**Mallepalli Rakesh Reddy**

**Assignment 2**

|  |
| --- |
|  |

**Data sets:** L ranges from 122.46 to 145.46 while P ranges from 76.24 to 191.73 and they do not have same units. So I used the z-score of the L and P parameters instead of their original values to normalize the effect of parameters on classification.

Then I generated 9 sets of training and testing data by randomly selecting the values from the given data.

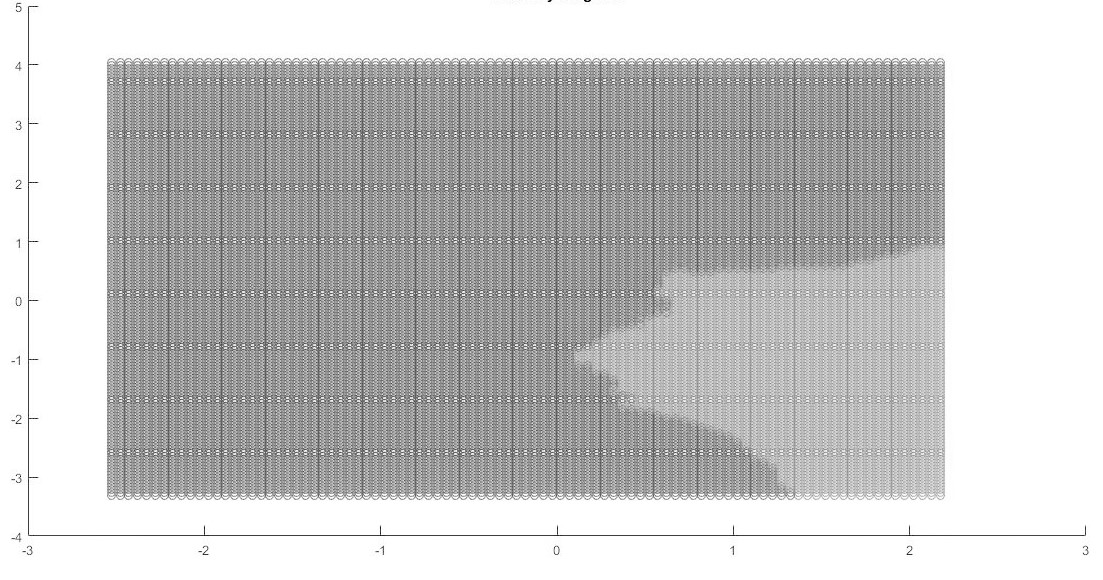
**KNN: Approach:** I used “at least 60% of neighbors in the winning class” augmentation for KNN. I choose k=17 ~ sqrt(n) where n=number of elements in training set as initial K value and if the winning class doesn’t have 60% by in first 17 nearest neighbors I incremented K till the 60% majority is reached or K reaches 31. The range of is selected to be 17 to 31 as smaller number of k increases the influence of noise and higher value of K is computationally expensive.

The performance graphs for KNN are shown in plots KNN:trail1 to KNN:trail9. The average performance for training and testing data is as shown below:

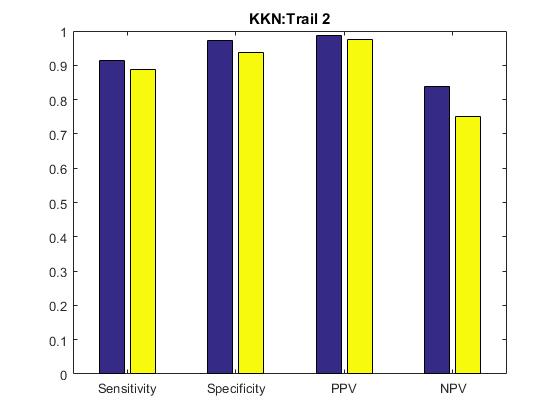
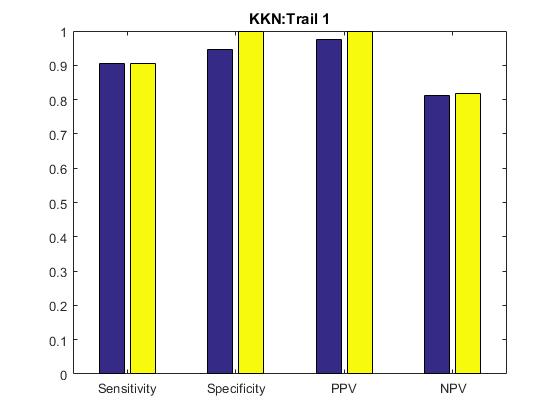
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Sensitivity** | | **Specificity** | | **PPV** | | **NPV** | |
|  | Training | Testing | Training | Testing | Training | Testing | Training | Testing |
| **Mean** | 0.9105 | 0.8916 | 0.9577 | 0.9502 | 0.9809 | 0.9732 | 0.8217 | 0.7925 |
| **STD** | 0.0107 | 0.0479 | 0.0196 | 0.0462 | 0.0082 | 0.0271 | 0.0184 | 0.1071 |

