Assignment 1: Supervised Learning

Ahmad Aldabbagh

September 25, 2016

1. Introduction

In this assignment, I compare the performance of five (5) supervised learning algorithms on classification problems. I have chosen two data sets for this assignment, one is 'breast cancer data' and the other is 'letter recognition'. The algorithms will be trained on a subset of these data sets and tested on the remainder of the data set.

2. Data Sets of Choice

The data sets I have chosen are 1) the 'Breast Cancer Wisconsin (Diagnostic) Data Set' and 2) the ' Letter Recognition Data Set'. Where both data sets have been acquired from the UCI Machine Learning Repository [1]; and they all have features extracted from them.

2.1 Breast Cancer Data Set

Breast cancer is one of the leading type of cancer in Women and accounts for 25% of all cancers in women [2]. It is suggested to detect the onset of breast cancer early before it causes symptoms where, if present, it usually has a small size and hasn't spread to other areas [3]. Cancer screening usually involves imaging using X-ray such as the use of Mammograms. However, such techniques are sometimes complemented by physical examinations to obtain more accurate and reliable results. These examinations often involve obtaining a sample of the tumor for analysis using fine needle aspiration (FNA) such as that shown in figure 1. The data set I obtained from the UCI repository [5] are digitized images of FNA breast mass. It has 569 instances and 30 features and two (2) output classes either **b**enign or **m**alignant.

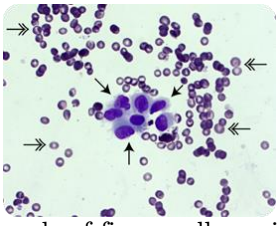

Figure 1: Example of fine needle aspiration image [4]

The reason I chose this data set, is that I wanted to have a small data set with a large number of features (curse of dimensionality!) and see how different algorithms perform/behave under these conditions. Another reason is that many papers cite it on the UCI website [5], which allows for a benchmark of comparing my performance to published results. The first thing I wanted to see was the distribution of features of both classes within the data set as shown in figure 2.

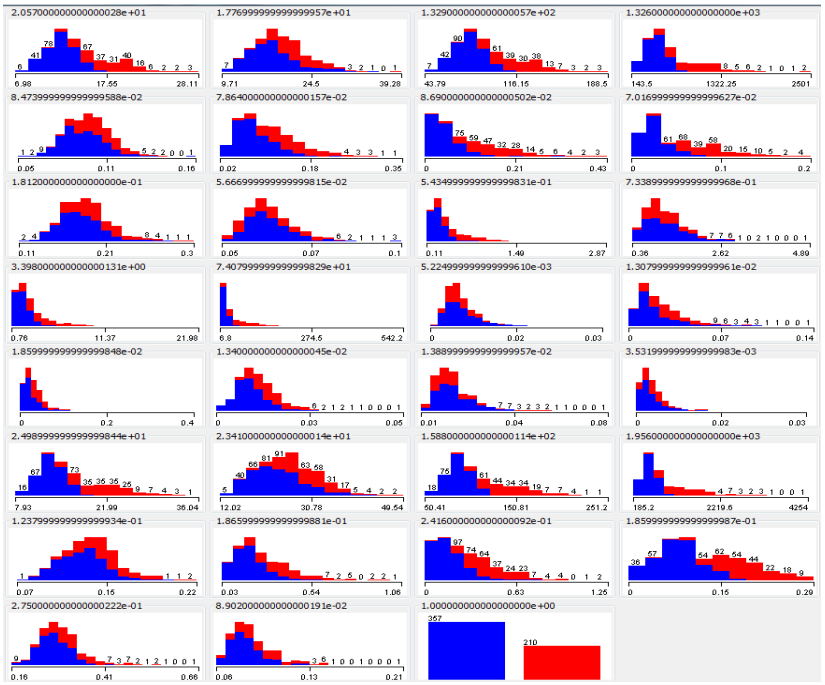

Figure 2: Distribution of features for Cancer Data. Each image represents a feature distribution except the last image which represents the class distribution

It can be seen that some features are similarly distributed among the classes while a few others aren't which seemed to be an interesting point to look for while analyzing the algorithms. Furthermore, since one class has a larger number of instances in the data set, I made sure while training the data to have a fair distribution of both classes in my subsets.

2.2 Letter Recognition Data Set

Optical Character Recognition (OCR) or Recognizing written letter is an area of interest in computer vision application. There are many uses for this kind of recognition. One particular use is for digitization of printed documents. Another use is for identification such as reading license plates, IDs etc. This data set tries to recognize such letters which are individually segmented and have been obtained from the UCI repository [6].

