Project 1: Supervised Learning

Tyler Bobik
Georgia Institute of Technology

Description of Classification Problem 1

Data	Attributes	Instances	Classes
diabetes	8	768	2

Description of Classification Problem 2

Data	Attributes	Instances	Classes
Spam	57	4601	2

1.0 Description of Classification Problem #1 (Diabetes)

I found this data set on the UCI machine learning database labeled "The Pima Indians Diabetes Data Set" it is multivariate and it 8 contains continuous type attributes, 1 discrete class and 768 instances. The purpose of the data set is to predict if someone will get diabetes or not. This data set was interesting to me because it has a pretty low number of attributes and a low number of instances. Also, there seems to be noise in this data set and I would like to see how the different learning algorithms are affected by this I was curious to see which of the different classification algorithms are affected by this. preform on this data given those facts. Additionally, this kind of problem is interesting to me personally because I am in the Intro to Health Informatics class and we are developing a web app to help diagnose diabetes and thought I could gain information and maybe even apply a machine learning technique I learn here to it.

2.0 Description of Classification Problem #1 (Spambase)

I found this data set on the UCI machine learning database labeled "Spambase Data Set" it is multivariate and it contains 57 continuous type attributes, 1 discrete class (spam or not) and 4601 instances. The purpose of the data set is to predict if an email is spam or not. Most of the attributes indicate whether a particular word or character was frequently occurring in the email. This data set was interesting to me because it has a pretty high number of attributes and a high number of instances. I was curious to see how different classification algorithms preform on this data given those facts.

Being new to machine learning, these two data sets were interesting me and drew my attention because, the diabetes data set had a low number of attributes and a low number of instances when compared to the spam dataset, where it has a high number of attributes and a high number of instances. With these two data sets together I was able discover many other interesting points when comparing the different classification algorithms which I will point out throughout this paper. On the left I will display the results of different classifiers for the diabetes data set and on the right I will display the results of the spambase data set.

1.1 Finding Decision Tree Parameters for Pruning (Parameter Optimization)

My first step in building my decision trees was to enable a type of "pruning" using RandomizedSearchCV in sklearn because currently pruning is not enabled in sklearn. I used RandomizedSearchCV in order to find the best parameters to use with my decision tree classifier. RandomizedSearchCV looked through random numbers from 1-20 for the following parameters with entropy always enabled; min_samples_split, max_depth, min_samples_leaf and max_leaf_nodes. The best parameter was chosen by the highest mean 10 fold cross validation score. I also decided to use entropy for the information gain to measure the quality of the split. The result for the best parameters was:

min_samples_split	11
max_depth	19
min_samples_leaf	14
max_leaf_nodes	18

Mean Validation Score	Test Run Time (Seconds)
0.760 (std: 0.042)	4.81

2.1 Finding Decision Tree Parameters for Pruning (Parameter Optimization)

The best score I found when pruning my decision tree for the Spam dataset was with GridSearchCv in sklearn. The grid search for each parameter was: "min_samples_split": [2, 10, 20],
"max_depth": [None, 2, 5, 10],
"min_samples_leaf": [1, 5, 10],
"max_leaf_nodes": [None, 5, 10, 20].
The best parameter was chosen by the highest mean 10 fold cross validation score. I also decided to use entropy for the information gain to measure the quality of the split. The result for the best parameters was:

min_samples_split	2
max_depth	10
min_samples_leaf	1
max_leaf_nodes	None

Mean Validation Score	Test Run Time (Seconds)
0.910 (std: 0.036)	53.49

More tree depth makes it possible for the model to extract more complicated behaviors from the data at the cost of additional complexity. That depth makes the best tradeoff between reproducing the underlying relationships and over fitting the problem. I thought it was interesting that the pruning of the spambase specifically, max_depth dataset stopped at 10 while the max_depth of the diabetes dataset stopped at 19. This was interesting to me because the spambase dataset has

many more features then the diabetes dataset and I figured this would create a bigger tree. When I saw that spambase only had a max_depth of 10 I thought that this might be under fitting the data even though we know that keeping a smaller tree is better I thought 10 was still too small. In the next section we will see if the decision tree for spambase is under fitting the data by missing relevant relations between the features and target outputs.

1.2 Splitting Data into Test/Training and Running Decision Tree with Pruning

After testing different testing and training sizes I discovered that a 30% test / 70% training split worked the best as it reduced bias and overfitting for this dataset. I used sklearn's cross validation split which splits the arrays into random train and test subsets. Running the Decision Tree Classifier with Entropy and the "pruning" parameters mentioned above I got the following results.

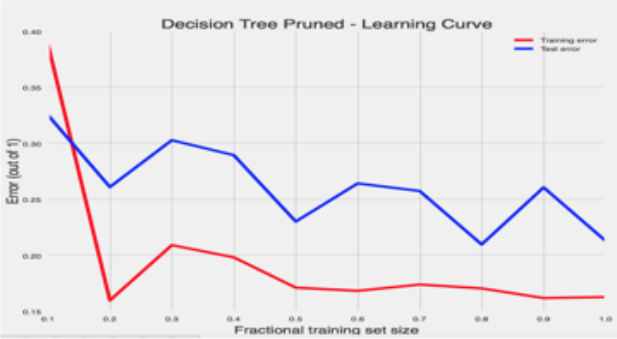
Train Accuracy	Test Accuracy	Test Run Time (Seconds)
0.8279	0.7027	0.0019

Test

	Precision	Recall	F1-score	Support
No Diabetes	0.82	0.82	0.82	197
Diabetes	0.63	0.63	0.63	95
Avg/Total	**0.76**	**0.76**	**0.76**	**292**

Train

	Precision	Recall	F1-score	Support
No Diabetes	0.84	0.90	0.87	281
Diabetes	0.79	0.70	0.74	155
Avg/Total	**0.83**	**0.83**	**0.83**	**436**



The average 10-fold cross validated test and training error plotted as fractional training set size. After pruning the percent of correct classifications using the trained model is 70%. You can see here as in the beginning there .1 - .2 of the fractional test size there is underfitting happening. This is because at that training set size the training set is not expressive enough to account for the data provided. As the training size increases you can see the error decrease for both the training error with the train-test gap not very large you can see that the model is not overfitting or underfitting but there it is some fluctuation in the test error curve, currently I do not know why this is occurring, I would be interested in doing more research this at a later time. The decision tree performs well on this data with pruning because this data is noisy and we learned that ID3 is good at handling noisy data when we modify the termination criteria, which I did in the step above.

2.2 Splitting Data into Test/Training and Running Decision Tree with Pruning

After testing different testing and training sizes I discovered that a 40% test / 60% training split worked the best as it reduced bias and overfitting for this dataset. I used sklearn's cross validation split which splits the arrays into random train and test subsets. Running the Decision Tree Classifier with Entropy and the "pruning" parameters mentioned above I got the following results.