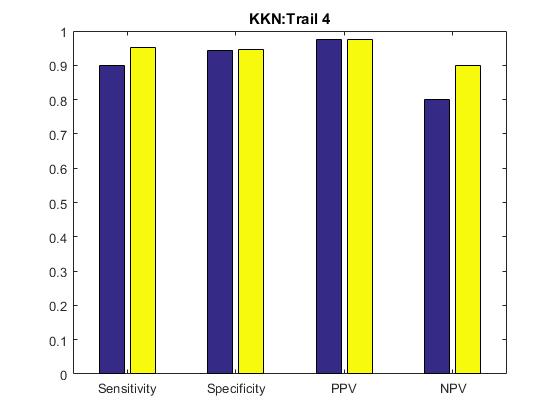
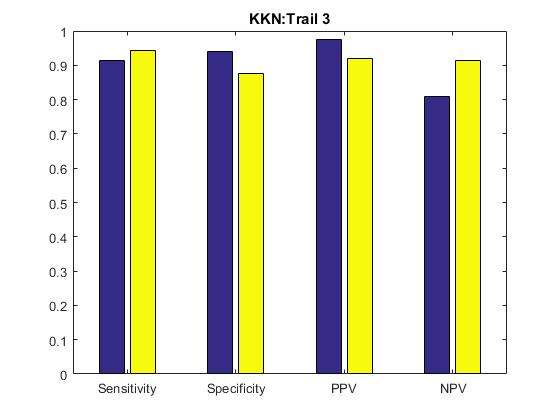
To draw the boundary between the two classes I took the training set of maximum performing trail (trail 4 in this case) and found the KNN for all the points in the feature space.

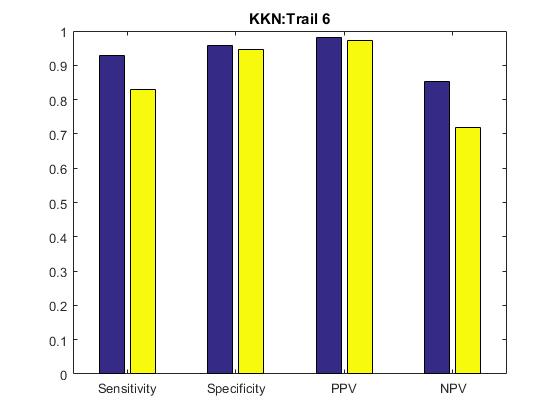
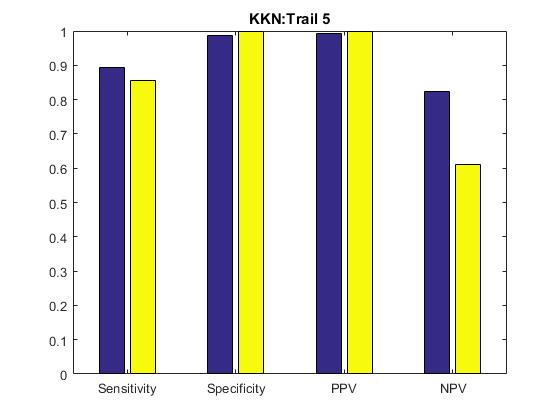
The boundary diagram is shown below. The points in darker shade represent class 1 and other points represent class 0.

