Assignment 1:
CS 7637

Ryan Johnson
ryan.johnson@gatech.edu

Abstract- 5 models were tuned on multiple parameters across two different datasets. Each model was able to improve its performance on the underlying dataset through tuning and these behaviors are explained in relation to the underlying mechanisms present in the model.

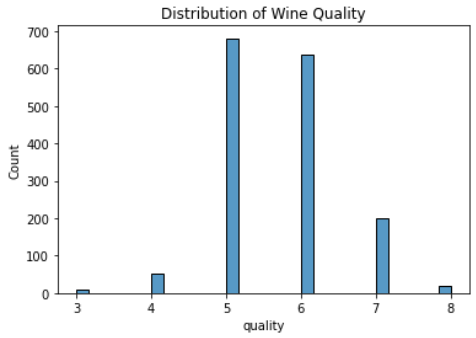
1 DATASETS

1.1 Wine dataset overview and practical interest

I have two datasets, both acquired from UC Irvine's Machine learning repository. The first is a red wine quality dataset. This contains 1,599 red wines sampled from Portugal's Vinho Verde wine region. Each wine has been chemically analyzed by 11 different features describing the wines' chemical make up like how acidic or alcoholic a wine is. These qualities are then labeled with a 1-10 quality score. After consuming the data, I then further narrowed the problem by categorizing wines with a quality greater than or equal to 7 as "high quality". The problem, then, is to try to identify these high quality wines as perceived by human sommeliers with their chemical makeup.

This is an interesting problem, practically, as wine is very difficult for humans to separate by quality. Most average people have difficulty describing or grading how good a wine is and often rely on professional tasters to help guide them to quality wine. There is, of course, some subjectivity in this. Perhaps a model that can accurately predict wine quality could be used as an objective basis for wine quality standards.

1.2 Why is wine dataset interesting for machine learning



From a machine learning standpoint, I find the core challenge of this problem is the uneven distribution of positive and negative labels. In our dataset, only 13.6% of wine reaches the benchmark of a high quality wine. This poses a challenge to machine learning models as we likely don't have a full explanation in the data of all the qualities that may make a wine high quality. This is simply because we don't have that many examples to explain our variability. Because of the difficulty in classification, we will see many of the models presented here struggle to perform beyond the baseline accuracy of labeling all values as poor quality wine and yielding 86.6% accuracy.

1.3 Cancer dataset overview and practical interest

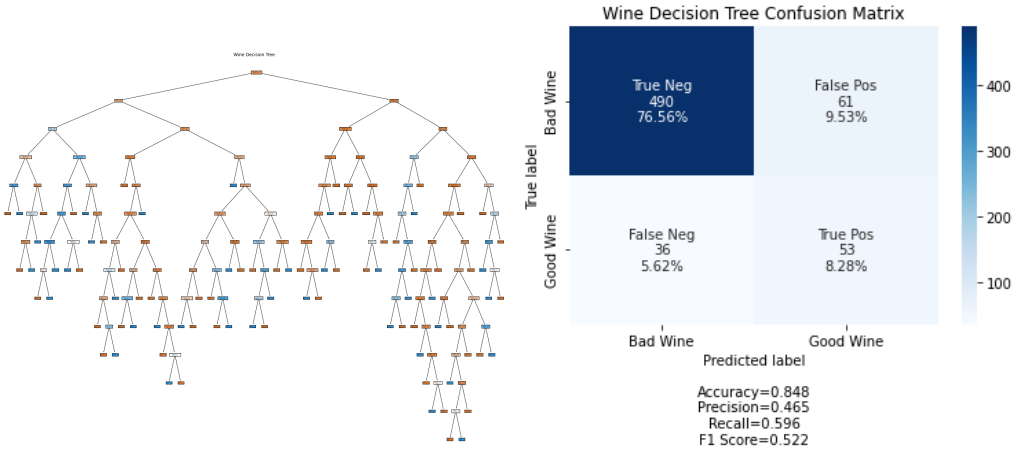
The second dataset is also from the UCI and consists of 32 measurements taken from images of breast tumors. These measurements are then labeled as either a malignant or benign tumor. In order to proceed with treatment, a tumor has to be correctly diagnosed as malignant or benign as their behavior and risk can vary wildly. Clinically, this is often done via an invasive biopsy which will always carry some level of surgical risk. A model that can label these different classes of tumor accurately could help avoid unnecessary surgeries and lead to a more dynamic and agile treatment plan.

