Supervised Learning

Cody Phrampus
cphrampus3@gatech.edu

1 Dataset 1 - Wine

This dataset contains 13 features describing wines from the same region in Italy but from 3 different cultivars. The classification of this dataset consists of mapping the 13 inputs to an integer 1 to 3 indicating the location. The set consists of 178 instances that span the classes with relative balance, the least covered class having around 50 instances and the most having around 70 (Wine). This dataset was chosen in order to foist the curse of dimensionality on the algorithms by providing many features, but not enough data to cover them adequately.

1.1 Decision Tree

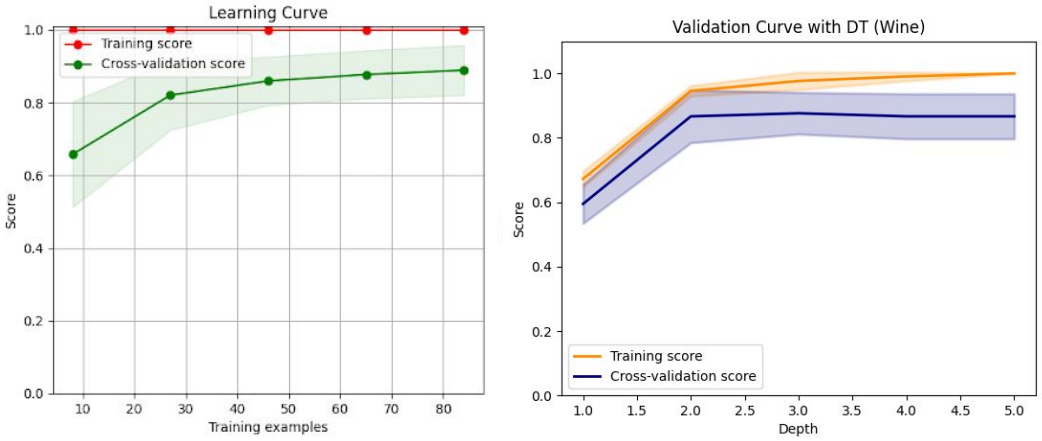

Figure 1, 2—initial learning curve (left) and validation curve over depth(right)

Looking at the initial learning curve, Figure 1, it could be that the tree is overfitting, due to a 100% training score, but the cross validation score is much higher than one would expect with an overfit tree, so it still seems plenty capable of predicting the rest of the data. Attempts to tune the tree via pre/post-pruning, the number of samples needed to split, etc. did not show any real improvement to the classifier's performance. This is strange, some amount of overfitting would be expected and could be combated with a slightly more generalized tree, trading some training points for testing points. However, the initial tree, allowed to grow unbounded, continued to perform the best among the candidate models. The validation curve showed a slight bump around depth 3, so, while this candidate did not perform the initial model, it was kept for scoring comparison out of curiosity. The most likely explanation for this performance is the

small, well defined universe of the problem allowing for a tree that should overfit to still "understand" the problem well enough to predict with high accuracy.

