The decision tree is called J48 in Weka, and this decision tree uses the "Entropy", a measure of the data disorder and the Gain to choose attributes to split on. The objective is to choose an attribute to maximize the Gain.
(More details are specified in https://pdfs.semanticscholar.org/2456/a979fbe8eea47b90d625c1a064162be5382e.pdf)

When testing some values, other values are always kept at default, which are given under the title of each section.

Dataset 1 - Car Evaluation Database *(more details in datasets/cars/carInfo)*
This dataset was derived from a simple hierarchical decision model (*in datasets/cars/carInfo)*. All attributes in this dataset are nominal values meaning there is a set of predefined values, and there are 4 labels/classes of car acceptabilities: unacc (unacceptable), acc (acceptable), good and vgood (very good). This is a multi-class classification problem.

This model evaluates cars according to the hierarchical decision model. This classification problem is interesting because it is based on a given and known hierarchical decision model and it would be interesting to experiment how well each algorithm can reconstruct this given hierarchical decision model. In this dataset, all attributes have semantic meanings embedded with their classes. It is simple for a human to say that a car with low price, and high comfort and safety must be ideal (very good). A car with high price, and low comfort and safety must be unacceptable. However, with many different levels of lows and highs given for each attribute, it is interesting to see how different learning algorithms can make associations among all the attributes and attempt to classify each instance.

Dataset 2 - White Wine Quality Dataset *(more details in datasets/wines/wineInfo)*
The inputs/attributes include objective tests, which are all numeric values (e.g. PH values, fixed acidity) and the output/label is based on sensory data (median of at least 3 evaluations made by wine experts). Each expert graded the wine quality between 0 (very bad) and 10 (very excellent). In the particular dataset I am using, I modified the labels/classes from classes of scores between 0 and 10 (inclusive) to classes of 'good' and 'not good'. All instances with a quality score lower than 6 are classified as 'not good' and all instances with a quality score equal or greater than 6 are classified as 'good'. This is thus a binary class classification problem.

This is a very interesting classification problem because, in contrast with the cars quality dataset, the attributes don't have many semantic meanings related to the classes. To put into perspective, it is almost impossible for anyone to infer the quality of a wine given all of its lab tests or objective tests values. Therefore, this classification problem is more about aiming to construct a new model to represent the correlation instead of reconstructing any existing model. After all, it is interesting to see how each algorithm can do in predicting the quality of a wine based on its lab/objective tests values. The quality of a wine is a very subjective indication, where different experts might have very different opinions on wine quality evaluations, and the test results are very objective numeric values. It is almost like classifying if a joke is funny based on values like how long it is, how many words are there or how many curse words are in it. This property of this problem makes this problem extremely interesting.

Car Evaluation Database - Decision Tree

Default Values: full-size training set, confidence factor = 0.25

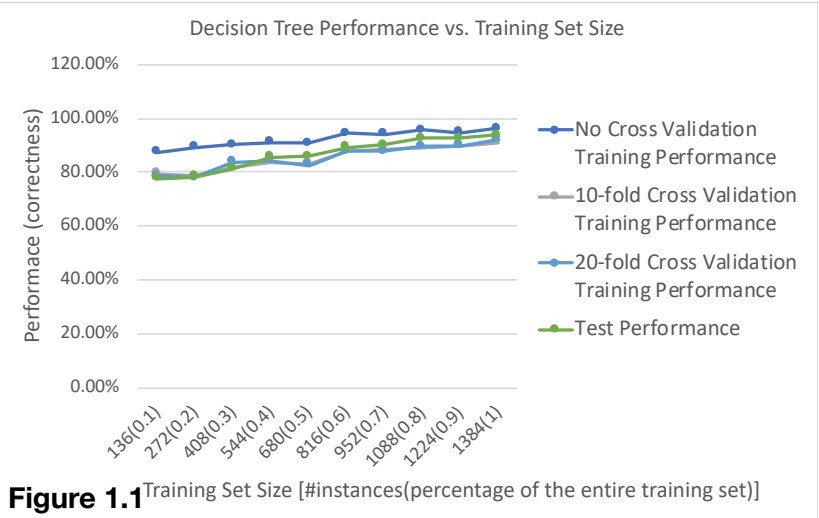


Figure 1.1

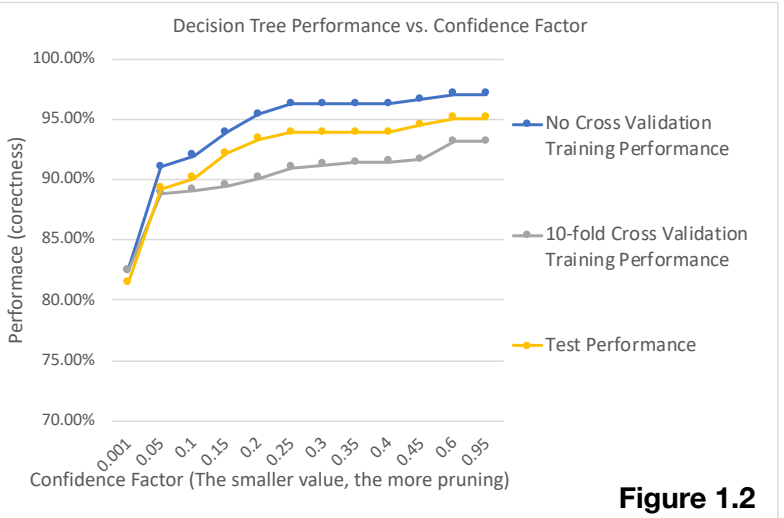


Figure 1.2

Car Evaluation Database - Neural Networks

Default Values: full-size training set, training time = 500, nodes in hidden layer = 12, hidden layer = 1, learning rate = 0.3, momentum = 0.2

