**Analysis Report for Assignment 1**

**OMSCS CS-7641**

**Mark Chang (xchang33)**

**Table of Contents**

**Part 0 -- Problem Introduction**

**Part 1 – Employee Turnover Problem**

1.1.1 Decision Tree

- 1.1.2 Pruning

- 1.1.3 Feature Selection

1.2.1 K-NN

- 1.2.2 Play with k, the number of neighbors

1.3.1 Support Vector Machine, kernels

1.4.1 ‘Adaboosted’ version of decision tree

- 1.4.2 Explore the number of iterations

1.5.1 Neural Network

**Part 2 – White Wine Quality**

2.1.1 Decision Tree

- 2.1.2 Pruning

- 2.1.3 Feature Selection

2.2.1 K-NN

2.3.1 Support Vector Machine, kernels

2.4.1 Boosting (Adaboost) version of the decision tree

2.5.1 Neural Network

**Part 3 -- Conclusion**

**Part 0 Problem introduction**

After researching and comparing among the potential classification problems, I found two interesting ones. First is the employee turnover problem and second is white wine quality analysis.

My bachelor’s background was in Human Resource Management. One of the most intriguing topics during that four years was how to achieve a high employee retention. We covered other topics too, such as performance assessment, salary management and human behavior. However, none of the topics, or lessons really taught me about the fundamental yet critical question: what do employees care the most at work and why they tend to leave? HR is a human related field, it’s expected to rely more on human behavioral data analytics. This dataset is a very good example. Besides, this is a binary classification problem and the dataset size is small, which is a good start for testing the classifiers. The data contains couple of categorical features, so data preprocessing is necessary.

The second problem is to predict the quality for white wine. This dataset contains 11 numerical attributes and the target attribute, ‘quality’, is a continuous score. It’s needed to be converted to a classification problem. Here for simplicity and reusability of the code, I also converted this problem to a binary classification. If the score is higher than 5, then the quality is considered “good”, otherwise “bad”. Also, the attributes of the dataset are not in the same range, so the data needs to be normalized for certain algorithms.

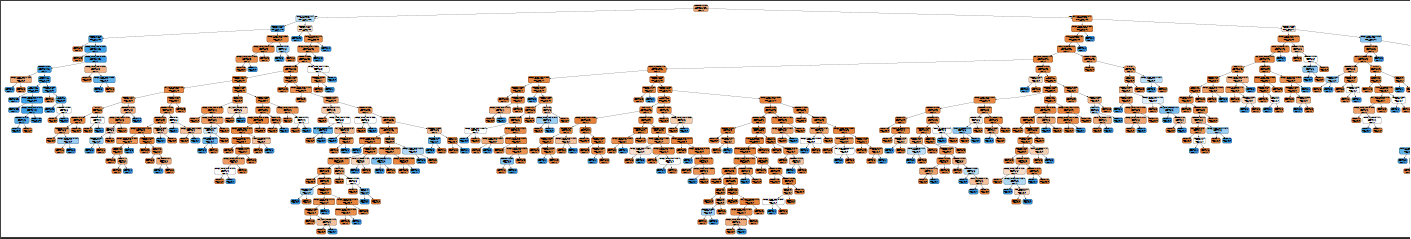
The rest of the report will split into 2 parts for the problems, 5 classifiers will be “implemented” and analyzed with various sets of hyper-parameters. I will compare their performances in terms of accuracy and runtime.

**Part I Employee turnover problem**

1.1.1Decision Tree

The following visualization is the first tree generated. This is a basic tree without any pruning and parameter tuning. The performance of the tree is as follows in the table. The technique of cross validation is employed here (cv = 5). The default split criterion is “gini”. This is a very huge tree, which cannot be even hosted on one page.

|  |  |
| --- | --- |
| Basic Tree | |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.975421 |
| CV\_Score\_STD | 0.00323 |
| Train\_Score | 1 |
| Test\_Score | 0.974545 |
| Precision | 0.934156 |
| Recall | 0.96105 |
| F1\_Score | 0.947412 |
| ROC\_AUC\_SCORE | 0.969912 |
| Runtime | 0.524626 |



1.1.2 Pruning

This is not necessarily perfect, I have a feeling that it could do a better if pruned in some way. Pruning is a set of ways to limit the growing of the tree. Here I tried to set an upper limit of the tree’s depth and the minimum number of samples required to be at a leaf node. The latter one guarantees that the model does not further split a node when the samples are exhausting.