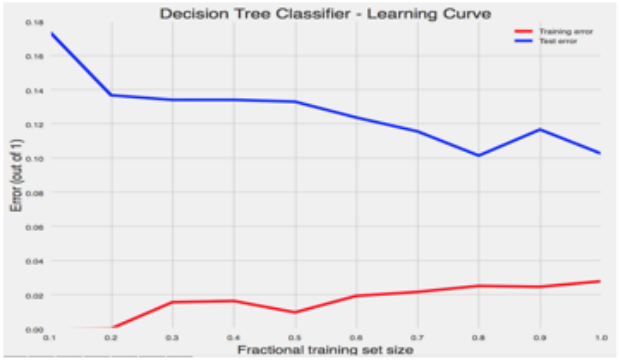
Train Accuracy	Test Accuracy	Test Run Time (Seconds)
0.9605	0.8908	2.0225

Test

	Precision	Recall	F1-score	Support
No Diabetes	0.91	0.91	0.91	1098
Diabetes	0.86	0.87	0.87	743
Avg/Total	**0.89**	**0.89**	**0.89**	**1841**

Train

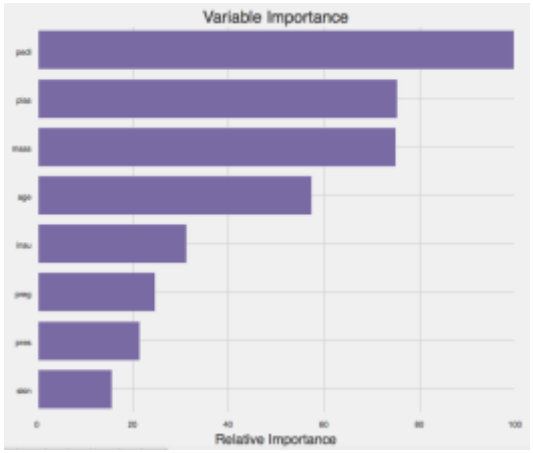
	Precision	Recall	F1-score	Support
No Diabetes	0.97	0.96	0.97	1690
Diabetes	0.94	0.96	0.95	1070
Avg/Total	**0.96**	**0.96**	**0.96**	**2760**



The average 10-fold cross validated test and training error plotted as fractional training set size. Here it is interesting to see that the data is not overfitting like I originally hypothesized. But, there might be a bit of bias, I say bit because the error rate is pretty low and in a high bias situation the error rate is usually higher. I would be curious to see if given more data if the training and test error would end up converging. My hypothesis for this is that there are a lot of irrelevant features in this dataset. What would happen if we we're to get more data for this data set, would the test and training error increase? If so, there might be a bias in the data. I could also try to change the test and training split to 30/70 and analyze the results.

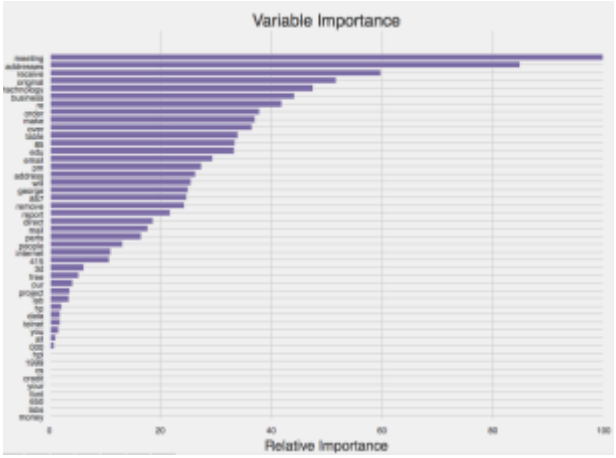
The first thing I noticed is that between the datasets I ran various train test split percentages and discovered that the smaller dataset diabetes performed better with more training data while the spam dataset did better with less training data. Additionally, I thought because the spambase pruning method limited the max_depth of the tees by almost half of that of the diabetes set it would be more similar in speed. Upon further inspection I believe this has to do with the size of the data sets

1.1.1 Variable Importance



You can see that there are some features that are not as important as the others, but I would not want to remove them because there is already such few features and if we can extrapolate any good data from those it is a good thing.

2.1.1 Variable Importance



You can see here that my above hypothesis was correct, that there is a number of unimportant features in this data set, some are even zero! In the future I would be interested to see if my analysis would improve if I removed these features.

1.3 Boosting Pruned Decision Trees

To test different parameters and different boosting algorithm's I used GridSearchCV in order to find the best scoring parameters using AdaBoostClassifier for SAMME and SAMM.R algorithms to boost my binary decision tree. The grid search was formed with n_estimators: 1, 10, 50, 100, 200, and also learning_rate: .1, .2, .3, .4 I used the same Decision Tree Classifier with the "pruned" parameters as above.

The best performing parameters are as follows:

Boosting Algorithm	Best Accuracy Score after Fitting data	Best Parameters	Test Run Time (Seconds) for both
Adaboost SAMME.R	0.741	n_estimators: 1, learning_rate: .3	
Adaboost SAMME	0.761	n_estimators: 10, learning_rate: .2	127.35

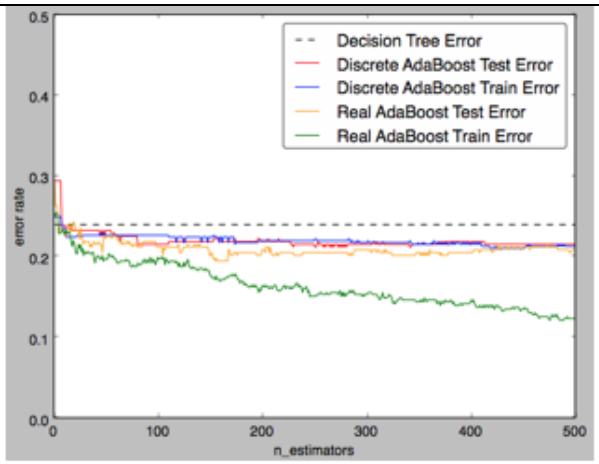
I decided to graph the learning curve for both of these with the learning rate and 500 estimators to see if one of these might be over fitting.

2.2 Boosting Pruned Decision Trees

To test different parameters and different boosting algorithm's I used GridSearchCV in order to find the best scoring parameters using AdaBoostClassifier for SAMME and SAMM.R algorithms. The grid search was formed with n_estimators: 1, 10, 50, 100, and also learning_rate: .1, .2, .3, .4 I used the same Decision Tree Classifier with the "pruned" parameters as above.

Boosting Algorithm	Best Accuracy Score after Fitting data	Best Parameters	Test Run Time (Seconds) for both
Adaboost SAMME.R	0.9446	n_estimators: 100, learning_rate: .3	
Adaboost SAMME	0.9481	n_estimators: 100, learning_rate: .4	588.37

I decided to graph the learning curve for both of these with the learning rate and 500 estimators to see if one of these might be over fitting.



Here you can see that the SAMME.R ends up over fitting the data almost right away and results in lower accuracy than SAMME, this is why I ended up choosing SAMME.

After applying the boosting algorithm (SAMME) I got these results.