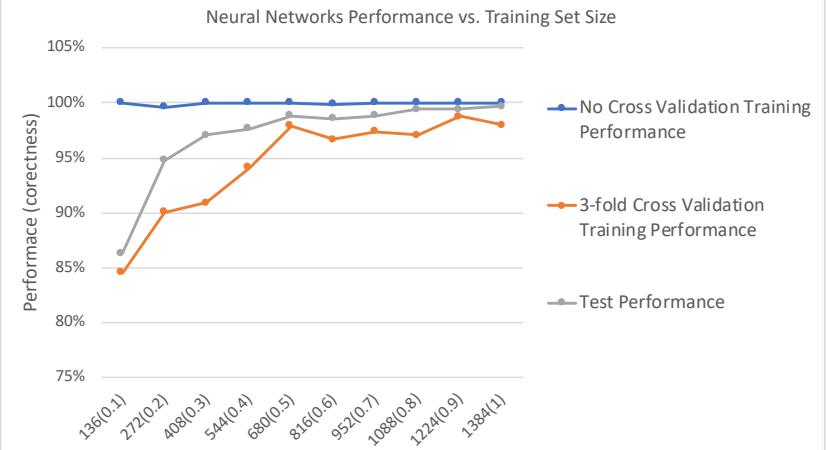


Figure 1.3

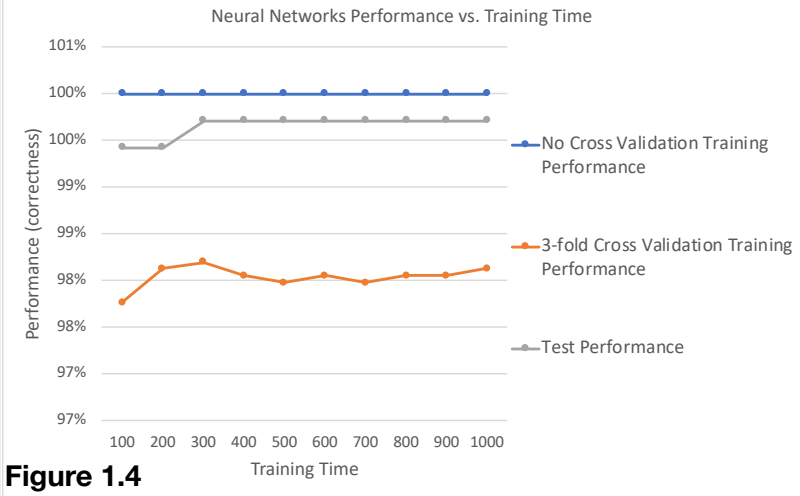


Figure 1.4

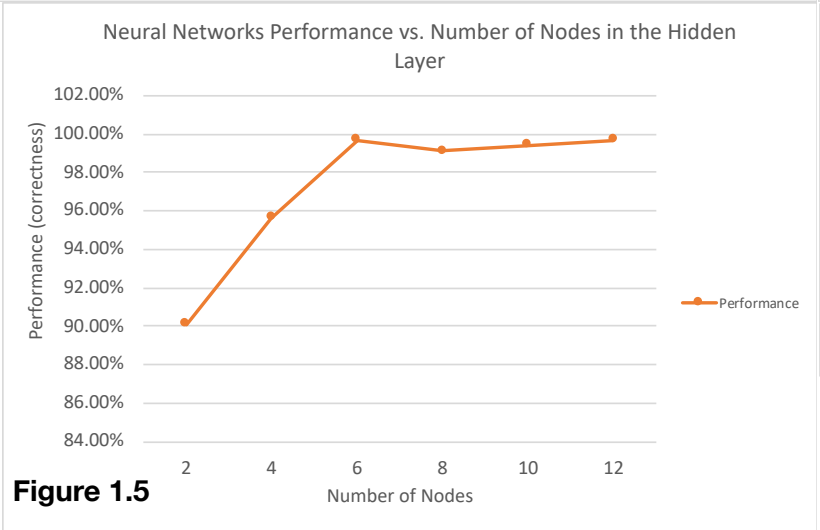


Figure 1.5

Performance	Learning Rate				
Momentum	0.1	0.3	0.5	0.7	0.9
0.2	99.7093%	99.7093%	99.7093%	99.7093%	100%
0.4	100%	99.7093%	99.4186%	99.7093%	100%
0.6	99.4186%	99.4186%	99.4186%	99.7093%	99.7093%
0.8	99.7093%	99.7093%	99.4186%	99.1279%	98.5465%
1	70.3488%	70.3488%	70.3488%	70.3488%	70.3488%

Figure 1.6

Car Evaluation Database - Boosted Decision Tree

Default Values: full-size training set, # of iterations = 10, confidence factor = 0.25

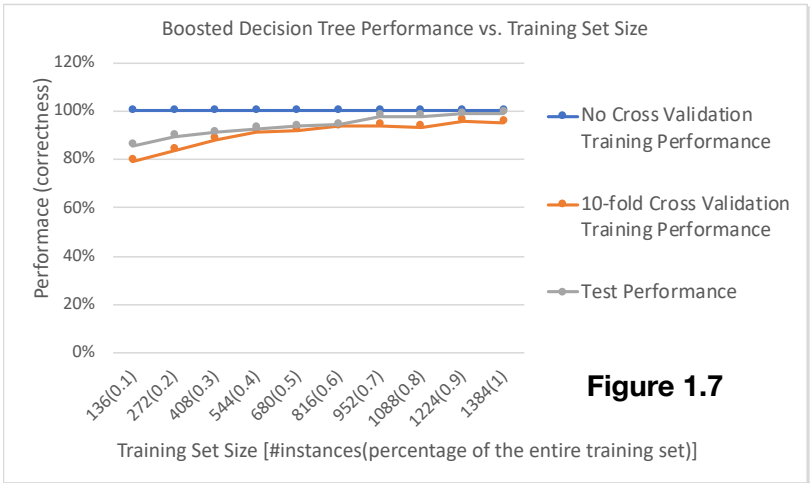


Figure 1.7

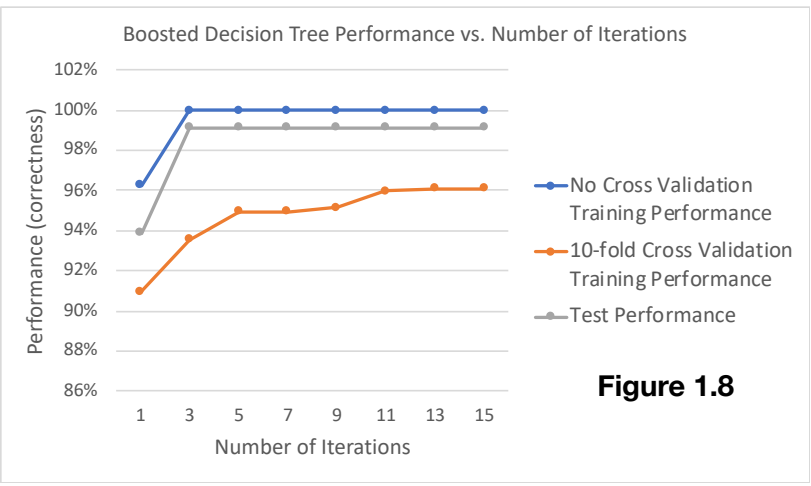


Figure 1.8

Confidence Factor	**Performance**
0.001	99.1279%
0.01	98.2558%
0.05	99.7093%
0.1	99.1279%
0.15	98.8372%

Figure 1.9

Car Evaluation Database - Support Vector Machine

Default Values: full-size training set, kernel = PolyKernel with exponent 1 (Linear)

