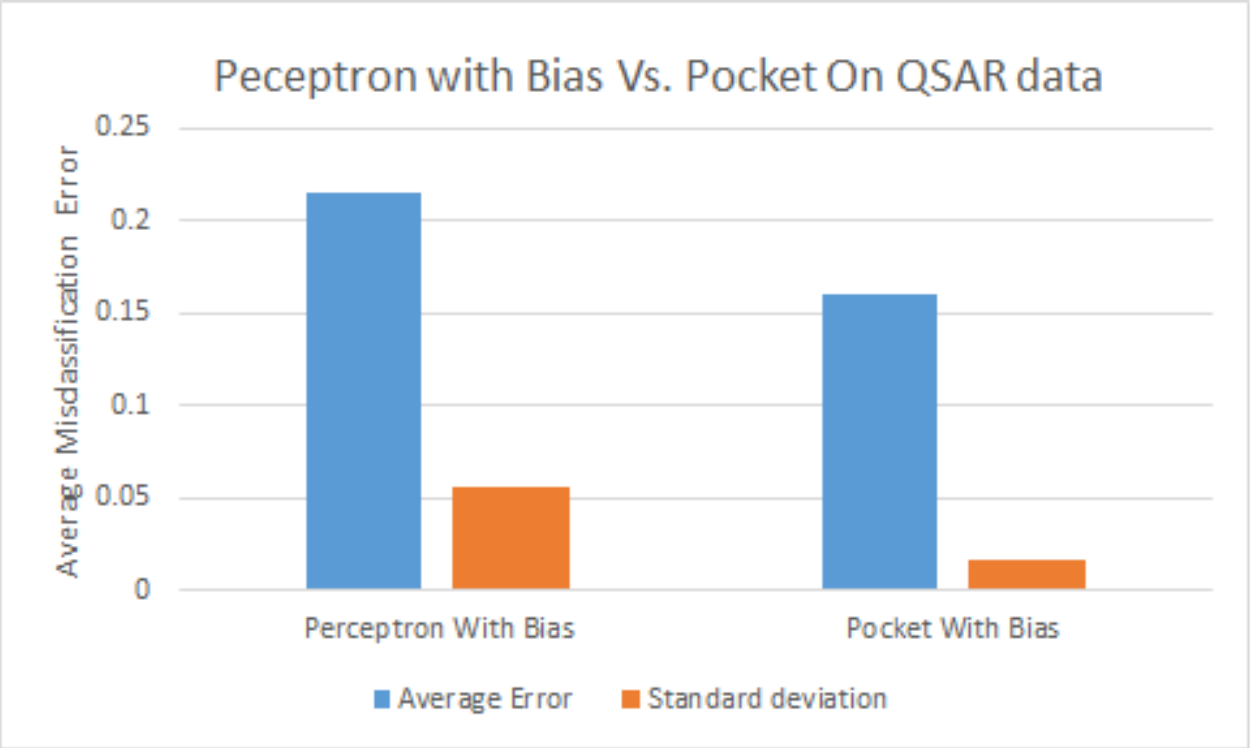

Figure 2: Comparison between Pocket and perceptron on QSAR

2.8.3 Conclusion

If we compare Perceptron with bias and pocket with bias on both datasets ,we can interpret that pocket algorithm performs way better than perceptron.Pocket just take more time to run but gives better accuracy on classification.

3 Part 2:Learning Curve

The learning curve presents the performance of the perceptron with bias algorithm on the Gisette data.It actually gives a point where machine started to learn.Considering the graph, when curve show the constant behaviour with respect to one of the axis,that poin indicates that model is learning from the data. The data set i used is segregated in 60:40 proportion i.e 60 percent of total data is used for training model while 40 percent is used for testing.As said i am reserving 40 percent of the data for testing. Keeping test data fixed ,i have taken varying amount of testing samples.X-axis represents the number of training examples while y-axis show the performance in term of "Out of sample error".
The following table shows the no of training samples and Out of sample error respectively for implementation of perceptron with bias algorithm.Total example in the test data is 2399.