To achieve the best tree, the GridSearchCV technique was implemented. Here, I need to specify a set of parameters: tree depth and minimum number of samples to split the leaf nodes. This technique helps to discover the best parameter combination in the list given.

Here are the potential parameters list. I tend to make it cover as many possibilities as possible.

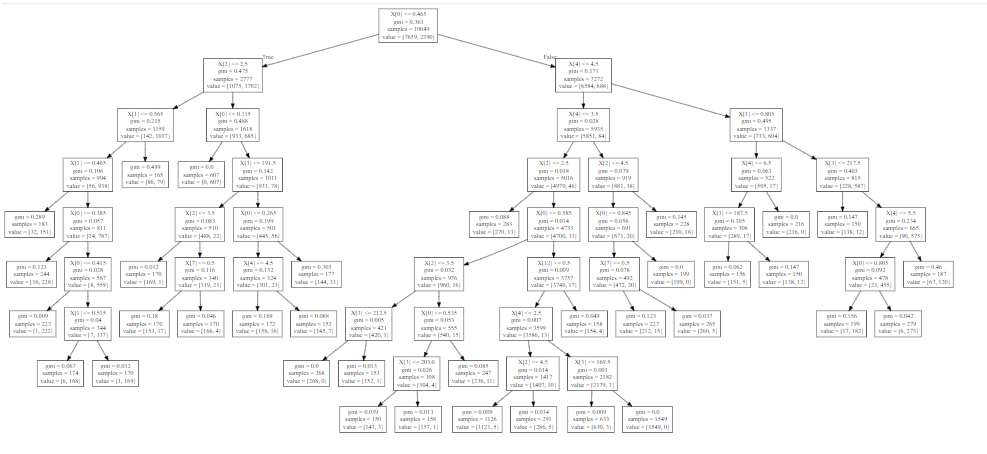
{'max\_depth': [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20], 'min\_samples\_leaf': [50, 100, 150, 200, 250, 300, 350, 400, 450]}

After the grid search (I set cv = 5 for consistency). Here return the best parameters:

{'max\_depth': 8, 'min\_samples\_leaf': 150}

|  |  |
| --- | --- |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.968256 |
| CV\_Score\_STD | 0.003137 |
| Train\_Score | 0.970246 |
| Test\_Score | 0.964646 |
| Precision | 0.948307 |
| Recall | 0.900931 |
| F1\_Score | 0.924012 |
| ROC\_AUC\_SCORE | 0.942771 |
| Runtime | 0.26997 |

Use above best parameters to fit the tree again and the new tree is much more simple without sacrificing significant performance. In fact, the runtime after pruning dropped dramatically by 50%.



1.1.3 Feature Selection

From the fitted decision tree, the importance of the attributes can be found. Due to the nature of decision tree, each feature provides different information gain as shown in the list below. I then dropped the features with an importance less than 0.05 and built the tree again (with pruning, otherwise it still would be a huge tree). This time, with the best parameters (5, 50), the accuracy stays the same, but the runtime drastically dropped.

|  |  |
| --- | --- |
| Features | Importance |
| satisfaction | 0.58045 |
| time\_spend\_company | 0.149701 |
| evaluation | 0.123546 |
| number\_of\_projects | 0.094283 |
| average\_montly\_hours | 0.051456 |
| salary | 0.000516 |
| management | 0.000048 |
| work\_accident | 0 |
| promotion | 0 |
| IT | 0 |
| RandD | 0 |
| accounting | 0 |
| hr | 0 |
| marketing | 0 |
| product\_mng | 0 |
| sales | 0 |
| support | 0 |

|  |  |
| --- | --- |
| Lean/Pruned Tree | |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.968256 |
| CV\_Score\_STD | 0.003137 |
| Train\_Score | 0.970246 |
| Test\_Score | 0.964646 |
| Precision | 0.948307 |
| Recall | 0.900931 |
| F1\_Score | 0.924012 |
| ROC\_AUC\_SCORE | 0.942771 |
| Runtime | 0.103695 |

|  |  |
| --- | --- |
| 5-NN | |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.941586 |
| CV\_Score\_STD | 0.005206 |
| Train\_Score | 0.959996 |
| Test\_Score | 0.935758 |
| Precision | 0.84603 |
| Recall | 0.893311 |
| F1\_Score | 0.869028 |
| ROC\_AUC\_SCORE | 0.921184 |
| Runtime | 6.465076 |

