Assignment 1

Joe Kraemer

I. Datasets

A. Diabetic Retinopathy

This dataset contains features extracted from Messidor images and the goal is to predict whether an image contains signs of diabetic retinopathy [1]. There are 19 numeric features and about 1100 data points. This data set does not have any trivially linearly dependent features.

B. Red Wine

The UCI Red Wine dataset has 11 features and 1600 data points [2]. The features describe various chemical attributes of the wine such as "citric acid", "residual sugar", and "chlorides". The goal is to predict the quality of the wine with a score of 1 – 10. However, there are only 6 different scores represented in the dataset. This data set does not have any trivially linearly dependent features.

Initially I checked to see if there were any strong correlations between the different features. I expected to see strong correlations between the different types of acid, and the overall pH level because this is a measure of acidity. I also expected a strong correlation between alcohol content and density because I know homebrewers measure alcohol content of their beer using density measurements. As I predicted alcohol content and density does have a quite high correlation of 0.5 but "fixed acidity" was actually higher with .67. Also as predicted "citric acid" and "fixed acidity" have high correlations with pH but "volatile acidity" does not.

II. Data Pre-Processing

We use the package imblearn to balance the datasets by using the Random Oversampler. The data is scaled based on the training data and the applied to the test data so that the neural networks can use the data properly. Because we are only scaling based on the training data, we are not biasing to the test data.

Even though the datasets are now balanced, we still use a weighted f1 score to completely eliminate any partially unbalanced fold. Although the dataset is now balanced, we will still used stratified k folds to ensure that there is an equal number of labels in each fold. We split the dataset into a test and train set. The training set will be used for hyperparameter validation.

III. Methods

The same general method was used for all the following experiments. I used a weighted f1 score to evaluate the performance of the algorithms because I wanted to accurately portray the precision and recall of datasets with unbalanced classes [3].

For all learning curves and scalability curves I used the following procedure. I used a Stratified Shuffle Split (70% for training and 30% held for validation) to execute 10 testing runs using stratified randomized folds. Being stratified ensures that relative class frequencies are attempted to be preserved in the train and validation fold. This will help balance data sets where there is an unequal distribution of target classes. If one class is particularly small and doesn't have any samples included in the training set, then the testing score is unrepresentative of the algorithm's performance. I chose a shuffle split over k-fold because it allows me to run an unlimited number of testing runs without decreasing the number of samples in the training and validation fold. This helped me reduce variance in my testing results. This is especially important when working with smaller datasets and algorithms that require high number of samples to be effective.

IV. Decision Tree

The decision trees in this section were implemented from the scikit-learn model and use the Gini impurity as the measurement to decide which feature to split on. The Gini impurity shows the odds of new data being incorrectly classified if it were given a random class given the class distribution in the training dataset.

A. Diabetic Retinopathy Dataset

In the learning curve of the Diabetic dataset (Figure 1) , we see that the the decision tree learner requires the full training dataset for the training score to match the validation score implying that the bias to the training dataset is eliminated.

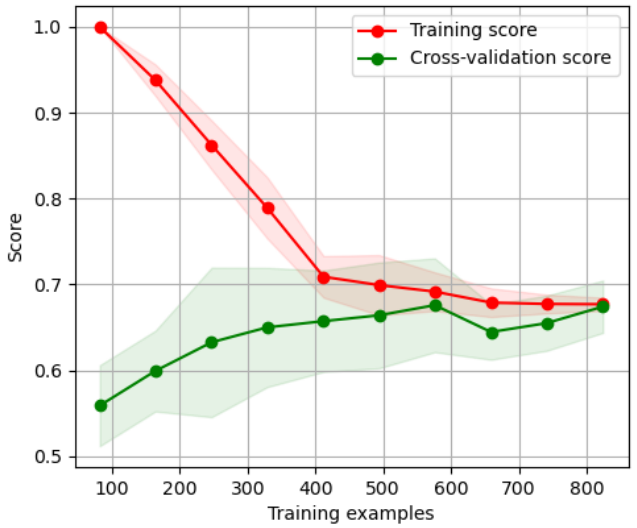

Fig. 1: Plot of the Learning Curve of a Decision Tree learner with hyperparameters of *leaf_size* = 1, *ccp_alpha* = 0.0, *max_depth* = 6 on the Diabetic Retinopathy Dataset.

The scalability of the decision tree learner on the Diabetic dataset is linearly increasing with additional training samples

(Figure 2). This is expected as the calculation time of the Gini impurity for each feature will increase with the amount of samples in the data. We would also expect the variance to stay roughly constant.