1.2 (Ada)Boosted-Decision Tree

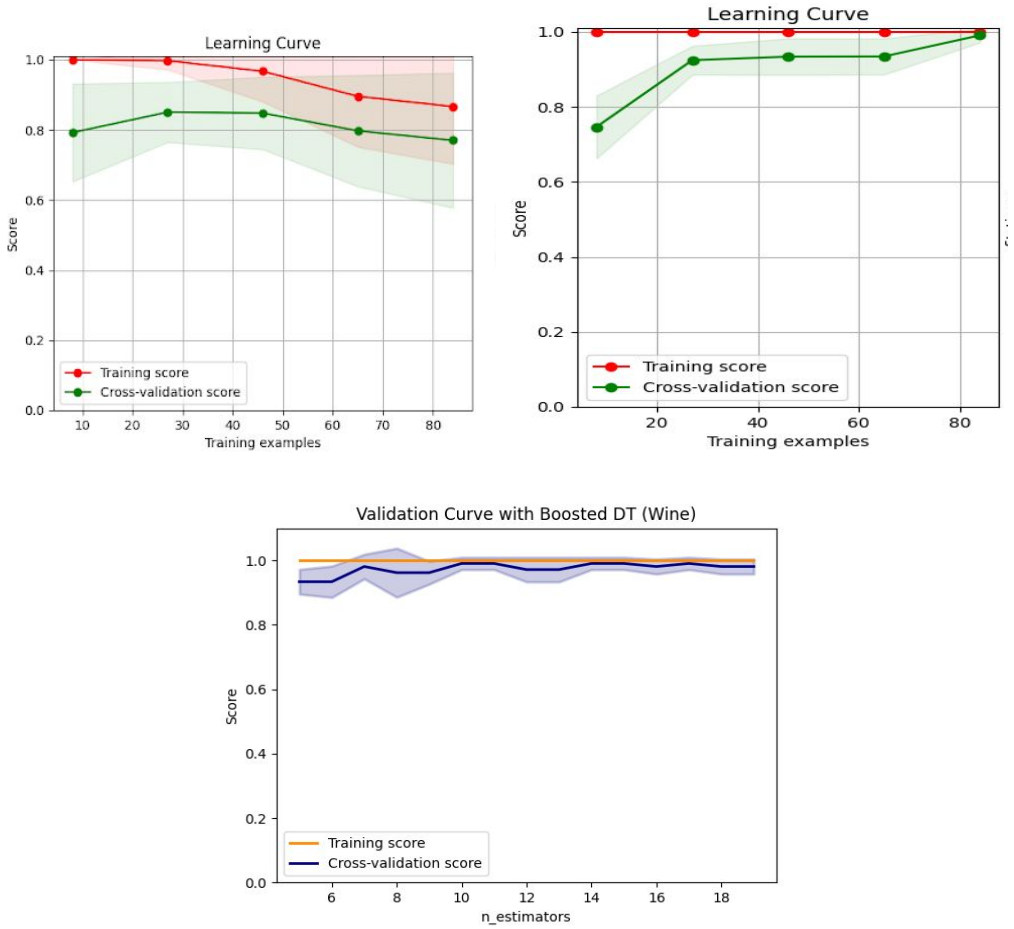

Figure 3, 4, 5—initial(left) vs tuned(right) learning curves, validation curve over number of estimators (bottom)

The initial run of the boosted decision trees performed well with a cv score just under 80%, but it should be possible to get some more improvement. The main feature of boosting to look at is how many trees are being thrown at the problem, but it is also important to see if the underlying trees can be improved, so fewer may be needed. THe underlying tree was tuned with max_depth and ccp_alpha. The depth was set to 2, compared to the original 1, and the alpha was set to 0.1. After this, the number of trees needed was found to be 10, as seen in Figure 5 above. These improvements brought the testing score to 99%. It is possible that a further refined tree could be found, but boosting would prefer more simpler trees to avoid overfiting.

1.3 SVM

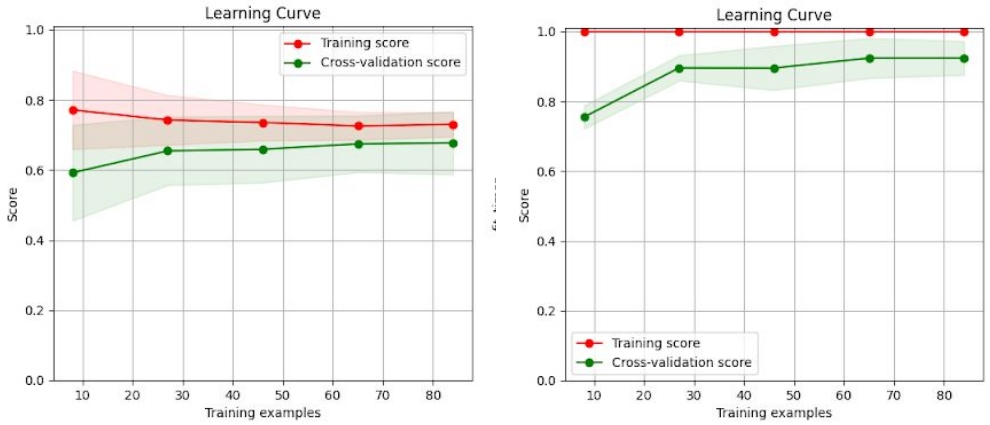

Figure 6, 7—initial(left) vs tuned(right) learning curves

Kernel	**Training**	**Testing**
Linear	*1.0*	0.95
Poly	*0.71*	0.75
RBF	*0.74*	0.70
Sigmoid	0.30	0.21

Table 1—SVM kernel tuning

As with boosting above, the initial curve is good, this time only getting to around 70% on both training and testing, remaining nearly parallel, but with a very distant convergence point. The initial curve uses the RBF kernel, the default in sklearn, but it can be seen in Table 1 that the linear kernel performs significantly better. The smoothing parameter "C" did not provide any additional benefits, so the default value of 1 was kept. If even better performance were desired, the best area for improvement would likely be a custom kernel leveraging feature domain knowledge to ensure the best separation of the classes in the feature space. This knowledge would likely present in the form of the important parameters and possibly the removal of some data there may cloud the classification.