Form the above table we could say that as we increase number of training examples, Eout (Out of Sample error) decreases and we get more accurate classification.The reason is when we train the model on the less number of samples,it does not learn much.Thus it is unable to make correct prediction of the large number of test data.The predict function of the perceptron is based on the dot product of the number of training

Number of training samples	Misclassified Examples	Eout
4	119	0.466
16	526	0.219
32	436	0.181
64	428	0.178
256	222	0.092
1024	116	0.0483
2042	99	0.0412
3691	82	0.0341

Table 3: Performance of Pecptron with Bias on Gisette

samples and respective weight vector. Hence for less no of samples,weight vector is not optimized and results in much higher Eout.

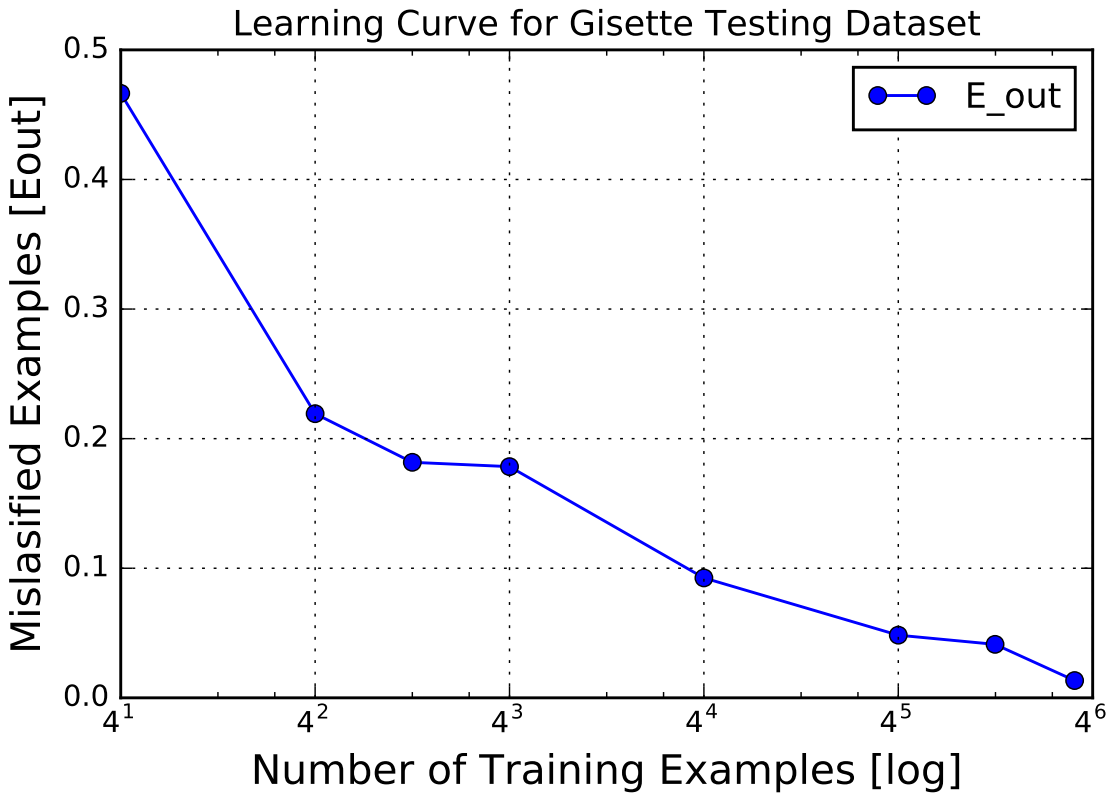

Figure 3: Learning Curve

The learning curve shown above represent the same logic.I have plotted curve by taking x-axis scale as log-base 4. he graph is almost linearly decreasing.However, we could observe that how better the machine become at predicting correct classes we increase the number of training samples.