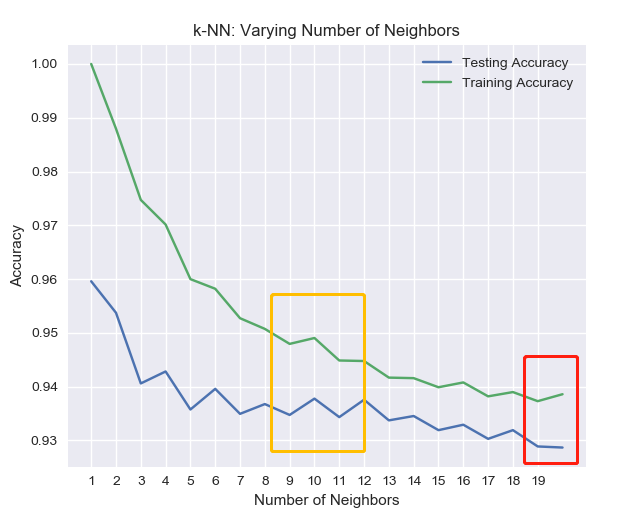
1.2.1 K-NN

Unlike decision tree, the data need to be scaled for K-NN, for better performance. Still, 5-fold cross-validation is embedded in the model. Here is the training result with 5 neighbors. We can tell that the runtime increased significantly than the Decision Tree. It demonstrates the ‘laziness’ of the K-NN algorithm.

1.2.2 Play with k, the number of neighbors

The following chart shows the trends of training accuracy and testing accuracy when k varies. For this problem, I tried k values from 1 to 20, the accuracy (both training and testing) decreases as k value increases. However, if k is too small, the model overfits. According to the chart, an “ideal” k value is 19, since when it’s 20, training and testing accuracies diverse. A good k value should keep the difference between training and testing accuracy as small as possible and keep itself small too. More k means more runtime. In this situation it seems k = 10 is a good fit (k = 11 sacrifices the accuracies). The right-handed table shows the performance of the knn when k = 10. As expected, the accuracies decrease compared to k = 5, but the difference between Train\_Score and Test\_Score decrease by 53.49%!

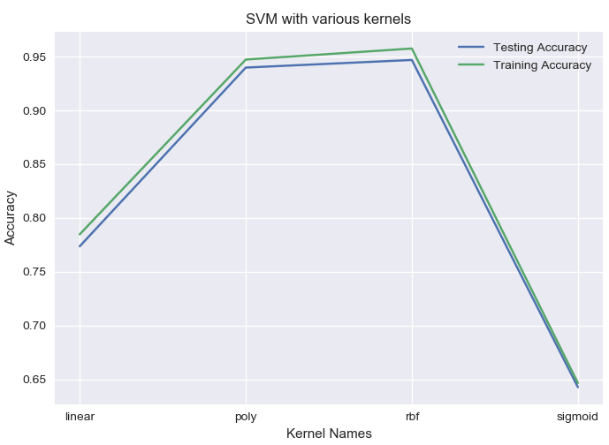
|  |  |
| --- | --- |
| 10-NN | |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.938103 |
| CV\_Score\_STD | 0.006772 |
| Train\_Score | 0.94905 |
| Test\_Score | 0.937778 |
| Precision | 0.865884 |
| Recall | 0.874682 |
| F1\_Score | 0.870261 |
| ROC\_AUC\_SCORE | 0.916115 |
| Runtime | 6.827148 |



1.3.1 Support Vector Machine, kernels

The SVM exploration concentrates more on the kernel functions. Below on the left is the performance of a basic SVM – with gamma = ‘auto’, C = 1.00 and kernel function as Gaussian Kernel. The right-sided chart shows the training and testing accuracy with different kernels.