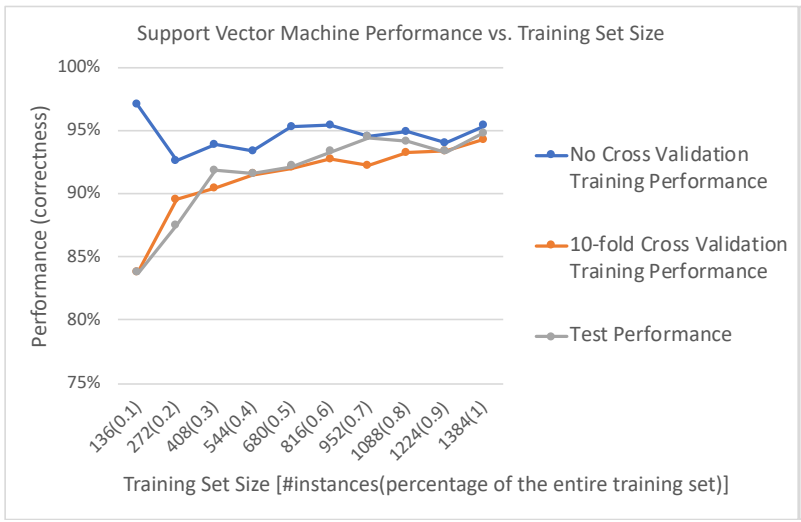


Figure 1.10

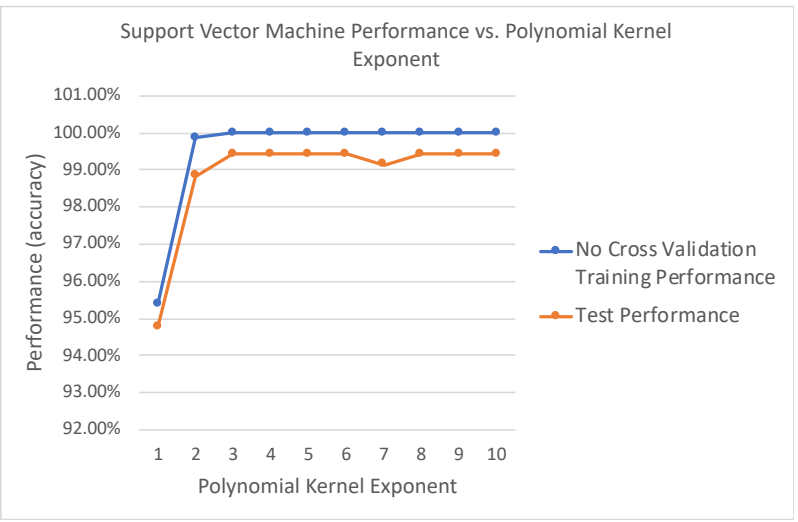


Figure 1.11

Car Evaluation Database - K-Nearest Neighbors
Default Values: full-size training set, k = 1

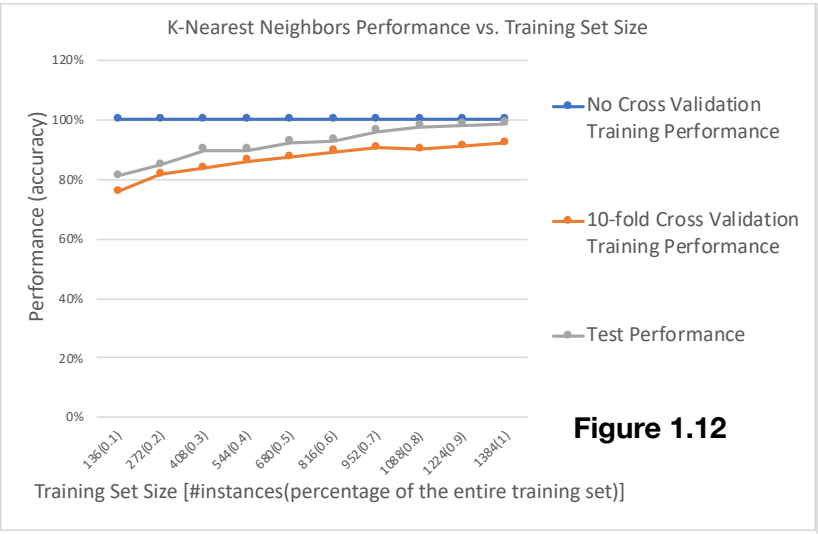


Figure 1.12

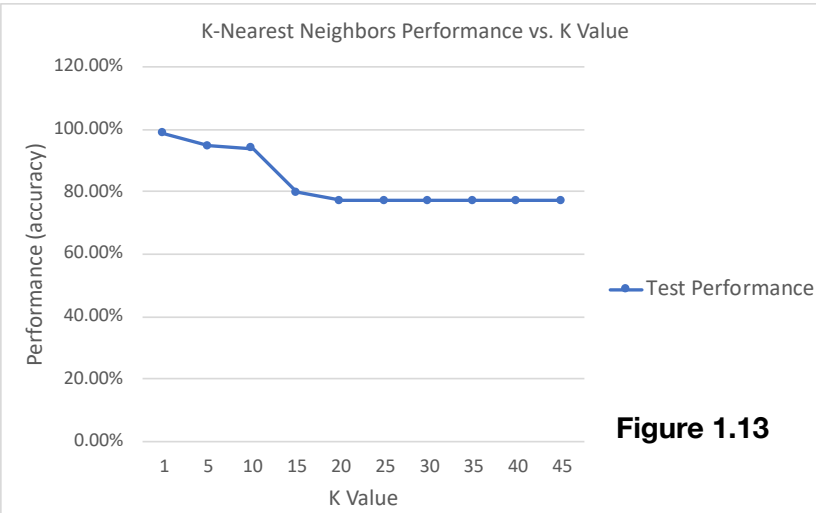


Figure 1.13

Car Evaluation Database - Analysis:
Decision Tree
The first graph (figure 1.1) is a performance vs training set size graph. Let's focus on the test performance line (green), which is an indicator of the actual performance of the trained model. The increase in training size did bring up the test performance by a considerate amount, which is expected.

The second graph (figure 1.2) shows that pruning actually hurts the performance. This is because the data is given based on an artificial decision model, so large amount of noisy data is very unlikely. the decision tree actually has a pretty accurate model to start with using all the non-noisy data, so pruning would hurt its accuracy by pruning off necessary branches.

