Assignment 1: CS7641 - Machine Learning

Saad Khan

September 18, 2015

1 Introduction

I intend to apply supervised learning algorithms to classify the quality of wine samples as being of high or low quality and to segregate type 2 diabetic patients from the ones with no symptoms. The algorithms I will be implementing for this analysis are decision trees, artificial neural networks, boosting, support vector machines and k-nearest neighbors.

2 Data Sets

The two data sets used are (a) a set of samples of chemical composition of wines with associated quality rating and (b) a data set containing samples with attributes categorizing whether a patient is type 2 diabetic or not.

2.1 Wine Quality Data Set

Although, wine preservation has been around since the late 1970s, there has been a recent surge in popularity of automated wine preservation systems in bars and restaurants. One name that comes to mind is 'Enomatic'. It is understandable that these machines are only as smart as the people who operate them but if machine learning is implemented, intelligent preservation systems can be built to classify different wines based on quality, along with their preservation. How great a wine is, depends on combined opinion of a group of experienced sommeliers.

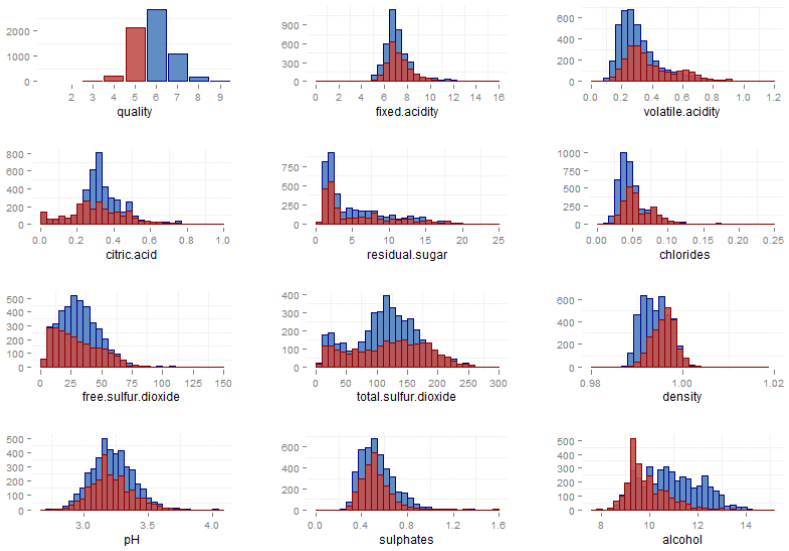

Figure 1: Attribute distribution of wine data set [Blue : high quality], [Red : low quality]

To accomplish this task of creating an intelligent system the data set I have used is part of the UCI machine learning repository. It is a combination of red and white 'Vinho Verde' wine samples taken from a study performed at the University of Minho in Portugal, contains 6497 instances and 12 attributes, with the classifying attribute 'quality' scoring between 0 and 10 (10 being the best). For the purpose of this classification task, I pre-processed the quality rating into 2 categories (using MS Excel), either being of low quality (rating 5 and below) or of high quality (rating of 6 and above).

As per the plot created using R above in Figure 1, blue represents 'high quality' while red represents low. At first glance, it is evident that samples with high alcohol content tend to have higher quality. Sulfate content and pH seem to be evenly distributed over the samples. In later sections we will explore in detail how this data fares against different learning algorithms.

2.2 Diabetes Data Set

On the contrary, machine learning is already helping medical professionals establish diagnosis to reduce the gap between human intelligence and available data. We can now apply these learning techniques to classify most vulnerable population at risk for diabetes. The escalating rates of type 2 diabetes, are likely due to the changes in the environment, coupled with changes in human behavior and lifestyle. One approach to reduce diabetes has been to study this disease within populations that have limited genetic and environmental variability.

To accomplish this, 2nd data set chosen is also from the UCI machine learning repository, in contrast to the first data set it is smaller including 768 instances and 8 features with the class feature classifying patients as being of class '1', tested positive for diabetes and class '0' with no symptoms of diabetes.

For ease of representation, I have replaced '1' and '0' with textual variables, 'tested positive' and 'tested negative' respectively.

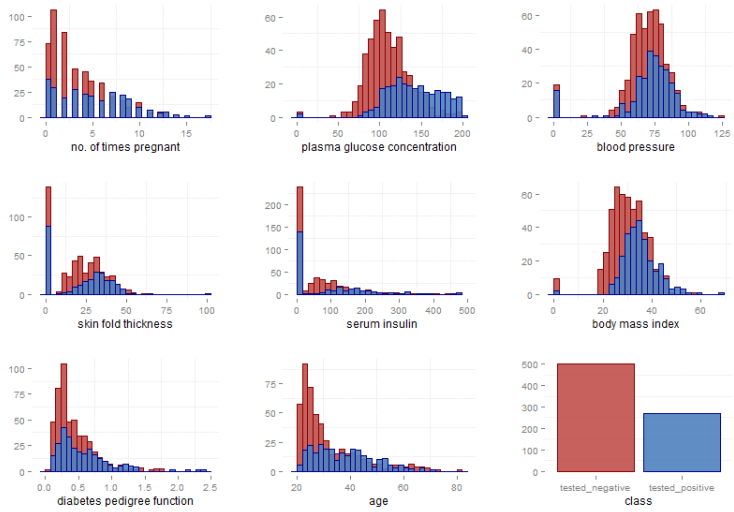

Figure 2: Attribute distribution of diabetes data set [Blue : tested positive], [Red : tested negative]

By looking at the distribution in Figure 2, its seems patients with higher plasma glucose concentration on the oral glucose test have lesser tendency of being diagnosed with type 2 diabetes. Main constraint during the study was that the patients were females with at least 21 years of age. This is also evident in the age histogram where patients of younger age show less symptoms of diabetes.

Now let us dive in to the algorithmic analysis of the 2 data sets using different supervised learning techniques.

