Author: Apurva Sharma CS7641
GTID: 902490301

Assignment 1 : Supervised Learning Techniques

Datasets Description and Motivation:

Soybean:

This dataset is collected from a survey of soybean diseases from an inspection of 683 instances and 36 (35 for description and 1 for class) attributes are used to describe each instance. Based on the symptoms, the plant is classified to have one out of the nineteen possible diseases. The data set contains some missing values.

SpamBase:

This is a dataset consisting of 4601 instances and (58) 57 + 1 class attributes with all of them continuous except the class label which is nominal. As opposed to Soybean, this dataset is complete. A majority of the columns are dedicated to indicate the frequency of select keywords/characters encountered mails. Attributes 55-57 measure run lengths of consecutive capitals. Based on this information, a mail can be marked either as spam (1) or non-spam (0).

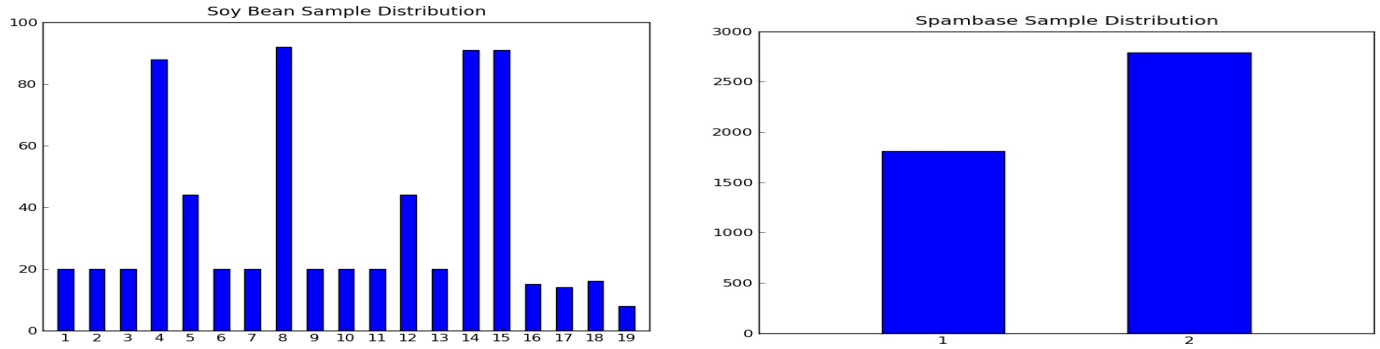


Figure 1. Sample Distributions for Soybean and Spambase datasets. The x-axis signifies the classification number while the y-axis signifies the number of samples belonging to the class.

What makes these datasets interesting:

Domain Point of View :

Soybean:

"Soybeans continue to occupy large acreages of land in the southern United States. Soybean acreage for 1987 was 19,000,000 acres. However soybean production has its problems, the most serious being diseases. Recognition and control of these diseases is vital. In 1987, the average loss from all pathogens was 12.8 percent, or 75.3 million bushels. At the average price of $5.50 per bushel, this equals a 414.3 million dollar loss!! "
(Source :http://cipm.ncsu.edu/ent/ssdw/soyatlas.htm)

The above statistic is staggering and provides ample motivation to try and identify diseases in soy plants on basis of observed symptoms. Once a disease is detected early, remedial actions like crop rotation, application of relevant fungicides, isolation etc. can be performed.

Spambase:

Spam is more than just a nuisance. Everybody gets it. Its a notorious eater of resources : right from time to network bandwidth. It accounts for about 45% of all email traffic. According to a study by the Radicati Research Group Inc., a research firm based in Palo Alto, California, spam costs businesses $20.5 billion annually in decreased productivity as well as in technical expenses. Nucleus Research estimates that the average loss per employee annually because of spam is approximately $1934. (Source : http://www.spamlaws.com/spam-stats.html). This is what prompted the need for spam filters in the first place. I want to see for first hand, how these machine learning techniques compare to clumsier methods like blacklists (eg. MAPS-RBL).

Machine Learning Point of View:

So, one of my key motivations was to pick up data sets which could differ from each other as much as possible. Factors in consideration included number of instances, number of attributes, missing values vs complete datasets, nominal attributes vs continuous real values, binary classification vs multi-classes. After a sufficiently thorough look up, I decided on these two data sets as they are different in all these aspects. Spambase interested me for one more reason. I have worked with spam before where I designed a naïve Baye's classifier for separating spam from non-spam. I really would like to see how these techniques compare to my previous approaches. One of the main motivations for picking up this dataset is that the classification can be fairly intuitive. After all, manually all of us can classify our mailboxes with 100% accuracy!

This, I think would make interpreting the results a lot more accurate, as well as let me play around with observing

results by dropping columns (since not all the columns are equally important, especially columns based on frequencies of particular words). In this case, it would be interesting to see which are the keywords which really matter in distinguishing spam from nonspam.
With respect to Soybean, I wanted to see how these algorithms perform when there are relatively fewer instances (683) with missing values and a fairly reasonable attribute count (36). What makes it even more challenging is the fact that there are 19 possible classifications. Add to this that for some classifications like herbicide injury, there are as few as 8 samples.
All these factors make these two datasets a fairly good choice for trying out the different algorithms.
With this in mind, we are all set to start with the experiments!

Decision Trees:
Decision trees try to create rules for classification on basis of observation of the training data. One of the key points in forming the tree is deciding which which attributes should be considered first. I looked up the source code of J48 algorithm in Weka to find out that it internally uses **information gain** to select which attributes move higher up the tree. I explore two post pruning methods : **subtree raising** and **reduced error pruning**. Both first grow the decision tree and then prune to reduce the height of the trees.
While subtree raising raises nodes upwards towards the roots, reduced error pruning prunes the subtree, replacing it by the leaf nodes with the majority and tests the accuracy. If the accuracy does not suffer, the prune operation is retained, else we switch back to the original tree.