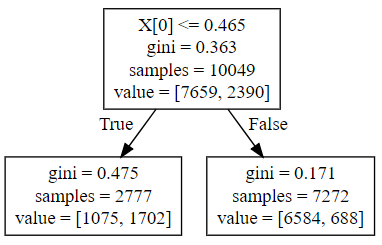
|  |  |
| --- | --- |
| SVM with Gaussian kernel | |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.951338 |
| CV\_Score\_STD | 0.004641 |
| Train\_Score | 0.957508 |
| Test\_Score | 0.946869 |
| ROC\_AUC\_SCORE | 0.926737 |
| Runtime | 6.093823 |



“rtf”, in the third position on the x-axis, stands for “Radial Basis Function kernel” also known as Gaussian kernel. Based on the performance of the 4 kernels, the linear kernel doesn’t do a good job, so maybe this is not a linear separable problem so either a polynomial or Gaussian kernel significantly increases the model’s performance.

1.4.1 ‘Adaboosted’ version of decision tree

In the first part exploration of the decision tree classifiers, pruning was implemented by limiting the tree depth and minimum samples for each leaf. In this section, one form of ensemble learning, boosting technique is implemented. Here a more ‘aggressive’ pruning will be used for each tree estimator (weak learners). The most intriguing hyperparameters are the number of the estimators, or iterations and the learning rate. As an example, the following is a fitted tree with depth = 1 and criterion = ‘gini’.



|  |  |
| --- | --- |
| Weak Learner | |
| SCORE NAME | SCORE |
| CV\_Score\_AVG | 0.82456 |
| CV\_Score\_STD | 0.008389 |
| Train\_Score | 0.82456 |
| Test\_Score | 0.812323 |
| Precision | 0.589616 |
| Recall | 0.701948 |
| F1\_Score | 0.640897 |
| ROC\_AUC\_SCORE | 0.774428 |
| Runtime | 0.050991 |

Apparently, this tree has relatively poor performance (weak learner). What about an adaboost classifier built on top of the learner? Here is the performance of the adaboosted version of the decision tree with the same pruning (depth = 1 and minimum samples of a leaf = 10). Iterations of 5000 and learning rate of 0.05 are the parameter values here.

|  |  |
| --- | --- |
| Adaboosted tree with 5000 iterations and learning rate of 0.05 | |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.960593 |
| CV\_Score\_STD | 0.003753 |
| Train\_Score | 0.962484 |
| Test\_Score | 0.960404 |
| Precision | 0.923474 |
| Recall | 0.909399 |
| F1\_Score | 0.916382 |
| ROC\_AUC\_SCORE | 0.942893 |
| Runtime | 148.1826 |

The accuracy substantially increases, and overfitting is largely improved. Compared to the previous pruned tree in 1.1.2, the test score is lower by 0.44% while the train and test accuracy difference dropped by 62.86%!

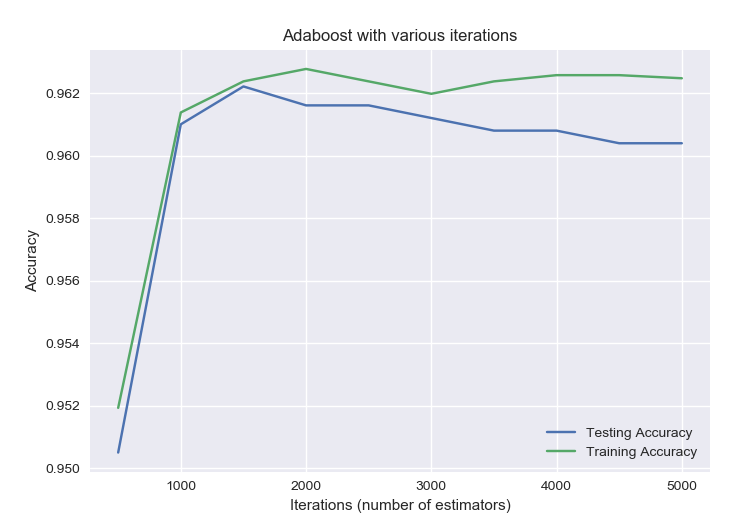
1.4.2 Explore the number of iterations

The number of iterations also states how many weak estimators are used in the adaboost classifier. From 1 estimator to many, the accuracy goes up. I would be intuitive if an iteration – accuracy learning curve is displayed.

Here is the result. It’s obvious that when number of iterations = 1500, the model performs best because of 1. least overfitting; 2. Relatively high accuracy. The curve almost doesn’t fluctuate, which means the learning rate is fine with the value of 0.05. But overfitting seems to emerge with more and more iterations.