3 Algorithmic Implementation

Weka 3.7.12, Explorer and CLI modules were used to implement all of the five algorithms. For the purpose of the classification task, the diabetes data set was used in its original form except the 'class' attribute values '1' and '0' were replaced by 'tested positive' and 'tested negative' respectively. On the other hand, the quality rating for the wine data set was converted in to 2 categories of high and low quality ratings as mentioned above in the overview.

Most of the algorithms were run with 10 fold cross validation, however, there were exceptions where 5 fold cross validation was used which is discussed in later sections. The reason for deploying cross validation was to efficiently utilize the available data (especially for the smaller data set) in order to try and build models that balance out bias/variance.

Clock times for the algorithms used in the analysis are generally average of 5 iterations for each configuration, however, there were exceptions which are mentioned in section 4. For consistent results random seed was kept constant at 1 in Weka. For analysis in this report, a particular data set is addressed by Wines $\gg$ and Diabetes $\gg$.

3.1 Decision Trees (with pruning)

J48, the decision tree algorithm used here is the C4.8 algorithm in Java ("J" for Java, 48 for C4.8, hence J48) and is an extension to the famous C4.5 algorithm. Here the algorithm was used with pruning - a useful technique to trim of some tree branches in order to avoid over-fitting the training data. J48 allows few ways to prune the tree, one of which is varying the confidence factor. Lowering the confidence factor helps induce more pruning but may decreases accuracy on training samples, which was counter-balanced here by using cross-validation (k=10). The relation of confidence factor, tree size and variation in accuracy is shown below.

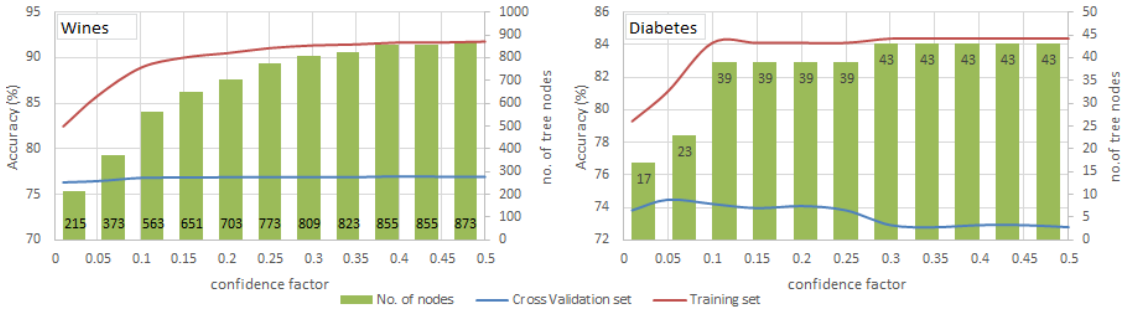

Figure 3: tree size and accuracy with respect to confidence factor [LEFT : Wines], [RIGHT : Diabetes]

3.1.1 Identifying relationship between confidence factor and tree size

Figure 3 shows, for both data sets, lower confidence factor values considerably reduce the tree size while compromising training set accuracy. Key is to strike a balance between the two, i.e. find a trade-off, which I tried to find by comparing 2 contrasting values for confidence factor[0.05, 0.25]. Value of 0.05 was chosen as it showed reduction in tree size without deteriorating the training accuracy too much. In contrast, 0.25 was chosen as both accuracy and tree size have reached a steady state at this point. Following learning curves in Figure 4 and Figure 6 will show the comparison and highlight over-fitting if there is any.

3.1.2 Contrasting confidence values factor to see over-fitting

Wines $\gg$ Learning curves with accuracy in Figure 4, for wine data set, generated at confidence factor(CF) of 0.05 do a better job to reduce over-fitting as training set size increases. In terms

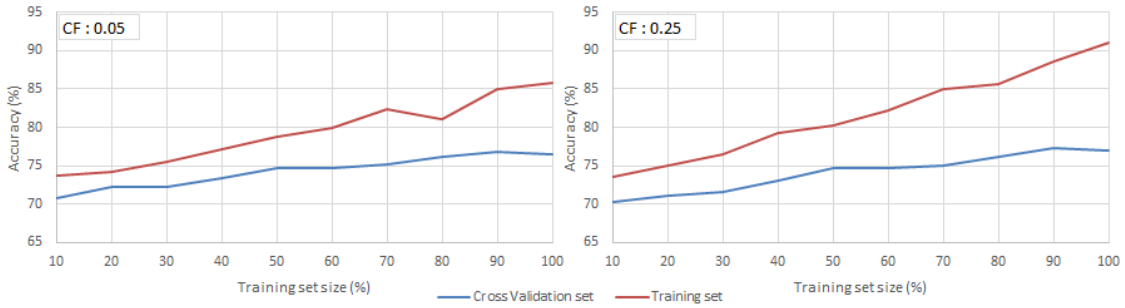

Figure 4: [Wine data set] accuracy w.r.t training set size keeping confidence factor(CF) constant

of clock times, the difference was less then 0.02 s (insignificant really) with CF of 0.05 being slightly quicker and for accuracy both configurations yielded $\approx 77\%$. Out of the two models, the one with confidence factor of 0.05 is clearly better as it reduced over-fitting and also produced a tree with half as many nodes, i.e. 373 vs. 773 for CF of 0.25. The complexity/size of the tree is shown in Figure 5 and root node is alcohol (not visible here though) as evident from the overview in section 2.

