Project 1: Supervised Learning

CS-7641 Machine Learning
Fall 2015
Name: Fujia Wu
GT Account: fwu35

1. Description of classification problems

1.1 Description of data sets

I selected two data sets for this project assignment: the Car Evaluation Data Set and the Mushroom Data Set. Both data sets come from the UCI Machine Learning Repository:

https://archive.ics.uci.edu/ml/datasets/Car+Evaluation
https://archive.ics.uci.edu/ml/datasets/Mushroom

The car data is a multi-class classification problem. The problem is to evaluate cars according to 6 attributes: buying price, maintenance price, number of doors, capacity in terms of persons, size of luggage boot and safety estimate of the car. Each attributes contains discrete category values, such as high, med and low. The class values are also discrete category data, including four classes: unacceptable, acceptable, good and very good. There are 1728 instances.

The mushroom data is a binary classification problem. The problem is to evaluate whether a type of mushroom is edible or poisonous based on its features, such as shape, color, odor, size, ring-type, etc. There are 22 attributes, all of which are discrete category values. The class values are only two, edible or poisonous. There are 8124 instances.

There are three reasons why I selected these two data sets before I started to learn them. First, since this assignment is about classification problem, I decided to select data that only with categorical values, rather than continuous numerical values. Therefore, I don't have to deal with the headache to decide how to convert the continuous values into discrete ones. Second, since this is an assignment, which has to be finished in a constrained time period and as the same time has to be rich enough and contains enough instances to perform learning and validation. So I decided to select data sets that have around 10-30 attributes and a few thousands instances. Third, I select a binary and a multi-class classification problem because I think one type may be more suitable for certain learning methods to learn versus others, therefore, these two data sets could help to differentiate the difference between different learning methods, I hope.

2 “Implementation” of learning algorithms

I use Matlab 2014b for this project. It provides all the tools needed for this project. Since for most of the machine learning methods in Matlab 2014b only accepts numerical values in the form of matrix, even for classification problems. So my program first converts the categorical data, represented by short strings, into discrete numerical data. My initial conversion method loops all the instances for a particular attribute. Whenever it sees a new value, it assigns an integer into the input matrix, which starts from 0 and increment to the total number of values. Then I normalized them into numerical values ranging from 0 to 1.

To track the performance of different learning methods, for each learning problem I divided the entire data into *k*-folded subsets randomly. Then I run the entire learning method for *k* times, with each time using 1 from the *k* folds as the test data and the other *k*-1 folds as the training data. For each training, I progressively increase the amount of data used for training so that I can later compute the performance of the trained model as a function of the training size (or learning cycle for boosting). Then I compute the averaged error rate on the training data and test data. For all problems in this assignment, I used $k = 10$, just to be consistent.

2.1 Decision trees

I used the function fitctree in Matlab 2014b for decision tree algorithm. Three options for the splitting criterion, *i.e.*, Gini's diversity index, twoing and cross entropy are explored and compared. From Figure 1 (left), it is seen that the three splitting criterions give about 10% difference in the error rate on the test set, while almost no difference on the training set for the car data set. In this particular case, Gini’s diversity index gives the better result compared to the other two.

Different levels of pruning are also explored and compared. Based on the node error (fraction of misclassified classes at a node), Matlab 2014b gives option for different levels of pruning. From Figure 1 (right), it is seen that as the pruning level increases, the performance of the decision tree decreases on the training data, but increases for the test data. It is interesting that at the maximum available pruning, the error rate of the tree on the training data and test data are almost same (around 0.3) and does not change with the training size. Upon examining the details of the tree, for the maximum pruning, I found there is only one node, i.e., there is roughly 30% percent of the data belongs to one class. This means for this particular data set, decision tree overfitting occurs quickly at a small training size.