The data set contains 20,000 instances with 16 features that represent 26 classes, one for each character of the alphabets. Furthermore, the data set has been cited by multiple references for a variety of applications.

The main reason for choosing this data set is that I wanted to have a data set that has a larger number of instances as well as having multiple classes instead of just having to deal with two binary classification problems. Figure 3 shows the distribution of features where it can be seen in the first plot in that all classes are nearly represented.

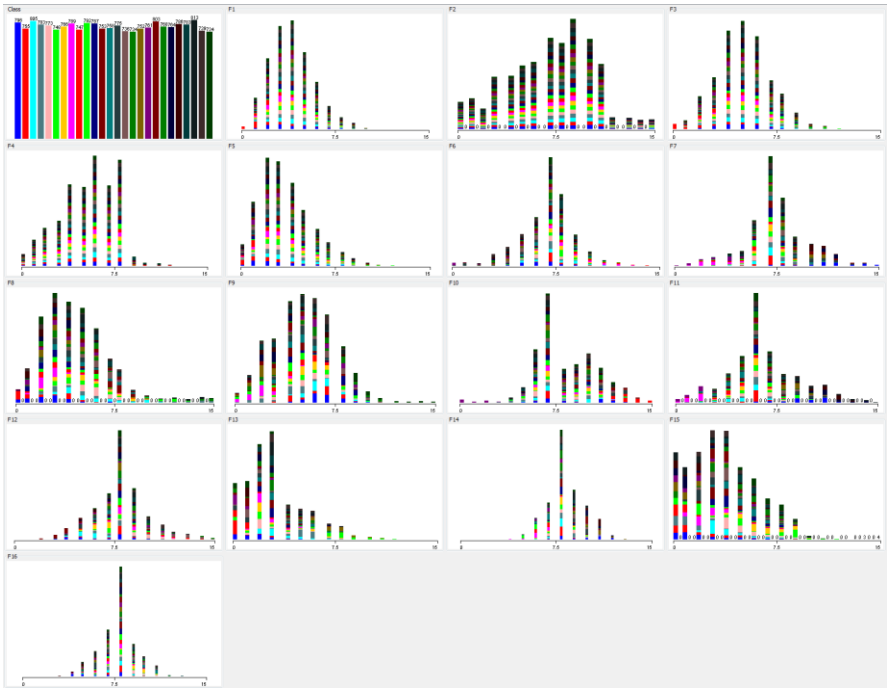

Figure 3: Distribution of features for Letter Data. Each image represents a feature distribution except the first image which represents the class distribution

2.3 Data Pre-processing

The data sets have been pre-processed to match the requirements of the programming language of choice and related packages (e.g. order of features & classes), which shall be explained further in the following section. Pre-processing started with changing class labels from strings to numbers. It also included normalizing features which resulted in yielding better results especially for the cancer data set. For instance, in an experiment with the cancer data set using KNN, the score of correctly classified instances increased from around 80% to around 95%. Which is quite understandable since KNN might give more weight to some features of lower variance. This would also be noticeable when using algorithms that leverage gradient-descent (many algorithms) to minimize the cost function resulting in a harder path to find the global minimum.

The used normalization/scaling was mean-normalization where the mean for a given feature (list of one kind of features) was subtracted from the feature and then the feature has been divided by the standard deviation.

3. Algorithms and Results

The development languages that have been used are Python, MATLAB and Weka. Specifically, Python (sklearn) is used for all algorithms except for neural networks, where MATLAB was used, and pruned decision trees, where Weka was used.

For each algorithm, the data sets have been split into training (60%) and testing (40%) data. Where testing data is not looked at till the very end. As for training data, 10-fold cross validation (20% of training) has been used to tweak the hyper-parameters of the

learner. To ensure a reasonable distribution of classes in training data. I looked at the histograms of the training set and testing sets for both data sets as shown in figure 4 below.

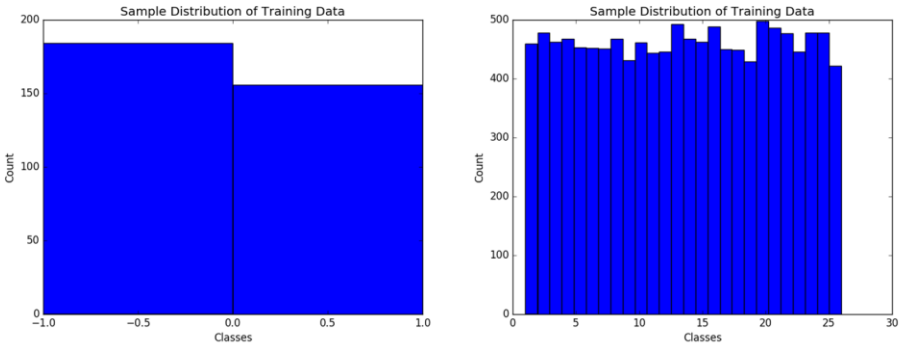

Figure 4: Distribution of Classes in training data. Cancer (left) and Letter (right)