1.4 Why is the cancer dataset interesting for machine learning

From a machine learning standpoint, we have a highly dimensional dataset that is linearly inseparable. This leads to relatively sparse data in this dimensional space. While not in the scope of this assignment, it's likely that some form of dimensionality reduction either through a method like PCA or feature selection

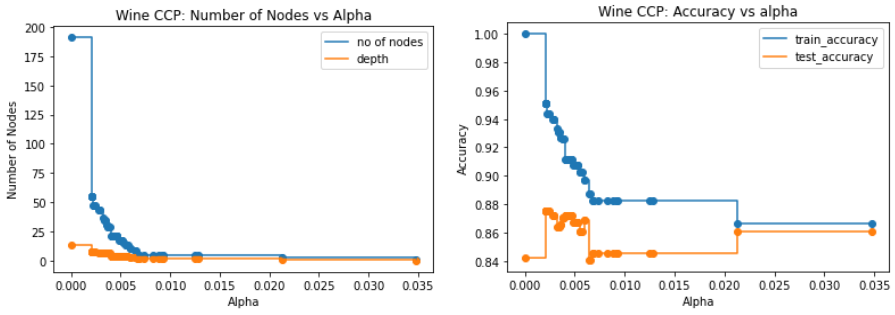
would be beneficial. However, I wanted to test how these different models would behave with highly dimensional data.

2 DECISION TREES



Before looking at tuning the decision tree, let's first see how a tree with no pruning performs. We fit a decision tree to our datasets using a 60/40 training/test split. K-folds were experimented with, but the benefits were marginal, and the behavior of the models was more transparent without them. We can see we've grown a massive tree on our wine dataset and have horrendously overfit our dataset. This makes sense as our tree is fitting many different features into hyper specific niches to perfectly explain our training dataset. However, when we then try to generalize our training set, we actually see worse performance than a naive model which would always predict "bad wine" and yield 86.6% accuracy.

2.1 Pruning the Decision Tree



In order to deal with this overfitting, minimal cost-complexity pruning was implemented. This penalizes our model for growing a large tree based on an alpha value. The larger the alpha, the more the model will prefer a smaller tree. Above, we see as alpha increases the number of total tree nodes decreases. We also see a consistent degradation in training performance. Interestingly, we cut the greatest number of nodes in that very first step. We also see the primary jump in accuracy with this first step up in alpha as well. When we think about the data we're representing, this makes sense. We're dealing with a handful of outlying positive data points. If the tree is trimmed too small, we can see the tree converges with aggressive pruning on the baseline prediction of 86.6%. The optimal pruning is therefore fairly conservative in order to allow for the tree to be flexible enough to accomodate our outliers.

dataset	model	splitting	treeSize	trainacc	testacc	trainTime	predictTime
wine	base	entropy	175	1	0.848	0.009	0.001
wine	base	gini	191	1	0.844	0.009	0.001
wine	pruned0.002	gini	63	0.955	0.877	0.011	0.001
wine	pruned0.005	ent	61	0.946	0.861	0.004	0
cancer	base	entropy	27	1	0.952	0.014	0.001
cancer	base	gini	37	1	0.939	0.005	0
cancer	pruned0.011	gini	9	0.956	0.956	0.011	0
cancer	pruned0.011	ent	9	0.988	0.961	0.012	0

In the cancer dataset, we can see the pruning is much more aggressive with alpha values up to 4x more aggressive. This yields a tree that is much more dramatically simplified to only 9 nodes while improving testing accuracy considerably. Again, this makes sense when we think about the core challenge in our data. The cancer dataset is difficult because of its dimensionality. Our decision tree learner effectively uses the pruning process to perform a type of dimensionality reduction as it limits the number of features included in the model.