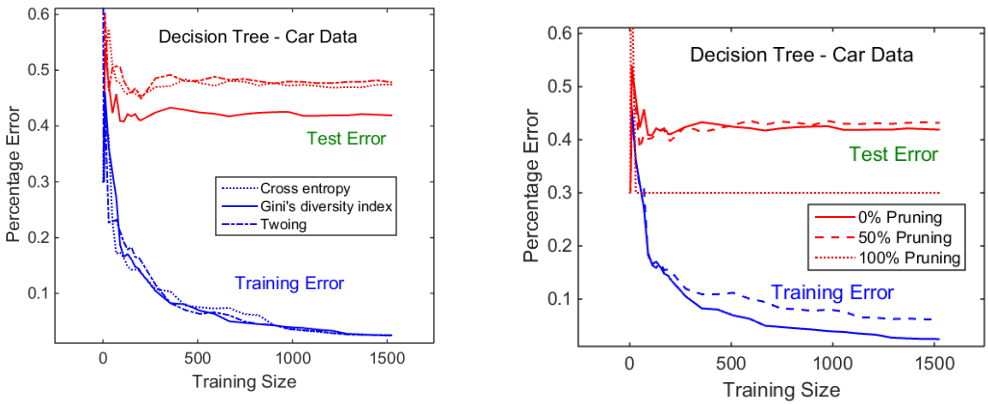

Figure 1: Comparison of different splitting criterions (left) and pruning level (right) for the performance of decision tree on the car data set

2.2 Neural Networks

I used the functions patternnet and train in Matlab 2014b to implement and train network networks learner. The learner is a two-layer forward network, with sigmoid hidden and softmax output neurons. For initial testing, I set the size of hidden layers to be the default (10) and compared 3 different learning algorithms: 1) gradient descent with momentum, 2) gradient descent with momentum and adaptive learning rate, and 3) scaled conjugate gradient backpropagation. They are all published learning algorithms in the literature. From Figure 3, it is seen that on the performance of car data set, the scaled conjugate gradient backpropagation is the best, followed by gradient descent with momentum and adaptive learning rate, and then gradient descent with momentum.

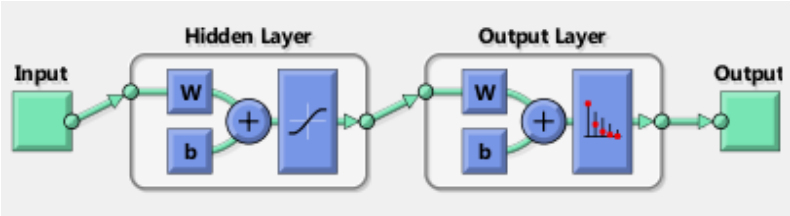

Figure 2: Structure of neural networks

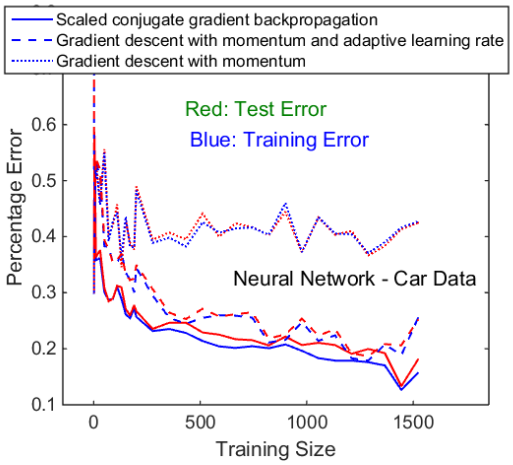

Figure 3: Comparison of three neural network algorithms on the car data set

2.3 Boosting

I used the function fitensemble in Matlab 2014b for boosting. The weaker learner is set up with the template creating function templateTree to create decision tree weak learners. For initial testing, I used the AdaBoostM1 boosting algorithm for the mushroom data set (binary classification problem), and AdaBoostM2 boosting algorithm for the car data set (multi-class classification problem). For the initial testing, the learning rate is set to 1 to boost the learning speed. I also compared the effect of pruning of decision tree weaker learners by turning the 'Prune' option on and off. Figure 4 plots the comparison of pruning on the performance on the car data set as a function of the training size.