As for any possible changes, I would suggest adding more non-noisy instances based on the decision model to the training set. Following the trend in figure 1.1, more non-noisy data should be able to increase the performance. An unpruned tree would also be better than a pruned tree for this problem. The highest test performance achieved for decision tree on this problem is **95.06%** with a confidence factor of 0.95 and all other values at default.
(0.57s for building model and 0.02s for testing on test set according Weka time)

Neural Networks
The first graph (figure 1.3) is the performance vs training set size graph. This one nicely illustrates that the test performance (gray) increases with the increase of the training set size. The curve converges at around 99% accuracy. It does seem a bit overfitted in the beginning because the model can't generalize enough information from such a small amount of data.

The second graph (figure 1.4) is a performance vs training time graph. The performance (gray) just reaches and stays at 99.71% accuracy when the training time reaches around 300. This shows that it doesn't require much training time for the model to adjust its weights to appropriate values, and more time doesn't help the neural networks either.

The third graph (figure 1.5) is a simple graph that shows the relationship between the performance and the number of nodes in the layer. I only experimented changing number of nodes with one hidden

layer because from the data collected, one hidden layer seems perfectly enough for this problem. The performance curve shows that the performance increases rapidly with the increase of the number of nodes in the hidden layer. However, at around 6 or 8 nodes, the increase of performance immediately slows down, which suggests to me that a layer of 6 to 8 nodes provides sufficient complexity for this problem.

Lastly, more detailed experiments and hyper-parameter tuning are performed. The table (figure 1.6) shows the performance of the model at different values of learning rate and momentum. For this part, the main purpose is to fine-tune the step size and how each step is taken when performing hill climbing on this problem. The model achieved 100% test accuracy a couple of times during this process.

Since there is no large amount of noisy data and the problem space is not extremely complicated, the neural networks is able to powerfully model the data and achieve a **100%** test accuracy with learning rate of 0.1, momentum of 0.4, and all other values at default. The only change I might try is trimming out even more noisy data, which might result in an even less complicated structure.
(3.78s for building model and 0.01s for testing on test set according Weka time)

Boosted Decision Tree
The first graph (figure 1.7) is a performance vs training set size graph, where the test performance (gray) shows an expected and gradual increase with the increase of training set size.

The second graph (figure 1.8) is a performance vs number of iterations graph. The test performance (gray) has a huge leap at 2nd iteration reaching 99.13% accuracy, and then the curve just stays at 99.13% accuracy despite the increasing number of iterations. This indicates that the curve is already converging at 2nd iteration. More iterations are not helping with the performance either.

The table (figure 1.9) shows the performance accuracy while adjusting the pruning of the trees. The confidence factor tested are all on the lower range because with boosted decision tree, the pruning can and should go more aggressive. The values show a peak of 99.7093% at confidence factor of 0.05.

If more data can be added, following the trend in figure 1.7, the performance still has a potential to increase. Pruning is definitely necessary here, confidence factor of 0.05 indicates a pretty aggressive pruning which agrees with the underlying knowledge for boosted decision tree, formed by simple models. The highest test performance achieved by boosted decision tree is **99.7093%** with J48 decision tree using a confidence factor of 0.05 and all other values at default.
(0.17s for building model and 0.01s for testing on test set according Weka time)

Support Vector Machine
The first graph (figure 1.10) is a performance vs training set size graph, where the test performance (gray) did have an overall trend of increasing with the increase of training set size. It did seem to have a bumpier increase when more data are added, which is expected because the kernel used here is linear so any noise or unbalanced data can cause some disturbance. Thus causing a bumpy increasing trend.

The second graph (figure 1.11) is a performance vs polynomial kernel exponent graph. The test performance curve (orange) nicely shows that when the exponent increases at first, the hyperplane gets significantly better at fitting the data, which is an expected behavior, with a peak at around degree 3, reaching 99.42% accuracy for the test performance. As the exponent keeps increasing, the performance seems to be not affected, which is maintained at 99.42%. This makes sense because it shouldn't have much influence on the performance with such slight increase in exponent values other than, theoretically, slowly bringing the curve into a decreasing trend. It is expected that as the exponent keeps increasing afterwards, the hyperplane will end up being more overfitted, which is not captured in this current frame of data. When tested with exponent of 20, the training performance is

still 100%, but the test performance dropped to 96.51%, which indicates overfitting. This proves what was just claimed: this curve will end up being overfitted if the exponent keeps increasing.

I wouldn't change anything other than keeping the exponent at around the 3 and 4. If more data is added to the training or test set, the exponent may need to be readjusted to fit all the data. The support vector machine is able to achieve **99.42%** accuracy with exponent at 3 and all other values at default. *(1.11s for building model and 0.08s for testing on test set according Weka time)*

K-Nearest Neighbors

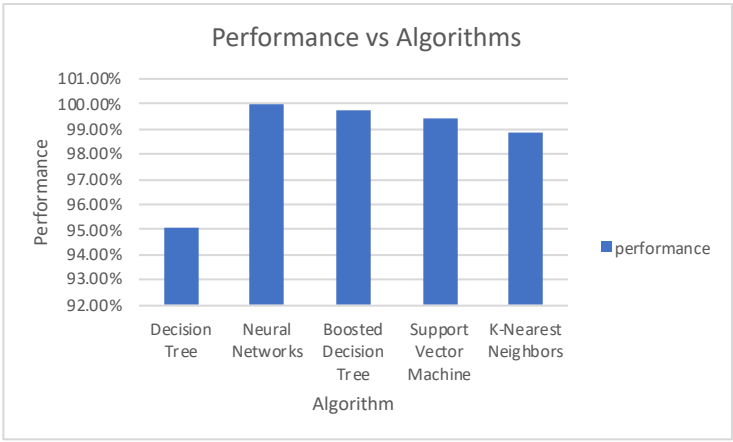
The first graph (figure 1.12) is a performance vs training set size graph, where the test performance (gray line) increases as the training set size increases, as expected. The 100% training accuracy is also expected due to the lazy nature of k-nearest neighbors.

The second graph (figure 1.13) is a performance vs k value graph. The performance curve has a significant drop when the k values goes from 10 to 15, which indicates that 15 might be a threshold of neighbors for this problem. From the graph, it is actually the best to have the k at 1, which is expected based on the nature of this classification problem. The dataset is based on an artificial decision model, so all the attributes have their importance in the decision making process. For each test instance, k-nearest neighbors takes in consideration of all attributes and pick the closest one, which highly aligns with the underlying knowledge of the dataset.

The k value for this problem should always be small because all the attributes provide valuable information. The closer ones are likely to be classified in the same class. The highest accuracy rate achieved for k-nearest neighbors is **98.84%** with k at 1 and all the other values at default. *(0s for building model and 0.1s for testing on test set according Weka time)*

Conclusion

From all the graphs above, it is notable that the cross validation performance does provide a better indicator of the actual test performance than the training performance. When the test set is not readily available in some cases, cross validation would be highly useful to predict the performance of the model. The only disadvantage for cross validation is that it might be extremely slow in cases when there is a large amount of data or the cross validation fold number is high. However, cross validation doesn't affect the actual test performance of the model because after cross validation is performed, the model still has to train on the entire training set before being used on the test set.