The performance of the adaboosted tree with iteration of 1500 is shown in the right-handed table. The difference between train and test accuracy decreases by 92% (compared to iteration = 5000)!

|  |  |
| --- | --- |
| Adaboosted tree with 1500 iterations and learning rate of 0.05 | |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.960892 |
| CV\_Score\_STD | 0.005193 |
| Train\_Score | 0.962384 |
| Test\_Score | 0.962222 |
| Precision | 0.931424 |
| Recall | 0.908552 |
| F1\_Score | 0.919846 |
| ROC\_AUC\_SCORE | 0.943796 |
| Runtime | 44.423687 |

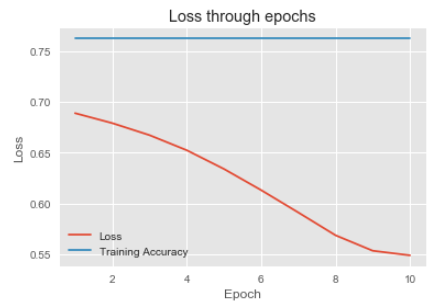


1.5.1 Neural Network

I used Keras to build sequential models with different layers, number of nodes, and activation functions. I tried 4 hidden layers with 5 nodes for each and 3 hidden layers with 12, 8 and 8 for each. Also, I tried both ‘tanh’ and ‘sigmoid’ as the activation functions for the output layer. Batch processing is implemented here (mini batches with 300 samples). The optimization algorithm I chose is Adam, it’s an alternative optimization algorithm for classic gradient decent or stochastic gradient decent.

However, my training accuracy does not improve in any cases, even after the less relevant attributes are dropped. The training accuracy is stuck at 76.22% and it seems all predicted results are “0”, which means the model predicts everyone to stay under all circumstances.

I’m still confused why the model does not work. More investigations should involve for this issue. Here is the learning curve for the network (4 layer, ‘relu’ as activation for the hidden layers and tanh for output layer; Adam as the optimization algorithm).