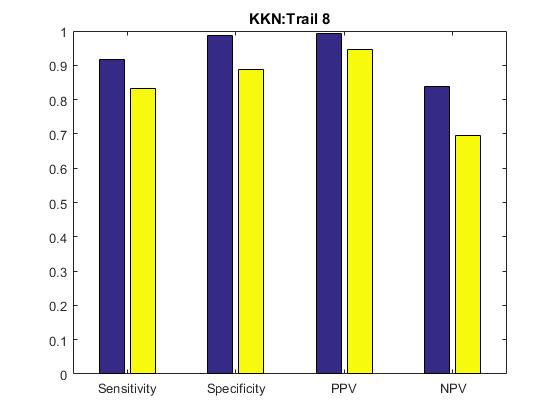
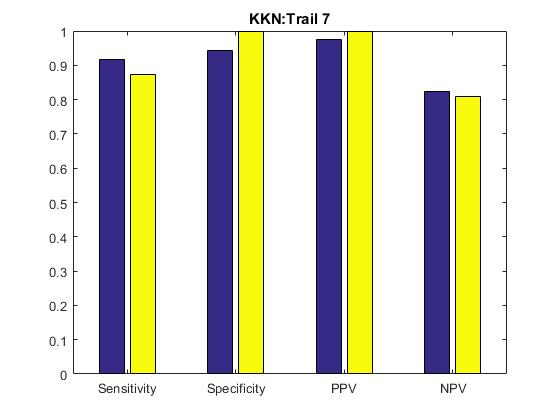


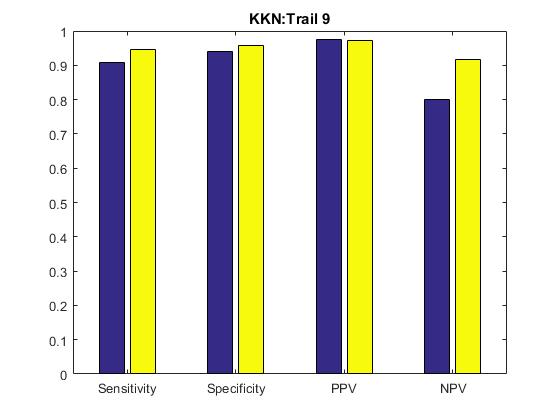
The performance metric bar charts are as shown below:









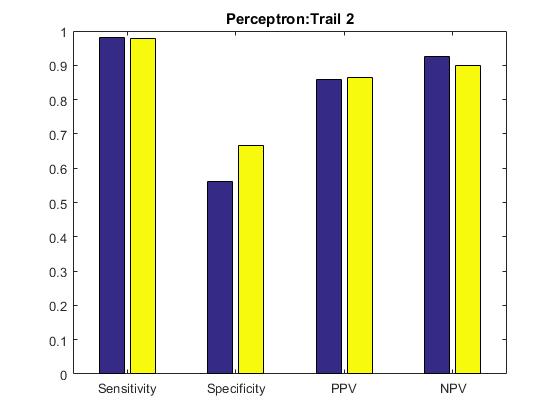
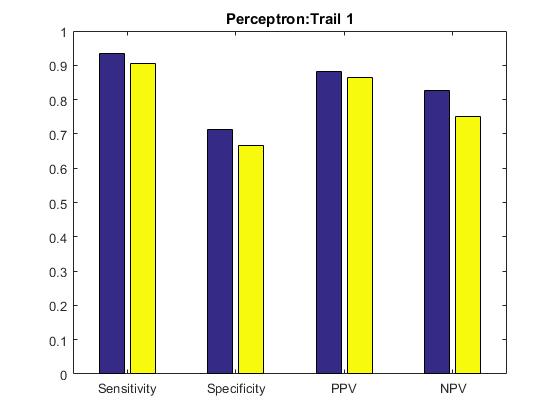


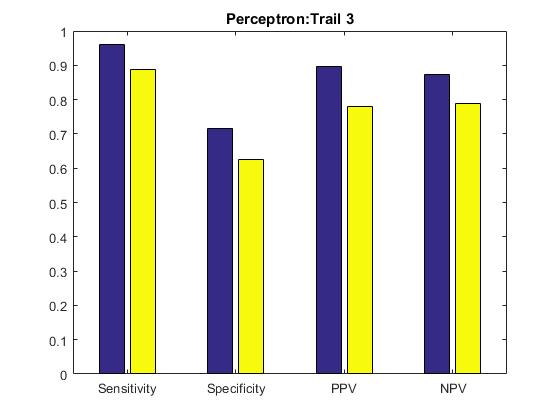
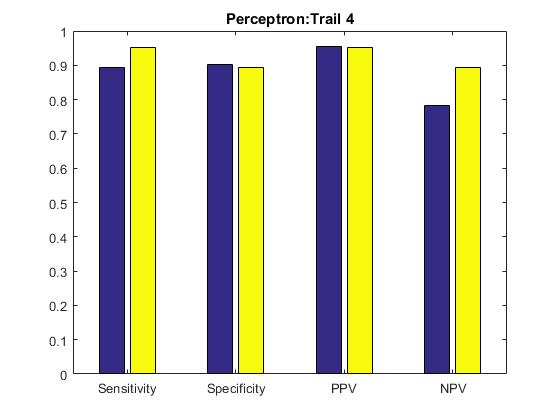
**Perceptron: Approach:**  I randomly choose the initial weights in range (0,1), learning rate as 0.1 and number of epochs as 200. I found that increasing the value of epochs above 200 doesn’t really have any significant influence on output.