4 Part 3:Data Normalization

The data sets are generally arbitrarily distributes and random in nature. Features mostly have different scales and units. Thus it is very difficult for any algorithm to classify them fairly. There is the possibility that result may be skewed towards some feature which we may not want to consider. Normalization, scaling and standardization are efficient way to confine data in to desire range. This data pre-processing techniques weighs all features equally in their representation. Hence, small valued features could equally contribute to the optimization as large valued features.

4.1 Scaling in the range(-1,+1)

"Normalization rescales the values into a defined range"[2].The following formula is for normalization.

$$X' = \frac{(X - min(X))}{max(X) - min(X)} \tag{4}$$

To rescale the data in the range of $[-1, +1]$,i have defined the range as $[a, b] = [-1, +1]$.The revised formula is gives as:

$$X' = (b - a)\frac{(X - min(X))}{max(X) - min(X)} + a \tag{5}$$

Where a is current minimum and b is current maximum.
In scaling we find maximum and minimum value of each feature and apply the above formula to fit the feature in the specified range.Implementation is given as follow.

```
def norm(X):
    X=np.array(X)
    norm_array=np.zeros(X.shape)

    #range [-1,+1]
    a=-1
    b=1

    for i in range(0,len(X[0])):
        #find minimum and maximum of each feature
        for j in range(len(X)):
            minimum=np.amin(X[i])
            maximum=np.amax(X[i])

        for j in range(len(X)):
            norm_array[j][i]=(((b-a)*(X[j][i]-minimum))/(maximum-minimum))+a

    min2=np.amin(norm_array)
    max2=np.amax(norm_array)

    print("Scaled heart data :",norm_array)
    print("min 2 :",min2)
    print("max2 :",max2)
    return norm_array
```

The loop in the above code iterates over the all values of each feature.It finds minimum and maximum value of the each feature and scale all the values to fit in the range of $[-1, +1]$.

Here's the result after scaling on heart data set.

```
original data :[[ 70. 1.  4. ...,  2.  3.  3.]
 [ 67.  0.  3. ...,  2.  0.  7.]
 [ 57.  1.  2. ...,  1.  0.  7.]
 ...,
 [ 56.  0.  2. ...,  2.  0.  3.]
 [ 57.  1.  4. ...,  2.  0.  6.]
 [ 67.  1.  4. ...,  2.  3.  3.]]

Scaled heart data : [[-0.56521739 -0.9964539 -0.96934866 ..., -0.98290598 -0.97345133
  -0.97446809]
 [-0.58385093 -1.       -0.97701149 ..., -0.98290598 -1.     -0.94042553]
 [-0.64596273 -0.9964539 -0.98467433 ..., -0.99145299 -1.    -0.94042553]
 ...,
 [-0.65217391 -1.       -0.98467433 ..., -0.98290598 -1.     -0.97446809]
 [-0.64596273 -0.9964539 -0.96934866 ..., -0.98290598 -1.    -0.94893617]
 [-0.58385093 -0.9964539 -0.96934866 ..., -0.98290598 -0.97345133
  -0.97446809]]

min 2 : -1.00469851214
max2 : 3.19330855019
Data set is linearly inseperable
In sample Error for Pocket (Ein) : 0.6172839506172839
Misclassified sample : 46.0
length of sample : 108
Eout : 0.42592592592592593
```

4.2 The Perceptron performance on normalized vs. non-normalized data

The table below shows the Eout after implementing perceptron with bias algorithm on normalized heart data and non normalized heart data.

	Normalized Data	Non-normalized Data
Eout	0.425	0.574
Accuracy	0.575	0.426

Table 4: Normalized Vs.Non normalized data

From the above table and graphs shown below we could interpret that the perceptron with bias performs better on scaled data than non-scaled data.If we consider Accuracy of both,its easy to say that performance in term of accuracy increased by 34%.