The diagram on the right shows the comparison of the performances for each algorithm on this classification problem. All of the algorithms are doing pretty well on this problem. The decision tree is doing the worst among all the algorithms, which is what I expected because decision tree is known to be relatively inaccurate comparing to other predictors. The neural networks, boosted decision tree and the support vector machine are all doing extremely well (>90% accuracy), but neural networks beats the other two with a perfect test performance accuracy. The neural networks seems very powerful on modeling this problem because only 1 hidden layer with 6 nodes are needed and training time of 300 is completely enough. This shows that the neural networks still has a lot more potential when, in the future, the decision model might become more complicated, for example, taking in consideration of gas or electronic and maximum speed. Overall, the **neural networks** seems like the optimal solution for this problem among all of the 5 algorithms.

White Wine Quality Dataset - Decision Tree

Default Values: full-size training set, confidence factor = 0.25

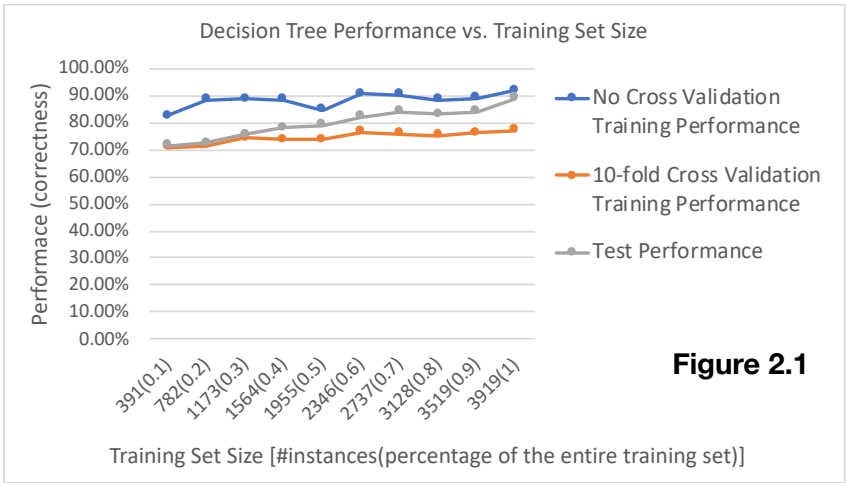


Figure 2.1

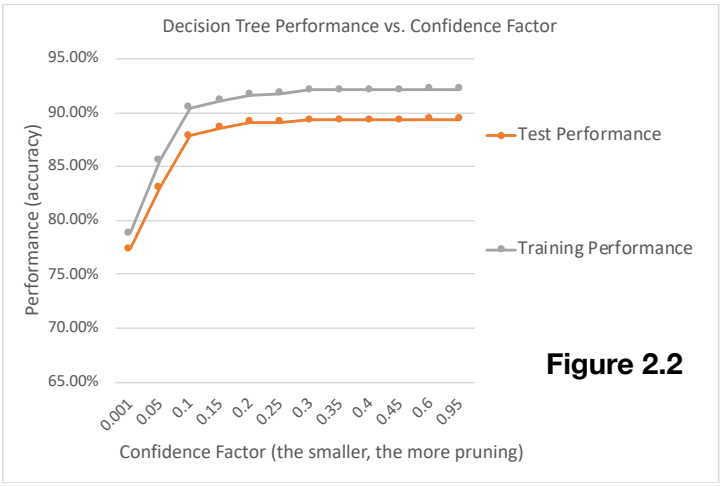


Figure 2.2

White Wine Quality Dataset - Neural Networks

Default Values: full-size training set, training time = 500, nodes in hidden layer = 6, hidden layer = 1, learning rate = 0.3, momentum = 0.2

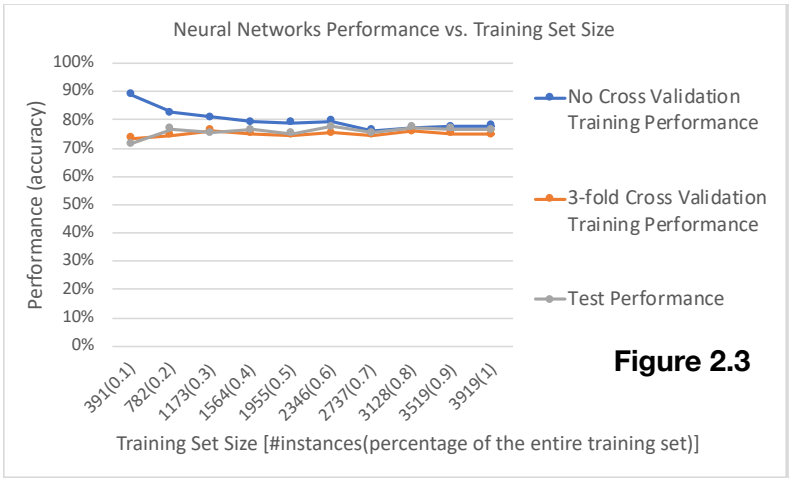


Figure 2.3

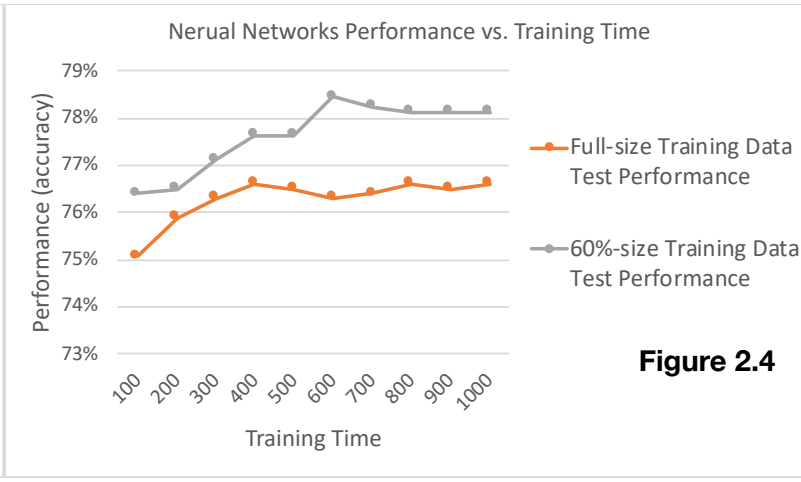


Figure 2.4

Performance	Number of Layers		
Number of Nodes	2	3	4
3	78.141%	78.856%	66.4964%
6	74.3616%	77.6302%	75.2809%
9	79.8774%	78.141%	76.0981%