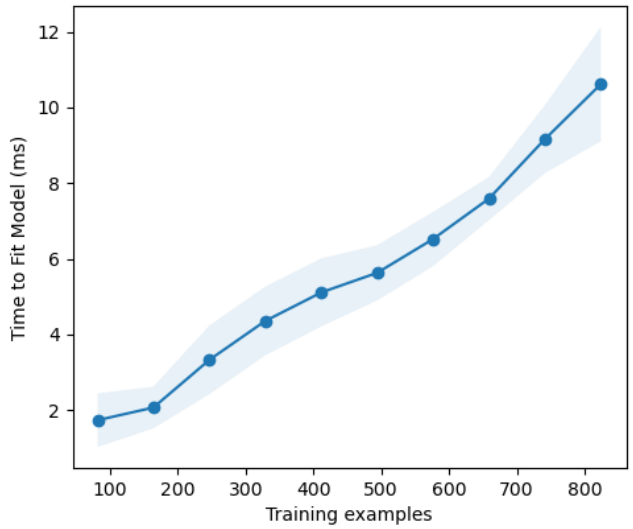

Fig. 2: Plot of the Scalability of a Decision Tree learner with hyperparameters of *leaf_size* = 1, *ccp_alpha* = 0.0, *max_depth* = 6 on the Diabetic Retinopathy Dataset.

In Figure 3, we see that the as we increase the minimum leaf size, the training bias decreases. This is because the learner is forced to generalize so the training score will decrease, but in turn will increase the validation score.

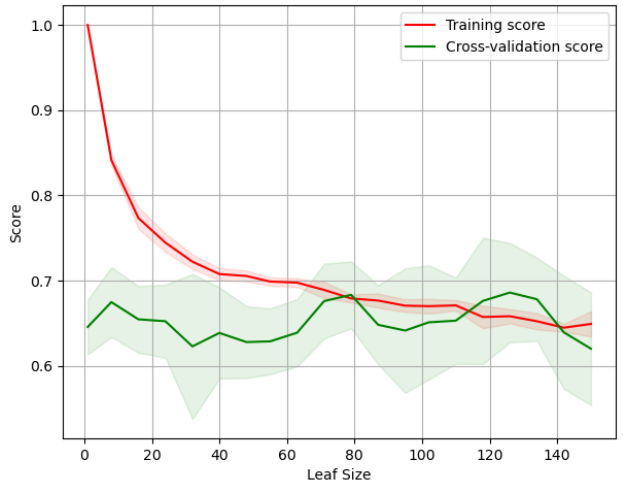

Fig. 3: Plot of the leaf size hyperparameter tuning of a Decision Tree learner with hyperparameters of *ccp_alpha* = 0.0 on the Diabetic Retinopathy Dataset.

B. Red Wine Dataset

In the Red Wine dataset, I would expect that the decision tree would be able to use less samples than the Diabetic dataset as there are less features in the Red Wine dataset. However, after using the full training dataset, the validation score is still lower than the training score implying that there still exists bias to the training data (Figure 4). Additionally, there is more variance with large sample sizes in the Red Wine dataset than in the Diabetic dataset (Figure 4). This could mean that the Red Wine dataset is simply more noisy than the Diabetic dataset. There are more classes in the Red Wine dataset and the score of wine is a slightly subjective criteria which both could lead to more variance.

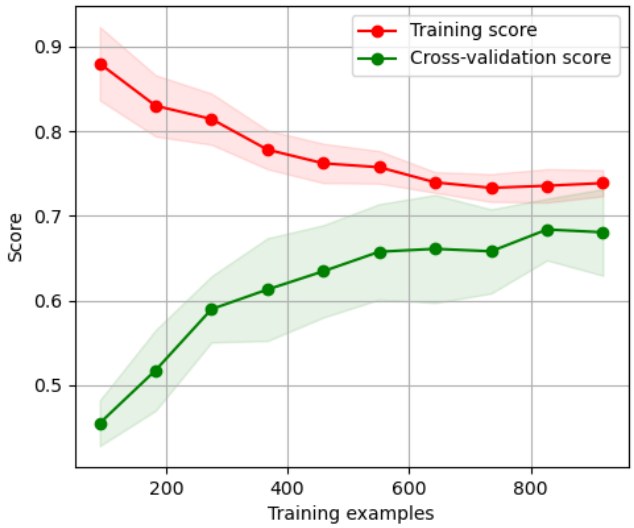

Fig. 4: Plot of the Learning Curve of a Decision Tree learner with hyperparameters of *leaf_size* = 1, *ccp_alpha* = 0.0, *max_depth* = 6 on the Red Wine Dataset.