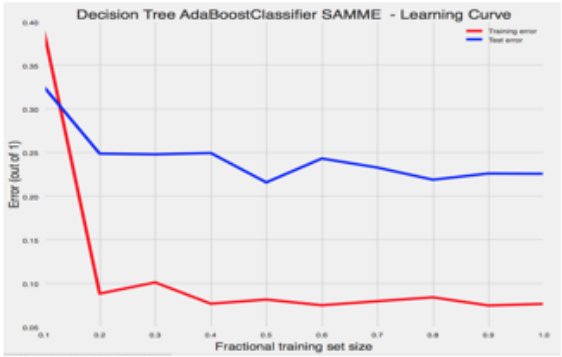
Boosting Algorithm	Parameters	Train Accuracy	Test Accuracy	Test Run Time (Seconds)
SAMME	n_estimators: 10, learning_rate: .2	0.8808	0.813	0.0360

Test

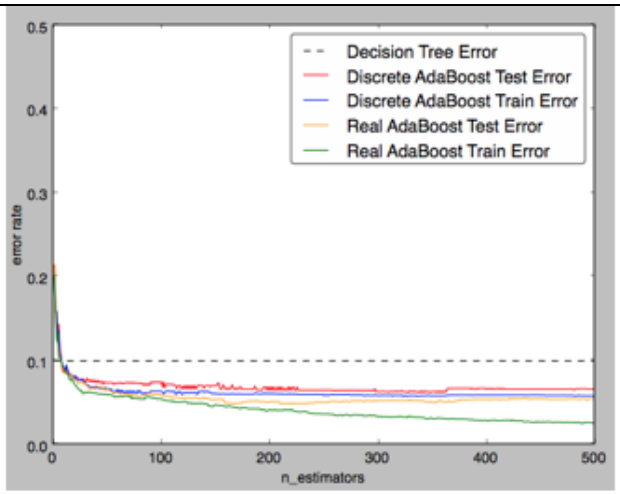
	Precision	Recall	F1-score	Support
No Diabetes	0.81	0.85	0.83	197
Diabetes	0.65	0.58	0.61	95
Avg/Total	**0.75**	**0.76**	**0.76**	**292**

Train

	Precision	Recall	F1-score	Support
No Diabetes	0.90	0.95	0.93	281
Diabetes	0.89	0.82	0.86	155
Avg/Total	**0.90**	**0.90**	**0.86**	**436**



The average 10-fold cross validated test and training error plotted as fractional training set size. You can see that the boosted tree increased the test accuracy by 11% which is significant, while the test error stayed about the same. Although the accuracy increased, the data is overfitting. You can see this in the large gap that is between the test and training error. I believe this is from noisy data because adaboost is particularly prone to overfitting when the data is noisy.



As you can see here they both fit the data pretty well but after 200 estimators SAMME.R starts to over fit very slightly after around 200 estimators. I chose SAMME.R over SAMME because it has a lower error rate.

After applying After applying the boosting algorithm (SAAME.R) I got these results.

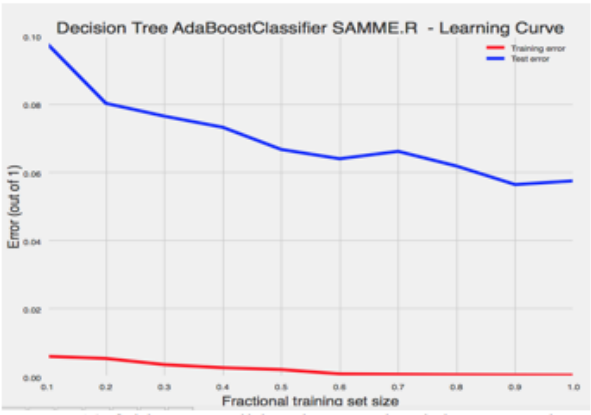
Boosting Algorithm	Parameters	Train Accuracy	Test Accuracy	Test Run Time (Seconds)
SAMME.R	n_estimators: 100, learning_rate: .3	0.9996	0.9451	3.07

Test

	Precision	Recall	F1-score	Support
No Spam	0.95	0.96	0.95	1098
Spam	0.94	0.93	0.93	743
Avg/Total	**0.95**	**0.95**	**0.95**	**1841**

Train

	Precision	Recall	F1-score	Support
No Spam	1.00	1.00	1.00	1690
Spam	1.00	0.96	1.00	1070
Avg/Total	**0.96**	**0.96**	**0.96**	**2760**



The average 10-fold cross validated test and training error plotted as fractional training set size. Wow, boosting pruned decision trees for the spam data set did really well with a 94% accuracy. To add to this there does not seem to be any under or over fitting, as the training error stays constant around .01 while the test error decreases as the training set size increases. This also indicates that this data set is not very noisy.

The second kernel **"linear"** optimal parameters are:

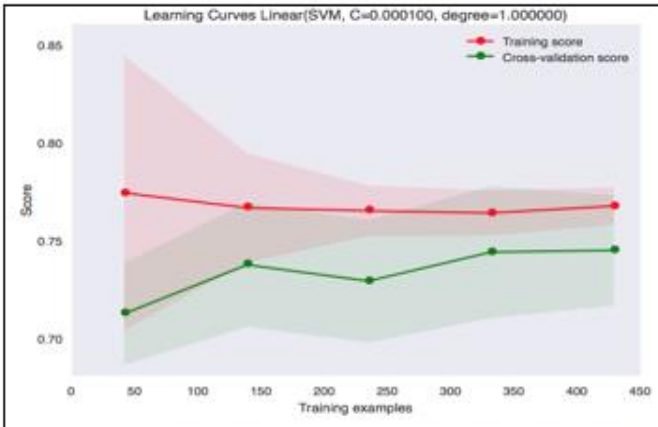
Degree	C	Train Accuracy	Test Accuracy	Test Run Time (Seconds)
1	1	0.7672	0.7922	3.75

Test

	Precision	Recall	F1-score	Support
No Diabetes	0.78	0.93	0.85	146
Diabetes	0.82	0.55	0.66	85
Avg/Total	**0.80**	**0.79**	**0.78**	**231**

Train

	Precision	Recall	F1-score	Support
No Diabetes	0.78	0.90	0.84	354
Diabetes	0.73	0.50	0.60	183
Avg/Total	**0.76**	**0.77**	**0.75**	**537**



The second kernel **"linear"** optimal parameters are:

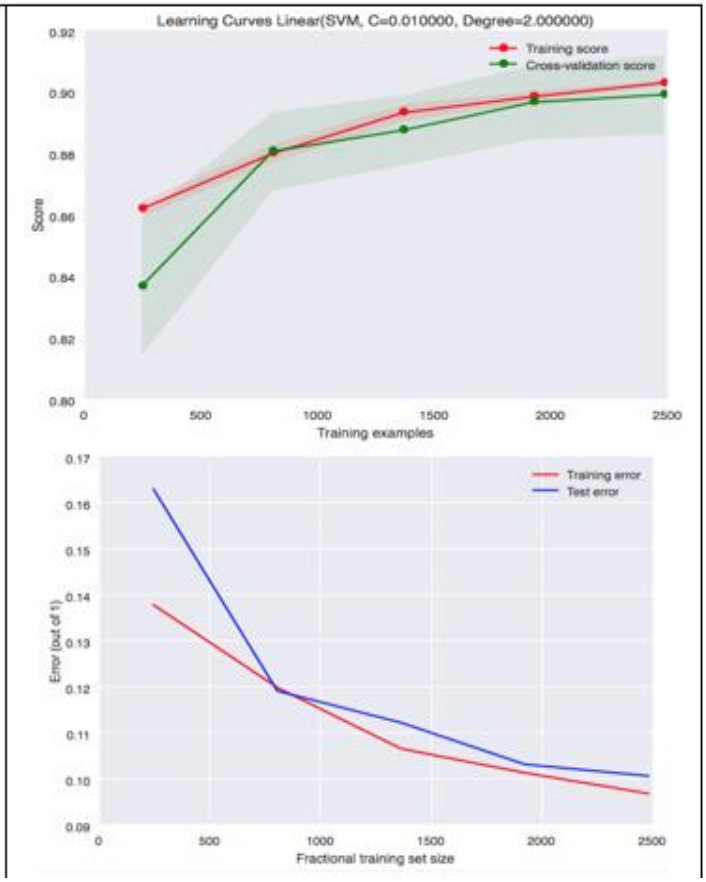
Degree	C	Train Accuracy	Test Accuracy	Test Run Time (Seconds)
2	0.0100	0.9032	0.9048	7.84

Test

	Precision	Recall	F1-score	Support
No Spam	0.90	0.94	0.92	1098
Spam	0.91	0.85	0.88	742
Avg/Total	**0.91**	**0.90**	**0.90**	**2760**

Train

	Precision	Recall	F1-score	Support
No Spam	0.90	0.94	0.92	1690
Spam	0.90	0.84	0.87	1070
Avg/Total	**0.90**	**0.90**	**0.90**	**2760**



1.4 Support Vector Machine

Two different Kernels were evaluated in this algorithm, rbf and linear. After attempting to do a GridSearch for the optimal parameters and having my code run for three hours and not getting any results I decided to manually adjusted the gamma and C for rbf and degree and C values to discover which pair fits the data best. I did this for two different kernels.