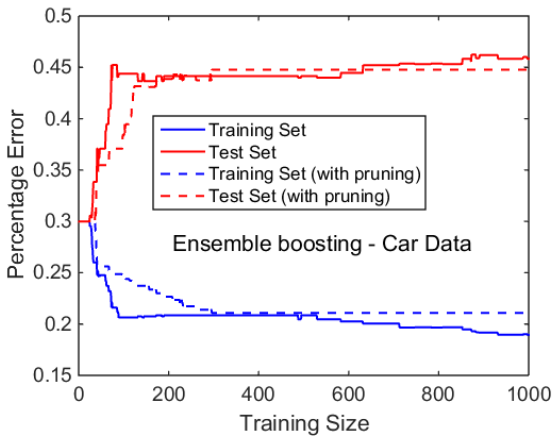

Figure 4: Effect of pruning of the decision tree weak learners on the final performance of boost ensemble learning on the car data set

2.4 Support Vector Machines

I used the functions fitcecoc and templateSVM in Matlab 2014b to implement support vector machine learning. The function templateSVM creates a support vector

machine learner template. Then fitcecoc actually performs the learning. Since the basic support vector machine algorithm is only for binary classification problem, the fitcecoc divides a multi-class problem into multiple binary classification problems using the 'Coding' option in the fitcecoc function. I used the 'onevsall' option. It treats one class as positive and the rest negative for each binary learner, and exhausts all combinations of positive class assignments. I have investigated three kernel functions: linear function, radial basis function and 3-order polynomial function. The comparison on their error rates on the car data set is shown in Figure 5.

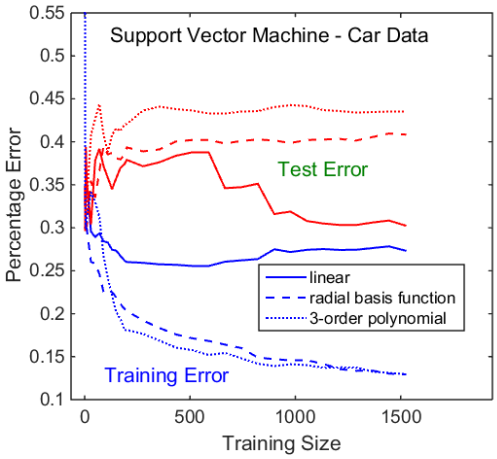

Figure 5: Training and test error for different kernel functions using SVM on the car data set

2.5 k-Nearest Neighbors

I used the function fitcknn in Matlab 2014b for *k*-nearest neighbor classification learner. I used the Euclidean distance as the distance measure, and put equal weight on different attributes since I do not put any bias at the beginning. I used different number of neighbors from 1 to 30. The results on the car data set are plotted in Figure 6. For this particular data set, although the training error rate increases with the number of nearest neighbors, but the test error rate decreases with the number of nearest neighbors.

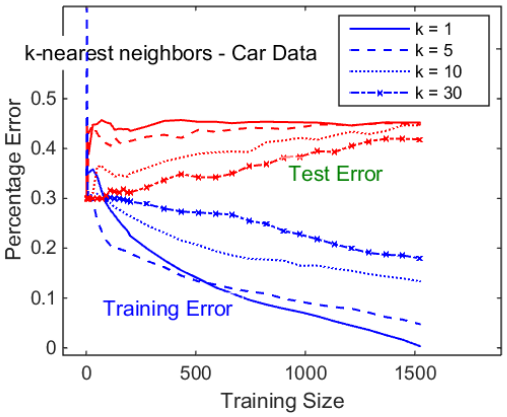

Figure 6: Training and test error using different numbers of nearest neighbors for kNN on the car data set

3. Comparison of Different Learning Methods

Figure 7 plots the comparison of the training and test error rates on the car data set for all five learning methods. It is seen that all training methods give reasonable error rates on the training data. For this particular data set, decision tree gives the lowest training error rate, followed by kNN, and then neural networks, boosting and SVM. For the test error rate, on the other hand, neural networks method gives the lowest error rate, which is almost the same as its error rate on the training set (only slightly higher). SVM method gives the second lowest test error rate. The other three methods, i.e., decision trees, kNN and boosting all have much higher test error rates compared to their training error rates. This means, these three training methods start to overfit rather quickly.