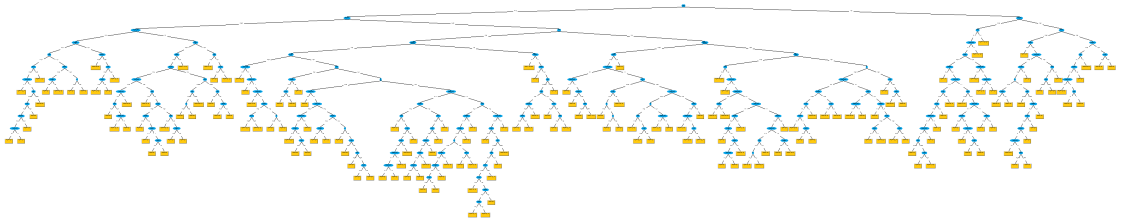

Figure 5: [Wine data set] pruned-decision tree with CF : 0.05

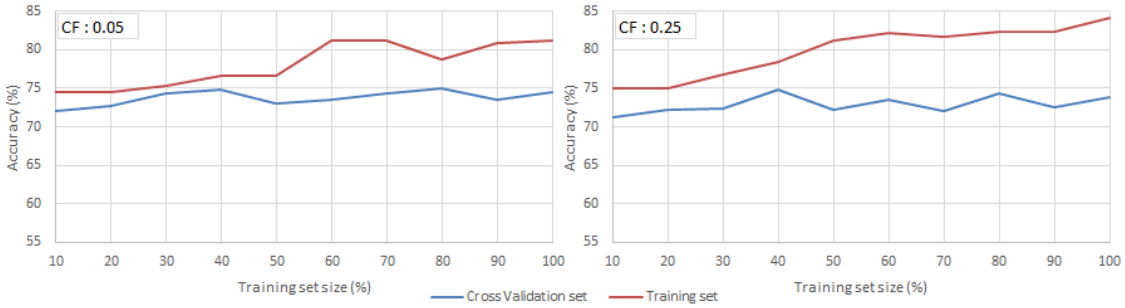

Figure 6: [Diabetes data set] accuracy w.r.t training set size keeping confidence factor(CF) constant

Diabetes ≫ This data set shows similar results for the two learning curves in figure 6 above, with configuration using confidence factor of 0.05 showing less over-fitting compared to the one with confidence factor of 0.25.

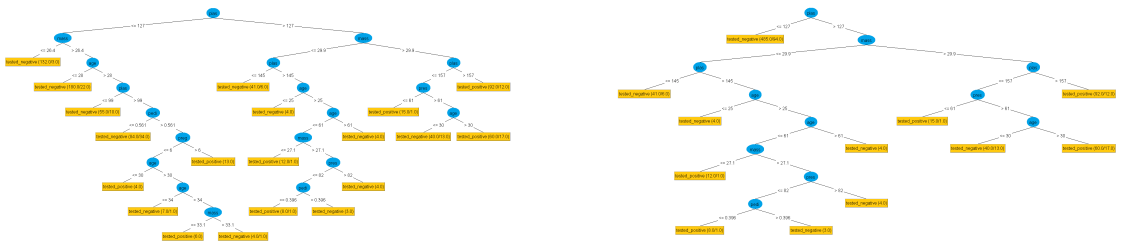

Figure 7: [Diabetes data set] (LEFT) Un-pruned tree, (RIGHT) pruned tree (CF : 0.05)

Accuracy for the CF : 0.05 configuration is slightly better than CF : 0.25, being 74.5% and 73.8% respectively. This makes configuration with CF : 0.05 a better option due its smaller/less

complex tree and a better fit on the validation set with reduced over-fitting. Figure 7 shows how pruning helped reduce the tree size by eliminating 5 nodes and 'plasma' was chosen as the root node, which I already highlighted in the overview section.

3.2 Neural Networks

Neural networks algorithm used for analysis was the 'Multilayer Perceptron' classifying model in Weka. Multilayer perceptron uses sigmoid function and few of the parameters that can be tweaked in Weka for variation in performance of the neural network are: hidden layers (H): no. of hidden layer in the network; learning rate (L) : amount with which weights are updated; momentum (M) : momentum applied to weights to minimize error, and training (N) : no. of iterations (epochs) to train the network.

3.2.1 Identifying the least error configuration

To choose the right configuration for the model, for both data sets L and M were kept constant (at first) at 0.1 each. While increasing no. of iterations (epochs), root mean square error (RMSE) vs. epochs for the cross-validation set was plotted for different combinations of hidden layers. This is shown in Figure 8 below with the naming convention is as: (eg: 2L : 6,8 interpreted as 2 hidden layers with 6 nodes in 1st layer and 8 nodes in 2nd layer).

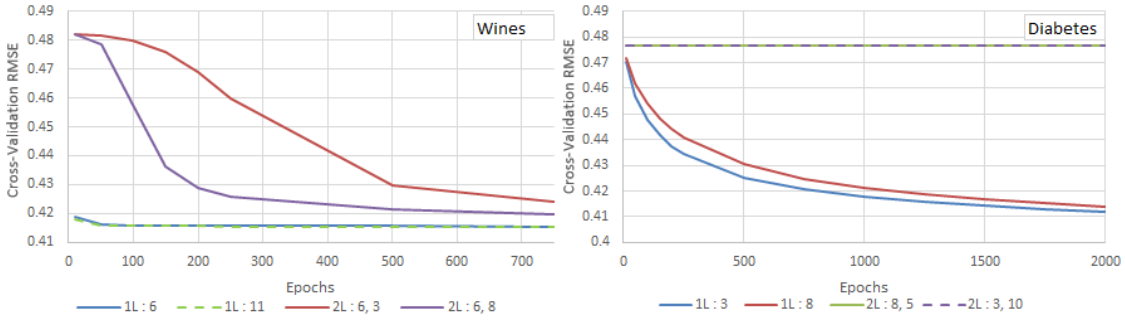

Figure 8: RMSE w.r.t. epochs for different hidden layers [LEFT : Wines], [RIGHT : Diabetes]

Wines ≫ RMSE plot with 1 layer and 6 nodes (blue solid curve) was almost identical to the one with 1 layer and 11 nodes (green dashed curve) [super imposed on each other]. Increasing the complexity of the network by adding an additional layer (2L : 6, 3 and 2L : 6, 8) resulted in bad RMSE for smaller no. of iterations while it was considerably reduced for higher number of iterations. I continued to plot 750 epochs with increments of 50 and of out these 4 plots, I selected 1L : 6 as network of choice because of its lesser complexity and lowest RMSE over 750 iterations. I did try to use more than 2 hidden layer but the error was quite high and it was taking a lot of time to train so I decided to stick to a simpler but effective approach (Ockham's razor).