The first kernel **"rbf"** optimal parameters are:

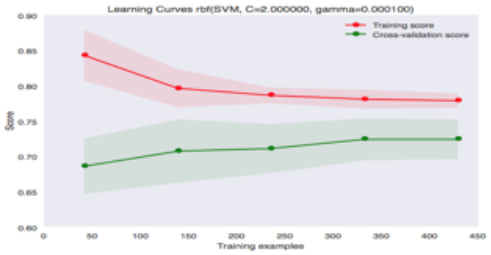
Gamma	C	Train Accuracy	Test Accuracy	Test Run Time (Seconds)
.0001	2	0.7728	0.7792	1.61

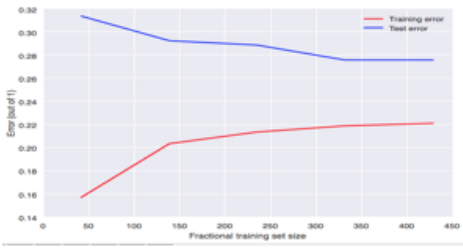
Test

	Precision	Recall	F1-score	Support
No Diabetes	0.77	0.94	0.84	146
Diabetes	0.83	0.51	0.63	85
Avg/Total	**0.79**	**0.78**	**0.76**	**231**

Train

	Precision	Recall	F1-score	Support
No Diabetes	0.77	0.93	0.84	354
Diabetes	0.77	0.48	0.59	183
Avg/Total	**0.77**	**0.77**	**0.76**	**537**





The second kernel **"linear"** optimal parameters are:

Degree	C	Train Accuracy	Test Accuracy	Test Run Time (Seconds)
1	1	0.7672	0.7922	3.75

Test

	Precision	Recall	F1-score	Support
No Diabetes	0.78	0.93	0.85	146
Diabetes	0.82	0.55	0.66	85
Avg/Total	**0.80**	**0.79**	**0.78**	**231**

Train

	Precision	Recall	F1-score	Support
No Diabetes	0.78	0.90	0.84	354
Diabetes	0.73	0.50	0.60	183
Avg/Total	**0.76**	**0.77**	**0.75**	**537**

2.3 Support Vector Machine

I also tried to run GridSearch to find the optimal parameters but also fan into the problem with very significant computing time, I left it my program running for three hours with this one too and decided to abandon GridSearch. Instead I manually adjusted the gamma and C for rbf and degree and C values for linear to discover which pair fits the data best.

The first kernel **"rbf"** optimal parameters are:

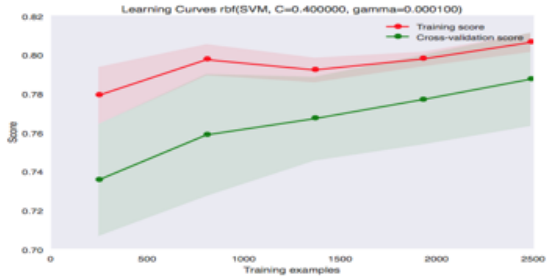
Gamma	C	Train Accuracy	Test Accuracy	Test Run Time (Seconds)
.0001	4	0.8138	0.7918	0.467

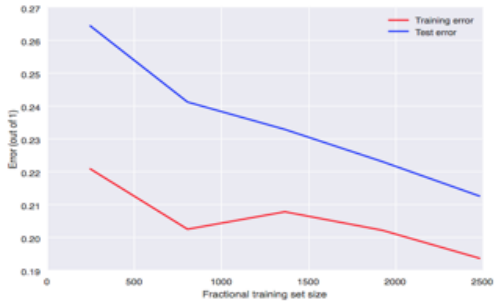
Test

	Precision	Recall	F1-score	Support
No Spam	0.80	0.87	0.83	1098
Spam	0.77	0.68	0.73	742
Avg/Total	**0.79**	**0.79**	**0.79**	**1840**

Train

	Precision	Recall	F1-score	Support
No Spam	0.82	0.89	0.85	1690
Spam	0.80	0.69	0.74	1070
Avg/Total	**0.81**	**0.81**	**0.81**	**2760**





The second kernel **"linear"** optimal parameters are:

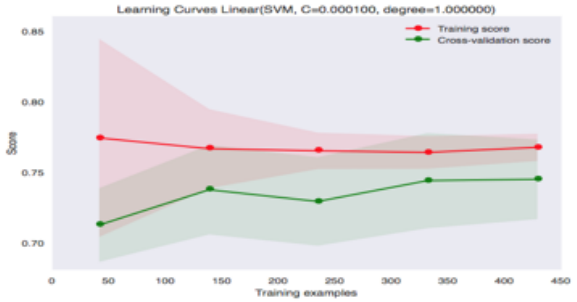
Degree	C	Train Accuracy	Test Accuracy	Test Run Time (Seconds)
2	0.0100	0.9032	0.9048	7.84

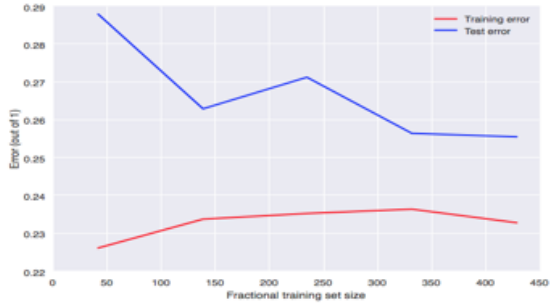
Test

	Precision	Recall	F1-score	Support
No Spam	0.90	0.94	0.92	1098
Spam	0.91	0.85	0.88	742
Avg/Total	**0.91**	**0.90**	**0.90**	**2760**

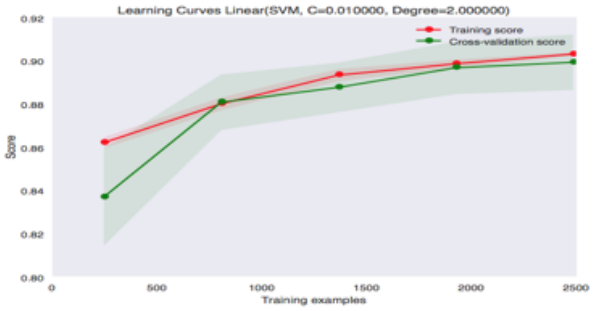
Train

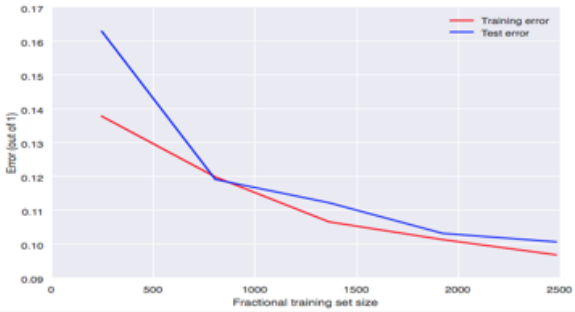
	Precision	Recall	F1-score	Support
No Spam	0.90	0.94	0.92	1690
Spam	0.90	0.84	0.87	1070
Avg/Total	**0.90**	**0.90**	**0.90**	**2760**