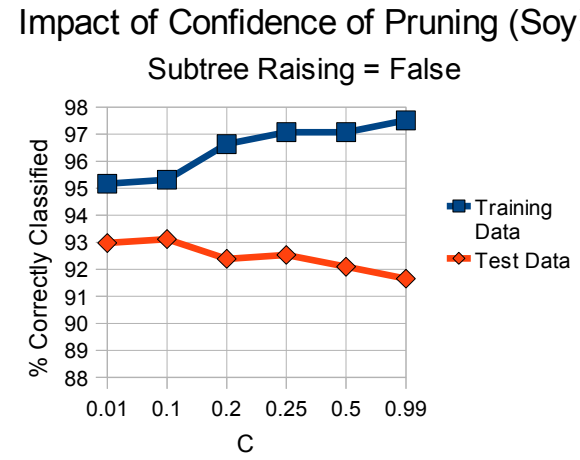
Intuitions before experiment:
1. It will be interesting to see the impact of the aggressiveness of pruning on the decision trees. Keeping occam's razor in mind that the simplest explanation, though not simpler than required yields the best results, I think the pruned trees will perform better than unpruned trees for the test data. However, very aggressive pruning may lead to over simplification of the decision tree, causing the performance to go down.
2. I also want to compare the different types of pruning techniques: mainly subtree raising and reduced error pruning. From whatever literature I have read about decision trees, subtree raising was not particularly recommended since it requires a lot more time and its relative success is not yet well established. I want to see how it performs for my datasets.

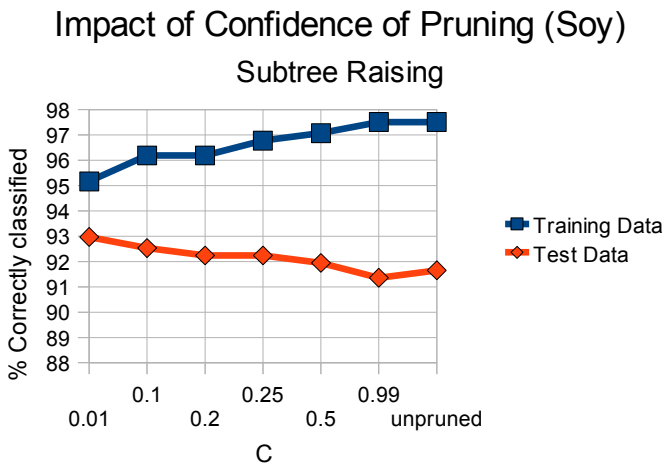
Experiment Setup:
I carry out 2 main experiments (which in turn involve multiple iterations with varying parameters).
1. I carry out pruning with different confidence values for both the data sets. I also see what the results for unpruned trees are.
Also, I see the impact of turning off Subtree raising to see, if It really has a big impact.
2. I also carry out reduced error pruning. Here I vary the number of folds for raising the tree before pruning to see the impact.

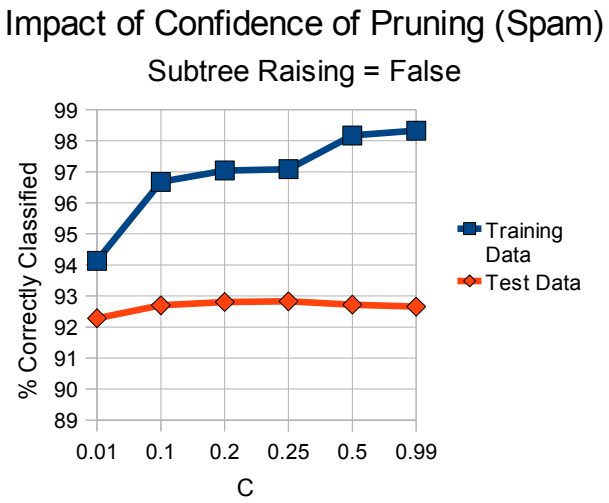
Analysis, Possible Explanations and Lessons Learnt:
Both the datasets give very consistent results. As expected, both the data sets give the best accuracy on the training data set when there is no pruning at all. However, this is hardly our main concern. We wish to see how the accuracy behaves on the test data set.
Accuracies are relatively lower for unpruned trees on test data. This is possibly because the tree is overfitting the training data.
As we reduce the values for C (i.e. prune more aggressively), the test data accuracy improves, but only to a point. After a point, the tree starts generalizing to a higher degrees than wanted, thereby reducing the accuracy.
The case for Soy data set is more interesting because the performance almost seems to increase monotonically as the pruning aggression increases (i.e. C decreases). This may be because of the fact that there are some key attributes which are giving a lot of information gain. They are placed much higher on the decision tree and are hence, less likely to be pruned. This also implies that there may be many attributes which are not rich sources of information, and pruning them is not having a negative impact on the accuracy of the tree.
The impact of subtree raising is also interesting, I note that subtree raising, in fact, does improve the accuracy of classification. This difference is however not very significant (in most cases <1%) and in fact, there are instances where not raising the subtree has better results!
But even more surprising is the fact that in spite of its criticism, the best result obtained by Subtree raising is much better than the best result of Reduced error pruning (at least according to my readings). This leaves us with very little to choose from the two techniques.
Another trend worth observing the reduce error pruning is that the number of folds that allow the growing of the trees has an impact on the accuracy. The trends indicate that as N increases, accuracy increases upto a point before starting to dip again. Logically, this makes sense. I explain this as : When we give really low values of N, we are not letting the tree grow enough before already starting pruning on it. This can lead to loss of information rich attributes, leading to a lesser efficiency. Similarly, if we give high values of N, we let the tree grow much more than needed. Then pruning removes attributes with less information, but some such attributes still remain, as the tree has grown

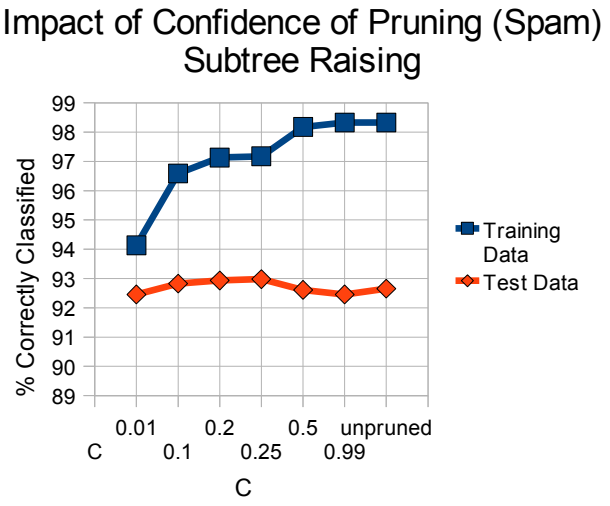
considerably big. This can lead to modeling of noise, and hence overfit, thereby causing a drop in the accuracy on the test data.