The scalability of the decision tree learner in on the Red Wine dataset is roughly linear and maintains a similiar amount of fit time variance throughout (Figure 5). This shape of the curve looks quite similar to the scalability plot of the Diabetic dataset (Figure 2). However, there is a difference in scale and slope as the Diabetic dataset takes longer and increases more with each additional training sample. This is expected as there are more features in the Diabetic dataset and computing the Gini impurity for each feature means that computation time should scale with each additional feature. This increase in computation time also implies that the model for the Diabetic dataset is more complex than the Red Wine dataset.

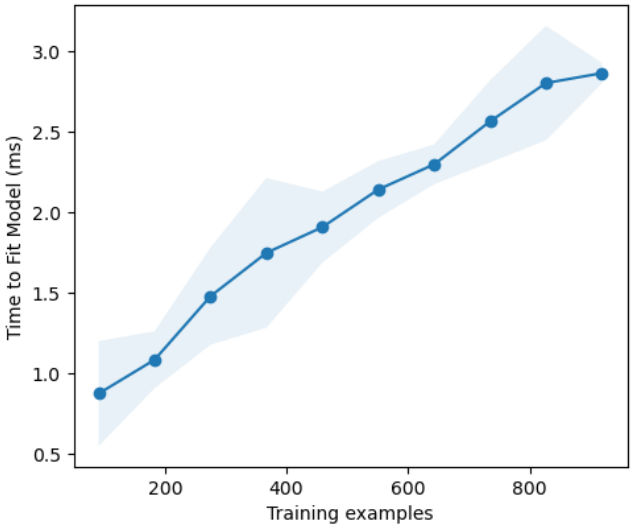

Fig. 5: Plot of the Scalability of a Decision Tree learner with hyperparameters of *leaf_size* = 1, *ccp_alpha* = 0.0, *max_depth* = 6 on the Red Wine Dataset.

In Figure 6, we see that the as we increase the CCP alpha, the training bias decreases. This is because the learner is starts pruning off the nodes that are the weakest and have the lowest alpha. This will in turn generalize the model and decrease bias to the training data.

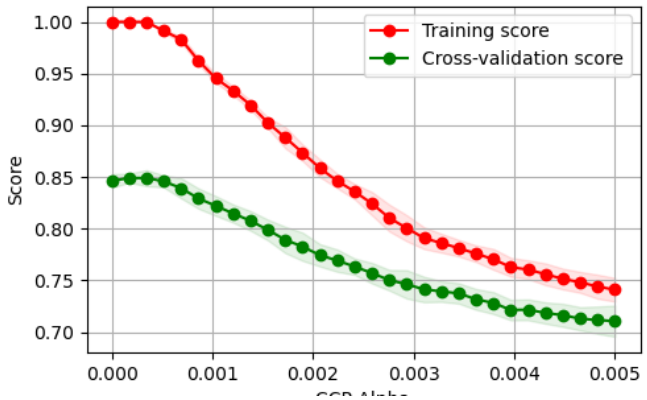

Fig. 6: Plot of the ccp alpha hyperparameter tuning of a Decision Tree learner with hyperparameters of *leaf_size* = 1 on the Red Wine Dataset.

V. K Nearest Neighbor

A. Diabetic Retinopathy Dataset

We can see that the learning curve of the Diabetic dataset has quite low bias even with 100 training samples. The training and validation scores track together along the whole curve. I expected the training score to be much higher than the validation score. I think this is because there is a low number classes in the dataset and the number of samples necessary is dependent on how complex the dataset is. I think that the variance in the learning curve could imply that the dataset is noisy or has outliers. KNN learners are particularly sensitive to noise and outliers because the final closest neighbor could incorrectly move a prediction to the wrong class. It could also imply that the dataset is difficult to cluster.