Figure 2.5

Number of Nodes indicates the number of nodes in the last hidden layer, the other ones are kept at 6 as default (ex. number of nodes = 3, number of layers = 3, the hidden layers look like: [6, 6, 3])

White Wine Quality Dataset - Boosted Decision Tree

Default Values: full-size training set, # of iterations = 10, confidence factor = 0.25

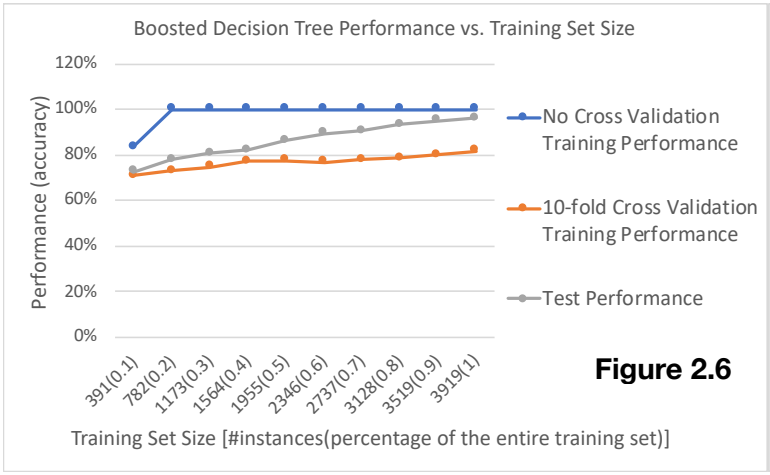


Figure 2.6

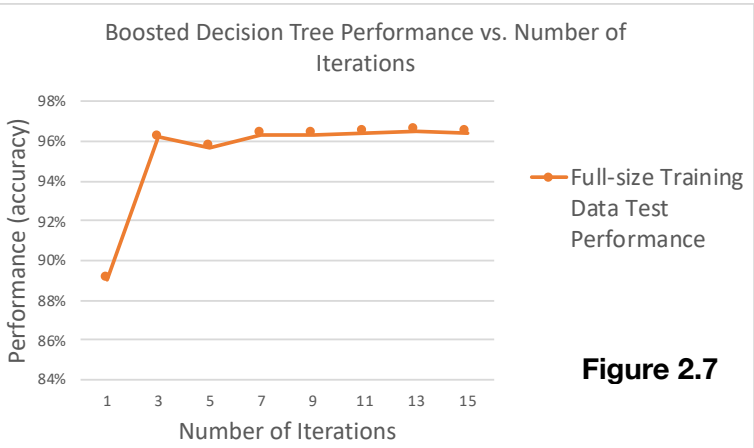


Figure 2.7

Confidence Factor	**Performance**
0.001	96.0163%
0.01	96.4249%
0.05	95.097%
0.1	97.1399%
0.15	95.6078%

Figure 2.8

White Wine Quality Dataset - Support Vector Machine

Default Values: full-size training set, kernel = PolyKernel with exponent 1 (Linear)

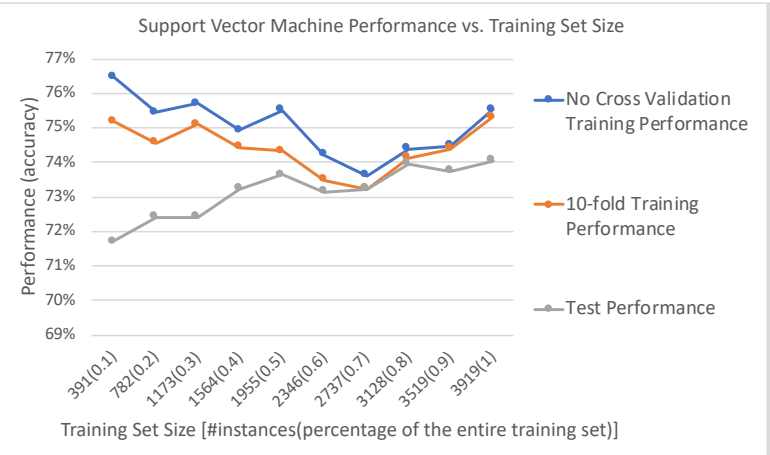


Figure 2.9

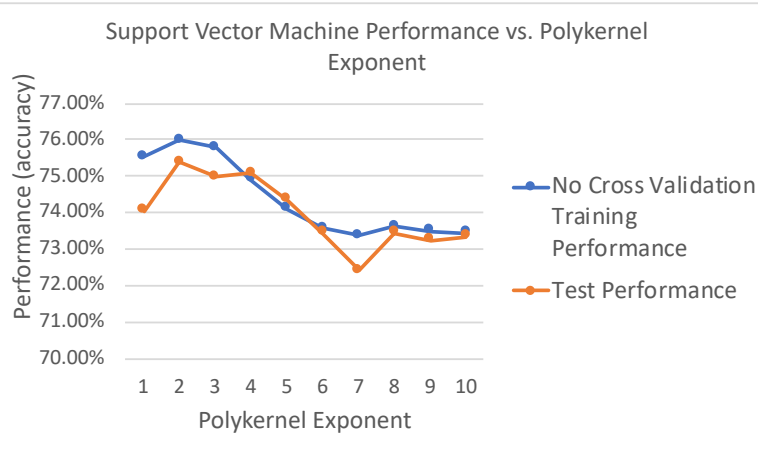


Figure 2.10

White Wine Quality Dataset - K-Nearest Neighbors
Default Values: full-size training set, k = 1