1.4 KNN

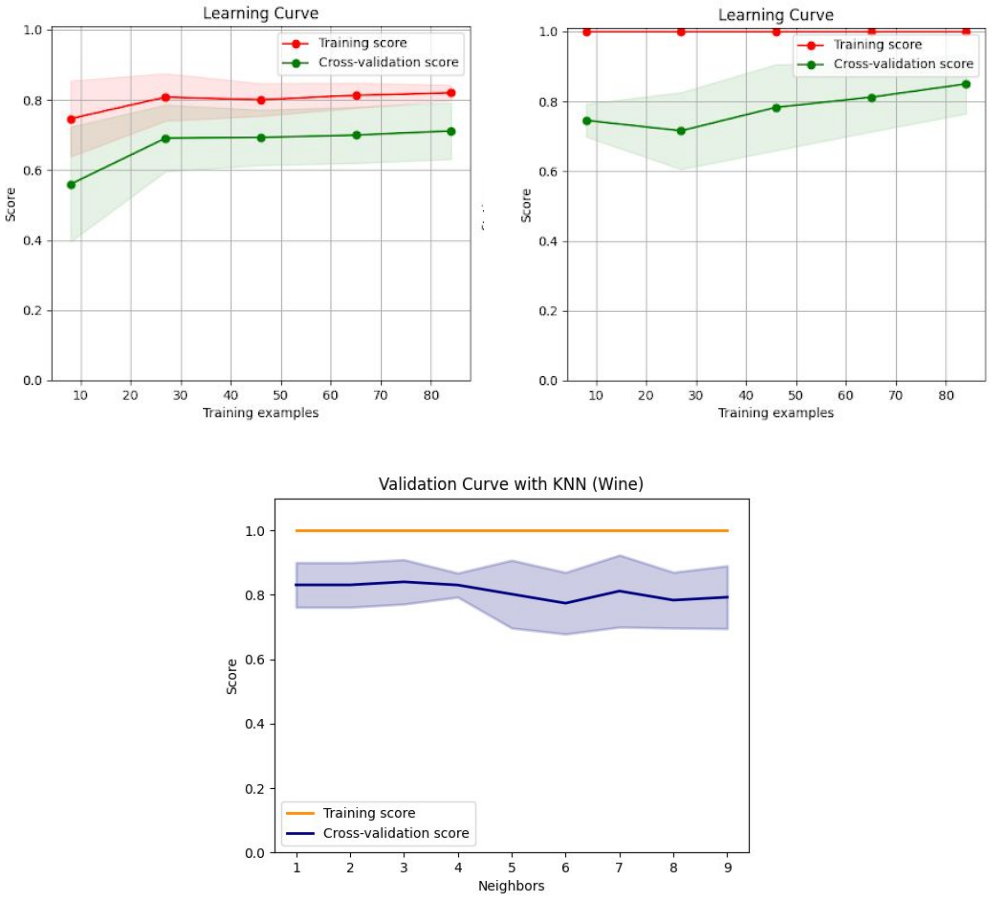

Figure 8, 9, 10—initial(left) vs tuned(right) learning curves, validation curve over number of neighbors (bottom)

Looking at the initial graph, it seems that after about 25 learning examples, the model ceases to improve. The default for KNN here was 5 neighbors of uniform weight. This provides two paths forward: 1. Adjust the number of neighbors and 2. Look at other weighting options, uniform weighting is much more likely to throw the prediction off due to a "distant" point having an equal say despite lack of similarity. Performing these changes yielded the results seen in Figure 9. The final model consisted of 3 manhattan-distance-weighted neighbors, bringing the cross validation score to around 85% and the training score to 100%, likely due to the limited number of classes behaving well in terms of spatial separation to ensure points that are alike have highly weighted votes and those that are distant are weighted lowly. If this were to continue, the next steps would be to leverage domain knowledge for a better, custom distance calculation in order to weight/ignore certain dimensions.