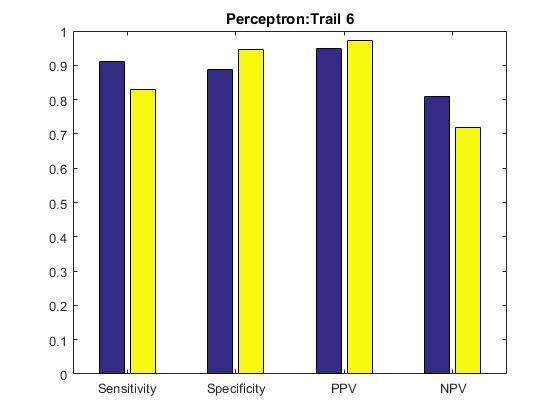
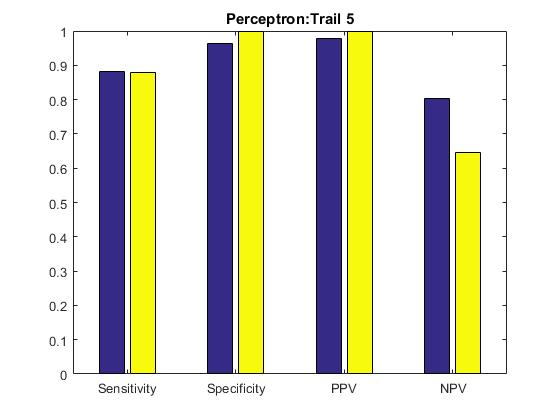
The mean and standard deviation of performance metrics are as below:

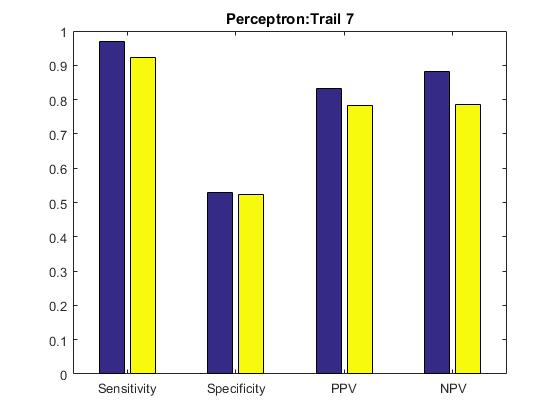
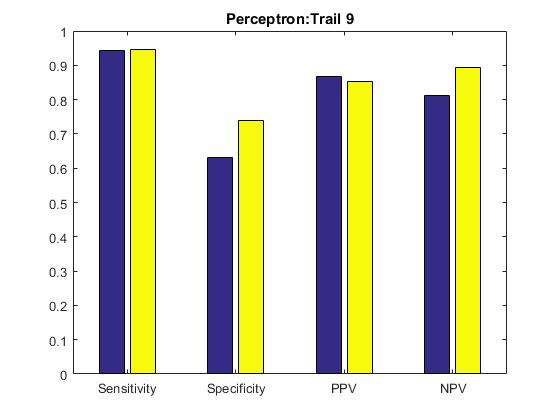
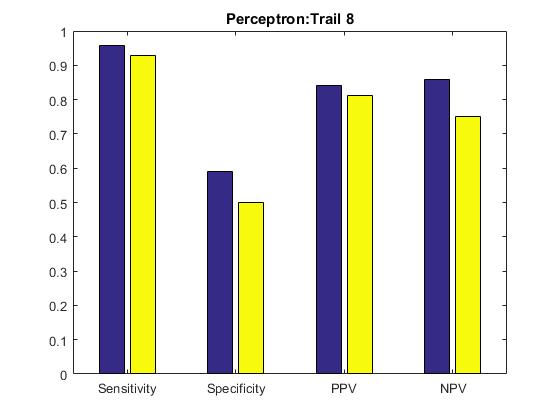
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Sensitivity** | | **Specificity** | | **PPV** | | **NPV** | |
|  | Training | Testing | Training | Testing | Training | Testing | Training | Testing |
| **mean** | 0.9368 | 0.9141 | 0.714 | 0.7177 | 0.8906 | 0.8751 | 0.8412 | 0.7924 |
| **STD** | 0.0351 | 0.0446 | 0.1708 | 0.1886 | 0.0597 | 0.0813 | 0.0461 | 0.0884 |