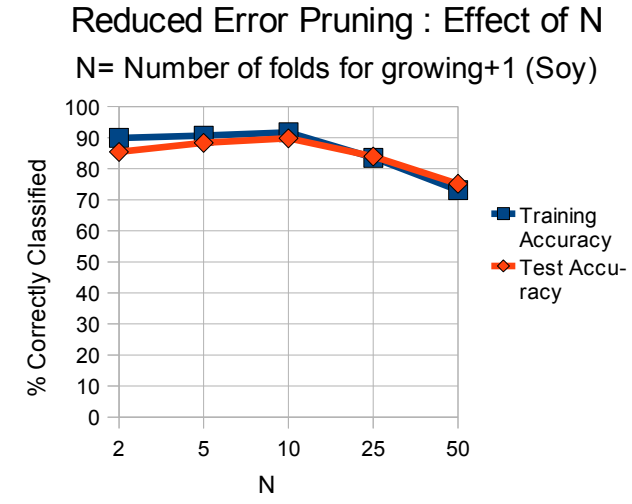
Figure 2 : Impact of Pruning on Accuracy











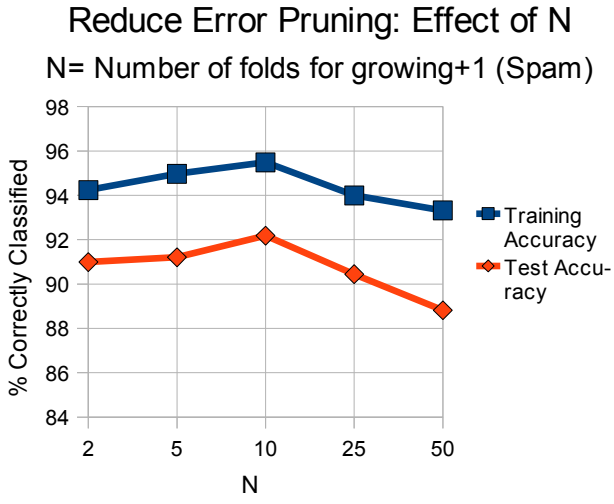


Figure 3: Impact of number of folds for raising tree on Accuracy in Reduced Error Pruning

Another observation I will carry out on the spam dataset for these experiments is the area under the ROC curve and the False positive rate for a non-spam mail being marked as spam. The reason I do it for this case is that this is a crucial measure for the effectiveness of a filter. A filter with a very high accuracy, but occasionally marking an important

mail as spam is not as desirable as as filter with lesser accuracy, but a lower false positives rate. Crux being, spam in inbox is tolerable, but important message in spam folder is not!

In the best case scenario for decision trees, I got a False Positive value (FP) for 0 (non spam) as 0.048 i.e. 4.8% of the non spam messages wrongly get marked as spam. How other techniques influence this parameter will be interesting to see.This value would be lesser if there were more training instances for non spam mails.

I do not do this analysis for soybean dataset as there are 19 possible classes, and I do not possess the domain knowledge to consider which diseases are more crucial.

Neural Nets :
I used the MultilayerPerceptron function in Weka to carry out the experiments. Some of the key parameters to consider while carrying out experiments on Neural Nets are:

1. **Number of Hidden Layers:** Values can be numbers as well as wildcards. 'a' = (attribs + classes) / 2, 'i' = attribs, 'o' = classes , 't' = attribs + classes.
2. **Momentum:** Interesting feature, Momentums are applied to weights during updating. Higher values sometime help “push” the function over local maxima. But care must be taken as they can push over the global maxima as well.
3. **Learning Rate:** Determines the amount by which weights are updated.
4. **Training time :** The number of epochs to train through.
5. **Decay:** Decay the learning rate as we get closer to the desired output so that we don't run the risk of overshooting beyond the global maximum.

Intuitions before experiment:
1. I expect the performance to get better with an increase in the number of hidden layers, particularly for the soybean dataset, since it has as many as 19 possible classes where odds of needing non linear decision surfaces for classification might be higher. What is a good number of hidden layers, however I don't know as of now. The experiments should give a better picture.
2. I am hoping that I land up in a case where the results get stuck up because of a local maxima. Variations in momentum would then come to the rescue, hopefully leading to a global maximum.
3. I expect to see poor performances for low values of learning rate. The justification for this can be done analogously to the steepest gradient descent algorithm. A really low learning rate forces the weights to be updated very slowly, thus it may not always be possible to reach the optimum value, since the experiment is bound by number of epochs. Similarly, very aggressive learning rates should also not yield the best results as the values may keep overshooting and then undershooting (continue oscillating) the optimal value.
4. The number of epochs should have a positive impact when increased. At least definitely for the training set. For the test set, this value may go down as the algorithm may have over fit.
5. Decay should come into picture especially when aggressive learning rates are set as the weights would update by smaller amounts when closing in on an optimal value.

Experiment Setup:
There are a lot of parameters to be considered while carrying out the experiments. I want to see the impact of all these parameters on the performance of the classification.
1. The core of my experiment is going to consider the impact of learning rates and the number of hidden layers.
2. Once a trend is seen, I will try to analyze the impact of momentum (particularly for cases where the accuracy may be unusually low) to confirm if the system is settling on a local maximum.
3. Also for high learning rates, I am going to use the option of enabling the decay of weights to see if the performance improves.
4. Finally an impact of the number of epochs will be studied.