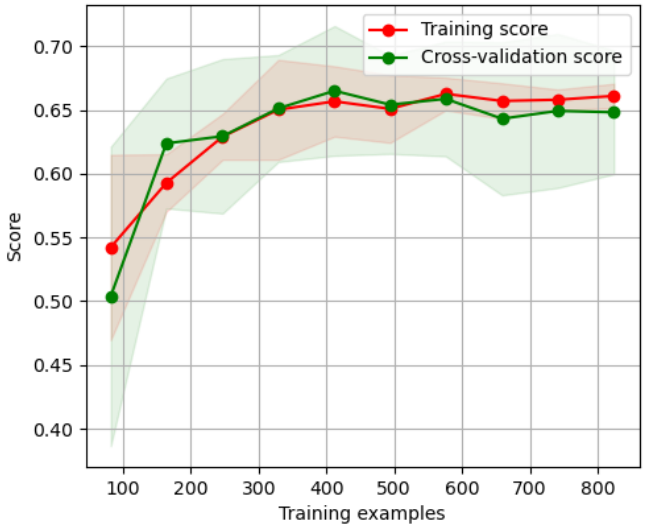

Fig. 7: Plot of the Learning Curve of a K Nearest Neighbor learner with hyperparameters of *n_neighbors* = 5, *algorithm* = '*ball_tree*' on the Diabetic Retinopathy Dataset.

KNN model's fit time should be relatively flat because with KNN there is no model being computed during training. As predicted, there is little increase in fit time with additional data in the Diabetic dataset (Figure 8). Instead a database of past examples is being created. The time should increase linearly as there is more data to save to memory, but there is no computation happening. This also explains the high variance as other tasks happening on the system could be affecting the fit times. The calculations happen during queries where the algorithm must compute the distances to the nearest neighbors and compute a final prediction.

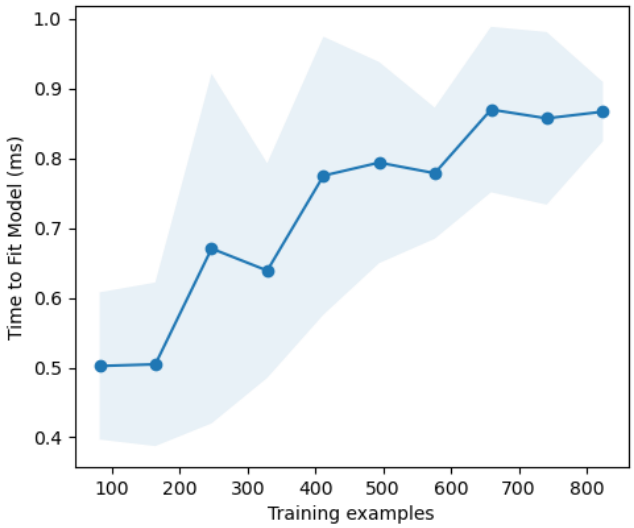

Fig. 8: Plot of the Scalability of a K Nearest Neighbor learner with hyperparameters of *n_neighbors* = 5, *algorithm* = '*ball_tree*' on the Diabetic Retinopathy Dataset.

With KNN neighbors algorithm, k represents the number of nearest neighbors included when evaluating a new prediction. Its expected that lower values of k will lead to overfit models

because the prediction is made using less training data. This can be seen in the Figure 9. We can see that the training score is higher than the cross-validation score until k=40. This value allows for a low amount of overfitting while preserving a high overall score.

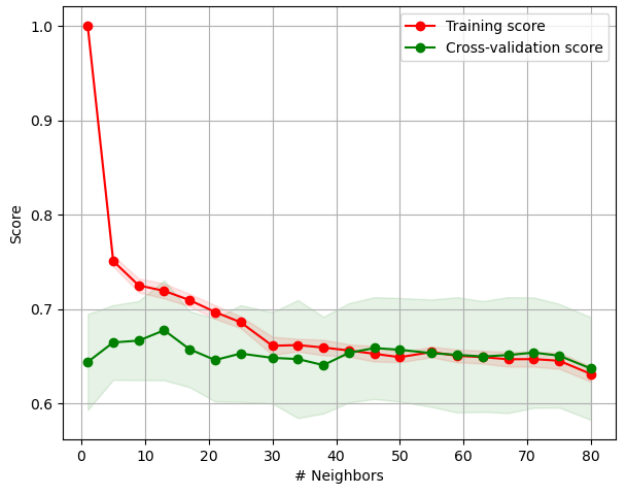

Fig. 9: Plot of the number of neighbors k of a K Nearest Neighbor learner with hyperparameters of *algorithm* = '*ball_tree*' on the Diabetic Retinopathy Dataset.

In Figure 10, we see a chart of various available distance calculations algorithms that can be used for Knn learners. I expected these to have drastically different outputs but they are all essentially the same. This shows that the distance calculation has a much smaller effect on the learner than the total number of neighbors.