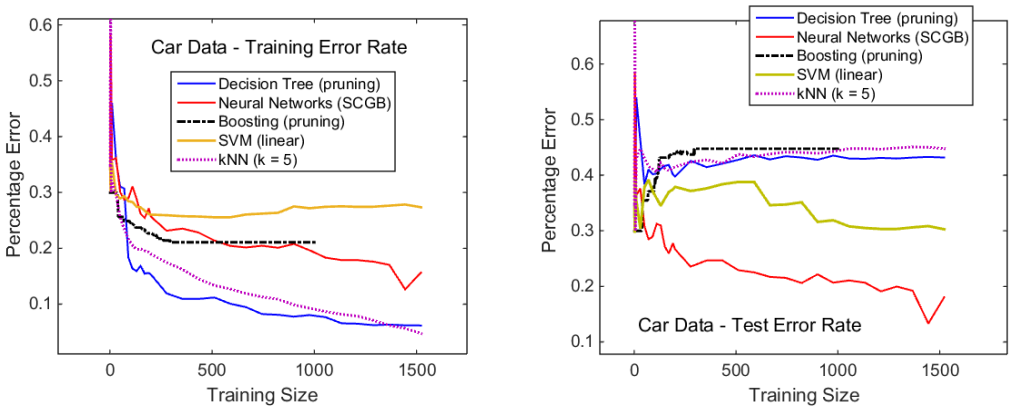

Figure 7: Comparison of all five learning methods on the car data set. Left: training error rate, Right: test error rate.

In addition to the car data set, I also investigated the performance of the five learning methods on the other data set, the mushroom data. Figure 8 plots the comparison of the

training and test error rates this data set for all five learning methods. It is seen that all learning methods have rather low error rate (less than 0.1) on the training data quickly as the training size increases. Boosting has the lowest error rate, followed by decision tree, neural networks, kNN and SVM. On the other hand, for the test data set, it seems only neural networks method gives acceptably low error rate. The other four methods, i.e., SVM, kNN, boosting and decision tree, all have rather high error rate on the test data set.

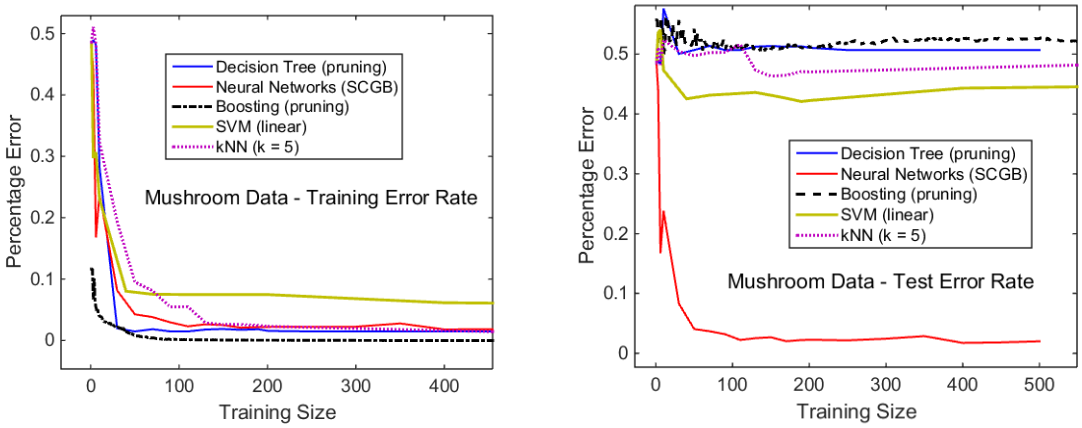

Figure 8: Comparison of all five learning methods on the mushroom data set. Left: training error rate, Right: test error rate.

4. Analyses and Improvements

From the above comparison, it seems that for both data sets only the neural networks method produces acceptable results, i.e., the test error rate is reasonably close to the training error rate. For the other four learning methods, the test error rate starts to increase substantially as the training size increases. This probably means neural networks method is a rather robust machine learning method.

Why do the other four methods fail to produce acceptable results similar to neural networks? To investigate the cause, I have spent extensive amount of time tuning the parameters for each method, hoping to make an improvement. For decision tree, I tried aggressive punning and using different splitting criterions. For SVM, I tried using different kernel functions and cross validation (using the 'CrossVal' option in Matlab 2014b). For boosting, I tried different learning rate, different weak learners (decision tree versus nearest neighbor). For kNN, I tried using different numbers of nearest neighbors and different weighting functions. These tuning generally makes an approximately 10-20% difference on the test error rates, sometimes improving them and sometimes worsening them. However, these tunings, at least those I tried, did not help to significantly decrease the test error rates and bring them close to the corresponding training error rate. For the sake of saving space and

my timing, I decided to not discuss them in detail, but focus on what I would like to discuss below.