1.5 ANN

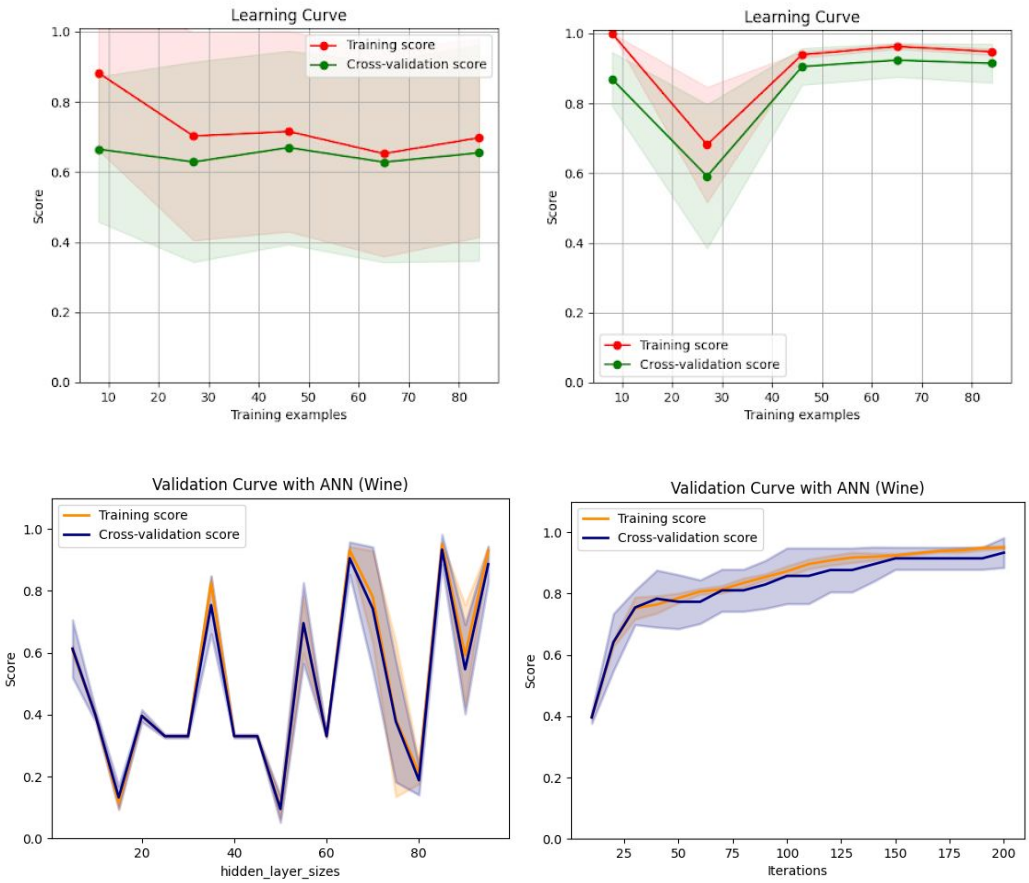

Figure 11, 12, 13, 14—initial(top left) vs tuned(top right) learning curves, validation curve over nodes in hidden layer (bottom left) and learning curve over iterations (bottom right)

The scoring on the initial model remained fairly consistent over the number of examples, with the exception of the initial drop in training score likely due to an unlucky next batch which did not fit well on the model thus far. In order to tune the network, the major point to look at was hidden layer depth and width, defaulting to a single layer of 100 nodes. The validation curve over hidden layer size showed that 85 nodes would be best, while still keeping a single layer of nodes. Tweaks to the learning rate, momentum, and alpha values showed no improvements. The model, default except for 85 nodes, ended up with 95% on training and 92% on testing, with the same unlucky second draw due to the determinism of the splits for the curve plotting. If this were to continue, I believe that the best next steps could be to examine multi-layer networks and the number of iterations to maximize learning but prevent overfitting. These were not examined yet as the results were deemed good enough and other models needed work.

1.6 Test Scores and comparison

Kernel	Test Score	Runtime (Seconds)
DT (overfit)	*0.92*	0.00
DT (pruned)	*0.76*	0.00
Boosting	*0.86*	0.03
SVM	*0.93*	0.03
KNN	*0.72*	0.00
ANN	0.93	0.16

Table 2—Dataset 1 model test scores and runtimes

As can be seen in Table 2 above, the tuned models performed fairly well overall on the held out testing set, the exceptions being the pruned decision tree and KNN. As mentioned prior, the pruned decision tree was included purely out of curiosity to see if the tree which should overfit would suffer on the testing data, but that is not the case likely due to the small universe of the problem as discussed in the decision tree section. KNN is likely suffering from the curse of dimensionality to some degree, as well as its inherent bias toward believing all features are important, where the others could decide to ignore them, either by zeroing out their weight, in the case of ANN, or ignoring the dimension entirely for the others.

2 Dataset 2 - Wine Quality

This dataset contains 11 features describing the quality wines from the north of Portugal. The classification of this dataset consists of mapping the 11 inputs to an integer 0 to 10 indicating the quality of the wine. The set consists of approximately 6,500 instances that span the classes without balance. There are no instances covering the classes 0-2 or 10 and the least populated class, 9, has only 5 instances compared to the most populated, 6, with over 2,800 (Wine quality). This set was chosen due to having a similar number of features as the first, but more data to adequately cover the classes. However, this is not truly the case due to the imbalance of the data mentioned previously. The intent of the dataset was to provide a counterpoint for dataset 1 by having a similar number of features, but more data to adequately describe them.