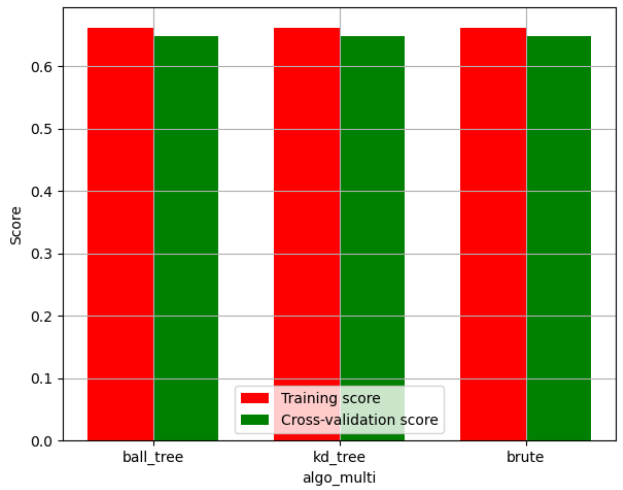

Fig. 10: Plot of different distance algorithms for K Nearest Neighbor learner with hyperparameters of k=40 on the Diabetic Retinopathy Dataset.

B. Red Wine Dataset

I expected that the KNN model would require more training samples for the Red Wine dataset because having more categories in the dataset makes it harder to generalize with less training samples. In the learning curve for the Red Wine dataset, we can see that the KNN model continues to improve even past 600 samples (Figure 11). This is partially because the Diabetic dataset is a 3 category classification problem where the Red Wine dataset has 6 categories. Having more categories in the dataset makes it harder to generalize with less training samples.

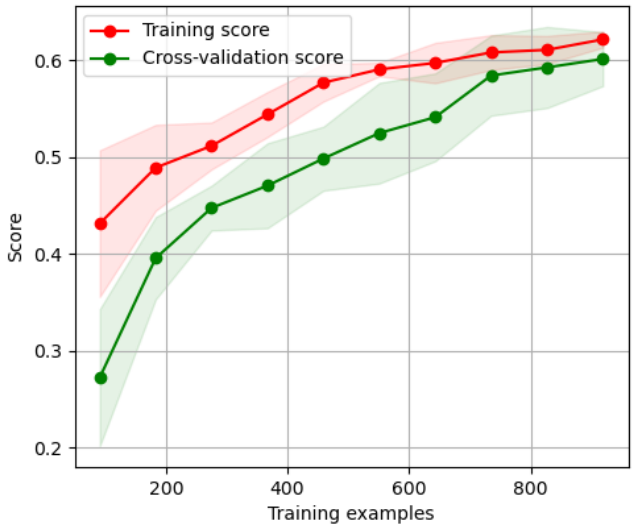

Fig. 11: Plot of the Learning Curve of a K Nearest Neighbor learner with hyperparameters of *n_neighbors* = 5, *algorithm* = '*ball_tree*' on the Red Wine Dataset.

Similar to the Diabetic scalability curve (Figure 8), the Red Wine dataset scalability curve (Figure 12) has a slight linear increase with high variance. Because K-NN learners do not do any calculations on the data during training, the learning curves should be very similar regardless of the dataset. We could expect that the Red Wine scalability curve should be lower because each training sample is smaller because there is less features. This is what the experimental data shows, but there is still a lot of variance in the data and this claim might require more additional testing to confirm this.

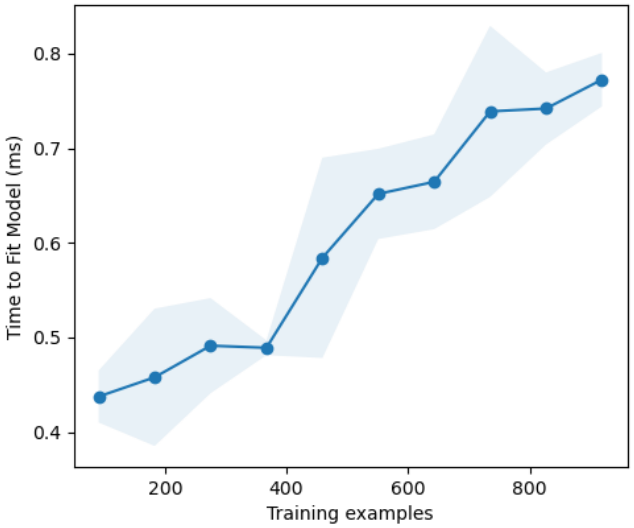

Fig. 12: Plot of the Scalability of a K Nearest Neighbor learner with hyperparameters of *n_neighbors* = 5, *algorithm* = '*ball_tree*' on the Red Wine Dataset.