SVM-Linear is a better choice because there is not as big of a gulf between training and test errors when compared to the SVM-rbf. SVM-rbf is clearly suffering from high variance (overfitting), it is demonstrated in the score learning curve where the training score and cross-validation score both decrease. To add to this, in the error learning curve, the test error is decreasing while the training error is increasing and there is a gap in between those two curves.
On the other hand, SVM-linear's cross-validation score is increasing while the training score pretty much stays the same, while the test error is decreasing and the training error increases only slightly then goes back down. Although this is less than ideal, it is still a better choice when compared to the SVM-rbf results. Comparing SVM-Linear to the pruned decision tree you can see the SVM-Linear does worse, I believe this might be because of my kernel choice, further research requires if my hypothesis is right or wrong. When comparing the SVM-Linear to the adaboosted tree algorithm, you can see that the adaboosted classifier does a bit better in fitting the model, this is because SVM-Linear does poorly with noisy data, this is a clear example of this happening. Although it does not fit the data as well I believe the boosted decision tree is still a better choice because it generates higher train and test accuracy.





SVM-Linear is a better choice than SVM-rbf because while both kernels have both training and cross-validation scores increasing, and both training and testing error are also decreasing, SVM-linear's curves are much closer together and has a higher accuracy in addition to lower error. This signals a better fit of the data. I believe that SVM-Linear does a good job fitting the data because this data set has low noise and a pretty large number of attributes compared to the size of the data set which is what kind of features SVM-Linear does well with. Additionally, because the data fits very well with the linear kernel we can see tell that the classes are separable by a hyperplane. An interesting point to be seen is when comparing this classifier to pruned decision trees, you can see that the accuracy is much worse using SVM-Linear and I believe this is because of the irrelevant features that were pruned using the pruned using the max_depth while the SVM-Linear tries to map these features, causing accuracy to drop.

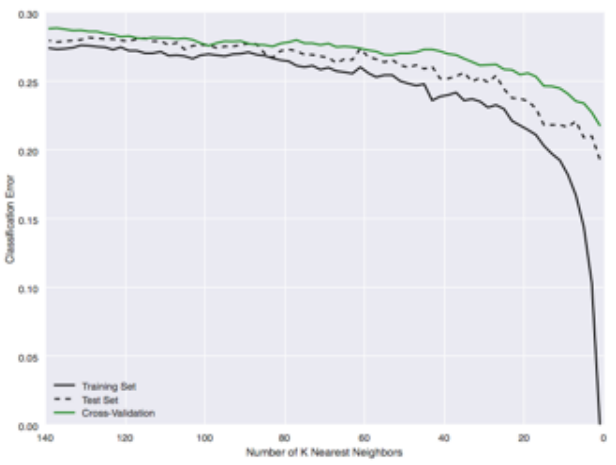
I thought it was interesting that with both data sets trying to find the optimal parameters using gridsearch took such a long time I believe not using this had an impact on my results, because I know that SVM's require parameter tuning to increase their accuracy with the test results. A conclusion that I came to after analyzing both results and comparing them to the other learners is that although SVM is a more complex classifying algorithm, just because it is more complex doesn't necessarily mean that it will be more efficient or give you better results.

1.2 K Nearest Neighbors

The first step I took for KNN is graphing the testing, training and cross-validation errors against 140 different K values.

2.2 K Nearest Neighbors

The first step I took for KNN is graphing the testing, training and cross-validation errors against 140 different K values.



With this graph you can see the best value for the lowest classification error for K lies between 9 and 18. I ran a grid search for the best K values and got K=15 for the test set and K=9 for the cross-validation. So, I decided to test and graph learning curves for both K values and see which one does better in terms of over or under fitting.

Results for K=9

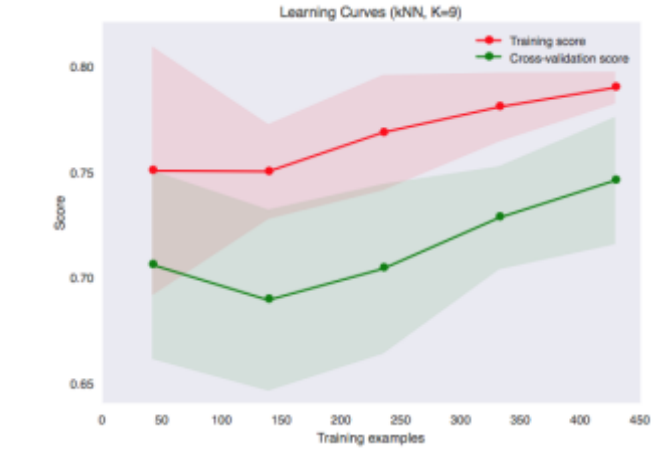
Train Accuracy	Test Accuracy	Test Run Time (Seconds)
0.7914	0.7489	0.00304

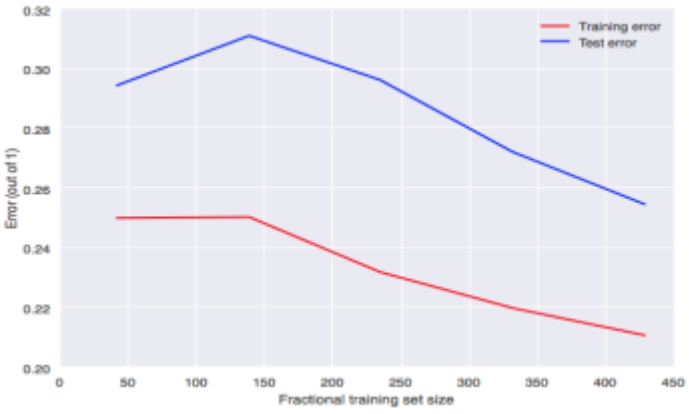
Test

	Precision	Recall	F1-score	Support
No Diabetes	0.80	0.84	0.82	157
Diabetes	0.62	0.55	0.59	74
Avg/Total	**0.74**	**0.75**	**0.74**	**231**

Train

	Precision	Recall	F1-score	Support
No Diabetes	0.82	0.87	0.84	343
Diabetes	0.74	0.65	0.69	194
Avg/Total	**0.79**	**0.79**	**0.79**	**537**





With this graph you can see the best value for the lowest classification error for K lies between 1 and 15. I ran a grid search for the best K values and got K=1 for the test set and K=1 for the cross-validation. So, I decided to test and graph learning curves for both K values and see which one does better in terms of over or under fitting

Results for K=1

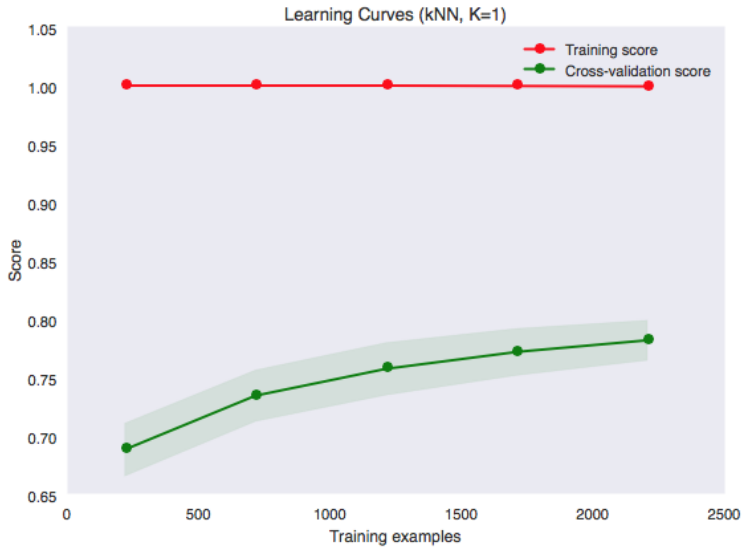
Train Accuracy	Test Accuracy	Test Run Time (Seconds)
0.9992	0.8065	0.0863