2.1 Decision Tree

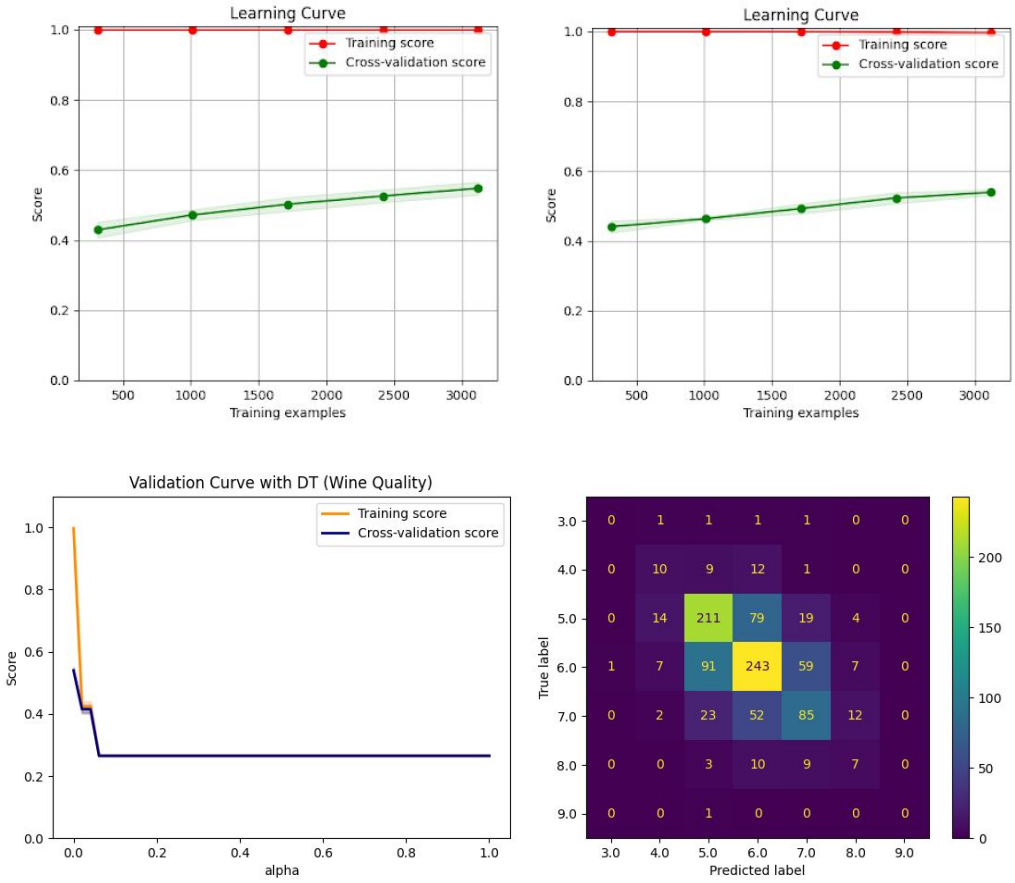

Figure 15, 16, 17, 18—initial(top left) vs tuned(top right) learning curves with weighted f1 score, validation curve over alpha (bottom left) and confusion matrix (bottom right)

Looking at the initial curve, what look to be clear signs of overfitting can be seen with 100% training score and a cross validation score in the 50's. However, typical tactics for decision trees yield no good results due to the imbalance in the class representation. The issue this poses is the fact that trimming the tree in some way regresses it back towards the more populous classes. It can be seen in the confusion matrix that the correct label is chosen most of the time for the 3 main classes of the data, 6, 5, and 7 respectively, with the remaining predictions falling in line with class distribution. However, being able to reduce a 7 class problem to only 3 should yield better results. The final model ended up being a 18 depth tree, as opposed to the 25 depth in the initial run. Further tuning would likely consist of checking parameters for required number to split and number of leaves, as well as the possibility of dimensionality reduction.