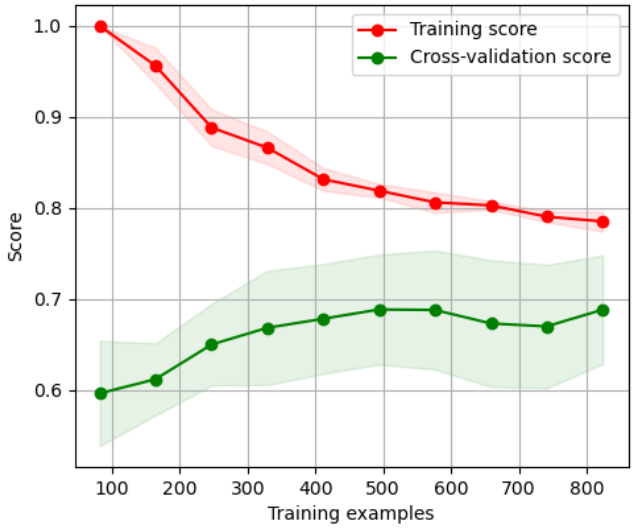

Fig. 13: Plot of the Learning Curve of a Ada Boost learner with hyperparameters of *n_estimators* = 65, *learning_rate* = 1 on the Diabetic Retinopathy Dataset.

VI. Boosting

A. Diabetic Retinopathy Dataset

At first I tried to tune the CCP Alpha parameter to prune the weak learners, but the results were poor as the learner ended up having far too much bias to the training data. The problem is that the AdaBoost algorithm prioritizes training on its errors during training. This will lead to a large bias if the Decision Trees are allowed to grow too large.

For AdaBoost, I expected that the model would tend to overfit in cases with few samples. This is because AdaBoost is built with weak learners (and in this case Decision Trees of *depth* = 1) and because it prioritizes fixing errors during training. So with a high amount of learners for a small amount of data, AdaBoost will overfit because there is too many learners. However, the bias to the training data should decrease as the ratio of training samples to number of weak learners increases. In Figure 13, we can see that this prediction is true. The training bias continues to decrease as we add samples. Curiously the variance in the validations score does not decrease while the variance decreases for the training score. This could imply that the learner needs more samples and also needs to generalize more.

I predicted that the AdaBoost learner would have a relatively flat scalability. This is because there is a constant number of weak learners so the number of computations should also be constant. Figure 14, shows that my prediction was not correct. There is a large spike variance and mean training time in the middle of the training sample size distribution. I believe this could be because there could be outliers in the training set that are more difficult to set the proper weak learning weight properly and so AdaBoost learner has to make more passes at updating these weights. At higher training sample sizes, the outliers might appear like a more normal distribution and be captured but the weak learners more completely and thus require less pass at updating the weights.

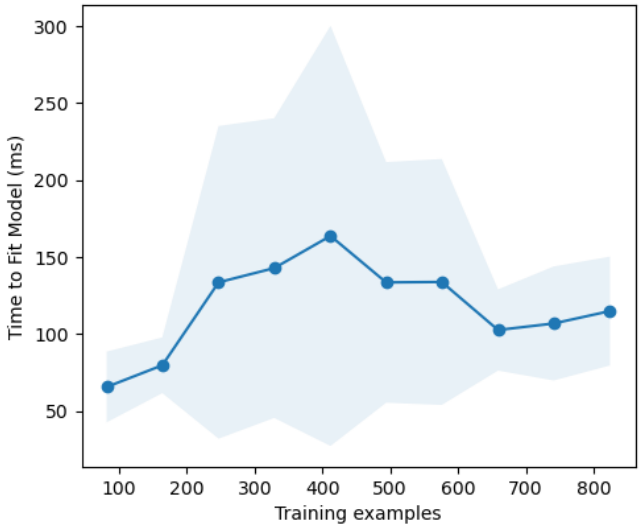

Fig. 14: Plot of the Scalability of a Ada Boost learner with hyperparameters of *n_estimators* = 65, *learning_rate* = 1 on the Diabetic Retinopathy Dataset.

I expect that as the number of estimators increases, we see that the gap between the training score and the cross-validation

score would decreases. In Figure 16, the number of estimators increases the validation score. This makes sense because as we increase the number of estimators we should be decreasing bias to the training data.