Analysis, Possible Explanations and Lessons Learnt:
From the Figure above, the impact of Learning rates is very clearly visible. It seems to be the most dominant factor in determining the overall accuracy of the classifier. I deliberately chose very exaggerated values : Learning rates as low as 0.00001 and as high as
0.999. While one may argue that such extreme values may not be practically chosen, but the idea of the experiment was to see how absurdly do the classifiers perform with such values.
For very low values of **learning rate**, the performance is relatively poor. I did not necessarily look at this as a bad thing. I tried increasing values for all other parameters, hoping that though the obvious reason was the low value for the rate, maybe I ran into a local minimum. Thus, I tried increasing the momentum values. I also increased the number of hidden layers as well as the number of epochs. Drastically increasing these parameters did not really change the accuracy, leading me to confirm that the low value of the learning rate, in fact, was the sole reason for such poor performance.

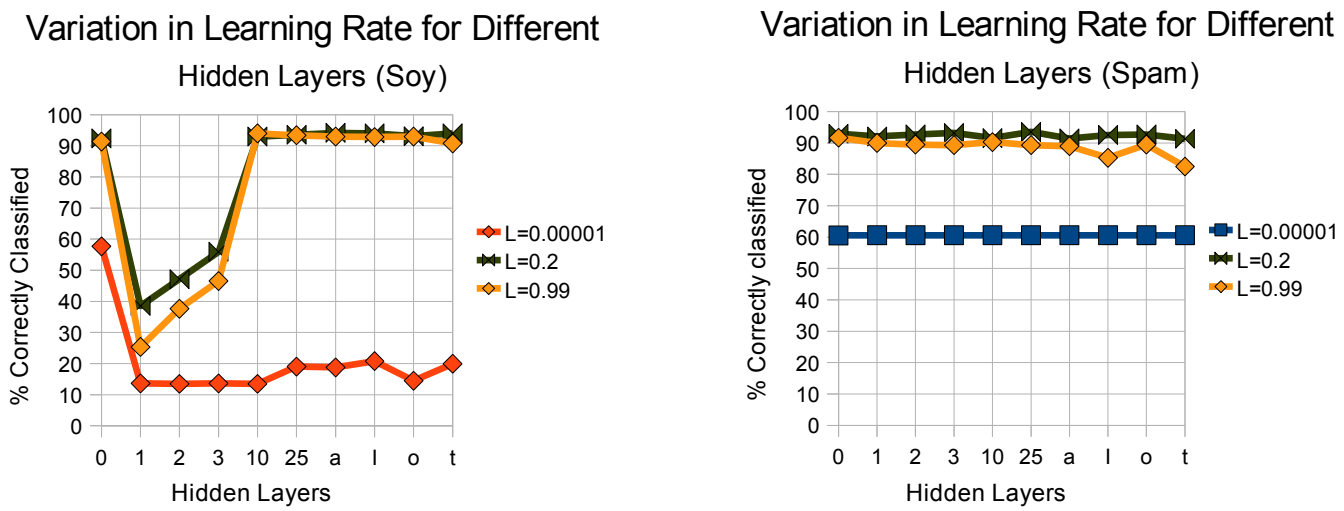


Figure 4: Impact of hidden layers on classification accuracy (for test data) for different learning rates

I was however, anticipating a much more poor performance for a learning rate of 0.999. Though expectedly not as good as accepted values like 0.2, the performance still is not as bad as I thought it would be. This in spite of the fact that no decay of weights was used. The behavior was even more counter intuitive when I enabled the option of **weight decay**. I was hoping to see an improvement in the performance of the classifier because decay makes particularly more sense in cases where the learning rate is initially high. However, doing so only lowered the performance even further. I tried to reason about this in all the ways I could from the view point of the algorithm mechanisms, but could not come up with a concrete reason. The only other possible explanation could be related to the nature of the data sets chosen. Understandably, the performance is relatively worse for the soybean data set. This may be because the number of training instances are much lesser, the number of possible classes is relatively high – 19. Add to this that the dataset has some missing values,thus it is understandable if the results are not as good as the spam dataset.
The number of **hidden layers** definitely has an impact on the accuracy, but not as dominant an impact as I thought there would be. In general, as the number of hidden layers increases, the performance improves. But there is one trend for the soy data set which doesn't make too much sense to me : For all learning rates, the performance with no hidden layers is significantly better than performance with few hidden layers. The performance falls sharply when number of hidden layers is increased to 1 and then slowly starts picking up as the value increases. Also noteworthy is that this is a trend only for the soybean dataset. The spambase dataset shows no such patterns. This means one of the datasets is not following the logical trend, or that the trend itself is supposed to vary for different datasets.
Finally, the **number of epochs** has a very notable and predictable behavior too. I increased the number of epochs to 10,000 for the soybean dataset. As expected, **over-fitting** was observed, as the training set accuracy increased to 99.85% but the test data accuracy dropped to 93.85%.

When talking about neural nets, I just have to mention about the time taken for training. It is significantly higher than any of the algorithms considered. I scripted a batch run of 20 experiments with varying parameters with a 10 fold cross validation which took about 14 hours to train!! The training time depends very heavily on the number of hidden layers and the number of epochs. Here is where a lesson is learnt. If training time is a criteria, then based on my observations, there is not a significant difference (~ 2%) between the accuracies of 'a' = (attribs + classes) / 2, 'i' = attribs, 'and 't' = attribs + classes layers.
If a little bit of compromise on accuracy is acceptable, choose the one that takes the least time (i.e. a or t).
Number of epochs has a great impact on the training time too. Raising the value to 10,000 took me about 6 hours to build the model for a relatively small dataset (soybean).

In the best case scenario for Multi layer perceptron, I got a False Positive value (FP) for 0 (non spam) as 0.099 i.e. 9.9% of the non spam messages wrongly get marked as spam. This is a not a very good number.