Diabetes ≫ Same was observed for the diabetes set, single layer network showed considerable reduction in RMSE while addition of 2 hidden layer showed no improvement with increasing epochs (flat error line for networks with 2 hidden layers). For this data set I continued till 2000 epochs and decided on 1L : 3 to be the network of choice, i.e. single layer with 3 nodes.

3.2.2 Putting learning rate and momentum into the equation

Once hidden layer for the network was chosen, I tried different combinations of learning rate (L) and momentum (M) to see if I could improve the accuracy to build a better model.

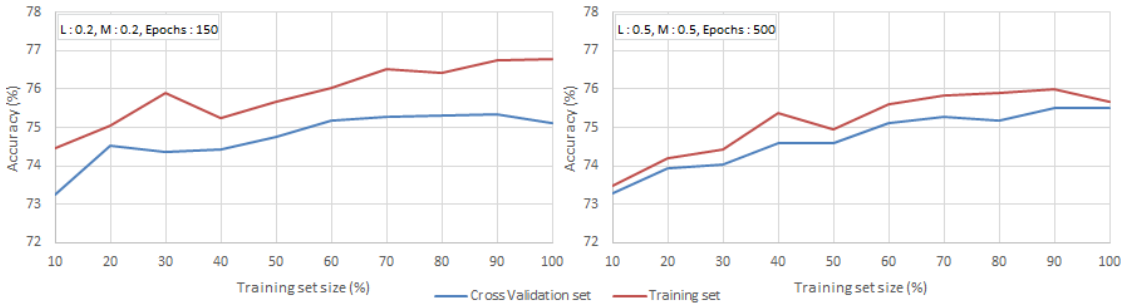

Figure 9: [Wine data set] accuracy w.r.t. training set size

Wines ≫ Variations in L & M were applied with equal values of L and M as L = M = 0.2 & L = M = 0.5. I observed improvement in accuracy by 1.5% for L = M = 0.5 and decided to plot learning curves for these 2 contrasting [L,M] pairs to view over-fitting.

Figure 9, plot on the right shows learning curve for L = M = 0.5 and 500 epochs while the one on left shows L = M = 0.2 with 150 epochs. It can be seen clearly that for plot on the right the cross-validation set generalizes well whereas the one on the left shows slight over-fitting. So for wine data set, model that I chose was: single hidden layer, 6 inner nodes, learning rate of 0.5, momentum of 0.5 with 500 iterations.

Diabetes ≫ For this data set, I took a different approach as it was smaller in size and I did not want to use higher values of L and M to make the model more complex so I tried 3 combinations of L and M to observe any reduction in RMSE.

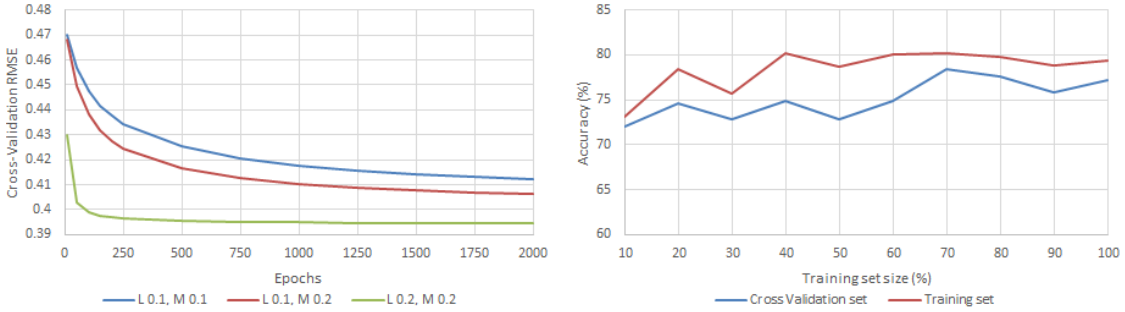

Figure 10: [Diabetes data set] [LEFT] RMSE w.r.t Epochs, [RIGHT] accuracy w.r.t training set size

Figure 10 [LEFT] shows the different combinations of L and M I tried. Cross-validation (CV) RMSE curve for L and M values (0.2 each) up till 2000 iterations showed considerable reduction in error. I chose this combination with the lowest error in order to view the performance. Figure 10 [RIGHT] shows the learning curve for this combination. Although, for medium sized training set the performance was a little jagged but accuracy is in high 70s and cross-validation set seems to generalize well with increasing training set size. For this data set, the following configuration gave me the best results: single hidden layer, 3 inner nodes, learning rate of 0.2, momentum of 0.2 and 500 epochs. I thought of using higher values for L = M ¿ 0.3 just like I did for the wine data set but using higher values for a smaller data set might cause the model to over-fit.

3.3 Boosting

Boosting was performed using the implementation of adaptive boosting ('AdaBoostM1') in Weka. J48 was used as the classifier but this time with an aggressive approach. Along with modifying confidence factor (CF) that aids in post pruning I also tried different combinations of 'minNumObj' (m) : the minimum number of instances per leaf. Setting this parameter to higher value like 3, 4 helps stop further splitting . Default is set to 2 in Weka, this was used in section 3.3 when analyzing decision trees.

