CSE 5523

HW2

Name: Xuecheng Liu

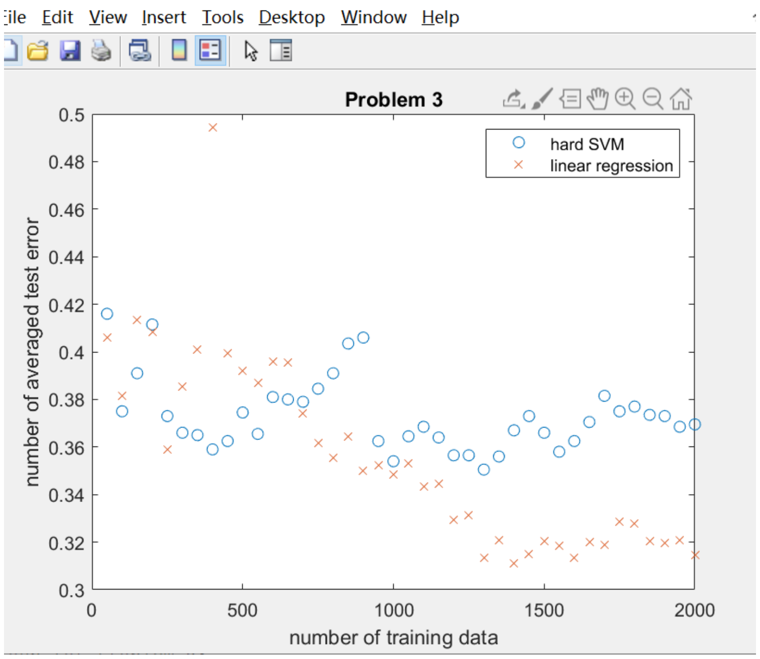
Problem 1

The accuracy with standard SVM over the test set is 92.800000%. For the least square linear classifier, the weight is calculated directly with normal equation and the accuracy is 92.850000%. Two methods roughly have the same performance.

Problem 2

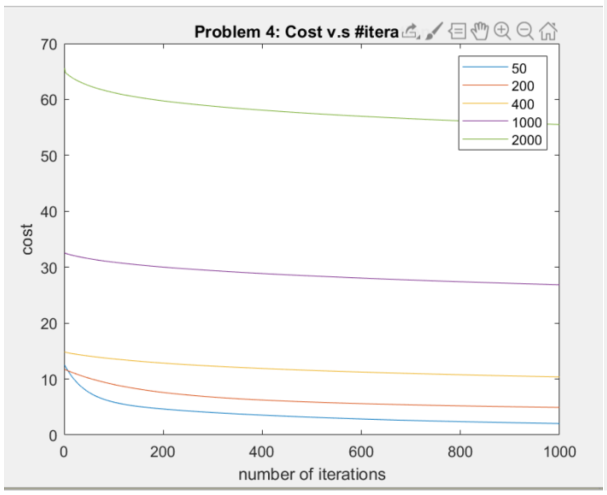
I trained the classifier with gradient for 1000 iterations and result in 94.000000% accuracy over the test set. It has a better performance compared with training the classifier with normal equations.

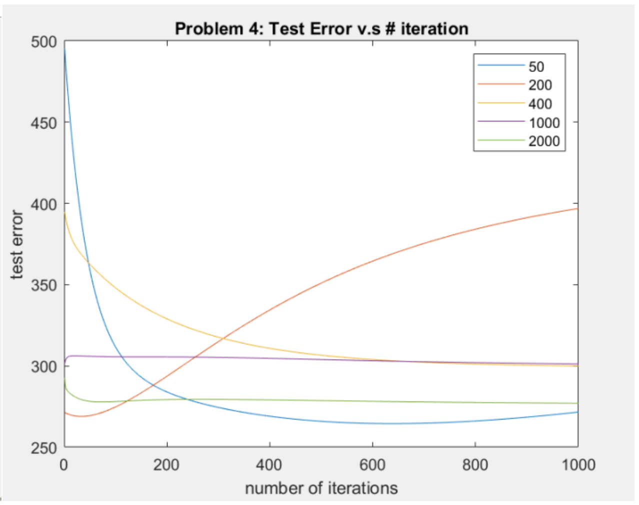
Problem 3



The graph above shows the number of averaged test errors versus the number of training data used to train each model. The result shows reducing the dimension of the training data lowers the accuracy of predictions which kind of make sense since we are using less features during the training. The result also shows hard SVM has a higher testing error. The reason for this due to the overfitting of training data when we pick C to be large, this causes the margin to be small and thus some data close to the boundary are miss classified. The result also shows the test error decreases in general as the model is trained with more data.