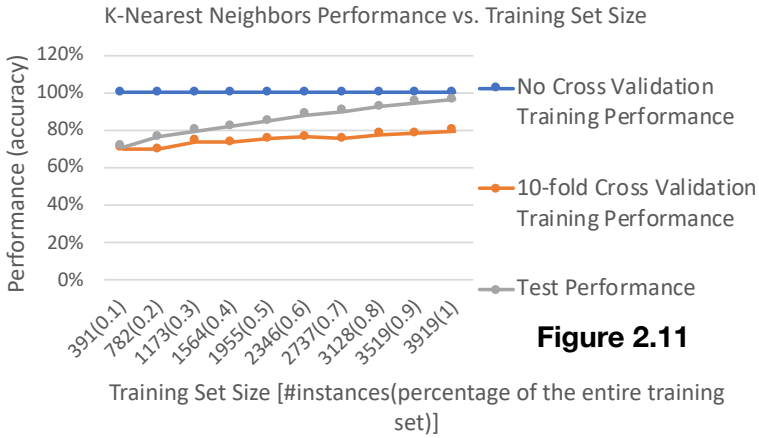


Figure 2.11

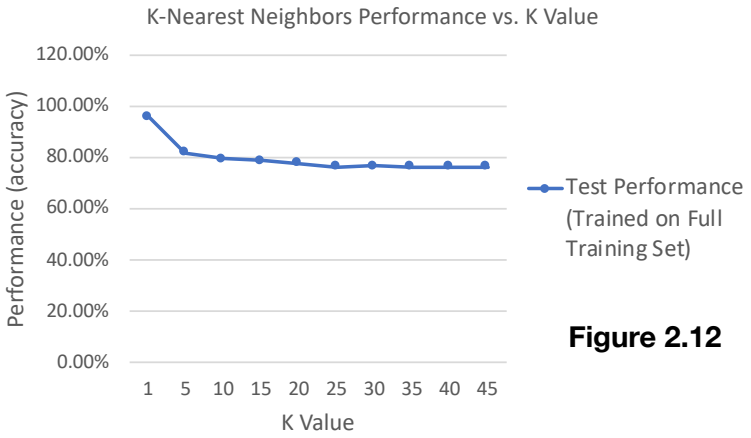


Figure 2.12

White Wine Quality Dataset - Analysis:
Decision Tree
The first graph (figure 2.1) is a performance vs training set size graph. The test performance (gray) has an increasing trend along with the increase of the training set size as expected.

The second graph (figure 2.2) is a performance vs confidence factor graph. The graph shows that the unpruned tree works better than the pruned tree. However, when the confidence factor is at around 0.3, both performance curves seem to be converging, which means the pruned branches at confidence factor of 0.3 are unnecessary to the performance, but they are not causing overfitting either. This shows that the training set and the test set contains very similar instances in terms of values and distributions, so that whatever model fits training set also fits the test set in the same way. The similarity of the two curves also suggests the same observation.

To use the decision tree on this problem, it would definitely help a lot if there are more training data because of the increasing tendency shown in figure 2.1. The tree should also be pruned with a confidence factor of around 0.3 to 0.6. Although the unpruned tree seems to work the best for now, in the future, if there are some more test data that don't necessary resemble the ones in the training set, this model risks being extremely overfitted. For now, the decision tree is able to achieve **89.38%** test accuracy with 0.95 confidence factor and all other values at default.
(8.87s for building model and 0.01s for testing on test set according Weka time)

Neural Networks
The first graph (figure 2.3) is a performance vs training set size graph. The test performance (gray) shows that the test accuracy is barely increasing while the training set size increases, and it is also very low (<80%) the whole time, which tells me that the neural networks is having a hard time generalizing the training data. This suggests to me that all the instances in each percentile of the training set are similar because the neural networks is struggling to generalize the information from 50% of the training set just as much as the full training set, which means that the additional 50% of the training set data is not making the generalization more clear at all; instead they are confusing the neural networks just as much as the previous 50% of the training set data. The similarity of instances mentioned here is not the similarity of the attribute values or classes of instances. The similarity here is that, across all instances, the attributes and the class relationship doesn't make sense to the neural networks. The neural networks is trying to plug the attribute values into some calculations along with

all the weights on each of the edges in order to reach a final classification. However, this is what the neural networks is having a hard time on: how to use the attribute values and the appropriate weights to formulate this massive calculation function. This means that this problem might just be difficult or even impossible to be modeled into a mathematical function. This is actually an underlying property of the dataset. This dataset's attributes are results from lab test, which are very objective values, but the classes are quality scores given by wine experts, which are very subjective values. This nature of the dataset perfectly explains why this relationship is extremely difficult to be mapped into a generalized function, which is exactly what makes this problem interesting. From just the graph alone, the neural networks has a peak at 77.63% accuracy when 60% of the training data is used.

The second graph (figure 2.4) is a performance vs training time graph. The model is actually doing a lot better with just 60% of the training set than with the full training set. For both curves, there is an obvious increase as the training time is increasing until the training time hits about 400. After the training time reaches 400, the neural networks seems to be converging on the full training set already as the curve (orange) is going flatter. On the other hand, the neural networks trained on 60% training set (gray) seems to be converging after training time reaches 400, but it had a leap when training time reaches 600, which actually looks a lot like jumping out of a local minimum. However, the highest accuracy reached is still around 78.45%.

The neural networks might not be doing well because it is not deep enough (the possible functions are not complex enough), so the table (figure 2.5) records the performance while the hidden layers and the number of nodes are modified. As the numbers show in the table, more hidden layers or more nodes doesn't really help the performance much either.

The "jumping out of a local minimum" in figure 2.4 actually inspired me to play around with the learning rate and momentum values. They are not showing here as a table because they didn't help the performance much either. Just like what I explained above, the goal of a neural networks is trying to adjust all of its weights so that these weights can perform the appropriate calculation on the input attributes, and output a good prediction. However, the underlying knowledge about this dataset goes against this nature of neural networks. Maybe a more complicated structure of neural networks can produce a better result, but for now, neural networks doesn't seem like the right fit for this problem because all the accuracies that the neural networks is able to achieve seem to be very low. The highest accuracy the neural networks is able to achieve is around **79.8774%** with 2 hidden layers of 6 nodes in the first one, 9 nodes on the second one, and all the other values at default.
(7.52s for building model and 0.01s for testing on test set according Weka time)

Boosted Decision Tree
The first graph (figure 2.6) is a performance vs training set size graph. The test performance (gray) seems to be increasing nicely with the increase of the training set size, which is reasonable.

The second (figure 2.7) is a performance vs number of iterations graph. The performance has a huge leap when the number of iterations changes from 1 to 3, and then starts to stay around 96% accuracy. This suggests to me that the boosted decision tree is actually doing a very good job on fitting this data. Just like what we talked about for neural networks, this problem might be a difficult problem to solve as a whole (with a massive function), but boosted decision tree, by nature, can dissect the training set into simpler sets to classify, which apparently makes the problem a lot easier as a whole.

The last table (figure 2.8) shows the performance of the boosted decision tree with different confidence factors. The boosted decision tree is able to achieve 97.1399% accuracy with a confidence factor of 0.1, which is a pretty aggressive pruning.