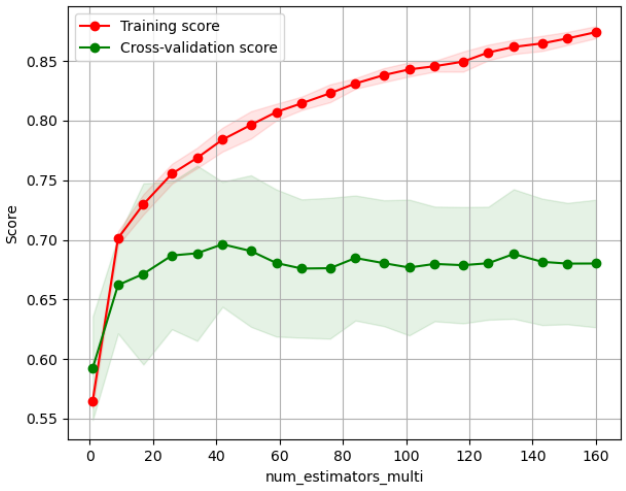

Fig. 15: Plot of the number of estimators hyperparameter validation plot of a Ada Boost learner with hyperparameters of *learning_rate* = 1 on the Diabetic Retinopathy Dataset.

I decided to change the pruning method on the AdaBoost from being a stump of *depth* = 1 to a very agressive cost complexity pruning. I expected the model to still overfit and bias the training data. In Figure 16, the low values of alpha are indeed very overfit, but after increasing the value, the pruning is aggressive enough to create a model that is not biased to the training data.

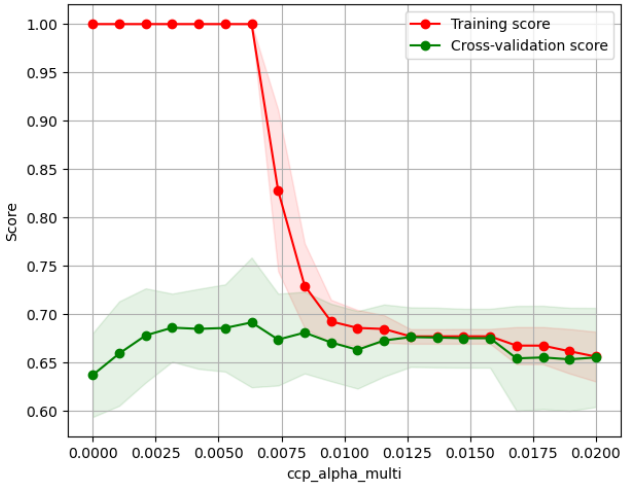

Fig. 16: Plot of the ccp alpha hyperparameter validation plot of a Ada Boost learner with hyperparameters of *learning_rate* = 1, and *n_estimators* = 65 on the Diabetic Retinopathy Dataset.

B. Red Wine Dataset

I expected that the learning curve for the Red Wine dataset would be similar to the Diabetic dataset and have an initially high bias and low validation score to the training set but then decrease the bias and increase the training score as the training samples increased. However, this learning curve is much different than the Diabetic dataset. The absolute score is low and there is large variance at all training sample sizes. I believe this is because the Red Wine dataset might be noisy and could has some outliers. Outliers and noisy would make the AdaBoost algorithm overfit on this data as the initial weak learners would classify these training samples incorrectly and the subsequent learners would attempt to fix these.

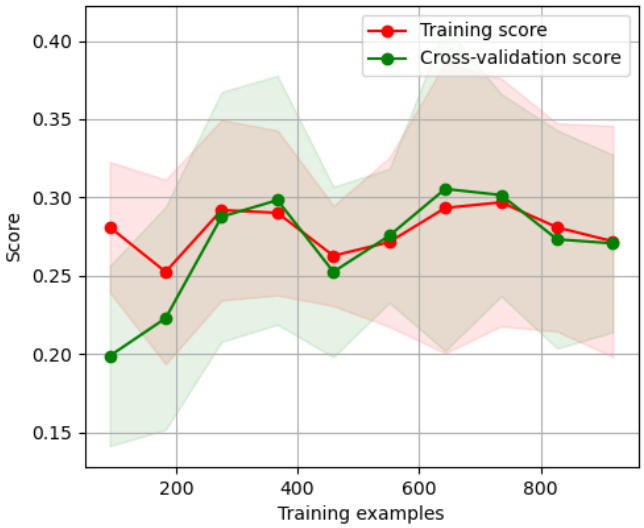

Fig. 17: Plot of the Learning Curve of a Ada Boost learner with hyperparameters of *n_estimators* = 65, *learning_rate* = 1 on the Red Wine Dataset.