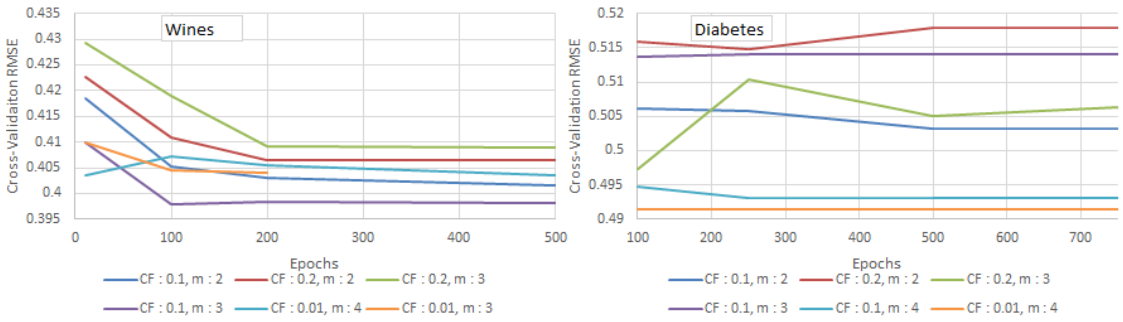

Figure 11: RMSE w.r.t Epochs with combinations of confidence factor and minimum branches per leaf

3.3.1 Quick analysis to identify the least error configuration

Wines ≫ As shown in Figure 11 [LEFT] I plotted cross-validation RMSE vs. epochs for few combinations of confidence factor and minNumobj. The lowest error curve I got was the one with CF : 0.1 and m : 3. I also tried using m = 5 on 2 different machines but Weka shutdown due to memory issues and I could only record RMSE for up to 200 iterations. So the maximum I could go for 500 iterations was m : 4 and least I could go for CF was 0.01.

Diabetes ≫ The RMSE plot for this data set is shown in Figure 11 [RIGHT] showed varied responses for different combinations and iterations but the most stable RMSE with the least value was the one with confidence factor of 0.01 and minNumObj of 4.

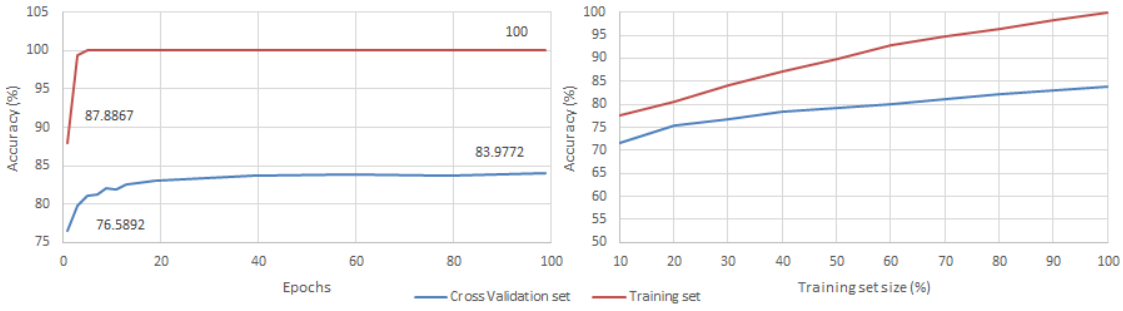

Figure 12: [Wine data set] [LEFT : accuracy w.r.t. epochs], [RIGHT : accuracy w.r.t. training set size]

3.3.2 Detailed iterative analysis and learning curves

Wines ≫ After the quick analysis, I did a detailed iterative analysis on the least error configuration (CF : 0.1, m : 3). For some reason, model showed improvement in classifying the instances for odd epochs so I created an accuracy plot in Figure 12 [LEFT] going from epochs : 1 to 15 with intervals of 2 and then from 19 to 99 with interval of 20. I got the best performance at 99 iterations even though training accuracy was at 100% long before that. The plot on Figure 12 [RIGHT] shows the learning curve with slight over-fitting for the best performing combination (CF : 0.1, m : 3, iterations : 99). The best performing algorithm for the wine data set was boosting. This will be further highlighted in the section 4 later.

Diabetes ≫ For the least error configuration of (CF : 0.01, m : 4) I plotted a detailed accuracy plot w.r.t. to no. of iterations from 1 to 25 and the best performance for the cross-validation set came with 8 iterations. This is shown in Figure 13 [LEFT] while Figure 13 [RIGHT] shows the learning curve for this best performing configuration i.e. ((CF : 0.1, m : 3, iterations : 8). This model is over-fitting as evident by the plot showing continuous increase in training set accuracy without any prominent increase in the cross-validation set accuracy.

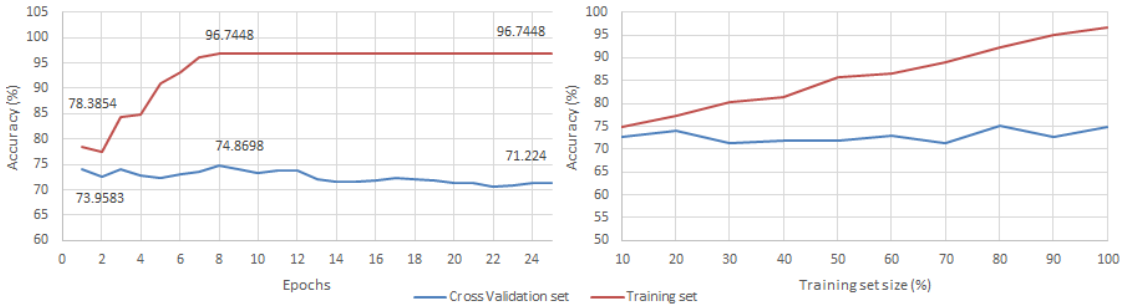

Figure 13: [Diabetes data set] [LEFT : accuracy w.r.t. epochs], [RIGHT : accuracy w.r.t. training set size]

3.4 Support Vector Machines

SVMs used are John Platt's sequential minimal optimization algorithm (SMO) implementation in Weka. The key parameter for SVMs is the kernel trick for which Weka provides few options. I explored SVMs using linear, polynomial with exponent 2 and radial basis function (RBF).

NOTE: Only for Wine data set, except for linear kernel, every other configuration was run using 5 fold cross-validation as it was taking a lot longer to perform 10 fold cross-validation.

3.4.1 Identifying the most consistent kernel

Linear kernel was setup using polykernel in SMO by putting the exponent as 1 and for the polynomial kernel I used exponent 2. For the RBF kernel, to control the bumpiness of the peaks in the hyperplane I explored 2 values of gamma (0.1 and 0.01) . I plotted RMSE curves for different kernel configurations with respect to complexity factor C with steps of 10 [10 to 100]. The plots for both data sets are shown below in Figure 14.