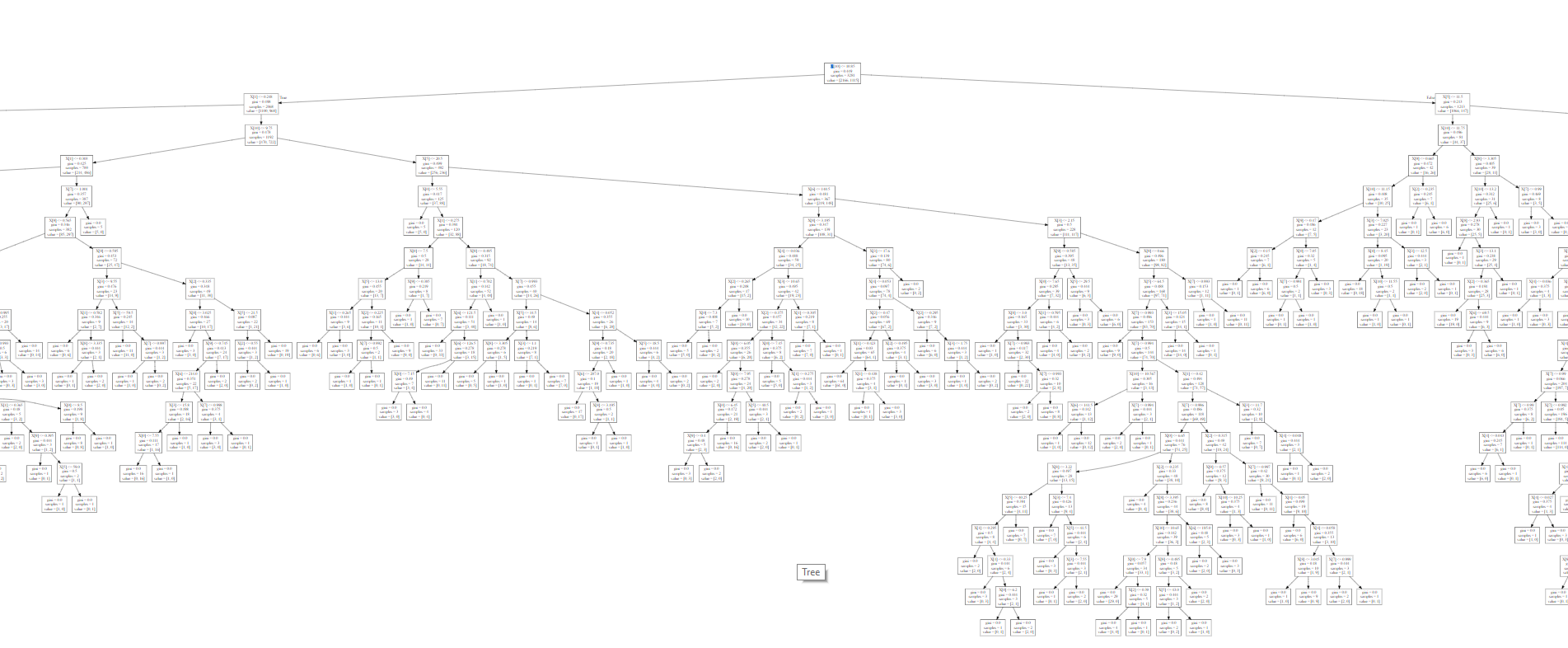
**Part II White Wine Quality**

2.1.1 Decision Tree

When looking at the basic tree (without specifying any hyperparameters), I found the tree built for wine quality does not perform as well as the one for employee turnover. I assume it’s caused by the data scales. To confirm the reason, I trained two trees, one with original data and the other with scaled data.

|  |  |
| --- | --- |
| Basic Tree before Scale | |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.760745 |
| CV\_Score\_STD | 0.008799 |
| Train\_Score | 1 |
| Test\_Score | 0.766852 |
| Precision | 0.633094 |
| Recall | 0.670476 |
| F1\_Score | 0.651249 |
| ROC\_AUC\_SCORE | 0.741832 |
| Runtime | 0.104009 |

|  |  |
| --- | --- |
| Basic Tree After Scale | |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.759831 |
| CV\_Score\_STD | 0.008449 |
| Train\_Score | 1 |
| Test\_Score | 0.765615 |
| Precision | 0.630824 |
| Recall | 0.670476 |
| F1\_Score | 0.650046 |
| ROC\_AUC\_SCORE | 0.740916 |
| Runtime | 0.104674 |

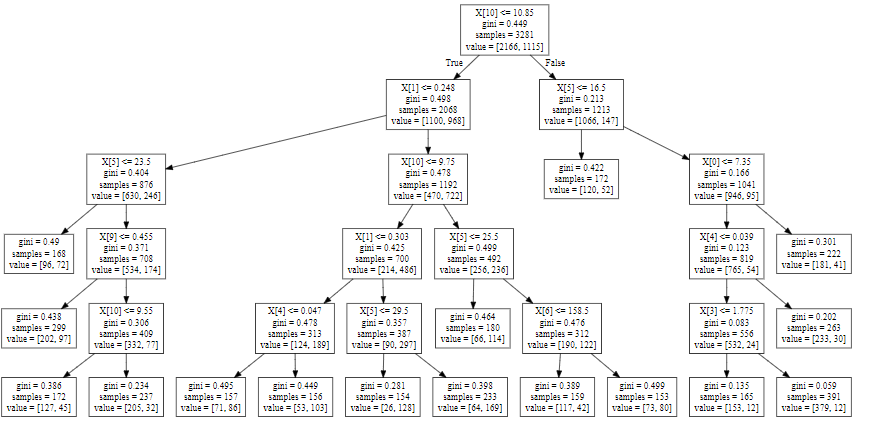


Scaling does not help. Does it mean Decision Tree does not care about training data scale?

2.1.2 Pruning

This tree before pruning is obviously overfitting the data. Here I used grid search to find the ‘best parameters’. The parameters to be used are {'max\_depth': 5, 'min\_samples\_leaf': 150}. Here is the result after pruning the tree using these parameters.

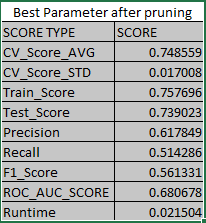
|  |  |
| --- | --- |
| After Pruning | |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.745815 |
| CV\_Score\_STD | 0.014814 |
| Train\_Score | 0.759829 |
| Test\_Score | 0.742733 |
| Precision | 0.607495 |
| Recall | 0.586667 |
| F1\_Score | 0.596899 |
| ROC\_AUC\_SCORE | 0.702216 |
| Runtime | 0.039439 |



Pruning decreases the variance issue of the model but does not improve the accuracy.