2.2 Using Different Splitting Metrics

In addition to pruning. I also experimented with using Entropy vs GINI splitting metrics. These metrics are very similar in that they both look at the probability of our positive class being in a given node. However, where GINI impurity looks at the summation of squared probability j, entropy looks at the sum of probability j scaled to the log of probability j. This means smaller populations can have a larger sway on the splitting metric. I think this weighting can most evidently be seen in how large the base gini vs entropy tree grew to be. With our nodes

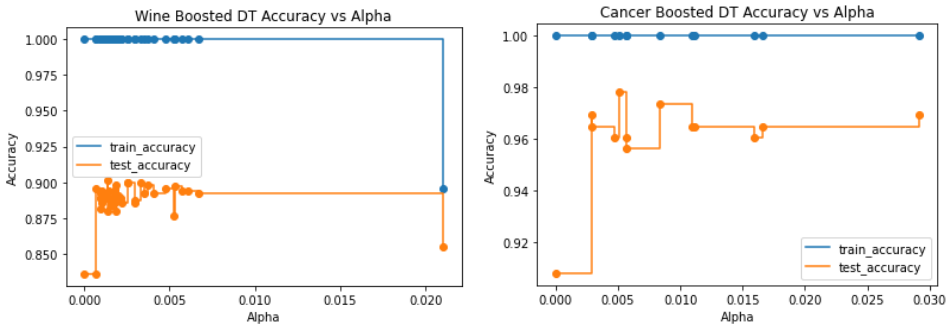
slightly favoring smaller probabilities, we reach a terminal node more quickly as the signal is slightly enhanced. This difference effectively disappears once we start pruning the trees, and the two splitting metrics match more closely. This should in theory come at the expense of compute time due to the need to calculate the log. This isn't really seen in my experiments as the datasets aren't large enough.

3 BOOSTING

To experiment with boosting, we pass through the same sci-kit learn decision tree algorithm used above to an adaboost classifier. This will run through additional iterations with emphasized weights on the harder examples to classify.

dataset	model	pruneAlpha	nEstimators	trainAcc	testAcc	trainTime	predictTime
wine	base			1	0.848	0.01	0
wine	base			1	0.844	0.008	0
wine	base	pruned0.002		0.955	0.877	0.01	0.001
wine	base	pruned0.005		0.946	0.861	0.013	0
wine	boosted	pruned0.001	10	1	0.873	0.115	0.005
wine	boosted	pruned0.001	98	1	0.906	1.182	0.048
cancer	base			1	0.952	0.01	0.001
cancer	base			1	0.939	0.011	0.001
cancer	base	pruned0.011		0.956	0.956	0.011	0
cancer	base	pruned0.011		0.988	0.952	0.011	0.001
cancer	boosted	pruned0.005	10	1	0.961	0.109	0.004
cancer	boosted	pruned0.005	94	1	0.982	0.944	0.03

3.1 Pruning the Boosted Trees

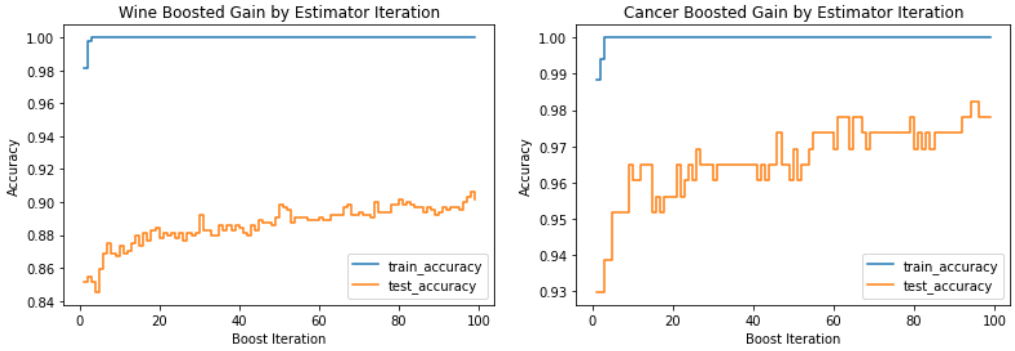


I ran the same pruning optimization from my decision trees with the boosting algorithm. I expected to see more aggressive pruning to yield better performance, but was surprised to see the opposite was true! In fact, it appears the boosted

trees preferred even slightly larger trees with less pruning in both use cases. Upon learning about the Adaboost classifier in depth, however, I think it makes quite a bit of sense. The Adaboost classifier is already generating mini trees called "stumps" to attempt to correct for the previous trees' shortcomings by applying new weights to the data. If we prune these stumps, we lose the ability to allow our trees to be specific enough to make corrections and therefore bias our model towards preferring minimal pruning.

3.2 Boosting Iterations

More in line with expectations, I experimented with finding the optimal number of boosting iterations across both use cases. In theory, we should see continuously improving performance with each new boost iteration.