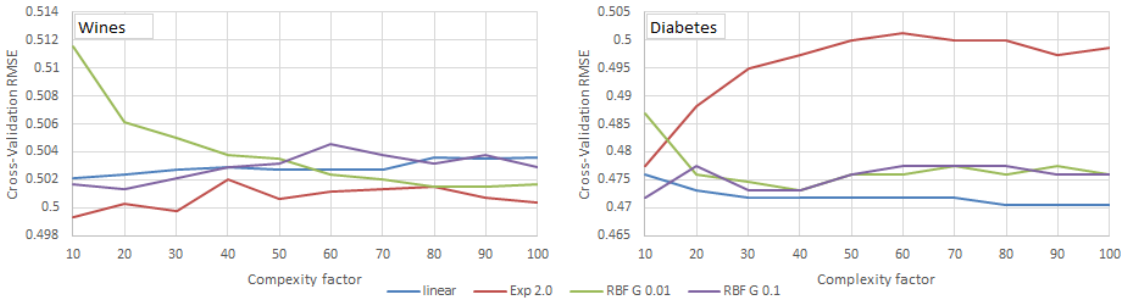

Figure 14: RMSE w.r.t complexity with combinations of different kernels [LEFT : Wines], [RIGHT: Diabetes]

Wines ≫ The least error I got was with using the polynomial kernel (exponent 2.0) configuration for wine data set. The worst performing configuration was with the RBF kernel (gamma = 0.01), especially for lower values of complexity C.

Diabetes ≫ In contrast to the wine data set, polynomial kernel (exp : 2.0) was the worst performing kernel in terms of RMSE values. Linear kernel consistently performed better compared to the other configurations.

3.4.2 Plotting learning curves with different configurations

For both data sets, I plotted learning curves using 2 contrasting configurations to highlight variations.

Wines ≫ For the wine data set, I plotted 2 curves for the polynomial kernel configuration using 2 different values of complexity factors C [10 and 100] to see the variation. Figure 15 [LEFT] shows that with higher complexity there is indication of over-fitting while Figure 15[RIGHT]

shows that the model with less complexity is a better fit. So the better performing SVM model for wine data set was polynomial kernel with complexity 10.

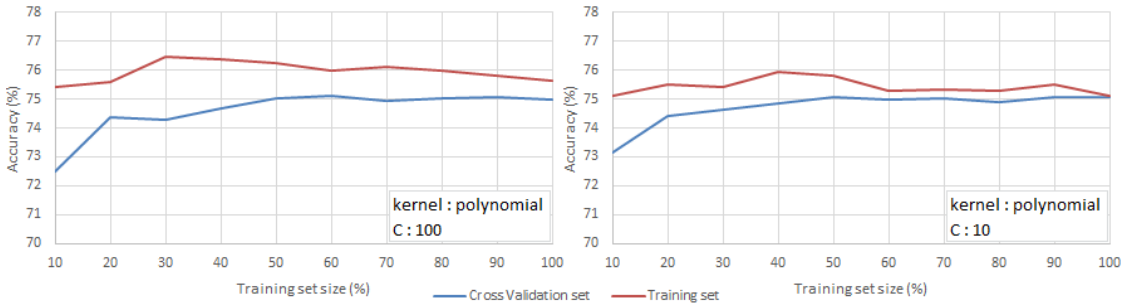

Figure 15: [Wine data set] accuracy w.r.t. training set size [LEFT : poly kernel with C: 100], [RIGHT : poly kernel with C : 10]

Diabetes ≫ In contrast, for diabetes data set I compared 2 different kernels with same complexity factor. Looking at the diabetes RMSE plot in Figure 14 [RIGHT] I chose to compare linear and the RBF kernel with gamma (G : 0.1) both at complexity (C : 100) as the error curves for both of these configuration were very similar. In figure 16 [LEFT] shows that the model is over-fitting for the parameters chosen while the plot in Figure 16 [RIGHT] shows that linear model works well for the diabetes data set. This algorithm gave the best performance with linear kernel and complexity C : 100. This was the best performing algorithm for the diabetes data set and will have detailed analysis in section 4.

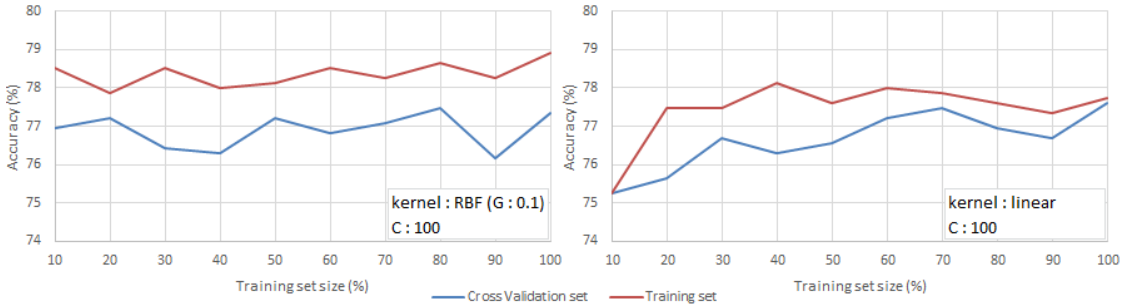

Figure 16: [Diabetes data set] accuracy w.r.t. training set size [LEFT : RBF kernel with G: 0.1], [RIGHT : linear kernel]

3.5 K-Nearest Neighbors

The implementation of the k-nearest neighbors used for analysis was IBk in Weka. Choosing the optimal k is almost impossible for a variety of problems, as the performance of a k-NN classifier varies significantly when k is changed as well as when distance metric is changed. For both data sets, to identify the value of k and the distance metric where the algorithm performed the best, I plotted cross-validation set accuracy w.r.t. 9 different values of k (2 to 10) for 2 different distance methods (euclidean and manhattan).

3.5.1 Best performing k

Wines ≫ Accuracy plot in Figure 17 [LEFT] shows how cross-validation set accuracy varies with changes values of k. For the 2 different distance metrics considered, the best performance came for k : 9 while second best was k : 5.