Problem 4

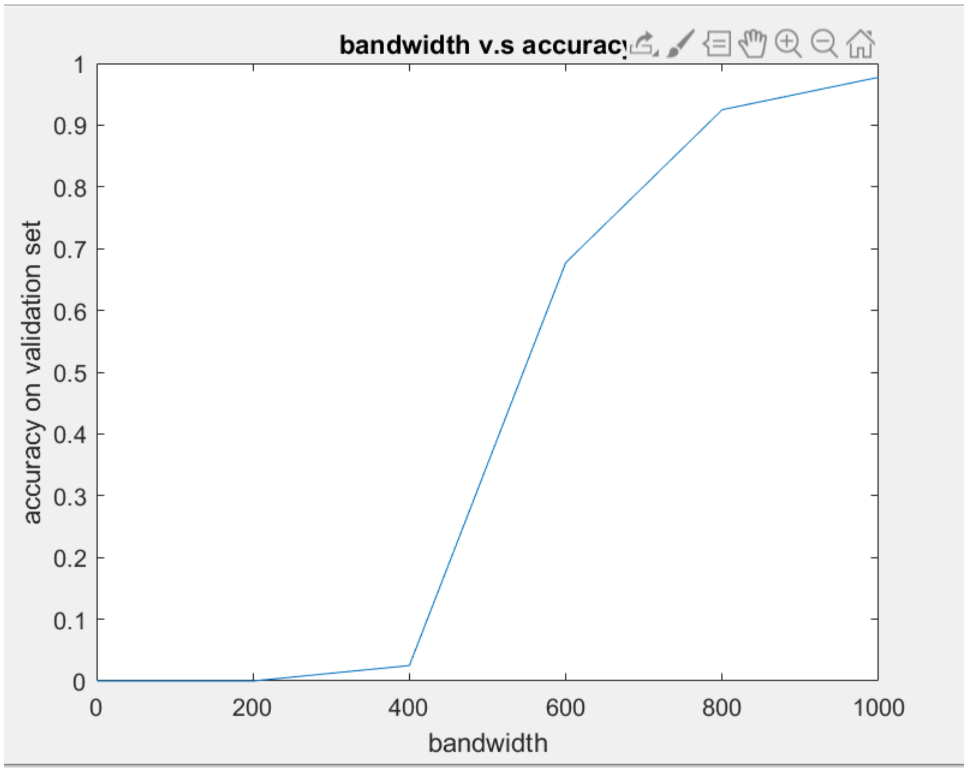




Observations for Problem 4

I plotted two figures, the first one is for Test Error v.s # iterations and the second one is loss v.s # iterations. As the second figure shows, the loss is decreasing after each iteration which means we are in the right track to train our model so that we can see how our model generalize to the testing data. From the first figure, there are several observations. First of all, the test error does not keep decreasing as the training goes on. Sometimes we have already reached the minimum loss with only several hundreds of iterations and the later training just overfitting the training data (cost for 400 training data shows this phenomenon). The general trend for other number of training examples seems okay since the cost generally decreasing as more training goes. However, the loss may blow up after more and more trainings and we can see this trend from the loss with 50 training examples. This may imply there is a point where we have the best generalized weights for test data and after that point we are just overfitting the training data.

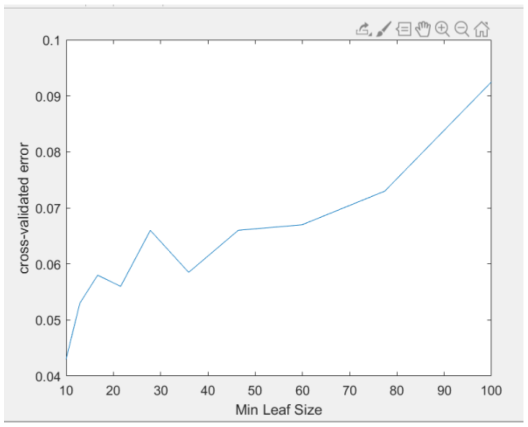
Problem 5



In this problem, I use the first 1600 training data to train the model and later 400 training data as the validation set. Also, the bandwidth I chose is 1, 200,400,600,800. As you can see from the figure above, the performance over the validation set increases as the increase of the bandwidth and reached almost 98% accuracy with 1000 as the bandwidth.