Boosting:

I perform Ada Boost (Adaptive Boosting) on the decision trees prepared in the earlier section. Since, we already know that we are supplementing the decision tree with boosting, we can choose to be more aggressive with the pruning of the tree, because we know that boosting will iteratively focus on the wrongly classified instances and improve the performance.
The parameters worth considering are the pruning confidence (C) and the number of iterations (N).

Intuitions before experiment:

1. I expect the basic trend to remain similar to what was observed for Decision Trees. But I expect the accuracy in each case to be pushed up because of the boosting.
2. As I vary the number of iterations, I expect the performance to get better as there are more runs for correcting the misclassified instances.

Experiment Setup:
This algorithm requires a fairly straightforward setup. I do two kinds of experiments:

1. For a constant number of iterations, vary the degree of pruning to observe the results.
2. For each experiment carried out in 1, carry the same experiment with different number of iterations to observe if the number of iterations has a significant impact on the accuracy.

Analysis, Possible Explanations and Lessons Learnt:

The results obtained seem to be highly dependent on the dataset chosen. For the Soybean dataset, the performance on applying boosting actually goes down!! This seems very counter intuitive. I did not believe this in the first go and tried tinkering with other values of C, but the pattern holds. This can be attributed to the fact that there may be symptoms common to different classes of diseases. So, in order to classify as one disease, a rule on a feature may have been modified which, in turn may have caused a misclassification on some other disease. Intuitively this can

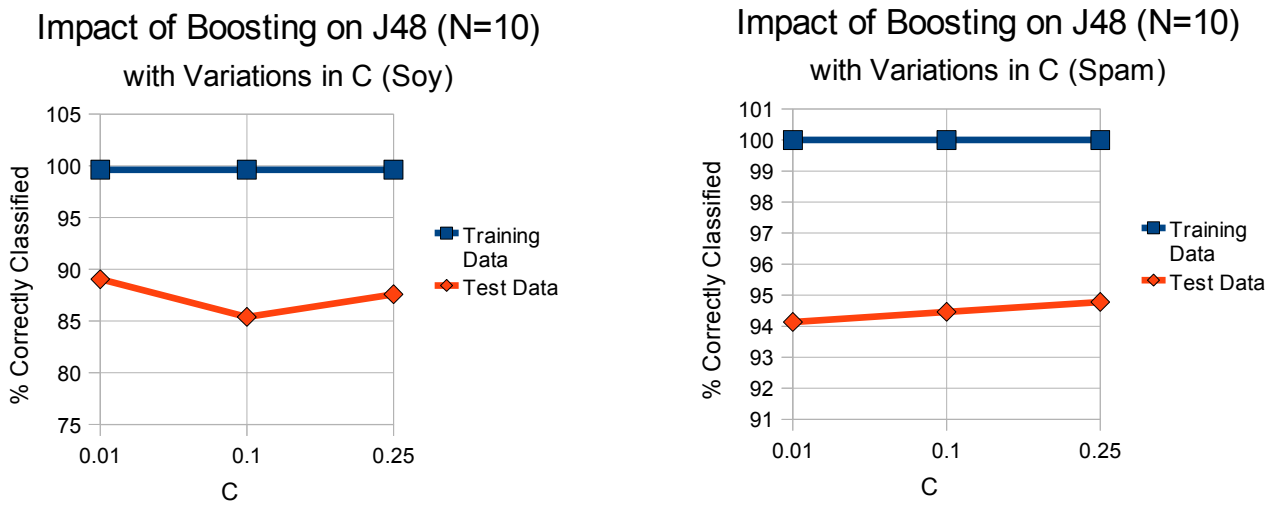


Figure 5 : Impact of Boosting on Decision Trees with variation in pruning

Value of C	J48 accuracy	J48 + Adaboost accuracy (N=50)
0.01	92.97%	89.05%
0.1	93.12%	86.13%
0.25	92.53%	89.78%

Table 1: J48 vs J48+AdaBoost test data accuracy for soybean dataset

Value of C	J48 accuracy	J48 + Adaboost accuracy (N=50)
0.01	92.46%	96.20%
0.1	92.83%	96.09%
0.25	92.98%	96.09%

Table 2: J48 vs J48+AdaBoost test data accuracy for spambase dataset

result in constant oscillations between rules pertaining to some attributes. However, I can not substantiate this reason with concrete data. It's merely an abduction.
Other possible reason is that the dataset has missing values, and boosting is trying to form contradictory rules on basis of these missing values.
A complete dataset like spambase gives very consistent results on the other hand. There is a substantial increase in

the accuracy of the filter. Though not displayed in the charts, another factor worth noting is that on the **train data** there is classification with 100% accuracy for almost all cases in the spambase dataset.
The number of iterations also has an impact on the accuracy of classification, Iteratively increasing the N values saw slow but steady improvement in the accuracy in both the datasets. I tried the values of N as 10,15,20,50 and the pattern was that the value stabilizes at some value of N (for eg, between 15 and 20 for soybean dataset) and then remains constant. After that, increasing the number of iterations has no effect on the accuracy.

KNN Algorithm:
This seems to be one of the most intuitive ways of going about classification/regression. Basically k- nearest neighbors (in terms of, say Euclidean Distance) of the query point are calculated. Then there are a variety of techniques on how to predict the value for the query point based on these k neighbors. Possible approaches include, mean of the neighbor values (regression & classification), weighted mean, etc.
I take into consideration the following three distance weighting approaches:
Unweighted: Here no weights are attached to the k neighbors while calculating the mean. This has the implication that all neighbors contribute equally, irrespective of their distance from the query point.
Weight = 1-d: Here, as the distance from the query point increases, there is a decline in the weight of the neighbor. This ensures points closer to the query point have a greater contribution from the one's further away.
Weight = 1/d : Here, the weight is inversely proportional to the distance. This achieves the same goal as the above option, though to a much more extent because of the inverse relation.