Diabetes ≫ On the other hand, for diabetes data set best performance came for k : 7. For this value of k , while accuracy for euclidean distance showed prominent increase, accuracy for manhattan distance was more flatter in comparison.

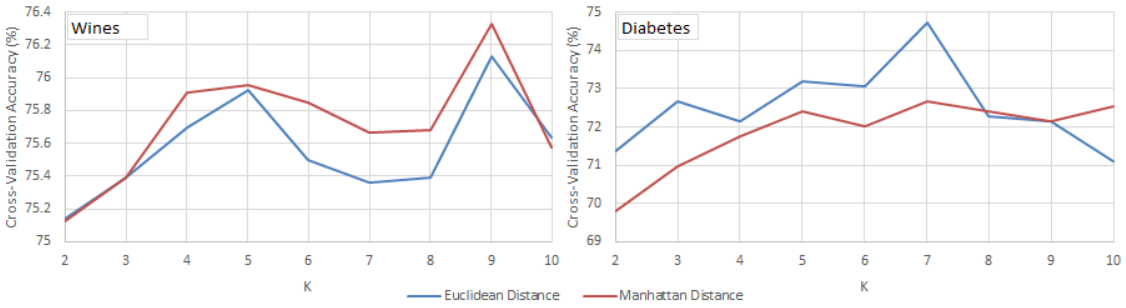

Figure 17: cross-validation set accuracy w.r.t. K [LEFT : Wines], [RIGHT : Diabetes]

3.5.2 Learning curves for best performing k

Wines ≫ Figure 18 shows the learning curves for k : 9 and the different distance metrics. By looking at the curves, there is not a lot to choose from. The accuracy at 76.1% and 76.3% is almost same and extent of over-fitting is also consistent. Even the model build and train times were almost same. However, other performance metrics such as precision and recall using manhattan distance came out to be slightly better than the one for euclidean distance so better choice for the model was k : 9 with distance metric : manhattan.

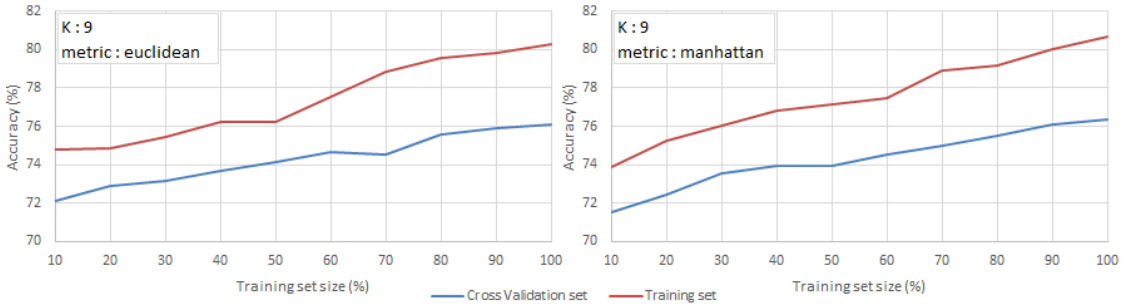

Figure 18: [Wines data set] accuracy w.r.t. training set size [LEFT : Euclidean], [RIGHT : Manhattan]

Diabetes ≫ Figure 19 shows the 2 learning curves for the different distance metrics tested with k : 7. Again, in this case too, there is not a lot to choose in terms of accuracy for smaller training set size but 100% of the training set was used there was definite improvement in cross-validation accuracy. For the model with distance metric as euclidean at 74.7% compared was better than the one for manhattan which was under 73%.

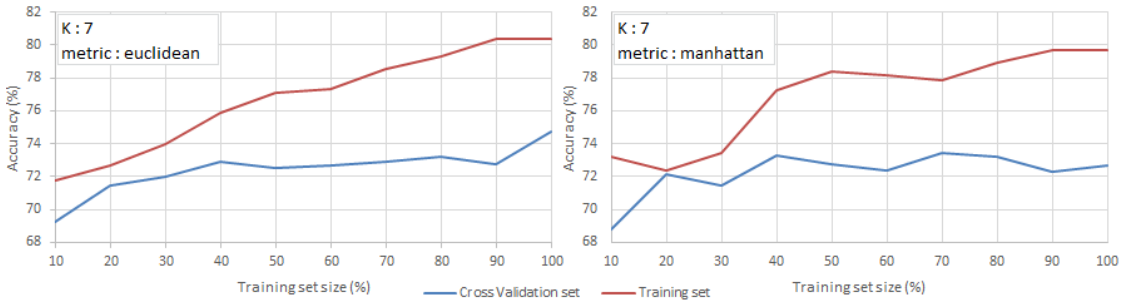

Figure 19: [Diabetes data set] accuracy w.r.t. training set size [LEFT : Euclidean], [RIGHT : Manhattan]

4 Comparison

Up till now we have been discussing performance of algorithms mainly based on their accuracy and root mean square error to predict a class. This section takes in to account few additional performance metrics that are critical towards analyzing a machine learning algorithm.

NOTE : Most of the times noted were average of 5 separate runs. And for analysis purpose times displayed in plots below are summation of both build and train times. In the case of boosting and support vector machines, for higher percentages of training set size, time for only 1 run was noted as it was quite time consuming to run 5 iterations typically taking more than 1 hour.