The performance metric bar charts are as shown below:

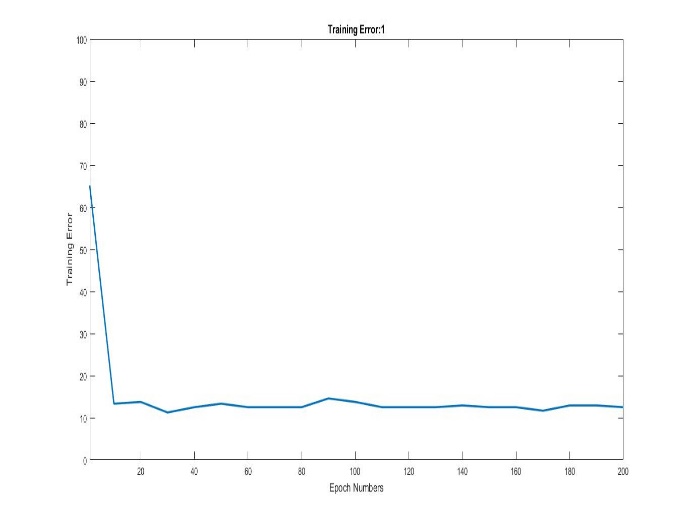
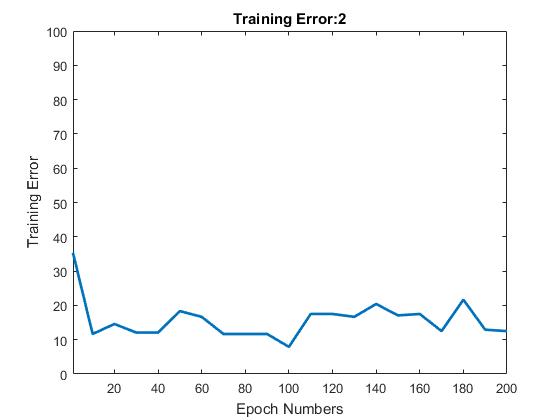


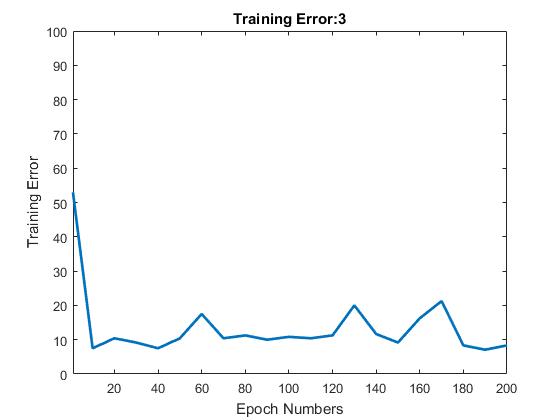
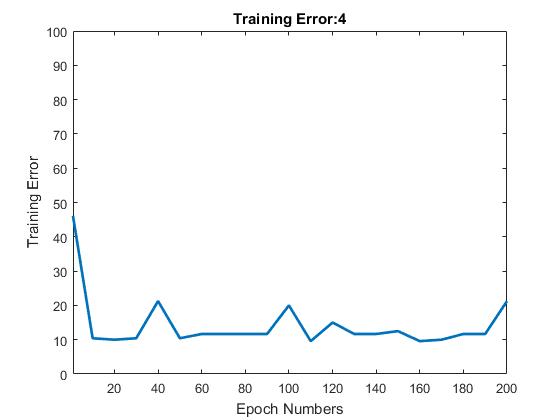
 