The metrics used to analyze/train each learner include: learning curves, parameter value vs. cross validation error. As for the evaluating the results, I have looked at precision, recall, f1-score, support and percentage of total data set correctly classified (Classification Score). Furthermore, a confusion matrix was generated for the algorithms to aid the analysis. However, the confusion matrix for the Letter data set will not be shown since it is large (26 x 26).

3.1 Support Vector Machines (SVM)

In training SVMs, I used Python's sklearn package implementation of the algorithm. It is based on LibSVM [7] and sklearn's implementation includes various kernels and allows for a lot of customization. One of the main advantages of SVM is its ability to handle large dimensions especially when the number of sample to number feature ratio is not that high.

When I was first wanted to train my SVM classifier for the data sets that I have chosen, I wanted to try all four kernels offered by SVM ('rbf', 'linear', 'poly', 'sigmoid'), along with various values for (C & γ) for both chosen data sets. However, after two days of the script running and not finishing, it was realized that this is not the best option so I decided to tweak my hyper-parameters for the radial basis function kernel. My approach was to use 10-fold cross validation with my training data with 0.2 splits. Where different parameters are used to train the classifier and then the classifier is tested on the validation set.

The basis for my approach in searching for the parameters, is a paper published by the same authors of the LibSVM library [8]. It outlines a structured approach on using SVMs. They recommend starting with an 'rbf' kernel and observe whether the results are satisfactory to justify trying a different kernel. It also mentions how to run a grid search by varying C & γ. The parameter C is varied between 2^{-5} and 2^{15} with multiples of two. As for γ, it was varied 2^{-3} and 2^{3}). While this might seem like a coarse search, it can be used to determine the range for conducting a fine search or, in case of good results, can be kept as is and choose which ever values that yielded in the least cross validation error.

To ensure I am not overfitting or using more training samples without improving the performance, I looked at the learning curves as shown in figure 5. These curves are based

on the best combination of parameters for that data set where the parameters have been obtained through training and validating on the cross validation set, as mentioned earlier.

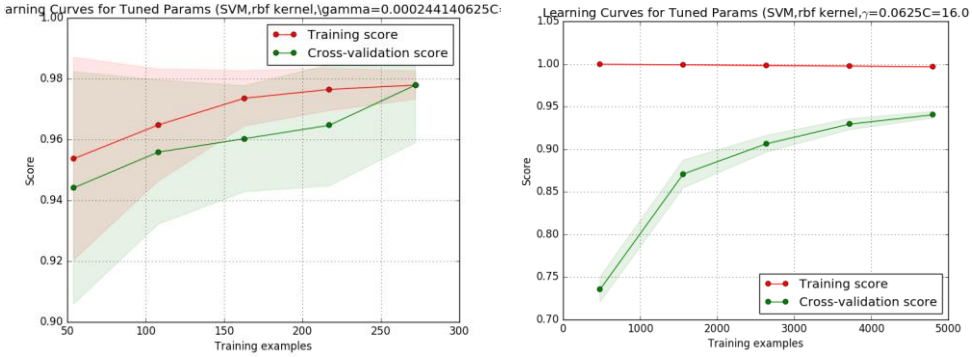

Figure 5: Learning curves for best performing parameters for Cancer Data (left) and Letter data (right)

While the learning curve for the cancer appears to have high bias due to the small gap between the training and cv curves, it can be argued however that this acceptable since the error is much less than the desired error.

In contrast, the graph on the right, for Letter data, appears to have high variance due to a larger gap between the training and cv curves but it still has a performance is higher than the desired performance especially for a large data set with a large number of classes.

Overall, both data sets had an acceptable performance where the cancer data set had a classification score of around 98.68% for C=16 and gamma=0.000244. The Letter data set had a score of 96.56% for C=16 and gamma=0.0625 which is a higher gamma than that of the cancer data set. From this we might conclude the letter data set tries to fit the support boundary while considering points that are close from the decision boundary which is expected since there are more than two classes. In contrast, the smaller value of gamma in the 2 class cancer data set gives influence to the points the are further away from the decision which is desirable to avoid over fitting Because giving more boundary to one class has a higher likelihood to encroach on the boundary of other classes since there 26 classes instead of 2. On the other hand, influence on the points that are further away from the decision boundary which is desirable to avoid over fitting especially that there are only two classes. Figure 6 illustrates this effect.