2.2 (Ada)Boosted-Decision Tree

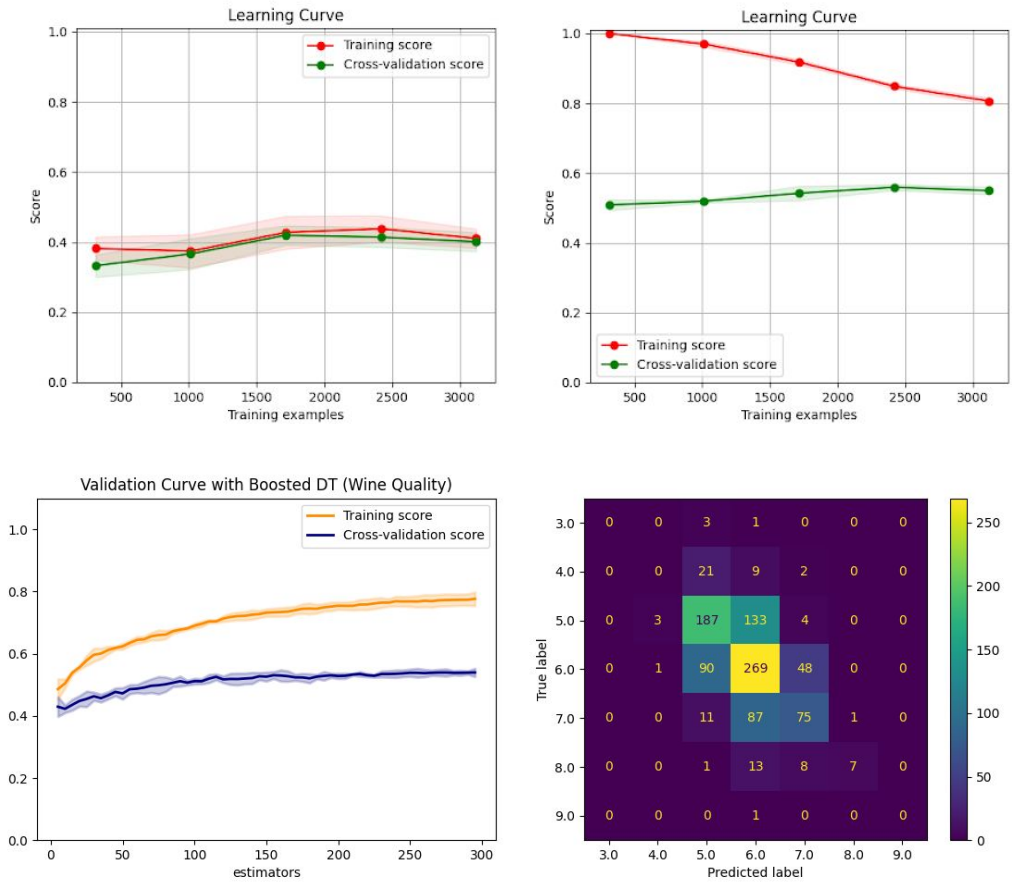

Figure 19, 20, 21, 22—initial(top left) vs tuned(top right) learning curves, validation curve over number of estimators (bottom left), confusion matrix (bottom right)

Looking at the initial graph, the testing and training scores stayed pretty much the same regardless of examples. However, in the same vein as the first dataset, the number of estimators and the estimators themselves can be examined. The first step was to expand the trees a bit to allow them to describe a bit more, which would hopefully also cut down on the number of estimators needed. The trees ended up being of depth 5, which seems large for the case of boosting, but gave the best performance. The number of estimators ended up needing to be capped for the sake of time and computing power to 500, over 1000 learners showed minimal progress towards the distant convergence at 70%. Further tuning could be done to the underlying estimator in terms of leaf size and when to split to reduce the estimators needed, or more and more estimators could thrown at the problem to leverage the fact that boosting tends to do better and better in terms of both scores, but it still may run into issues of class representation among the data.

2.3 SVM

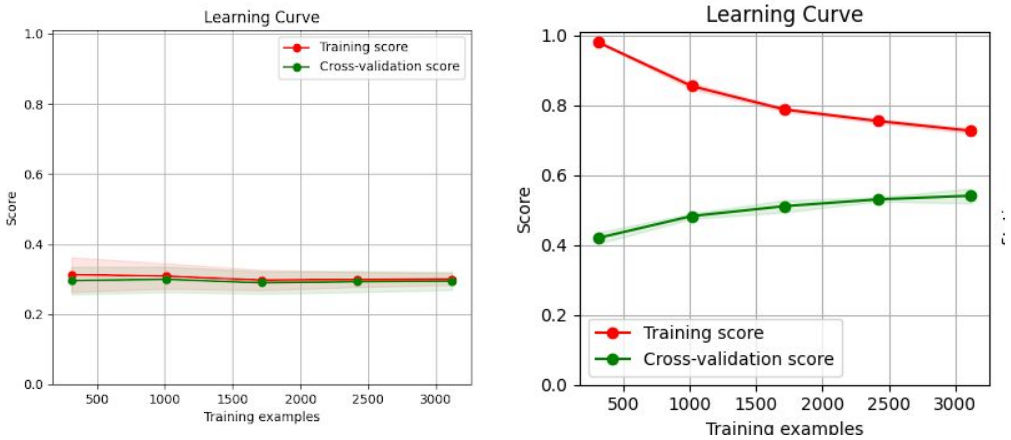

Figure 23, 24, 25—initial(left) vs tuned(right) learning curves

Kernel	**Training**	**Testing**
Linear	*0.54*	0.51
Poly	*0.44*	0.43
RBF	*0.44*	0.43
Sigmoid	0.38	0.37

Table 3—SVM kernel tuning

Using the default parameters, C of 1 and an RBF kernel, the training and testing scores are basically on top of each other at around 30%. Looking at Table 3, it seems that a linear kernel may again be the best for the problem. However, this is neglecting other parameter tuning which is more easily done via coarse grid searches over each kernel. The grid search, which included C for all kernels for smoothing and gamma for rbf, gave that an rbf kernel with a C of 10k and a gamma of 0.001 would yield the best results. Using these parameters, the trajectory of the model shows that given enough examples, it would converge around 65%. Further tuning would likely require more specific domain knowledge in order to be able to construct a kernel better able to separate the data, perhaps in addition to removing extraneous dimensions to make the task easier.