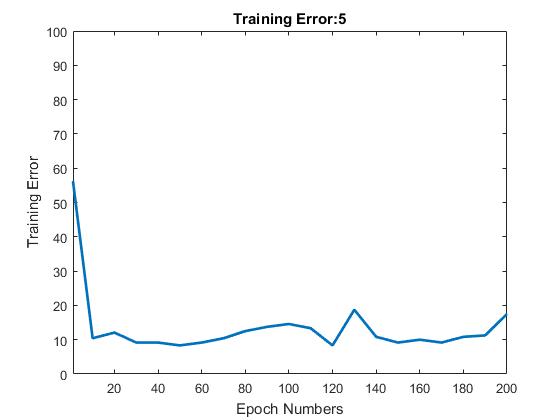
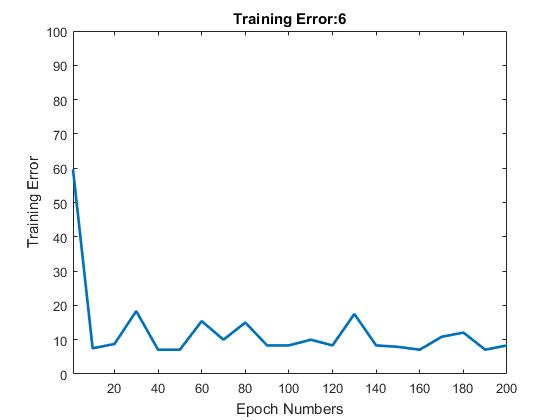


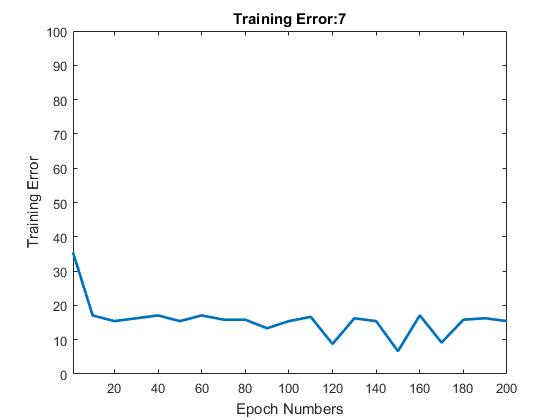
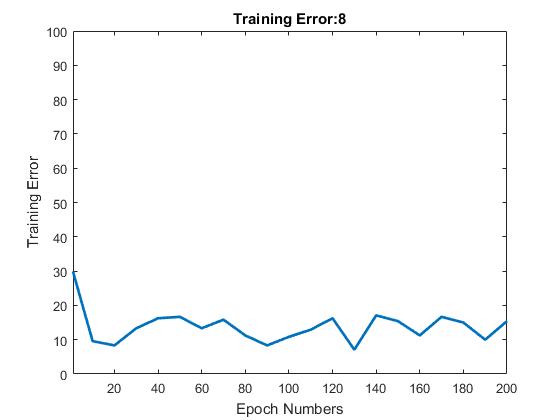
 

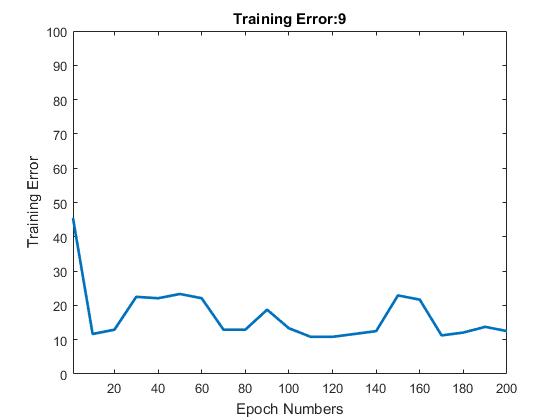
Trial-Wise Training Error graphs for the Perceptron are as shown below:

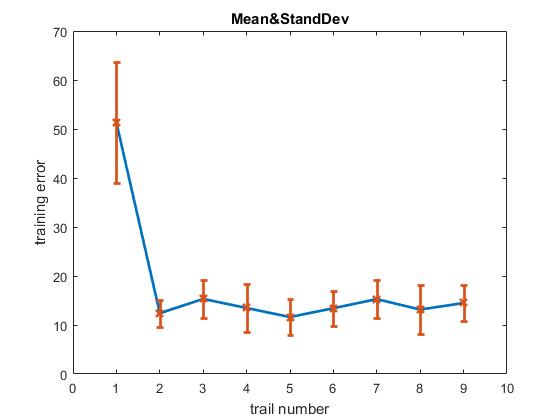
 



Mean Training Error for the Perceptron is as below:



**Analysis:**

The performance metrics of two algorithms are very similar. However the metrics of KNN deviated very less from the mean, therefore I think KNN is a better algorithm of these two for this data. (For this type of implementation.)

Both the algorithms seems equally resource intense. The KNN helps to classify where the data is not linearly separable and looking at the performance metrics I think that the two classes are not completely linearly separable.

**Code to generate data sets:**

****

**Code for KNN:**

****

****

**Perceptron Code:**

****

****