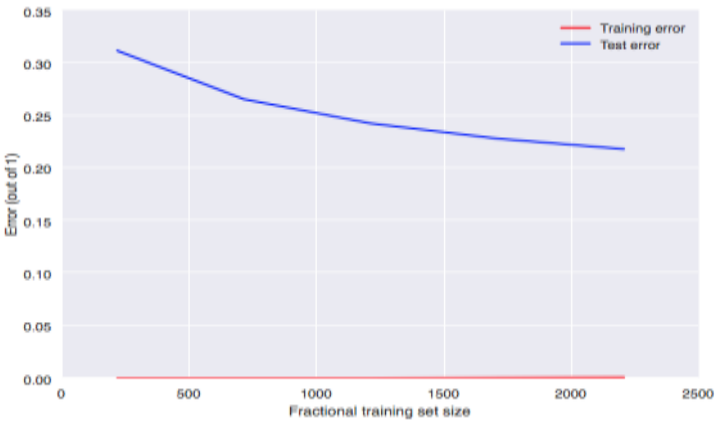
Test

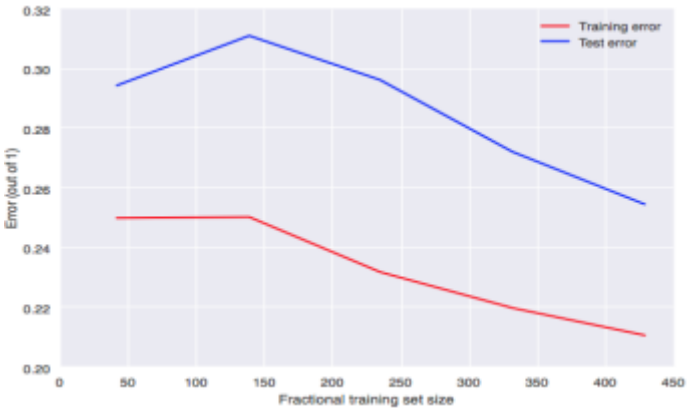
	Precision	Recall	F1-score	Support
No Spam	0.84	0.83	0.84	1098
Spam	0.76	0.77	0.76	742
Avg/Total	**0.81**	**0.81**	**0.81**	**1840**

Train

	Precision	Recall	F1-score	Support
No Spam	1.00	1.00	1.00	1690
Spam	1.00	1.00	1.00	1070
Avg/Total	**1.00**	**1.00**	**1.00**	**2760**







Results for K=15

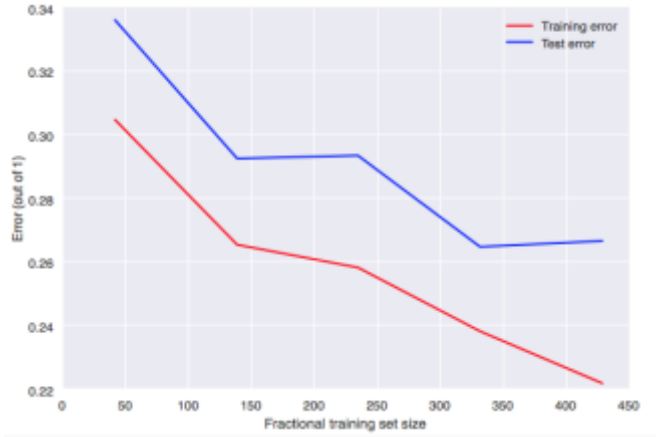
Train Accuracy	Test Accuracy	Test Run Time (Seconds)
0.7877	0.7575	0.01208

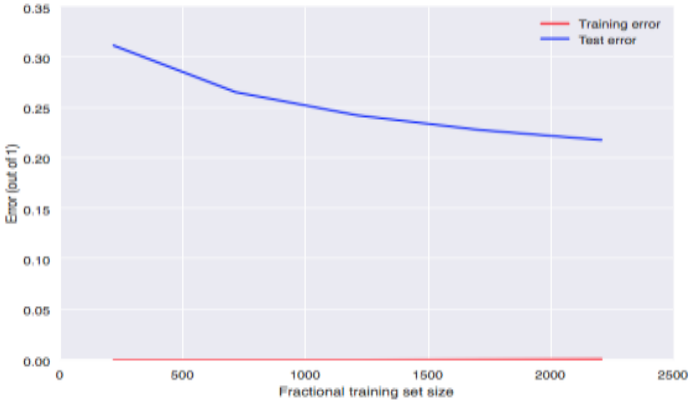
Test

	Precision	Recall	F1-score	Support
No Diabetes	0.79	0.89	0.83	157
Diabetes	0.67	0.49	0.56	74
Avg/Total	**0.75**	**0.76**	**0.75**	**231**

Train

	Precision	Recall	F1-score	Support
No Diabetes	0.81	0.88	0.84	343
Diabetes	0.74	0.63	0.68	194
Avg/Total	**0.78**	**0.79**	**0.79**	**537**





Results for K=15

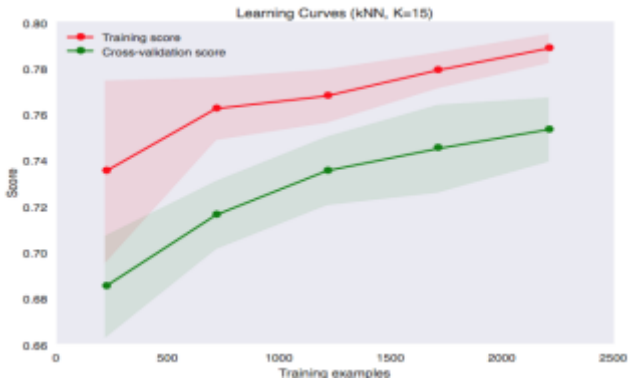
Train Accuracy	Test Accuracy	Test Run Time (Seconds)
0.7967	0.7815	0.1461

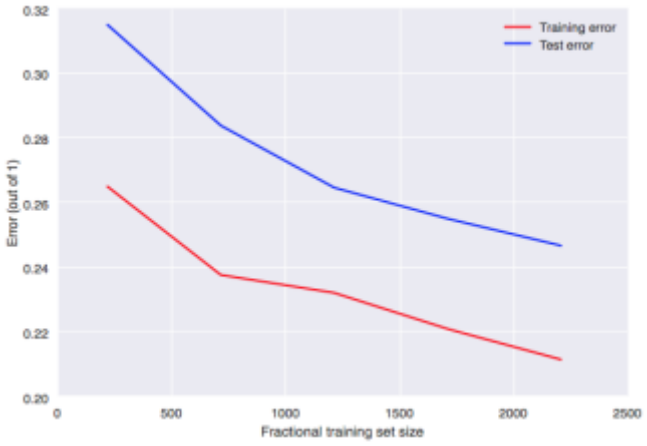
Test

	Precision	Recall	F1-score	Support
No Spam	0.80	0.84	0.82	1098
Spam	0.74	0.70	0.72	742
Avg/Total	**0.78**	**0.78**	**0.78**	**1840**