The main drawbacks I have seen in SVM is the training time as well as the time it takes to complete a hyper-parameter search, which was the highest in comparison to all the algorithms I have tested.

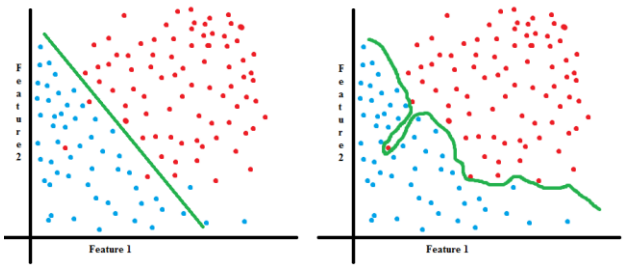

Figure 6: smaller gamma (left) and higher gamma (right)

Finally, below are the summary reports for the data sets.

Cancer Data Set:

	precision	recall	f1-score	support
Class 1	0.99	0.99	0.99	173
Class 2	0.96	0.98	0.97	55
avg / total	0.99	0.99	0.99	228

Confusion matrix:
[[171 2]
[1 54]]

Letter Data Set (average scores since details are too long for 26 classes):

avg / total	0.97	0.97	0.97	8000

3.2 K-Nearest Neighbor (KNN)

KNN is a non-parametric fitting model that classifies points based on their closest counter parts from the training data. The definition of closest varies and depends on the distance metric (Euclidean, Manhattan, etc) chosen when training the learner. The most important parameter however, is the value of 'k' which is the number of neighbors to consider when returning the result (whether classification or regression). I implemented the learner using Python's sklearn which has uniform weights by default and uses the Euclidean distance by default.

I had two variations of the learner where one was weighted and the other wasn't and for both, the value of 'K' was varied from 1 to 59 while observing the effect on the cross-validation data.

The curves shown in figure 7 below illustrate the effect of 'K' for both the weighted and un-weighted cases.

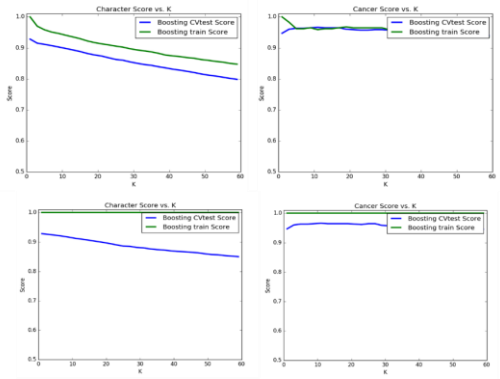

Figure 7: Un-Weighted KNN (top) and Weighted KNN (bottom)

Although the graphs look a bit different, both the unweighted and weighted learners resulted in the same values for K which are K=1 for Letter Data Set and K=11 for the Cancer Data Set. While the value of K=1 looks as if it over fitted it actually yielded good results and it can be seen in figure 7 that in the Letter data set as K increases so does the error. It, k=1, can also be problematic in the case where there are two instances with the same distance, so it would be more appropriate to go a higher value of k. As for the Cancer data set, varying the value of K seem to have little effect on increasing the performance. However, this is not true for large values of K where one might be looking

at values from the other class but this also that there is a good separation between the two classes that allows for such tolerance.

Learning curves for the cancer set seems to have more variance in the weighted learner than that in the un-weighted learner. While the learning curves for the Letter Data Set seemed to have similar variance in both cases as shown in figure 8. However, all are in acceptable error range.

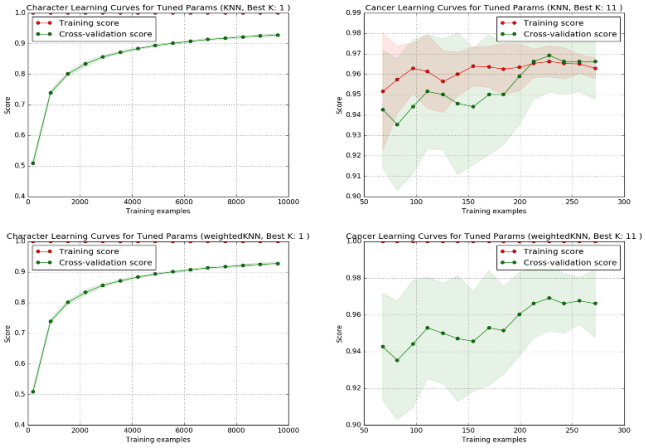

Figure 8: Learning Curves for Un-Weighted KNN (top) and Weighted KNN (bottom)

Finally, the classification scores for weighted vs. un-weighted are very similar as shown below. As a matter of fact, the confusion matrix for the cancer data set in both cases is identical. Time was much faster than SVM where it only took a few seconds to finish.