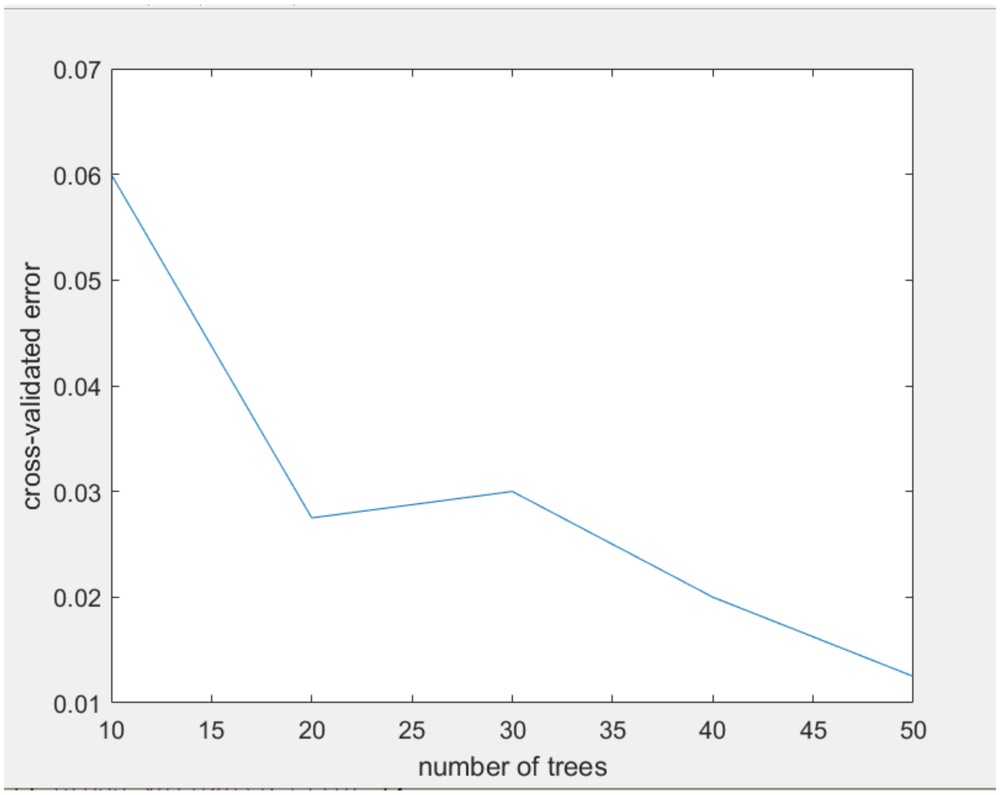
Problem 6

For decision trees, I will treat min leaf size as the hyperparameter and choose the best one over the cross validation.



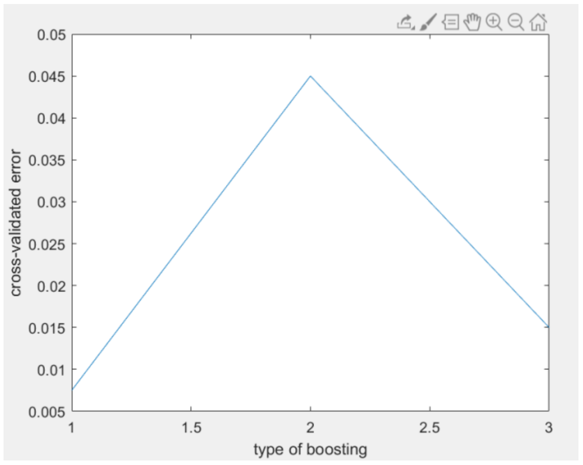
As you can see from the figure above, as the number of leaf size increases, the error over the validation set blows up. And min leaf size = 10 has the lowest error. Then I will train the tree with min leaf size = 10 and the accuracy is 93.75% over the test set.

For the bagged decision tree, I used number of trees as the hyperparameter during the cross validation.



The figure above shows that as the number of trees increase, the cross-validation error decreases in general. Then I picked 50 as the value for the hyperparameter and the performance over the test set is 98.25%.

For the boosted decision tree, the hyperparameter I tuned in the cross validation is type of boosting, where I chose from Adaboost, RUSboost and LogitBoost. The error over the validation set is shown below, where 1 represents Adaboost, 2 represents RUSboost and 3 represent LogitBoost.



The figure above shows Adaboost performs the best. And the performance of adaboosted decision tree over the test set is 98.45%.