This is in fact exactly the real observed behavior seen in the models! While there is some noise from iteration to iteration, we see steady improvement across both our problems to the tune of about 3% in improved accuracy. In the case of the wine quality classification, each iteration is effectively emphasizing the quality data points making the overall model more sensitive and capable of identifying quality wines compared to the base decision trees.

Similarly, we see a very impressive classification performance in the cancer problem with our model approaching perfection in the test dataset. In the theme of dimensionality reduction being beneficial to our model performance, Adaboost performs a type of reduction even more aggressively than pruning the trees. By generating and weighting decision 'stumps' the algorithm can effectively prioritize the most important features and deprioritize less important ones. Quite the accomplishment!

4 NEURAL NET

I experimented on a multilayer perceptron which was trained looking at the effects of learning rate and momentum as well as by training epochs. Unlike the decision tree and boosted model, the datasets were scaled in order to remove any incidental bias when applying weights to the different perceptron nodes.

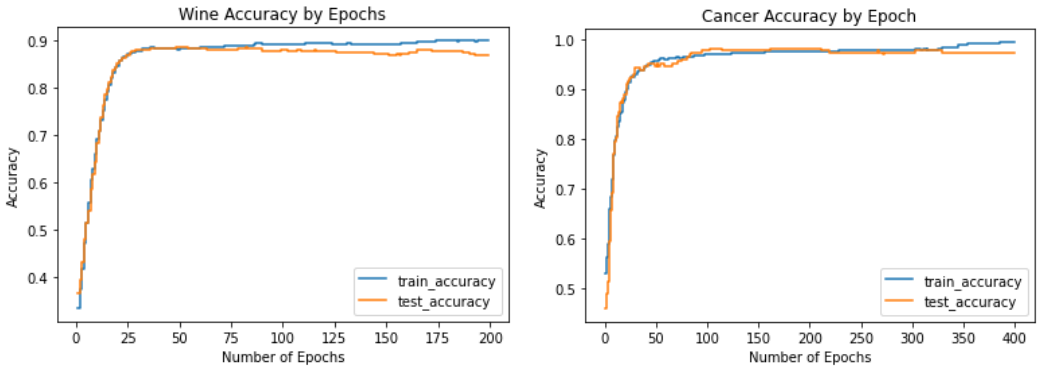
4.1 Learning Rate and Momentum

dataset	learningRate	momentum	epochs	trainAcc	testAcc
wine	0.01	0	50	0.899	0.869
wine	0.01	0.5	50	0.899	0.869
wine	0.01	1	50	0.899	0.869
wine	0.1	0	50	0.909	0.856
wine	0.1	0.5	50	0.909	0.856
wine	0.1	1	50	0.909	0.856
wine	0.3	0	50	0.908	0.842
wine	0.3	0.5	50	0.908	0.842
wine	0.3	1	50	0.908	0.842
wine	0.5	0	50	0.867	0.861
wine	0.5	0.5	50	0.867	0.861
wine	0.5	1	50	0.867	0.861
wine	0.001	0.9	49	0.881	0.888

dataset	learningRate	momentum	epochs	trainAcc	testAcc
cancer	0.01	0	50	0.988	0.982
cancer	0.01	0.5	50	0.988	0.982
cancer	0.01	1	50	0.988	0.982
cancer	0.1	0	50	0.997	0.969
cancer	0.1	0.5	50	0.997	0.969
cancer	0.1	1	50	0.997	0.969
cancer	0.3	0	50	1	0.965
cancer	0.3	0.5	50	1	0.965
cancer	0.3	1	50	1	0.965
cancer	0.5	0	50	0.613	0.649
cancer	0.5	0.5	50	0.613	0.649
cancer	0.5	1	50	0.613	0.649
cancer	0.001	0.9	102	0.971	0.982

We see relatively different behavior between the wine and cancer datasets. Most interestingly, in the cancer problem, we see a relatively high learning rate of .5 across all levels of momentum performs very poorly. This is because the model just doesn't have the level of granularity to settle into a performance optima. Every time it makes a tweak, it over adjusts and can't settle closer into a better performing model. Meanwhile, our smaller learning rates are able to ease into very strong performance.