	Un-weighted	Weighted
Cancer	97.81 %	97.81 %
Letter	93.7 %	93.7 %

Table 1: Classification Scores

	Un-weighted (precision-recall-f1-score-support)				Weighted (precision-recall-f1-score-support)			
Cancer	0.98	0.98	0.98	228	0.98	0.98	0.98	228
Letter	0.94	0.94	0.94	8000	0.94	0.94	0.94	8000

Table 2: Summary Reports for knn (avg/total)

3.3 Neural Networks

In implementing neural networks, I have used MATLAB which offers a variety of functions to be used as the train function of the neural network. I experimented with Gradient Descent with Momentum (traingdm) and Scaled Conjugate Gradient Backpropagation (trainscg) with two different experiments. The first related to examining the effect of the momentum and learning rate as well as the layers. The second experiment is varying the hidden layer size while fixing the number of layers.

-Gradient Descent with Momentum (MC) and Learning Rate (LR):

#	Layer Size(s)	LR	MC	Epochs	Perf	Train perf	Val perf	Test perf	Time	functio n	trainfncn
1	40-26-10	0.1	0.9	1470	0.269	N/A	N/A	0.2683	02:41	pattern et	traingdm
2*	20-20	0.1	0.9	1699	0.269	0.269	0.269	0.269	01:40	pattern et	traingdm

3	20-20	0.1	0.5	1699	0.269	0.269	0.269	0.269	01:40	Pattern et	traingdm
4	20-20	0.1	0.0	1699	0.269	0.269	0.269	0.269	01:50	Pattern et	traingdm
5	20-20	0.1	0.5	1699	0.0375	0.0375	0.0375	0.0375	00:58	fitnet	traingdm
6	20-20	0.1	0.9	1699	0.0372	0.0372	0.0372	0.0371	00:58	fitnet	traingdm
7	20-20	0.4	0.9	1699	0.0365	0.0365	0.0365	0.0365	01:10	fitnet	traingdm
8	20-20	0.7	0.9	7 (stopped auto)	0.486	0.486	0.488	0.485	00:01	fitnet	traingdm
9	20-20	0.5	0.9	1699	0.0363	0.0363	0.0363	0.0363	01:08	fitnet	traingdm
10	20-20	0.01	0.9	1699	0.0544	0.0544	0.0546	0.0542	00:56	fitnet	traingdm
11	20-20	0.9	0.0	1699	0.0357	0.0357	0.0356	0.0357	00:58	Fitnet	traingdm
12	20	0.9	0.0	1699	0.0343	0.0343	0.0342	0.0342	00:40	fitnet	traingdm
13	20	0.01	0.9	1699	0.121	0.121	0.119	0.120	00:44	Fitnet	traingdm
14	20	0.1	0.9	1699	0.0418	0.0418	0.0421	0.0417	00:49	Fitnet	traingdm
15	20	0.5	0.5	1699	0.0353	0.0353	0.0354	0.353	00:50	Fitnet	traingdm
16	20	0.1	0.9	1699	0.2689	0.2689	0.2689	0.2689	01:46	pattern et	traingdm

Table 3: 'traingdm' Experiment Summary (with some removed for space)

It can be seen above in number 2 vs. 1 that three layers vs. two layers does not offer much improvement in performance. Which was mentioned in Mitchell Ch. 6 that most nets with 2 hidden layers offer quite good performance. Where having more than 3 is not usually necessary. This is expected, since the data set produced good results in other simpler algorithms.

As for the number of epochs relation with performance it was found in experiment 2*, even after 10000 epochs without overfitting the CV set, the performance is around 0.268 which is not worth the additional iterations. Moreover, it was noticed that the run time is even higher than the 3 layered network as shown in number 1.

When removing momentum, It was noted that removing the momentum did not have much effect on the performance of the data set. However, it decreased the runtime by around 10 seconds while getting the same results. The main purpose for it, is to allow the network to escape from some local minima.