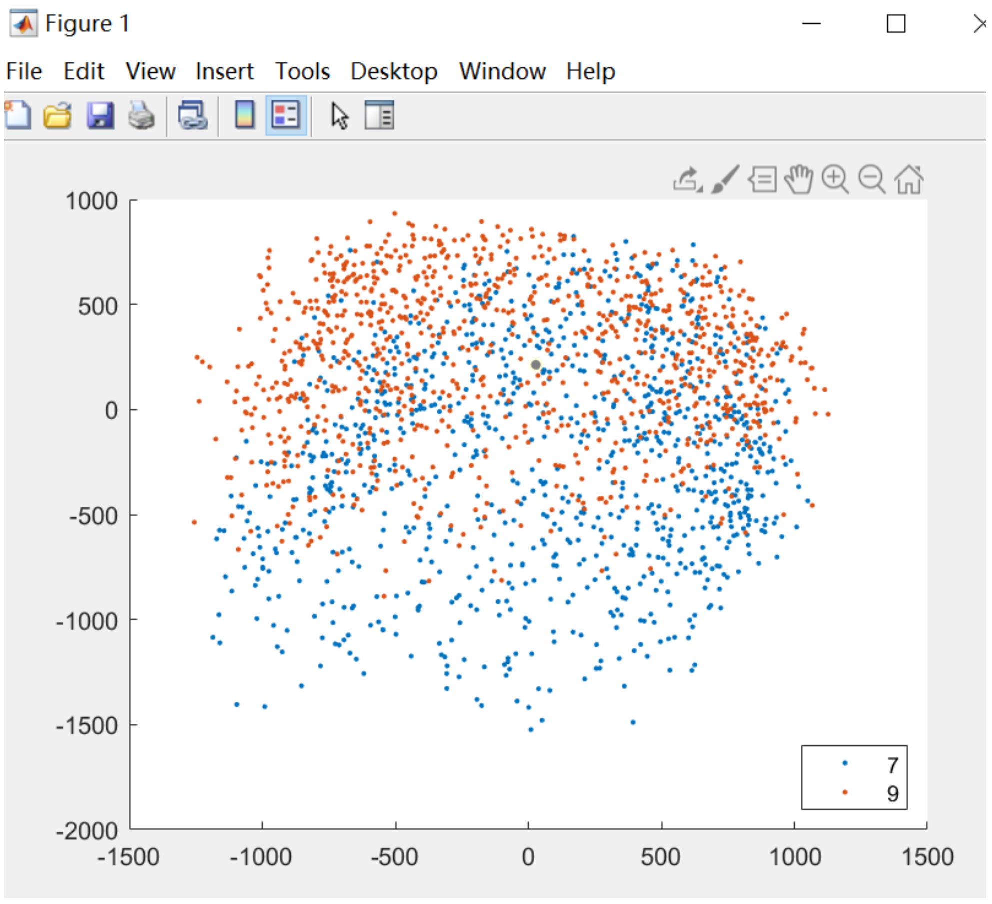
In summary, the result above shows both bagging and boosting improves the performance of the decision tree and convert a week learner into a strong learner.

Problem 7

Part (1)

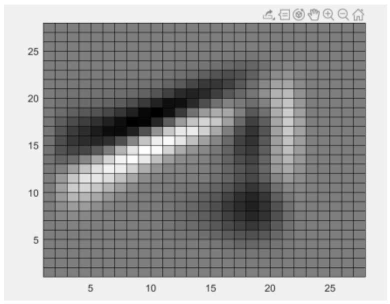
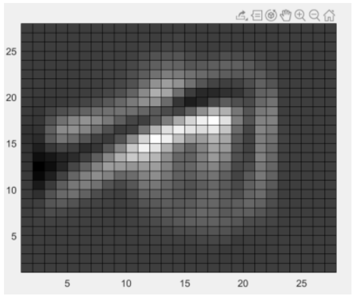
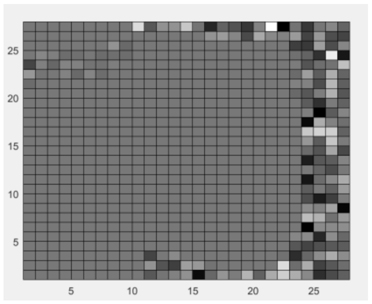
Data visualization with 2 dimensions. Implementation is in hw2_P7.m

After reducing the feature dimensions, 7 and 9 are generally lie in different parts of the space where 7 mainly groups in the bottom part while 9 mainly groups in the top part.