After thinking about it for an extensive amount time, I started to think about the representation of the data itself, rather than the learning methods. I then realized that for these four learning methods, i.e., decision tree, boosting (with decision tree as weak learner), SVM, kNN, all largely rely on the "ordering" information of the inputs in order to make effective predictions. For example, for the car data, the first predictor is the car pricing, which is represented by "vhigh", "high", "med" and "low". Here I treat them just as distinct categorical values without the ordering information. But what happens in reality, i.e., in the "true" model, the order do matters. Without the ordering information, the learner might be able to generate a good model for the training data, but could not generalize to the unseen data, which could just be random data if the ordering information is messed up.

To solve this issue, I altered my program to manually assign "vhigh", "high", "med" and "low" values 1, 2/3, 1/3 and 0, and similarly do the same for other attributes. With this ordering information in the inputs, I ran all the learning methods again. This time all five methods improved their performance significantly on the test data set!

Figure 9 plots the comparison of all five learning methods before and after the ordering information has been manually assigned in the data set. It is seen that after the ordering information is given, the test error rates of all five methods have been significantly decreased and is much closer to the training error rates. This clearly indicates that without human inputs that decide the ordering information for the data in prior, the test data seem quite random to the models training from the training data, but if the ordering information is given, the entire data set is much more regular and physically make sense, which is why all methods significantly improved.

Figure 10 plots the new training and test error rates for the five learning methods after the ordering information has been given in the car data set. It is seen all the methods now have gained acceptable performance on both the training set and test set. What is nice is that in all cases, the test error rates are very close to the corresponding training error rates. Now, if we define learning methods that have the lower test error rate to be better, from Figure 10 (right) it is seen that decision tree is the best, followed by kNN, neural networks, and then SVM and boosting. The fact that decision tree performs the best is not surprising because upon reading the original car data set is generated from a tree-based model.