It was noticed that as I increased the learning rate, the performance increased at the fixed epochs of 1699 but it also meant that it might have overshoot where I set the validation check to 6. Such as the case with experiment 8. When I ran that again I got a performance of 0.360 after going through the 1699 epochs. This was made clear when I increased the learning rate to 0.9, I got a score of 0.0356 with a time of 01:01 but in some instance I get a bad result also quickly. To overcome this, one can use an adaptive learning rate that is tuned to try using a step that quickens reaching a local optimum such as a L-BFGS algorithm [http://web.stanford.edu/class/cs294a/sparseAutoencoder˙2011new.pdf]. Further, the variability in results can be attributed to the change in initial conditions every time the algorithm is run.

-Scaled Conjugate Gradient Backpropagation

To see how the effect of the number size of hidden layers affects the performance, I decided to use the 'trainscg' function while changing the Size of hidden layers upto 300. Training automatically stops when generalization stops improving, as indicated by an increase in the cross-entropy error of the validation samples which was used as performance metric along the %Error.

Size of Hidden Layer	% Error on Cancer Test, validation and testing Data	% Error on Character Test, validation and **testing** Data
2	2.71, 1.75, 2.81 (21 Epochs)	67.98, 68.30, **67.43** (58 Epochs)
8	2.16, 0.80, 4.93 (33 Epochs)	24.04, 24.69, **25.61**(197 Epochs)

12	1.35, 0.00, 2.11 (26 Epochs)	17.13, 16.95, **18.94** (269 Epochs)
25	2.17, 0.00, 4.22 (29 Epochs)	08.03, 10.00, **10.44** (229 Epochs)
39	2.71, 10.5, 3.52 (11 Epochs)	06.51, 7.14, **9.24** (157 Epochs)
50	2.17, 0.00, 3.52 (20 Epochs)	03.89, 7.199, **8.58** (151 Epochs)
70	2.71, 0.00, 1.41 (23 Epochs)	02.04, 07.45, **6.94** (141 Epochs)

Table 4: 'trainscg' Experiment Summary (some omitted for space)

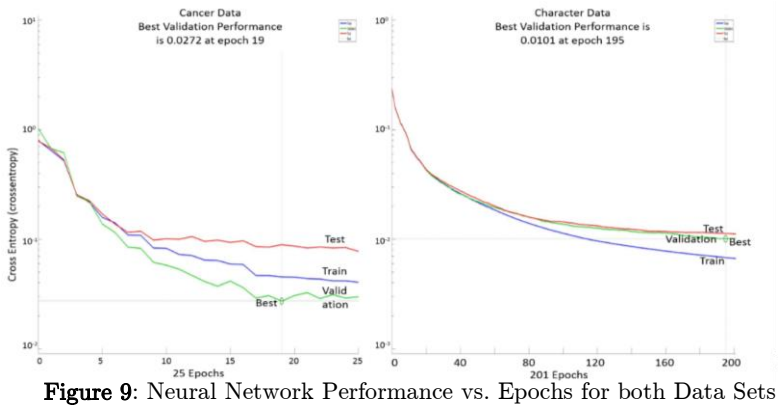

Figure 9: Neural Network Performance vs. Epochs for both Data Sets

The best classification error for the cancer data is 2.11% for a size of 12. Whereas the letter data had an error of 9.24% for a size of 39. Neural networks tend to overfit when their size is large where it fits the noise in the data. However, if the data is complex, it could be the case where a larger size is required similar to our letter data set where it contains many clustered classes. The reason is, a higher size would allow for a higher degree of freedom for the network to fit the data. To avoid overfitting in those cases, techniques such as regularization can be used in a manner similar to gamma (where a smaller regularization parameter makes the boundary more sensitive to closer points). As for time, in single layer networks most did not take more than a minute to run and this number increased as the number of layers increased and learning rate decreased with the max being around 10 mins.

3.4 Boosting

Boosting has been implemented in Python's sklearn. It implements a version of AdaBoost that begins by fitting a classifier on the data set and keep on fitting additional copies of the classifier on the set while adjusting weights of instances depending on their difficulty to classify. The main parameter can be tweaked and observed vs. performance is the learning rate. Another is the maximum number of estimators that determine when boosting is terminated. I have varied learning rate from 0.001 to 3.000 with 0.02 increments to determine the best learning rates for my data sets. Figure 10 shows this relationship.

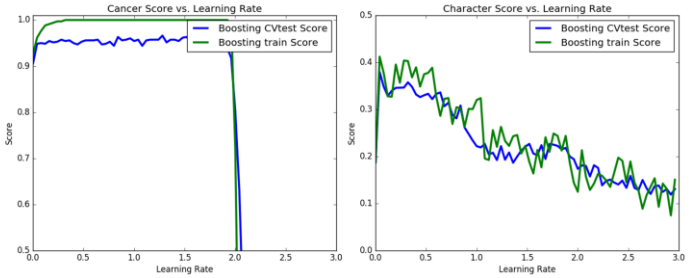

Figure 10: Learning Rate vs. Performance for Cancer Data (left) and Letter Data (right)

For the cancer data set, a high range of learning rates below 2 yield good results. However, in the Letter data, only a very small range yield the highest results but those results are poor when compared to the cancer data sets' results or when compared to other algorithms. The following graph (figure 11) shows the best of those learning rates plotted in a learning curve.