4.2.1 Reason behind increase in the performance

I have considered processed version of heart data set which has 270 samples.The features of the heart data has varied values. All the values are defined on different scale and differs in the overall range.Thus,it's become difficult to classify the data with minimum error.
The scaling fits the data in the specific range,so the every feature in the data is in defined range.Thus,chances of misclassifications data get reduced than original data which has wide range of values.
Hence,scaling reduces the chances of the features which are in the greater range of values dominating the features having smaller range of values but greater importance towards classification.Also it reduces the calculations required in the inner products of the feature vectors.

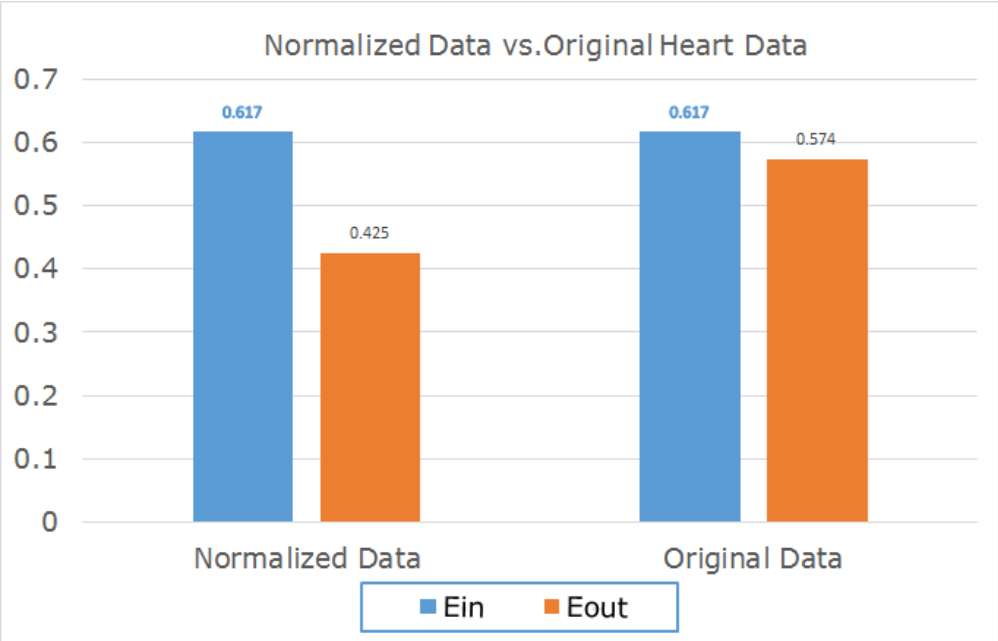

Figure 4: Learning Curve

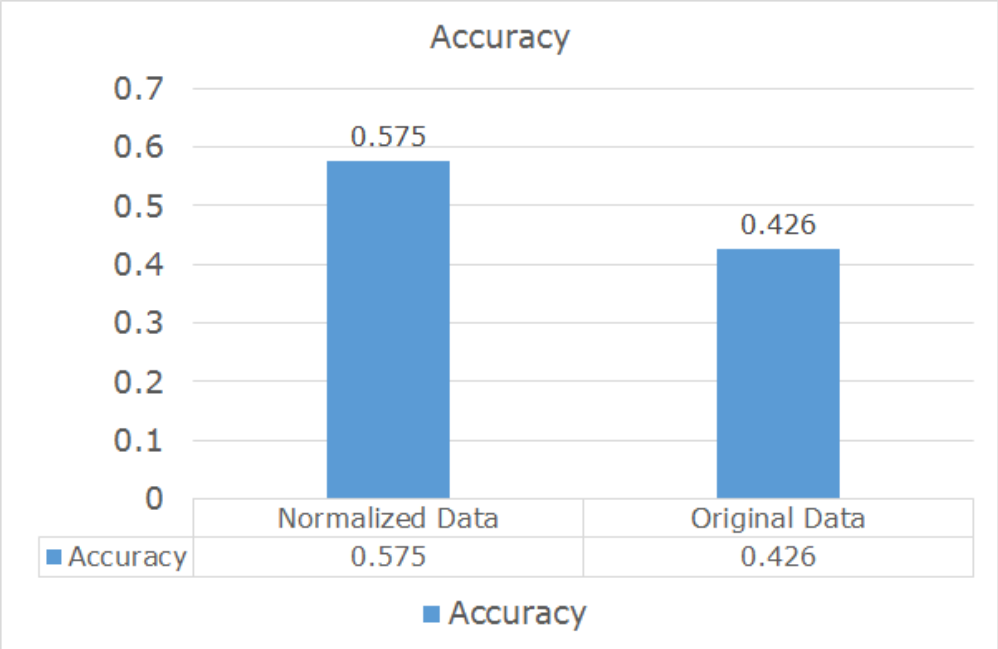

Figure 5: Learning Curve

4.3 Standardization

In normalization we rescale the features to fit in the desired range.While in standardization,we transform the values such that they have mean 0 and standard deviation 1.Thus in standardize data,we could figure out features which are most important based on the certain distance measure.Secondly,The weight also plays im-

portant role in determining the importance of the feature.If we reduce the range effect on weight vector,then it will give the better judgement of the feature.Moreover,in normalization their are chances of loosing outlier data which is reduced when data is standardized.
To achieve standardization we calculate the mean and standard deviation for each feature.Using the following formula we could calculate standardization for the training set.

$$X' = \frac{(X - \mu)}{\sigma} \tag{6}$$