4.2 Epochs



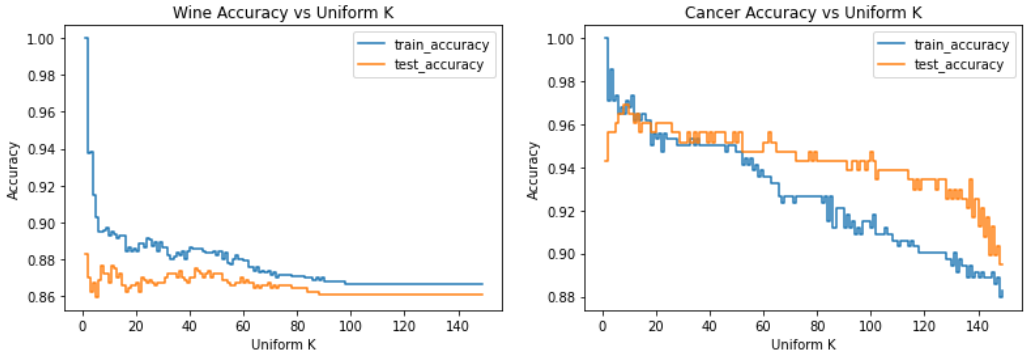
An epoch is one complete pass over the training dataset. Each pass our model tweaks the weights of our perceptrons in order to better fit the training data. We can see very similar behavior in both problems here with training accuracy continuing to increase with every additional epoch. However, particularly in the wine problem, we see overfitting is achieved relatively quickly as testing accuracy falls away! In both problems we actually terminate before reaching complete convergence on the optimal network for the training set. We could let our network keep training until it reaches this optima, but given we can see clear overfitting occurring in both models, it makes sense to limit our training epochs.

5 K-NEAREST NEIGHBORS

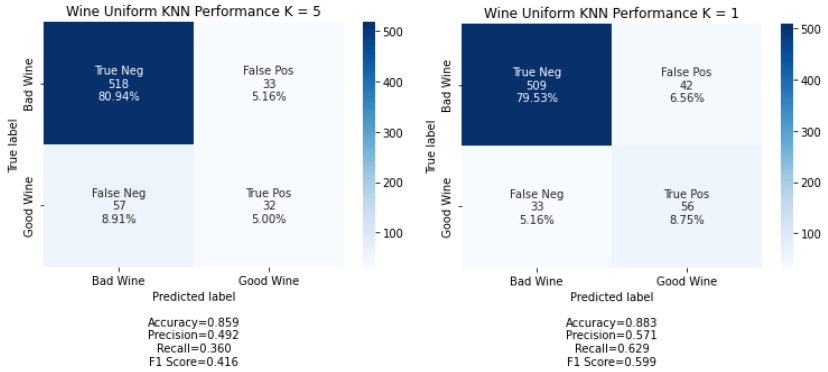
dataset	k	weighting	trainAcc	testAcc
wine	5	uniform	0.903	0.859
wine	5	weighted	1	0.878
wine	1	uniform	1	0.883
wine	8	weighted	1	0.902
cancer	5	uniform	0.974	0.961
cancer	5	weighted	0.974	0.961
cancer	8	uniform	0.965	0.969
cancer	10	weighted	1	0.974

A KNN model was implemented and tuned to two parameters: # of points in k and a distance vs uniform weighted k. The above show the results of the models after tuning. As always a 60/40 training/test split was used. Additionally, because we are primarily interested in distance from point to point, we standardize our data to the same scale. This way, we don't accidentally make one feature more important than any other arbitrarily.