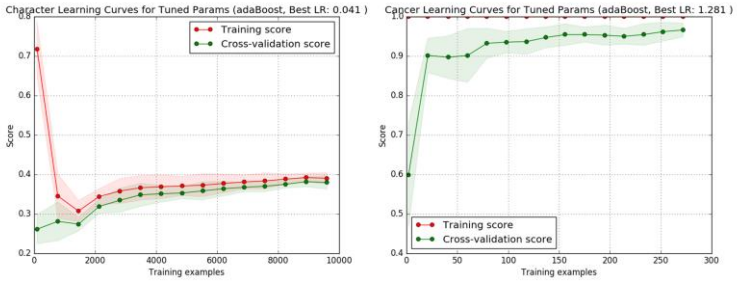

Figure 11: Boosting Learning Curves for Letter Data (left) and Cancer Data (right)

It can be seen in these graphs that both data sets seem to have a higher bias with the difference that the cancer data set has an acceptable error rate while the Letter data does not. Had I not seen the results of other algorithms, I would have thought that maybe adding more features to the letter data set would produce better results. It could be that boosting simply does not perform well in a high class data such as the letter data. The following table (table 5) illustrates the performance.

	Classification Score (% correct)	Summary report (avg/total) (precision-recall-f1-score-support)			
Cancer	97.37 %	0.97	0.97	0.97	228
Letter	40.60%	0.54	0.41	0.40	8000

Table 5: Boosting performance

3.5 Decision Trees

The unpruned decision tree algorithms I used was in Python's sklearn. It implements a CART algorithm which essentially is similar to C4.5 but it supports regression and does not compute rule sets. It constructs binary trees using feature and threshold resulting in the highest information gain for each node. The main variable I experimented with was the maximum depth parameter and I also varied the minimum number of samples required for a leaf to be at a leaf node. Figure 12 shows how that affects the performance for both data sets.

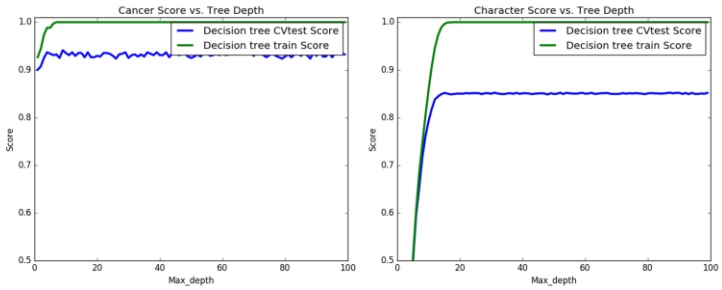

Figure 12: Tree Depth vs. Performance for Cancer Data (left) and Letter Data (right)

It can be seen that in both cases, the performance does not increase beyond a certain tree depth so it is good to find such depth in order to not explore the tree any further and

save on computation time. While the optimum might occur in higher depths, it was actually found that earlier values have similar results to the actual optimum. For example the best max depth for cancer was found to be at 18 but this is not a reasonable choice because if we inspect figure 12, we can say that a depth of around 10 is enough to yield results similar to the found depth. This applies to the other data set as well. This made me realize that simple search should be the only way to choose your parameters but should be a guide and other means should be used to assess the tradeoff between choosing something of cheaper computation without sacrificing much of the performance.

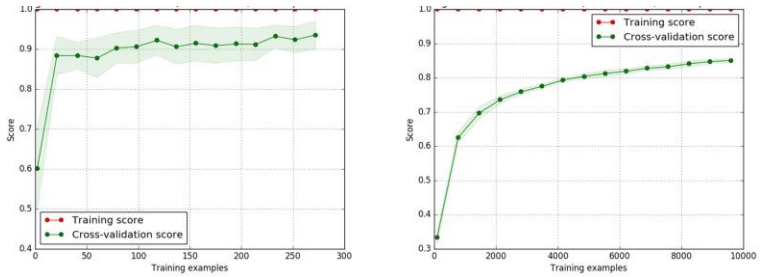

Figure 13: Decision Tree Learning Curves for Cancer Data (left) and Letter Data (right)

The learning curves for Cancer data show a fairly good performance. Whereas that for Letter data has some sort of variance that might be rectified if a feature selection algorithm is used such as PCA.