where μ is mean and σ is the standard deviation for each feature.
To get correct result,we can save standard deviation and mean on training data and use them while standardizing test data.This will reduce the out of sample error.

Implementation of Standardization.

```
def standardize(X):
   X=np.array(X)


   for i in range(0,len(X[0])):
          mean=np.mean(X[:,i])
          stdev=np.std(X[:,i])
          X[:,i]=(X[:,i]-mean)/float(stdev)



   min2=np.amin(X)
   max2=np.amax(X)

   print("Scaled heart data :",X)
   print("min 2 :",min2)
   print("max2 :",max2)
   return X
```

I have tried this function on heart data set.I got the range of -34026 to 6.09300.

```
{original data : }[[ 70. 1.  4. ...,  2.  3.  3.]
 [ 67.  0.  3. ...,  2.  0.  7.]
 [ 57.  1.  2. ...,  1.  0.  7.]
 ...,
 [ 56.  0.  2. ...,  2.  0.  3.]
 [ 57.  1.  4. ...,  2.  0.  6.]
 [ 67.  1.  4. ...,  2.  3.  3.]]


{Scaled heart data :} [[ 1.71209356 0.6894997 0.87092765 ..., 0.67641928 2.47268219
  -0.87570581]
 [ 1.38213977 -1.45032695 -0.18355874 ..., 0.67641928 -0.71153494
   1.18927733]
 [ 0.2822938  0.6894997 -1.23804513 ..., -0.95423434 -0.71153494
   1.18927733]
 ...,
 [ 0.1723092 -1.45032695 -1.23804513 ..., 0.67641928 -0.71153494
  -0.87570581]
 [ 0.2822938  0.6894997  0.87092765 ..., 0.67641928 -0.71153494
   0.67303154]
 [ 1.38213977 0.6894997  0.87092765 ..., 0.67641928 2.47268219
```

```
  -0.87570581]]

min 2 : -3.40260930001
max2 : 6.09300449751

Data set is linearly inseperable
```

References

[1] How normalization works, http://stats.stackexchange.com/questions/41704/how-and-why-do-normalization-and-feature-scaling-work.

[2] Standardization vs. Normalization,http://www.dataminingblog.com/ standardization-vs-normalization/.

[3] Perceptron-based learning algorithms - IEEE Xplore, S Gallant, 1990, cited 439.