The boosted decision tree seems to have a better approach to the problem than the neural networks. The boosted decision tree tries to dissect the problem into a set of simpler problems to solve, and the neural networks is just trying to solve everything at once. The boosted decision tree can actually

achieve an accuracy of **97.1399%** accuracy with J48 tree and some aggressive pruning (confidence factor = 0.1) and all the other values at default.
(1.8s for building model and 0.02s for testing on test set according Weka time)

Support Vector Machine
The first graph (figure 2.9) is a performance vs training set set graph. Although the test performance curve (gray) seems very bumpy from the graph, it still shows an overall increasing trend. This curve really reminds me of the test performance curve (gray) of the neural networks in figure 2.3, which stays around 70% to 80% thought out the changes of the training set size, whereas this curve only stays around 71% to 74% as the training set size is increasing. The behaviors of the two training performances (blue and orange) seem very similar and interesting, and this behavior can be caused by many reasons, such as unbalanced distribution of data or noisy data. The initial decreasing trend simply describes that the linear kernel function (k=1) is doing worse on separating the data as more linearly inseparable data are used for training. The later increasing trend might indicate that the data added in the later phases are more linearly separable thus bringing up the training performances.

The second graph (figure 2.10) a performance vs polynomial kernel exponent graph. Let's focus on the test performance (orange) as this one matters more to the overall performance. The curve shows an interesting behavior. When the exponent goes from 1 to 2, the test performance boosts up a little bit, which makes sense, and then starts to drop as the exponent increases. This means that as the support vector machine tries to separate the data with more complex kernel functions, it actually ends up misclassifying more instances in the test set. The similar behavior in the training performance (blue) indicates that it is not a case of overfitting. This behavior again reminds me of the the behavior when the neural networks is trying to solve this problem. The model just seems confused about all the data points. With higher complexity, the hyperplane seems to misclassify more instances. This indicates that the support vector machine is on the completely wrong path when attempting to generate a hyperplane to separate the data. Therefore, the more degree is given to the polynomial kernel function, the more mistakes it makes based on this wrong path. Later when the exponent gets even bigger, the performance seems not to drop so much anymore. This is because given such a high degree of complexity, the hyperplane can actually reach a precision at certain regions to successfully separate some of the complex data pints; however, the accuracy is still very low (around 73%-74%)

The behaviors observed from support vector machine in this problem space seem to have many similarities with the ones from neural networks, which actually makes perfect sense. Both of these algorithms are trying to classify all the instances using one massive mathematical function. The neural networks is using the combinations of attribute values and weights to calculate the classification, whereas the support vector machine is using a hyperplane to represent the mathematical function, the goal of which is to separate all the classes in the problem space. This also reconfirms my explanations above for neural networks: this problem is way too complex to be modeled into any function because it is using objective numbers to predict subjective classes. Likewise, support vector machine is not recommended to solve this problem either. From the data collected alone, the highest accuracy achieved by support vector machine is **75.38%** with exponent at 2 and all the other values at default.
(6.58s for building model and 0.14s for testing on test set according Weka time)

K-Nearest Neighbors
The first graph (figure 2.11) is the performance vs training set size graph. The test performance curve (gray) nicely shows that the test performance increases as the training set size increases, which is what's expected. The training performance without cross validation (blue) is constantly at 100% which agrees with the lazy nature of the k-nearest neighbors.

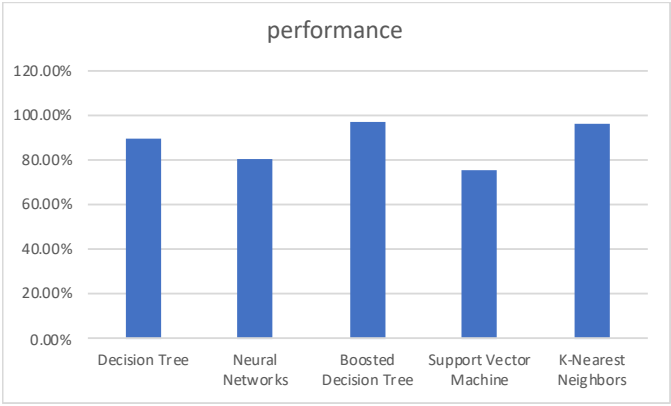
The second graph (figure 2.12) is the performance vs k value graph. The performance actually decreases as the k value increases, which means that the performance gets worse, when the k-nearest neighbors takes more neighbors into consideration when making a prediction. This behavior proves

that this is such a complex problem space that all instances seem to be extremely blended and mixed together. Even looking at 5 neighbors instead of 1 is causing a 14.2% drop in the performance.

Although this is a complex problem space, the k-nearest neighbors seems to be able to solve this with the simplest hyper-parameters (k=1). The nature of k-nearest neighbors with k equals to 1 is that the model doesn't care about how complex the problem space is or how mixed the instances are in the problem space; instead, it simply looks for the closet point in the problem space. This algorithm intrinsically avoids the most difficult part of this classification problem, which is the complexity of the problem space. The way k-nearest neighbors solves this problem makes semantic meanings too. The objective test values are like the identification of a particular wine, and the quality of the wine is how humans perceive the wine. The closer the objective test values are for two wines, the more similar they are in terms of contents. The more similar they are in terms of contents and in the way they are formed, the more difficult it is for humans to tell them apart with just human sense. Thus resulting in similar qualities according to human senses. The k-nearest neighbors is able to achieve **96.22%** with k value at 1 and all the other values at default.
(0.01s for building model and 0.4s for testing on test set according Weka time)

Conclusion

Just like the last dataset, from all the graphs above, it is notable that the cross validation performance does provide a better indicator of the actual test performance than the training performance. This dataset also gives me an interesting experience with all the algorithms while allowing me to understand each one more. From this dataset, it really helped me to notice many differences among these machine learning algorithms. For example, the neural networks is always known to be effective, powerful and able to model complicated problems spaces; however, in this case, neural networks seems to be not that useful in this extremely complex problem space; instead, the seemingly simple algorithm - k-nearest neighbors - solves this problem in the easiest way possible with a high accuracy.



The diagram on the right shows the comparison of the performances for each algorithm on this classification problem. The boosted decision tree and k-nearest neighbors seem to have higher performances among all the algorithms used. The boosted decision tree has 97.1399% accuracy and the k-nearest neighbors has 96.22% accuracy. The boosted decision tree's performance is only around 0.92% higher than the k-nearest neighbors, which is extremely trivial. For the boosted decision tree, the time taken to build the model is around 1.8s and the time taken to test the test set is around 0.02s (Weka time). For the k-nearest neighbors, the time taken to build the model is 0.01s and the time taken to test the test set is around 0.4s (Weka time), which is 20 times more than the test time for boosted decision tree. With all the analysis, the boosted decision tree does seem slightly better than the k-nearest neighbors with an acceptable model building time, a much faster testing time and a slightly higher performance. However, the k-nearest neighbors would also be a great solution for this problem. K-nearest neighbors would definitely work better with more non-noisy training data. Overall, the **boosted decision tree** seems to be the best solution for this problem with 97.1399% accuracy at 10 iterations and confidence factor at 0.1.