Pruning had positive effects on both data sets where it decreased the size of the trees in both data sets which is desirable. It also had a positive improvement on the cancer performance. On the other hand, it did not increase the score of the Letter data set but it reduced its tree size.

	Classification Score (% correct)	Number of Leaf Nodes	Size of Tree	Time to build model (s)
Cancer	91.90 %	22	43	0.02
Letter	87.45%	2238	4475	0.6

Table 6: Decision Tree Performance (no pruning)

	Classification Score (% correct)	Number of Leaf Nodes	Size of Tree	Time to build model (s)
Cancer	92.08 %	6	11	0.08
Letter	87.21 %	1949	3897	7.49

Table 7: Decision Tree Performance (with pruning)

Conclusion

In this assignment a comparison of five major algorithms has been made. The table below shows a comparison summary of the best obtained results on both the data sets.

Algorithm	Cancer (Score)	Letter (Score)	Cancer (train time)	Letter (train time)	Cancer (test time)	Letter (test time)
SVM	98.68%	96.56%	0.013 s	12.91 s	0.001 s	2.705 s
KNN	97.81%	93.70%	0.001 s	00.07 s	0.006 s	1.057 s
Neural Net	97.89%	90.79%	0.001 s	24 s	Included in train time	Included in train time
Boosting	97.37%	40.60%	0.177 s	1.251 s	0.006 s	0.476 s
Decision Tree	92.08%	87.21%	0.08 s	7.49 s	0.001 s	0.002 s

Table 8: Decision Tree Performance (with pruning)

In the table above it can be seen how each of the algorithms perform with respect to others in both accuracy and time. In the case of the first data set, it seems that it performed fairly well on all algorithms so I tried using the extraTreesClassifier in sklearn to determine the most valuable feature in the data set and was able to achieve more than 93% accuracy. Also, we can say that the data set might not contain a lot of noise and based on the results, automated classification systems might be deployed to aid physicians with high reliability (of course, more data will need to be evaluated first). The letter data set performed fairly well on most algorithms as well but had poor performance when classified using boosting. Surprisingly, KNN outperformed neural nets in the Letter data but when I inspected two of the main features, it seemed as if classes are clustered together, as shown in figure 14, which might explain the performance.

Furthermore, SVMs had the best performance in both data sets but had the only issue of large time required to find the optimum hyper-parameters. Its performance in both can be also attributed to SVMs adaptability to data sets with a lot of features.

Finally, it was shown that supervised methods can provide valuable insights in understanding labeled data, in the case there is a distribution correlated with the labels. Also, these techniques can prove useful in a wide range of areas but need plenty of tweaking, experimentation and understanding to yield positive results.

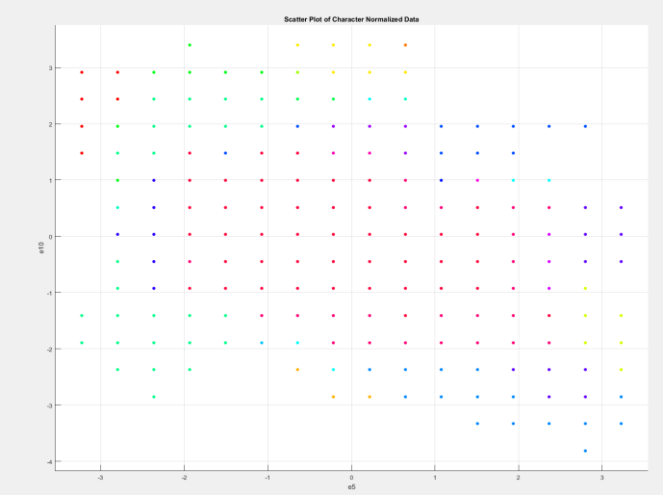

Figure 14: Distribution of Letter Data for top 2 features

References

[1] https://archive.ics.uci.edu/ml/index.html
[2] http://www.wcrf.org/int/cancer-facts-figures/data-specific-cancers/breast-cancer- statistics
[3]http://www.cancer.org/cancer/breastcancer/moreinformation/breastcancerearlydetection/breast-cancer-early-detection-acs-recs
[4] http://www.breastimaging.vcu.edu/services/guided/fineneedle.html
[5]https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29
[6] https://archive.ics.uci.edu/ml/datasets/Letter+Recognition
[7] http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf
[8] http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf
[9] Mitchell, T. M. (1997). Machine Learning. New York: McGraw-Hill.