Intuitions before experiment:
1. The classification accuracy should depend on values of k. My intuition says that k=1 may not give consistently good results because there may be cases where there are two results at almost the same distance but in opposite directions. Then, the result would be biased towards the point which is marginally closer. The results should get better with increasing k, but only upto a particular point, after which performance should degrade, as points not that close start contributing to the predictions as well.
2. Unweighted distances approach should not yield results as good as the other two approaches as it makes sense for the closer points to have a greater influence in predicting the values for the query point. Amongst (1-d) and (1/d) approaches it's not that easy to make a prediction. Both achieve the same target though to a different extent. It will be interesting to see how these two compare against each other.

Experiment Setup :
I carry out the following experiment:
For each distance weighting approach, vary the values of k to see how the classification accuracy is impacted. Then, compare these trends for each approach and draw conclusions.

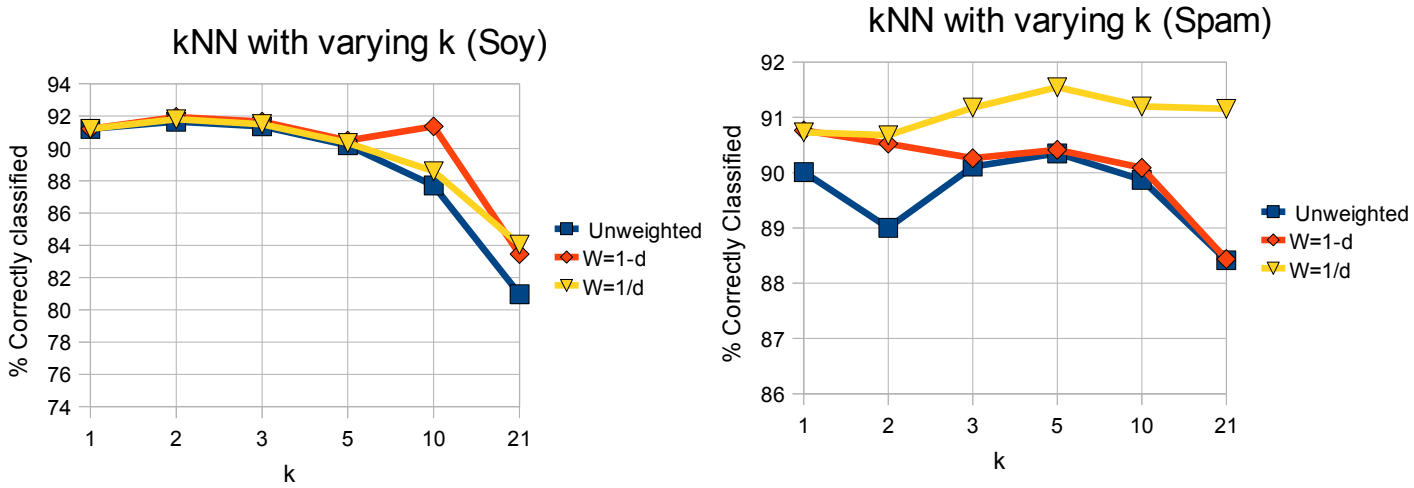


Figure 6: Comparison of classification accuracy with different values for k for the three distance weighting approaches

Analysis, Possible Explanations and Lessons Learnt:
The results of the experiments show some pretty interesting trends. First of all, some of my intuitions are re-affirmed. One of them being, there is a performance improvement with an increase in the value of k only upto a limit. I imagine this value of k to depend on how the samples are distributed. One case where I imagine considering k of higher values

is if there are multiple clusters in the distribution and the query point lies in a region in between, but not a part of any of the clusters. Choosing a very small k would bias the result unfavorably towards the closest point.
As for deciding what the best distance weighting strategy is, my experiments on spam data sets tend to suggest (w=1/d) strongly. I can understand unweighted strategy not performing as well for higher values of k. The experiments seem to suggest that as value for k increases, we really need to cut down the contributions of distant points pretty aggressively.
While I am convinced about w=1/d being a really good strategy, I am still a little skeptical about making claims like "its always better than 1-d" because I think this also depends on how the data is distributed. For example, for the soy dataset, for k between 10 and 21, w=1-d seems to be a better option. However, I would give more importance to the spam dataset results because it is a complete dataset as opposed to soybean which has missing values.

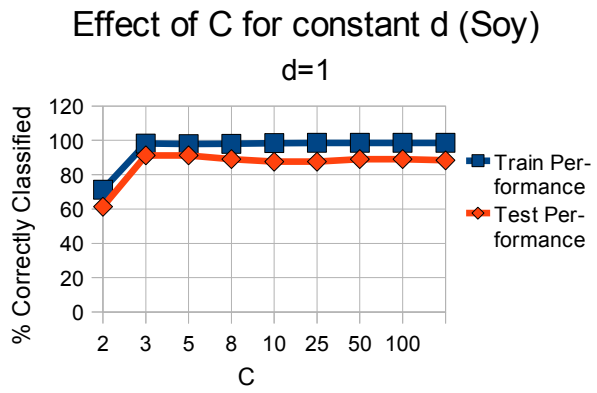
I had two more interested questions while trying to analyze the results related to kNN, which required some reading up.
1. For the spambase dataset. It may be a good idea to add weights not only to distances, but also to attributes. Clearly frequencies of words like "free", "viagra" etc. hold much more importance than frequencies of words like "hi", "the" etc. Hence, there should be some mechanisms to address this issue.
2. The second question crossed my mind was with respect to the soybean dataset which has some missing values. How does kNN deal with such missing values. I looked this up online as well as in the Weka source code. Weka has a thumb rule as: if both – the query point and the neighbor have the attribute missing, take the difference of the attribute as 0. If only one of the values is missing, the difference is taken as 1 for nominal attributes, 1 for numeric attributes (for normalization) and max-min otherwise.
A fair bit of lessons were learnt from this algorithm since the results exhibited such clear patterns for one of the datasets (spam). One lesson is, that there is no way of predicting a good value of k just by inspection. We must take multiple runs with varying typical k values and selecting the most consistently good one. However, we can say that very large values of k should generally be avoided. The other lesson is that its almost always a good idea to employ some kind of a distance weighting scheme. W=1/d seems to be a fairly good scheme.
In the best case scenario for kNN, I got a False Positive value (FP) for 0 (non spam) as 0.12 i.e. 12% of the non spam messages wrongly get marked as spam. This is a not a very good number.