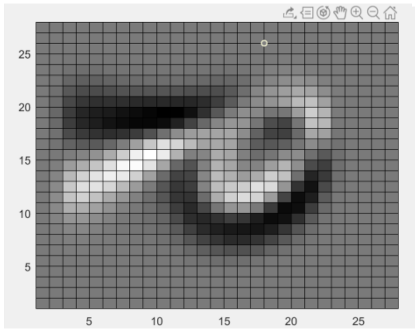
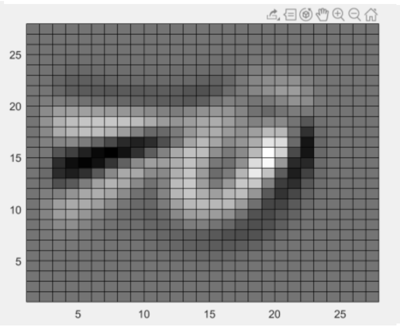
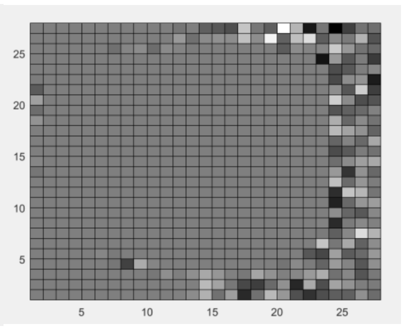
Part (2)

"Eigendigits" for '7' class

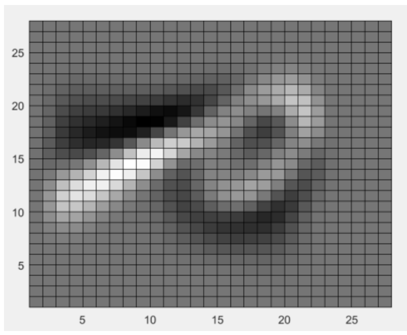
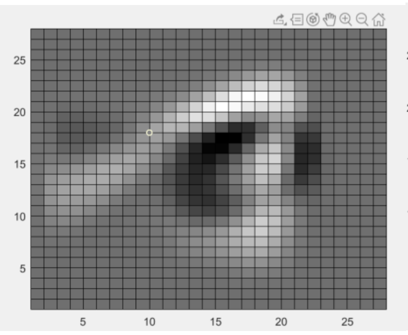
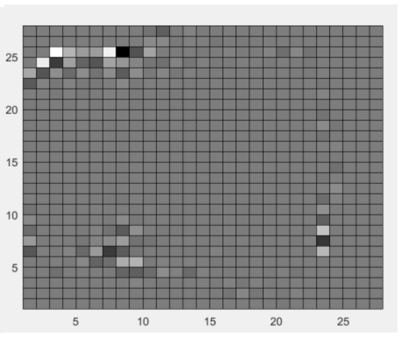
1st principal component 5th principal component 600th principal component

“Eigendigits” for ‘7’ class

1st principal component 5th principal component 784th principal component

“Eigtndigits” for both class

1st principal component 2nd principal component 500th principal component

Observations: As you can see from the above pictures, the first several eigenvectors generally capture the feature of both the 7’s and 9’s classes as we can recognize the digits easily in the first two pictures for each class. The eigendigit for both classes looks funny as it has the characteristic from both classes and we can still recognize them. However, as the loadings get less important, the images start to get vague and looks more like random noise for each of the classes.

Problem 8

Outline to use K-mean as a classification method: Firstly, I will use all the training data and apply K-mean to them. Let we first pick K = 2. The MATLAB function will return the cluster for each class and the center for each cluster. Then I will go through each cluster and see which class takes majority portion of that cluster and assign a label to that cluster based on the majority. During the test phase, for each testing data point, I will calculate the distance from that data point to the center of each of each cluster. Then I will pick the cluster which has the shortest distance to that point and assign the label of that cluster to that point. Finally, I will go through every single point in the test set and compare the predicted label with the actual label. Then I will repeat this process for K = 5, 10, 50. The result is showing below.

For K = 2, the accuracy over the test set is 59.10%,

For K = 5, the accuracy over the test set is 72.60%,

For K = 10, the accuracy over the test set is 86.65%,

For K = 50, the accuracy over the test set is 93.70%.

Thus, as the number of clusters increases, K means did a pretty good job in classification with the method described above. I think the above method is reasonable to be used to compare the results between classification and clustering, although we are kind of cheating in the process to labelling the clusters. This method will not work if we cannot visualize the high dimensional training data and assign a label to each cluster. Also, one of the most significant benefit with using K mean is training takes relatively less time than other classification algorithms.