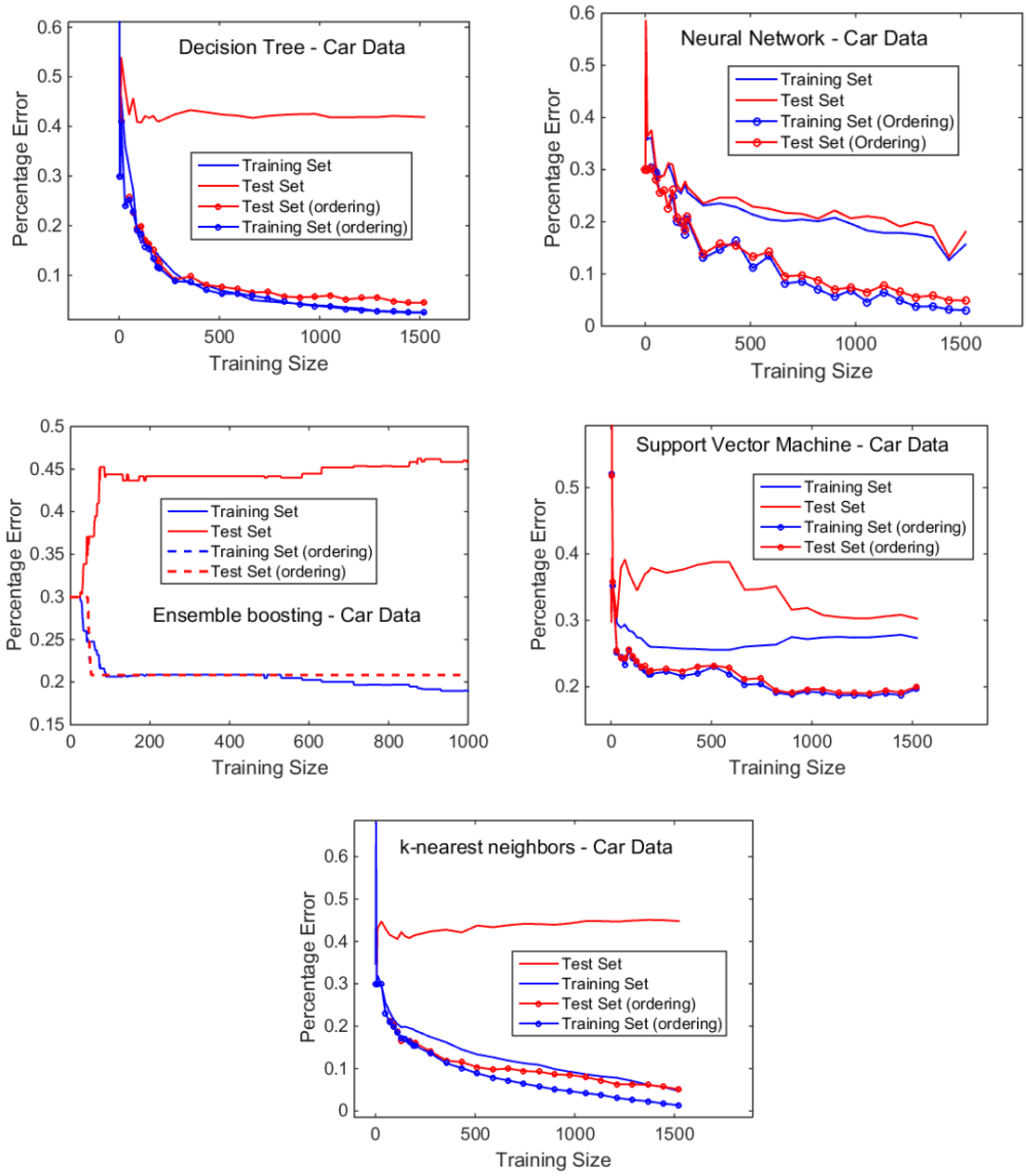

Figure 9: Comparison of all five learning methods before and after the ordering information is manually assigned for the car data set

On the other hand, if we define performance to be not only having low test error rate but also robust on the ordering and noise in the inputs, I would say in this case neural networks method has the best performance because it also gives low error rates when the ordering information for the attributes is not given, which all other methods failed to do. This means neural network method is not sensitive to the ordering of the numerical values in the input matrix, but the other four methods are.

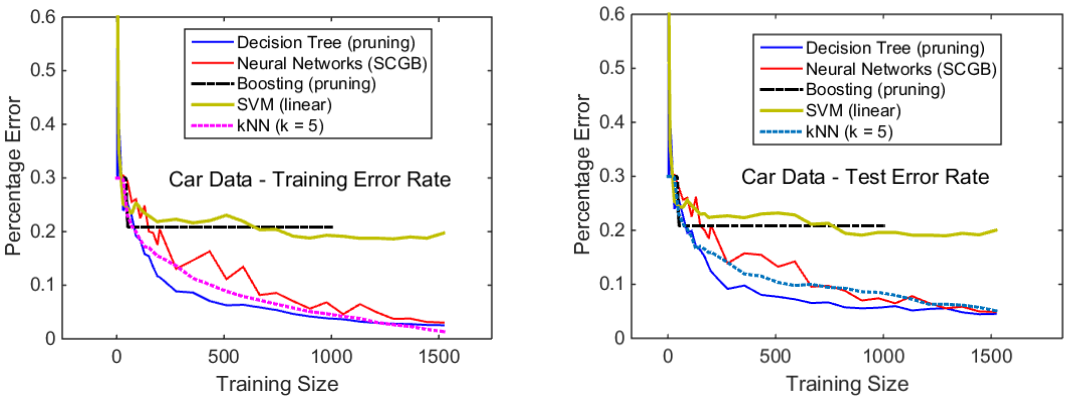

Figure 10: Comparison of all five learning methods on the data after the ordering information has been given in the original data

Can the performance on the mushroom data set be improved in the same way? However, upon looking into the mushroom data set, it is noted that it seems to be difficult to assign the values of its attributes to numerical values with ordering information. For example, color and shape. Different colors or shapes do not mean there should be any particular order among them.

Then what to do? One solution, which might be expensive, is to try all the possible ordering for all the values for all attributes, and use the model that gives the best performance. Then depending the number of values for an attribute and the total number of attributes, this method might not be feasible because it will be too expensive. Another solution is to randomly explore different possible combination of ordering select the best result. Since I discovered the importance of the ordering information quite late in this project, I did not have enough time to implement the approaches that I proposed here.

Finally, I also did not conduct a throughout analysis and recording of the running time for different learning methods. After running them for so many times, a general impression is that kNN is the fastest, followed by decision tree. These two methods are generally quick. Neural networks, boosting and SVM take much longer time, depending on the specific parameters they use.