Support Vector Machines:
I found this algorithm to be the most intriguing of the lot and spent a good chunk of time trying to understand it. I finally understood what these are about, why we need kernel functions at all, and what do the parameters mean.
I chose the following two kernel methods:
Polynomial Kernel : K(x,y)=<x,y>^d where varying values of d are supposed to vary the curvature (flexibility) of the decision boundary (d=1 signifies a linear classifier not suited for non linearly separable values).
RBF Kernel : Where the parameter ɣ (gamma) determines the width of the RBF kernel.
Independent of the kernel chosen, SVM's also take a parameter C which signifies penalty assigned to the errors. A higher value for C signifies a higher penalty, implying a smaller margin as the costs of misclassification are high. Similarly, a small value for C implies a smaller cost for misclassification, thereby not compromising the margin width to a great extent.

Intuitions before experiment:
As per my readings on the subject, I expect the following trends:
1. For low values of C, I expect to see a poor performance in the classification on the training set. Relatively speaking, the corresponding test set performance should be not too bad. Keyword here : **Relatively.**
2. As C increases, performance should pick up on the training set, as we would be considering the idiosyncrasies of points because of high penalty. This reduces margin and should have a negative impact on the test set performance.
3. For Polynomial Kernel, as d increases, I expect the performance to increase with an increase in d, but only to an extent. Once, a value for d, good enough to separate the points is established, the performance should not get any better, but time required to build the model should increase.
4. For RBF Kernel, the parameter ɣ should be examined. I expect a similar trend as 3 here, but in this case, I expect the test dataset performance to go down for very high values, as a lot of idiosyncrasies would be modeled too.

Results:
Exp1 (Polynomial Kernel):
Keep d=1 constant , vary C to see the impact on accuracy

Analysis, Possible Explanations and Lessons Learnt:
As expected, for a constant d, as C increases, Training Dataset performance increases and stabilizes. There is a mild dip in the performance for test data for the Soybean dataset. The Spambase dataset behaves more erratically, with an expected dip in the test data classification, but then for sufficiently large C, the value picks up and stabilizes. I find this



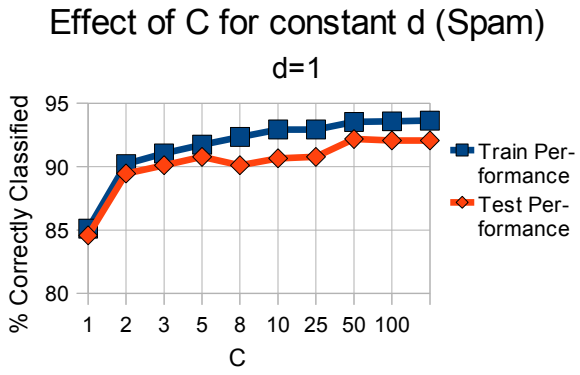
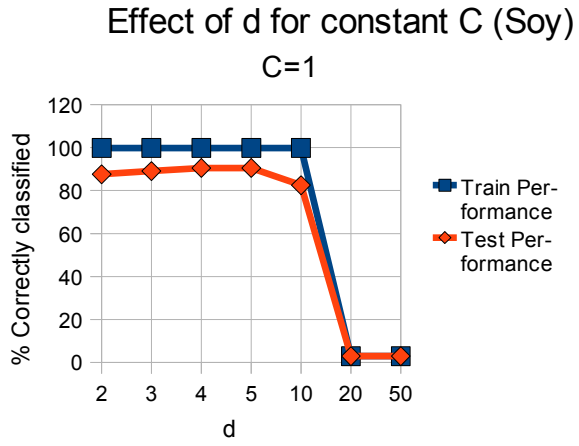


Figure 7: Comparison of data sets for variation in C

behavior strange and counter intuitive. One possible explanation may be that we are pure lucky as the test points may be well away from the margin of the hyperplane , thus not falling in the “interesting regions”. Another fact worth noting is, that the performance is better for the soy dataset than spam. One possible explanation for this may be that the number of instances is much larger, and they may be scattered around more inseparably (i.e. mixed) than the soy dataset. Lesson learnt is that very high values for C may hamper the performance for test data by setting the margin width lower.

Exp2 (Polynomial Kernel):
Keep C constant , vary d to see the impact on accuracy:
Analysis, Possible Explanations and Lessons Learnt:
The effect of d was particularly staggering, especially for the Soybean dataset. With an increase in d, the accuracy rose to an extent but then decreased not only for the test set, but also the training set. This is significantly different than my intuition that once the degree is high enough to separate the classes, then further increase should not increase the performance, but shouldn't effect it adversely either. The result is consistent even for spam dataset. So there may be more to the explanation than just the nature of the dataset. I tried exploring this a lot, but am yet to come up with a satisfactory explanation for this.
Lesson learnt seems to be that if there is a stable performance, there is no point shooting for higher degrees, considering that the running time increases with an increase in d, with substantially diminishing returns after a point, even to the extent of dropping the
performance significantly.



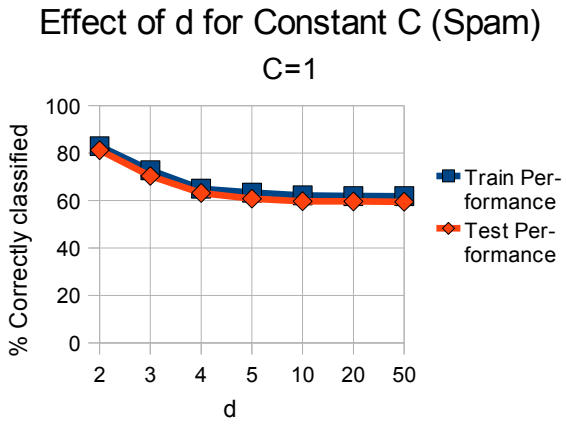


Figure 7 : Comparison of data sets for variation in C

Exp3 (RBF Kernel):
Keep γ =1 constant , vary C to see the impact on accuracy:
Analysis, Possible Explanations and Lessons Learnt:
The nature of curves is very similar to experiment 1, but a lot more gradual. For the spam dataset, after a peak at C=25, there is a very gradual decline in accuracy. Since C is independent of the kernel chosen, this makes sense.