4.1 Wines data set (Performance and Time Analysis)

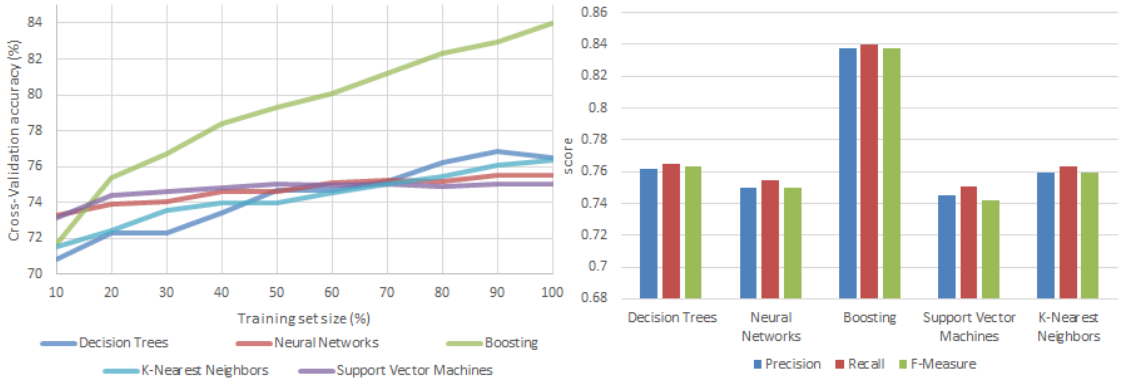

Figure 20: [Wines data set] [LEFT : accuracy w.r.t. training set size], [RIGHT : Performance metrics]

Figure 20 [LEFT] was plotted using the cross-validation set accuracy for the best performing models for each algorithm discussed in the section 3. This plot shows that best performing algorithm was boosting with pruned decision trees while all of the other 4 algorithms had similar performances. This comparison is also evident from the precision, recall and F-measure bar graphs for the complete data set in Figure 20 [RIGHT] and clearly boosting is by far the best performing algorithm.

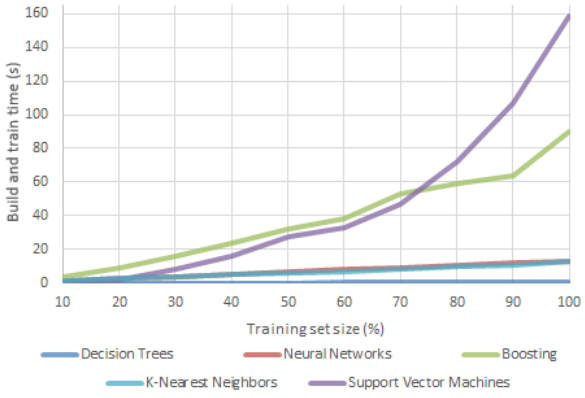

Figure 21: [Wines data set] build and test time w.r.t. training set size

Figure 21 shows the build and training times of these algorithms. Two of the slowest algorithms were supports vector machines and boosting. In my opinion the loss in time we see for boosting is compensated by gain in performance mentioned above. So for the wine data set the best performing algorithm is boosting.

4.2 Diabetes data set (Performance and Time Analysis)

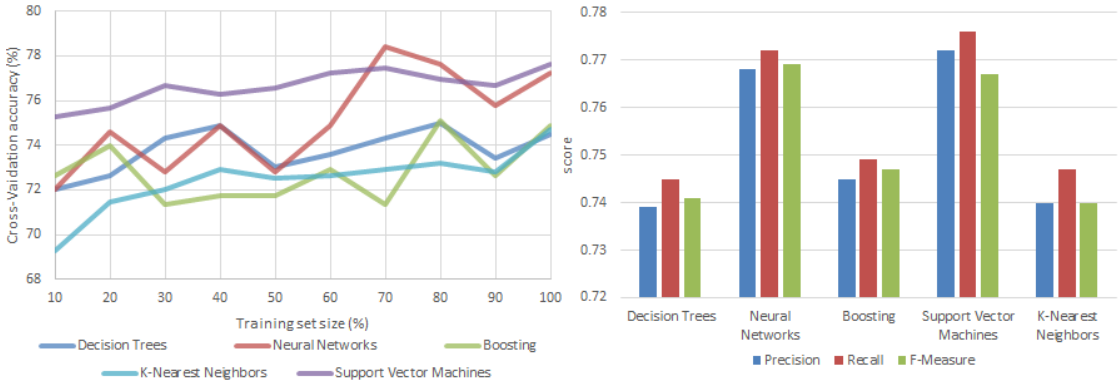

Figure 22: [Diabetes data set] [LEFT : accuracy w.r.t. training set size], [RIGHT : Performance metrics]

For the diabetes data set things do not seem to be as straight forward. In Figure 22 [LEFT], all five algorithms gave similar performance in terms of cross-validation accuracy set with neural networks and support vector machines being the top two algorithms for larger percentages of the training set. This can also be seen in the precision, recall and F-measure bar plot in Figure 22 [RIGHT] with very slight advantage to support vector machines.

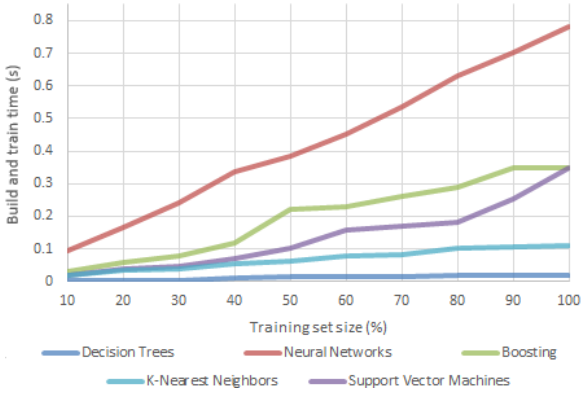

Figure 23: [Diabetes data set] build and test time w.r.t. training set size

Figure 23 gives a clear picture in terms of time taken to build and train the algorithm. Although, neural networks performed very well for this data set but took a lot of time. This tilted my vote in favor of support vector machines. With slightly better performance compared to neural networks in terms of accuracy, precision, recall and taking almost half the time, support vector machines was the best performing algorithm for the diabetes data set.

5 Conclusion

Overall this assignment was an amalgamation of a lot of aspects of supervised learning, ranging from *overview of the classification problems, model complexity* to *learning curves for better performing models, techniques such as cross-validation to avoid over-fitting and performance analysis of the algorithms* based on metrics such as precision, recall and F-measure along with accuracy.

The analysis performed in this report is one of the many ways to compare algorithms. In majority of the cases I used RMSE curves and in some I used accuracy for cross-validation set in order to get a quick idea about the direction I should take in order to improve the performance of an algorithm while working on my classification problems.