Train

	Precision	Recall	F1-score	Support
No Spam	0.83	0.84	0.84	1690
Spam	0.75	0.72	0.73	1070
Avg/Total	**0.80**	**0.80**	**0.80**	**2760**





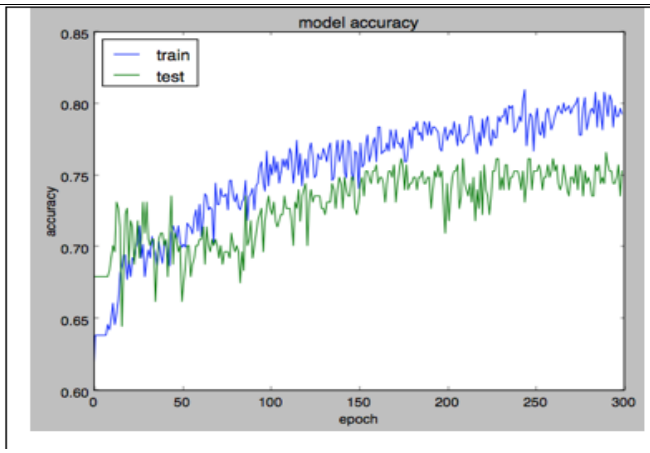
K =15 is better for this data set because both training and cross-validation scores increasing, and both training and testing error are also decreasing. The reason I don't choose k=9 is because although both training and cross-validation scores increasing, and both training and testing error are also decreasing, the curves are not as close together, showing that k=9 does not fit the data as well as k=9. Judging from the classification curve at the top of this section I believe if we went much higher than K=15 we would end up biasing our model on the test set. Another interesting find I made on this data set with KNN was another piece of evidence of a noise data set. This is shown in the K from range 1K to 5K because the accuracy changes a lot, and that is seen as a possible sign of noisy data.

With K=1 it is clear that the model has high variance, this is because with very low K vales each point only considers itself when making new classifications when evaluating on the training set it is making perfect predictions, which you can see in the learning curve with the error rate where the training set is at zero percent error and 100% accuracy on the training set, which is a good indicator of overfitting. So, K=1 would not be a good K value for this data set.

I wonder if K=1 would ever be a good value for a data set. We see that increasing K to 15 decrease that variance while not increasing bias by too much, but if we were to increase K more we would see that it would become high bias. For K=15, it is not as fast as K=1, this is because having K=15 generates a more complex model, for this data set, you are trading off speed for accuracy.

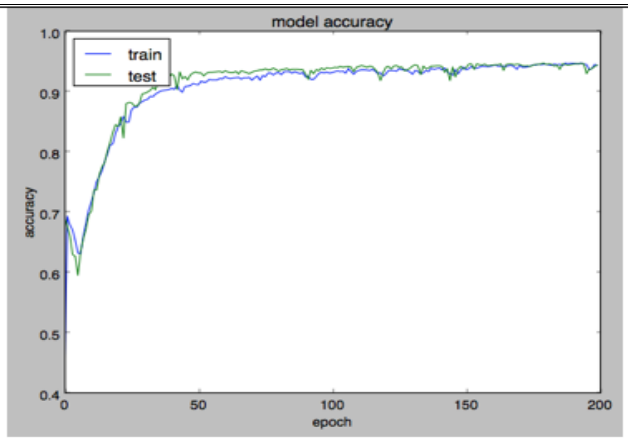
1.2 Neural Network

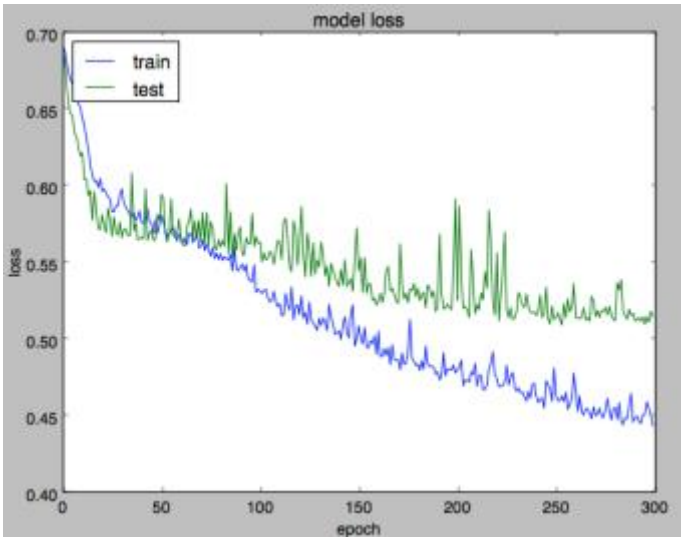
I created this neural network with the Library Keras. After testing different number of hidden layers, activations and number of nodes I found what I considered to be optimal parameters. The amount of feedback loops for backpropagation are called epochs. I initialized the network with a dense function which connects the entirety of the nodes and chose the weights of the network to be randomly selected from a uniform distribution. the activation I chose to use on the first and third layer is relu, with the second layer being linear and the last layer being a sigmoid output layer. The reason we use sigmoid as the output layer is so that the output of the network is between one and zero with a threshold of .05.
The first hidden network has 15 neurons with activation relu
The second hidden network layer has 6 neurons with activation linear
The third hidden network layer has 4 neurons with activation relu
The fourth hidden network layer has 1 neuron with activation sigmoid
In order to compile this model, I used binary cross entropy as the loss function to assess the weights in conjunction with the gradient decent algorithm.
I graphed the accuracy and loss for my train and test over 300 and 10,000 epochs with batch size 20 and returned these learning curves.



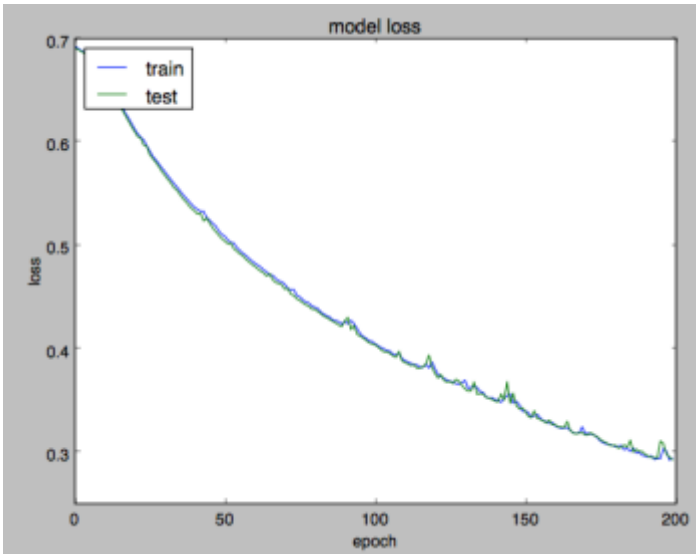
2.2 Neural Network

I created this neural network with the Library Keras. After testing different number of hidden layers, activations and number of nodes I found what I considered to be optimal parameters. I initialized the network with a dense function which connects the entirety of the nodes and chose the weights of the network to be randomly selected from a uniform distribution. the activation I chose to use on the first layer was relu, with the second layer being softmax and the last layer being a sigmoid output layer. The reason we use sigmoid as the output layer is so that the output of the network is between one and zero with a threshold of .05.
The first hidden network has 20 neurons with activation relu
The second hidden network layer has 6 neurons with activation softmax
The third hidden network layer has 1 neuron with activation sigmoid
In order to compile this model, I used binary cross entropy as the loss function to assess the weights in conjunction with the gradient decent algorithm.
I graphed the accuracy and loss for my train and test over 200 and 10,000 epochs with batch size 400 and returned these learning curves.