2.4 KNN

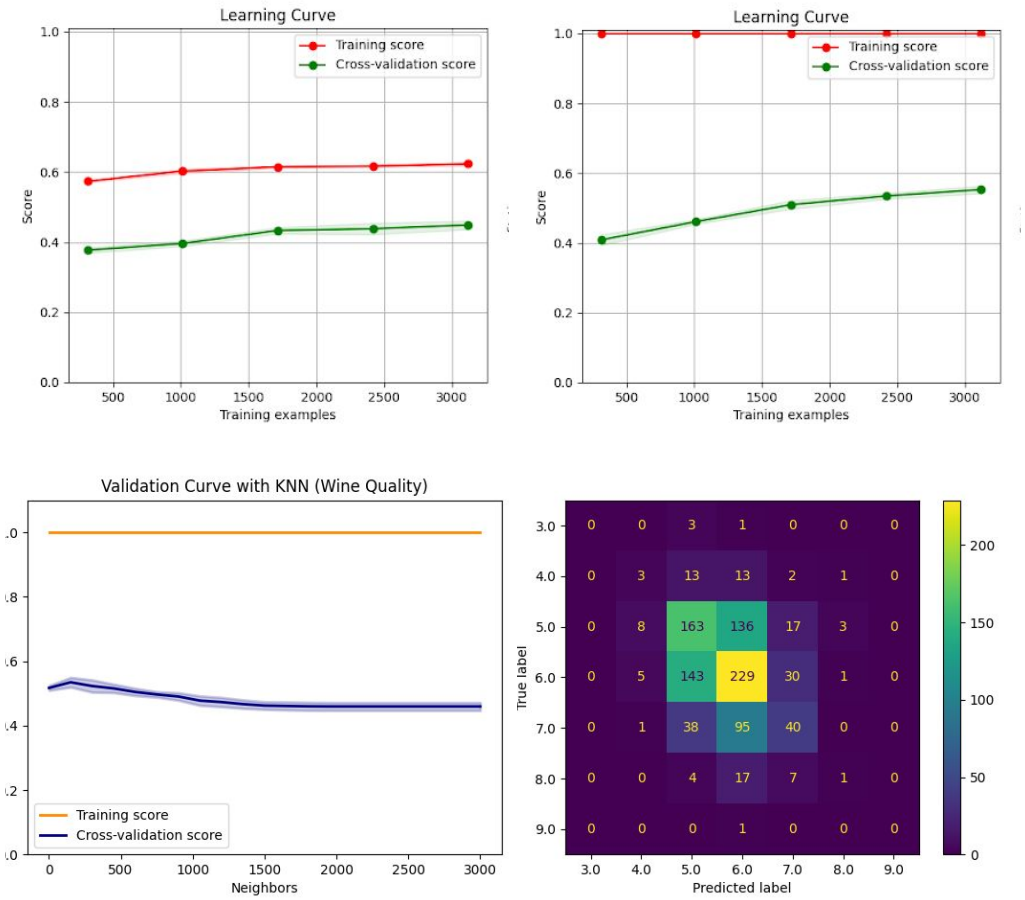

Figure 26, 27, 28, 29—initial(top left) vs tuned(top right) learning curves, validation curve over number of neighbors (bottom left) and confusion matrix (bottom right)

The initial run showed mediocre performance with the training score around 60% and testing around 40%. The default parameters use uniform weighting, which can lead to some imbalance when dealing with high dimensional spaces due to equal say, as mentioned previously. The first thing done to combat this is to switch to distance weighting to allow closer items to have more say, since they are more likely in the same class space. The second source of improvement is the number of neighbors to consider, as more may help or hinder the calculation, especially in terms of items close due to a "useless" dimension. Using these two improvements, 14 manhattan weighted neighbors and a leaf size of 10, the model was able to get to a 55% weighted f1 testing score which is not great, but is better than before. The most likely issue with the model is the curse of dimensionality and the fact that all dimensions will be considered with equal weight. To combat this, custom weighting of the dimensions would be necessary based on specific domain knowledge in order to compress/expand certain dimensions. The confusion

matrix also shows that it is getting the correct label fairly often but is still very close, especially in terms of predicting 5 or correctly getting 7.