Exp4 (RBF Kernel):
Keep C constant , vary γ to see the impact on accuracy:

Analysis, Possible Explanations and Lessons Learnt:
This experiment gave me really consistent results. All the intuitions I had prior to starting my experiments proved correct, almost picture perfect correct! As guessed, the accuracy on the training data goes up as the value for γ increases, which is understandable as the locality for support vector increases, giving a greater curvature of decision boundaries, almost following the whims of the training set. This has a very clear effect on the test data, where this increase causes a drop in the performance on the test set.
The dataset for soy gives the most dramatic results!! Notice how small the changes are in gamma for such substantial dips in the test data classification.
Note : I have chosen the values of C in either dataset which gave the best performance for a constant Gamma (1). This gives a clear indication that the algorithm has over-fit the training data in this case.
The lesson learnt here is a very valuable one. Great care must be taken when deciding the value for gamma, because in certain data sets, very small variations in the value can result in very different values for the test data accuracy (which is what we really care about).
In the best case scenario for SVMs, I got a False Positive value (FP) for 0 (non spam) as 0.104 i.e. 10.4% of the non spam messages wrongly get marked as spam. This is a not a very good number.

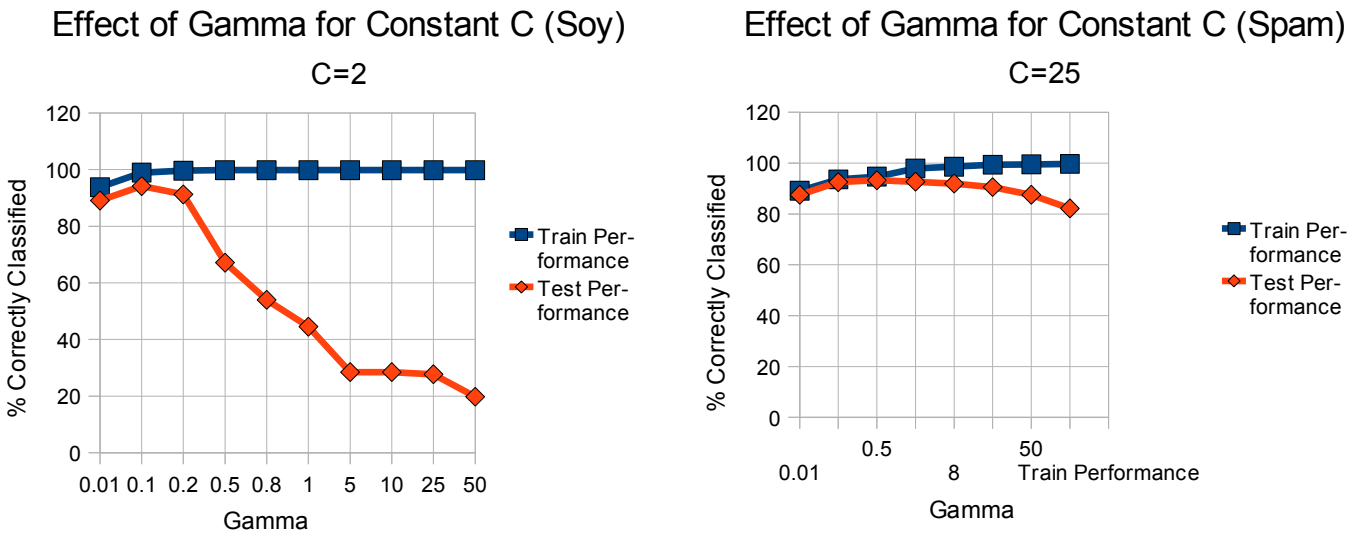


Figure 8: Comparison of data sets for variation in Gamma

Overall Summary:

This assignment involved carrying out a lot of experiments (at the end of the last experiment, I had about 700mb of results!) on various types of supervised learning techniques. The datasets chosen behaved well to a great extent, giving consistently for many patterns. However, there were cases where factors like missing values, fewer training instances etc. did result in counter intuitive results.
Each algorithm has its own parameters which can be tweaked to get reasonably good results. For example, I did not face a case where the disparity between the maximum accuracy for algorithms was more than 10%.
This actually says a lot about the nature of these algorithms – there is no one truly good or one truly bad algorithm- the results in fact, to a great extent depend upon how the parameters of the algorithm are manipulated.
Amongst all the algorithms, SVM's and Decision Trees (with Boosting) interested me the most.
However, boosting seems to be a really effective technique which can be applied to most of the algorithms and should be tried.
For the soybean dataset, I got the highest accuracy for SVMs with RBF kernel and Gamma 0.1 (94.16%) which is a very good result considering the dataset has relatively less instances, more attributes, 19 classes and missing values.
For the spambase dataset, I got the maximum accuracy for J48 trees with AdaBoost (96.2%).
In terms of time, training Multilayer Perceptrons easily took far more time than other algorithms. Whereas, it did not occur in the best results for either dataset, it seems to be fairly robust giving much smoother accuracy curves.
Also, most of my experiments were conducted on a 80-20 percentage split as well as cross validation with 10 folds. While publishing the results in the report, I chose cross validation because I think it gives a much more balanced pictures as there maybe chances that the test split may sometimes pick out values suitable to the classification rules, giving results better than what it's capable of. On the other hand, if this is performed 10 times, such cases can be ruled out to an extent. This becomes particularly useful for datasets with missing values like soybean.