2.1.3 Feature Selection

The decision tree ranks the importance of the features as listed below. The features with importance weights less than 0.03 are eliminated and the tree was built again. As the results shown below.

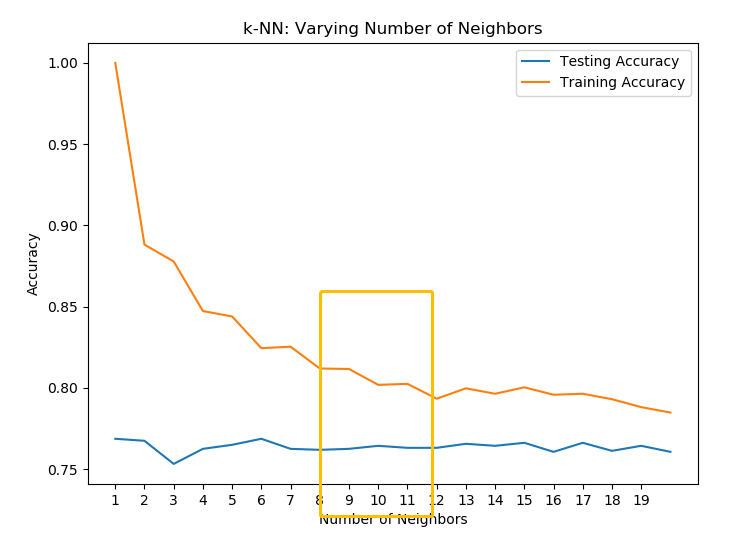


|  |  |
| --- | --- |
| Feature | Importance |
| alcohol | 0.543606 |
| volatile acidity | 0.294541 |
| free sulfur dioxide | 0.095869 |
| total sulfur dioxide | 0.026539 |
| sulphates | 0.016281 |
| fixed acidity | 0.012524 |
| chlorides | 0.009598 |
| residual sugar | 0.001043 |
| citric acid | 0 |
| density 0 | 0 |
| pH 0 | 0 |

2.2.1 K-NN

As 1.2.1 and 1.2.2 implies, finding best k value for K-NN algorithm is critical so here is the correlation of the accuracy and values of k. It shows that when k = 10, the bias and variance are balanced. The 10-NN model performs better than decision tree, both on training and testing accuracies.

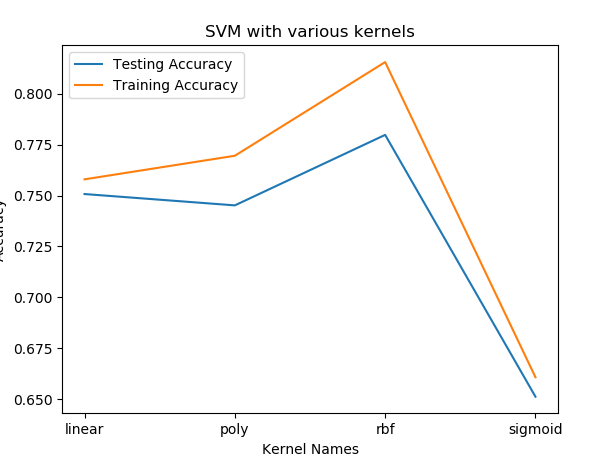
|  |  |
| --- | --- |
| 10-NN | |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.769279 |
| CV\_Score\_STD | 0.011408 |
| Train\_Score | 0.80189 |
| Test\_Score | 0.764378 |
| Precision | 0.68 |
| Recall | 0.518095 |
| F1\_Score | 0.588108 |
| ROC\_AUC\_SCORE | 0.70044 |
| Runtime | 0.62492 |