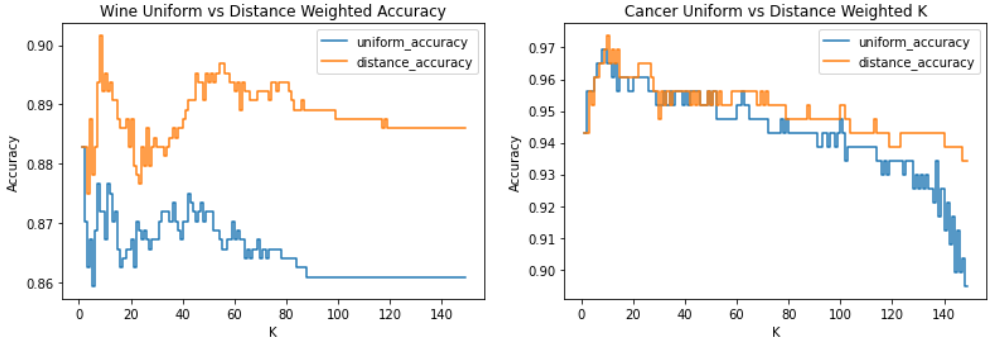
5.1 K and Uniform Weighting



We have two very different behaviors in the two problems as the # of neighbors increases in a uniform k weighting. In the wine dataset, we actually reach the optimal number of neighbors at 1. How does that make any sense? Well, with our positive values behaving as outliers, they aren't necessarily clustered together, and even where a couple may be, there could be many more negative values close by. So, the second we start to include more neighbors, we start including more negative examples making it harder to correctly identify a quality wine. This can be seen in detail if we look at where the model is missing. When K = 5, we have a much higher false negative rate. By altering K to 1, we become drastically more sensitive to positive data points, though there is some overfitting here as our false positive rate increases slightly.



5.2 K and Distance Weighting

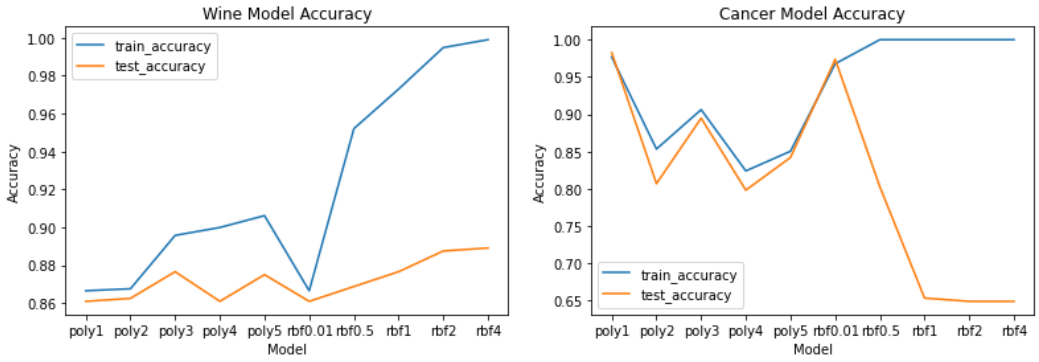


Perhaps just as interestingly, we see the wine dataset behavior change once we weight our neighbors according to their distance. Here, we actually see a significant spike in performance above uniform weighting. This is because we keep the highest weighting at k = 1, but we still allow our neighbors to weigh in. This has the effect of peaking to see if there are any other positive examples

nearby without unduly too heavily weighting an equivalent number of negative data points. This then has the effect of further reducing false positives while improving the models ability to recognize true negatives.

6 SUPPORT VECTOR MACHINES

A support vector classifier was implemented and tuned to two different kernels with different hyper parameters. As usual a 60/40 training/test split was used. Similarly to our MLP and KNN models, the data was scaled to avoid accidentally favoring one feature over another.



6.1 RBF vs Poly Kernels

Two different kernels were used in the support vector classifier, and each kernel seems to favor an opposite model. The "poly" kernel uses a polynomial function with a tuned degree to split our data once the data is projected to a dimension where it can be separated. In the wine dataset, we see marginal improvements in our testing set, but I think this kernel is poorly suited to the outlier challenge. Conversely, it appears a first degree polynomial actually works quite well at separating the cancer data.

The RBF kernel is used for non-linear problems and this can be seen by the differing performance above. In the cancer dataset, where our poly kernel shows a fairly linear solution, the RBF kernel does quite poorly as gamma increases. However, the wine data, with its scattered outliers, seem somewhat more receptive to the 'circling' of data the rbf kernel is capable of.