2.5 ANN

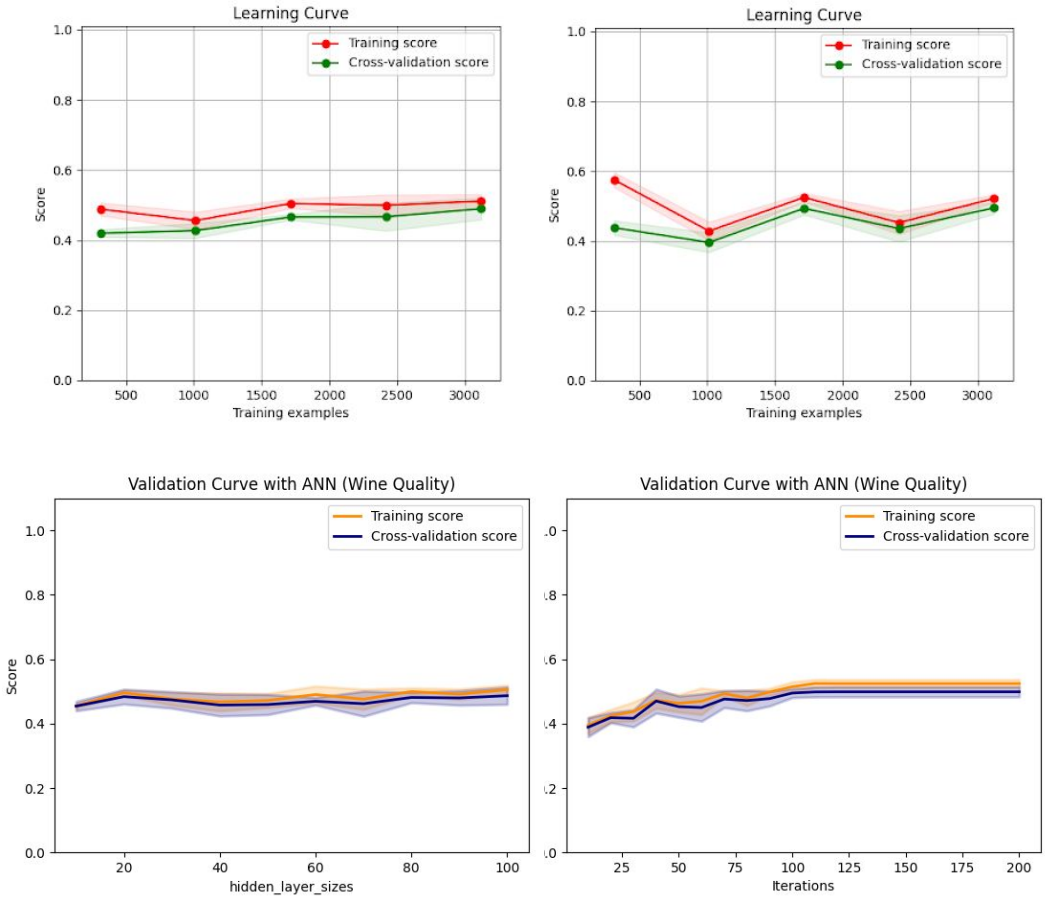

Figure 30, 31, 32—initial(top left) vs tuned(top right) learning curves, validation curve over hidden layer width (bottom left) and learning curve over iterations (bottom right)

The initial curve with default parameters, a single layer with 100 nodes, does not perform that well, getting just about 50% score for both training and validation. Grid searching and validation curves showed that the performance would be better with over 100 nodes, leading to a hidden layer size of 290. This only yielded about a 2% increase. Further tuning for momentum, an additional hidden layer, and alpha did not improve the score either. The model reached best performance around 110 iterations and showed no steep drop indicating overfitting. The most likely cause for the poor performance is the number of features and the classes being inadequately covered by the available data. The best thing to "tune" to improve

performance would likely be the data themselves in order to remove extraneous dimensions that cloud the underlying function as well as better covering the class space, if possible.

2.6 Test Scores and comparison

Kernel	Test Score	Runtime (seconds)
DT	*0.57*	0.05
Boosting	*0.55*	7.67
SVM	*0.55*	70.74
KNN	*0.60*	0.04
ANN	0.51	4.44

Table 4—Dataset 2 test scores and runtimes

The algorithms for dataset 2 performed relatively closely in terms of score and further showed the runtime differences due to the larger volume of data. It is clear that KNN came out on top in both score and runtime. This seems to indicate some amount of clustering of the data within the full feature space and that, to some degree, these data near a given point are, indeed, similar. However, as mentioned in the analyses above, the data are imbalanced towards 2 classes, leading to a lack of coverage of the class space and there may be additional dimensional problems confusing the data. As the dataset write up suggests, feature selection/weighting would likely be beneficial by removing the "haziness" around the class in the reduced feature space (Wine Quality).

3 Comparing Datasets

These 2 datasets were chosen due to having a comparable number of features, one with relatively few examples and one with many more with the expectation that the one with fewer examples would fall short due to coverage limitations. However, this expectation was flawed due to balancing the feature space but not the class space. The one with a small number of well defined/covered and balanced classes outperformed the other with many more classes, not all well covered and highly imbalanced. A better side by side comparison would have locked both in sync and varied only the number of examples given in order to illustrate the curse of dimensionality. A better way to do this may have been to isolate the 3 main classes of dataset 2.

4 References

1. UCI machine Learning Repository: Wine data set. (n.d.). Retrieved February 18, 2021, from http://archive.ics.uci.edu/ml/datasets/Wine
2. UCI machine Learning Repository: Wine quality data set. (n.d.). Retrieved February 18, 2021, from http://archive.ics.uci.edu/ml/datasets/Wine+Quality