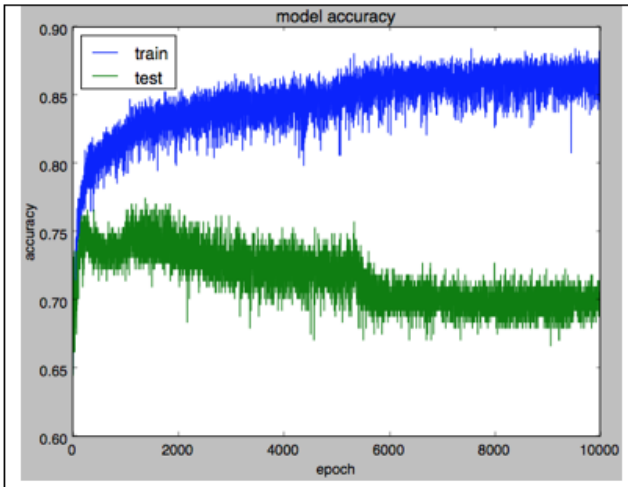


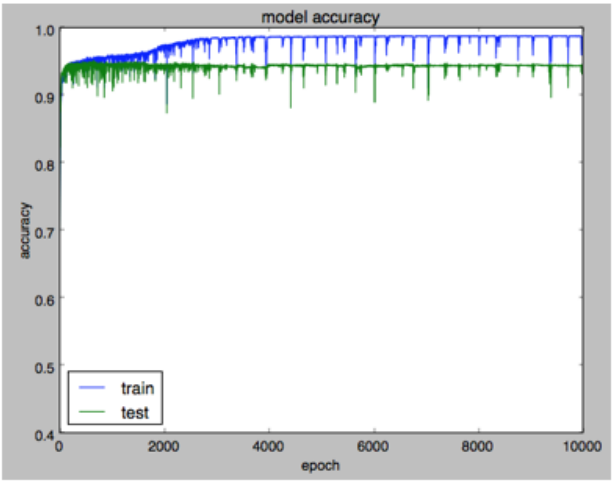


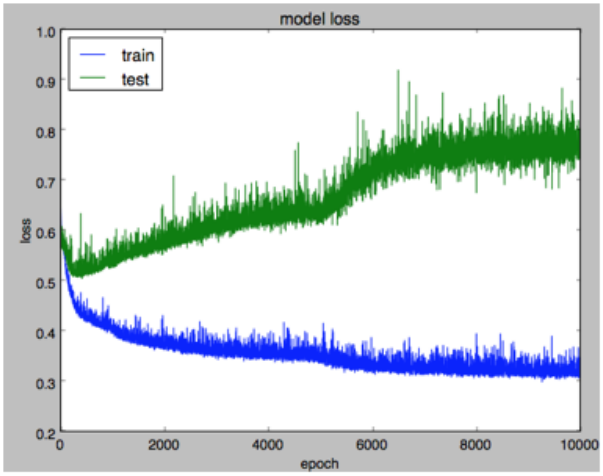
Train Accuracy	Test Accuracy	Test Run Time (Seconds)
0.7932	0.7532	7.44

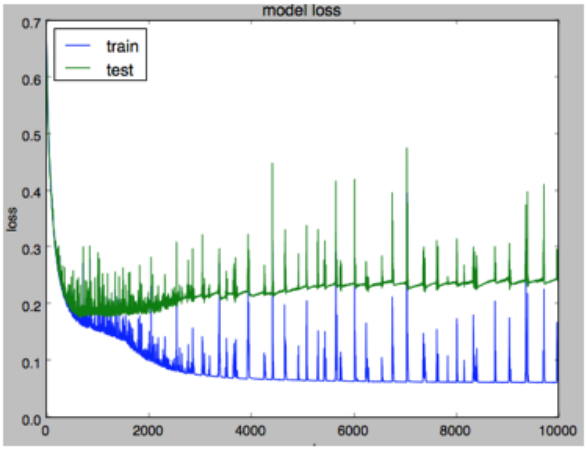


Train Accuracy	Test Accuracy	Test Run Time (Seconds)
0.9442	0.9442	4.58









Train Accuracy	Test Accuracy	Test Run Time (Seconds)
0.8715	0.7056	142.76

I used the loss graph with 10,000 epochs to see where I should stop the number of epochs. It is clear to see with the plot that the best number of epochs is around 300 because train and test loss are both decreasing, and we are looking for a model were we want to minimize loss because the lower the loss the closer our predictions are to the true labels. You can see that after around 500 epochs the training loss is dropping significantly and this is due to overfitting. When I ran the neural network with 300 epochs, you can see that I was correct in my initial analyzation and the model with 300 epochs is a good fit. When experimenting with what to choose in terms of the number of layers and type of activation to use I discovered that what I choose for the type of activation was much more important. Additionally, you can see how noisy this data set is with these graphs, although this could be mitigated by using a bigger batch size, when I tried that I got significantly worse results. I believe this is because with a smaller batch size the noise allows the model escape local minima.

Train Accuracy	Test Accuracy	Test Run Time (Seconds)
0.9855	0.9462	131.52

I used the loss graph with 10,000 epochs to see where I should stop the number of epochs. It is clear to see with the plot that the best number of epochs is around 500 because train and test loss are both decreasing, and we are looking for a model were we want to minimize loss because the lower the loss the closer our predictions are to the true labels. You can see that after around 400 epochs the training loss is dropping significantly and this is due to overfitting. When I ran the neural network with 200 epochs, you can see that I was correct in my initial analyzation and the model with 200 epochs is a good fit. When experimenting with what to choose in terms of the batch size for this data set that a higher batch set lowered the model loss significantly.

Conclusion

	Best Stats of Diabetes Classification Algorithms			Best Stats of Spambase Classification Algorithms		
Classification Algorithm	**Training Accuracy**	**Test Accuracy**	**Run Time (Seconds)**	**Training Accuracy**	**Test Accuracy**	**Run Time (Seconds)**
Pruned Decision Trees	82.79%	70.27%	0.0019	96.05%	89.08%	2.0225
Adaboost	88.08%	81.30%	0.0369	99.96%	94.51%	3.07
SVM	76.72%	79.22%	3.75	90.32%	90.48%	7.84
KNN	78.77%	75.75%	0.01208	79.67%	78.15%	0.1462
Neural Network	79.32%	75.32%	7.44	94.42%	94.42%	4.58

In conclusion, the best classification algorithm for the Diabetes data set is, Adaboost with the pruned decision trees. I chose this algorithm because of the high training and test accuracy along with the great run time. Although the adaboost classifier did not fit the data as well as the SVM I believe the boosted decision tree I believe that t is still a better choice because it generates higher accuracy on the training and test accuracy, the adaboost also ranked in a much faster run time which is important to consider as well. As for the Spambase data set, the best classification algorithm was also adaboost with the pruned decision trees. Although neural networks is a close second I don't believe that it is superior from my results because it has a lower test and train accuracy along with a longer running time. These results really surprised me because my initial intuition was that the more complex algorithms such as SVM and neural networks would perform better than decision trees, it is apparent that just because they are not as complex, they should not be overlooked in the slightest.