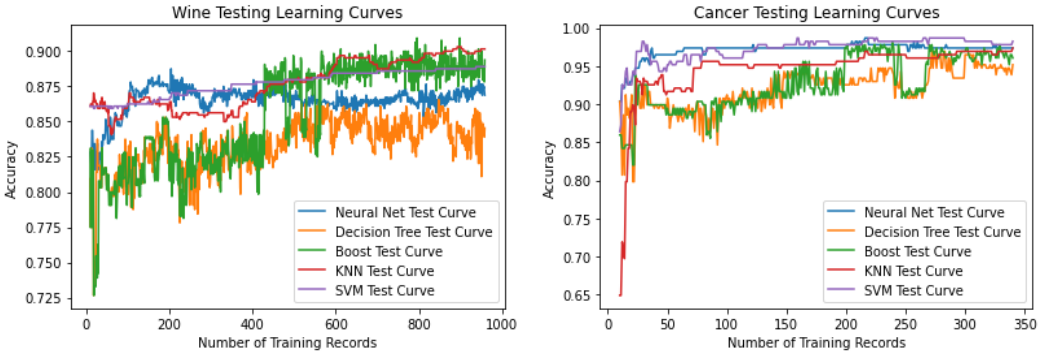
7 MODEL ANALYSIS

5.1 Performance Across all Models

Dataset	Model	Training Acc	Testing Acc		Dataset	Model	Training Acc	Testing Acc	
wine	MLP	0.881	0.888		cancer	MLP	0.971	0.982	*
wine	SVM	0.999	0.889		cancer	SVM	0.977	0.982	*
wine	KNN	1.000	0.902	*	cancer	KNN	1.000	0.974	
wine	BOOST	1.000	0.906	*	cancer	BOOST	1.000	0.982	*
wine	TREE	0.955	0.877		cancer	TREE	0.988	0.961	

When we take the best tuned model from each section we've looked at we can see overall performance. After conducting experiments on each of the models I'm overall quite surprised at how well the models performed across the board. At a high level, based on accuracy and time to run, it looks like KNN performs best for wine and SVM for cancer. If time and space allowed, I think one way to better tune and improve the models would be to take a closer look at other accuracy metrics like F1 score. Especially in the wine dataset, diving further into the sensitivity and specificity of the models could draw further comparisons than have already been raised.

5.2 Model Learning Curves

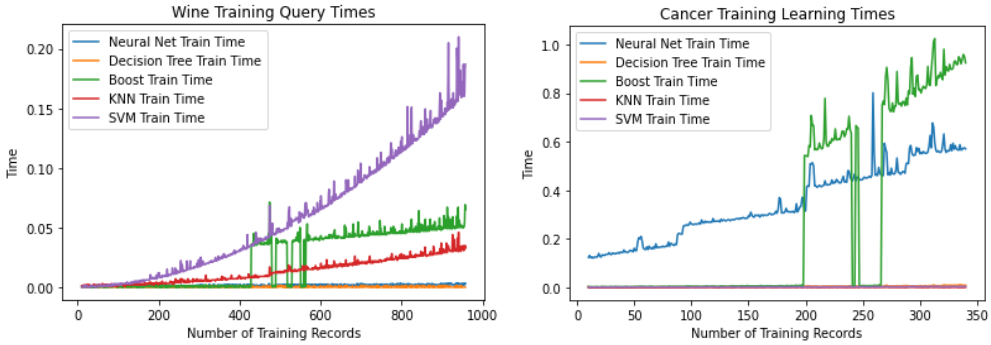


Above are the learning curves for both problems across the best versions of the experimented models after tuning. We see the testing accuracy based on being trained on an increasing number of records. There's a lot of really interesting information held in these charts! In the wine dataset, we can see the neural network trains the fastest, but also begins to overfit. Meanwhile, our boosted decision trees don't begin to really see an advantage over a single tree until a stark increase where the boost takes a large leap above the decision trees. I think this is because some of the most difficult to predict records are in this range.

Until those appear in the training dataset, there may be limited upside to boosting. Meanwhile, we can see the KKN model initially degrades in performance before steadily improving as more records are added. This makes sense as we simply don't have many examples in the training set to indicate a positive neighbor. As more outliers get added, there's more potential for one to neighbor an outlier in the training set.

In the cancer set, we see all models rapidly improve over even the first 50 records in our training dataset. I believe this relates to the separability of the datasets in high dimensional space. It's telling that our support vector classifier performs best and does so early as it should be best suited to separating highly dimensional data as long as the first records are representative of the rest of the training set.

5.3 Model Learning Times



When looking at training times, we see the multilayer perceptron is the most consistently computationally expensive model. This makes sense given the backpropagation task over many iterations. Most surprising was the KNN training time was so long in the wine problem. The model should in theory be the quickest to train (and is indeed the fastest in the cancer dataset). Even after controlling for several different untuned parameters, I was unable to find a definitive explanation for this behavior. The stark change in performance in the boosting algorithm in both problems was puzzling to me. However, the jumps in performance neatly match with the jump in training time. I think this is further evidence that the difficult cases are added into our training sets at these points. Their introduction means more potential upside to boosting, but also more work to do as the boosting algorithm tries to correct previous trees.