2.3.1 Support Vector Machine, kernels

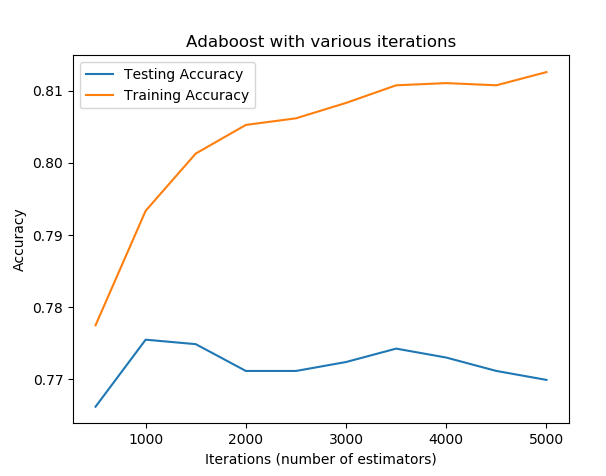
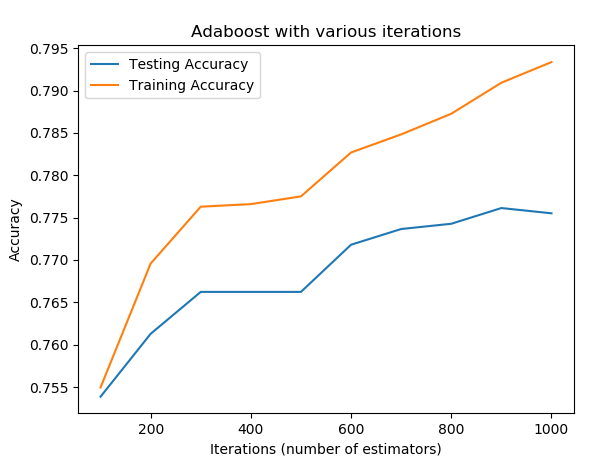
The left chart suggests that Gaussian kernel fits the best. The svm model with Gaussian kernel is by far the strongest predictive model for wine data.

|  |  |
| --- | --- |
| SVM with Gaussian kernel | |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.783605 |
| CV\_Score\_STD | 0.007524 |
| Train\_Score | 0.815605 |
| Test\_Score | 0.779839 |
| Precision | 0.683297 |
| Recall | 0.6 |
| F1\_Score | 0.638945 |
| ROC\_AUC\_SCORE | 0.73315 |
| Runtime | 1.660538 |



2.4.1 Boosting (Adaboost) version of the decision tree

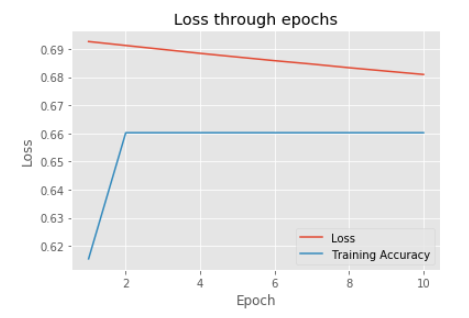
First, find the best number of iterations. From the charts, 900 seems to be a fair iteration. The result of adaboosted tree with iteration = 900 and learning rate = 0.05 is shown in the table. The ensembled model does not outperform svm with Gaussian kernel. Still, it shows that overfitting comes with excessive iterations.



|  |  |
| --- | --- |
| Adaboosted tree with 900 iterations and learning rate of 0.05 | |
| SCORE TYPE | SCORE |
| CV\_Score\_AVG | 0.765324 |
| CV\_Score\_STD | 0.024624 |
| Train\_Score | 0.790917 |
| Test\_Score | 0.776129 |
| Precision | 0.670146 |
| Recall | 0.611429 |
| F1\_Score | 0.639442 |
| ROC\_AUC\_SCORE | 0.73337 |
| Runtime | 16.246744 |

2.5.1 Neural Network

Similar with HR turnover problem, the result of the neural network is the same here, although the value of loss and accuracy are not exactly the same, but they follow the similar pattern: decreasing loss does not improve the model accuracy.



Part 3 Conclusion

5 models, each for 2 datasets, were implemented in this assignment. Here are my conclusions:

1. Data size is critical for machine learning;
2. Use Cross Validation as possible as I can;
3. Adaboost is a great version of ensemble learning;
4. Data does not have to be scaled for KNN;
5. KNN is computation consuming;
6. Gaussian kernel is robust for these two problems;
7. Decision Tree is very expressive, readable and intuitive;
8. Decision Tree helps on feature selection;
9. Data preprocessing is important;
10. Neural networks are worth exploring more