Similar to the diabetic dataset, I expected the scalability of the AdaBoost learner to be relatively flat. In Figure 18, the curve looks more similar to my prediction than the Diabetic dataset. I still see high variance in the fit times, but curiously, these are at the high and low ends of the training sample size. The opposite of the Diabetic dataset. I still think that the variance is due to outliers in the datasets which requires more weak learner weight calculations. The time to mean fit the Red Wine dataset is higher than the Diabetic dataset. I think that this because there are more weak learners which requires more computation because each weak learner needs to calculate the feature to split on by calculating the Gini Impurity as well as an additional weight for the AdaBoost summation.

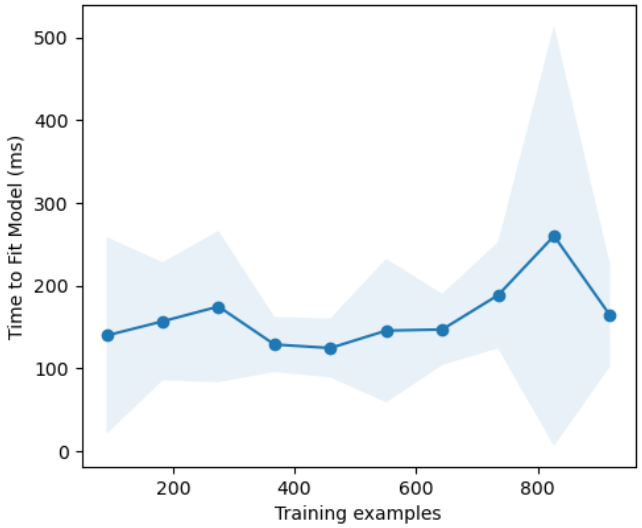

Fig. 18: Plot of the Scalability of a Ada Boost learner with hyperparameters of *n_estimators* = 65, *learning_rate* = 1 on the Red Wine Dataset.

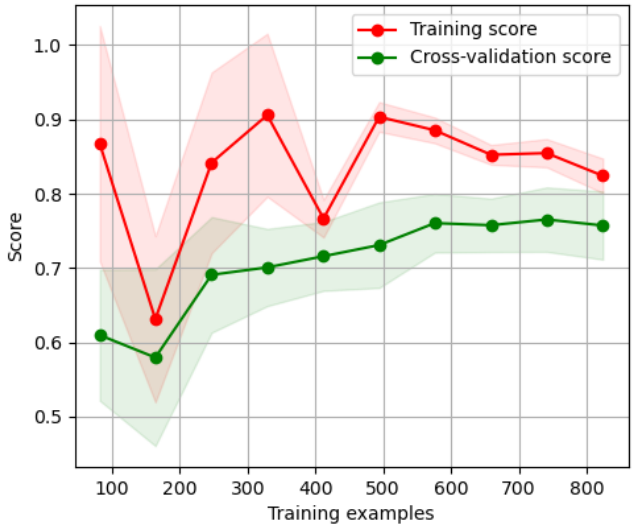

Fig. 19: Plot of the Learning Curve of a Neural Network learner with hyperparameters of *n_layers* = 3, *n_nodes* = 30, *alpha* = 1e-5, *solver* = 'Stochastic Gradient Descent' on the Diabetic Retinopathy Dataset.

VII. Neural Networks

First I created a Grid Search to determine the ideal hyperparameter selection for the dataset. All testing was done using scikit-learn's MLPClassifier using Stochastic Gradient Descent.

A. Diabetic Retinopathy Dataset

I expected the learning curve for the Diabetic dataset to be initially overfit to the training data, but then reduce bias as the training sample size increased. In Figure 19, we can see that there is a huge amount of variance when the sample size is low. Additionally, there is a large gap between the training score and the testing score when the sample size is low implying that there is a large amount of bias to the training data. This makes sense because if there is a small amount of training data, the neural networks tend to overfit to the training data because they are reducing the error on too few training samples. The early variance could be from outliers that the neural network is overfitting too.

As the sample size increases, the mean variance decreases and the experimental variance also reduces. We can also see the gap between the training score and the validation curve decreases immensely implying that the algorithm has generalized and decreased the bias to the training data. We can see that the testing score is tapering off, but it continues to grow. In this way, the neural network could benefit from even more samples.

I expect that the scalability of the neural network will increase with the number of training samples. This is because the loss is calculated on all of the training data before each iteration and back propogation through the neural network. Figure 20, shows that the scalability of the model does indeed increase, but there was a large amount of variance. The variance also increases with sample size. This could be because the neural network is optimizing a local minimum instead of the global minimum which could be a different local minimum on each experimental run due to the random initialization of the neural networks' weights.

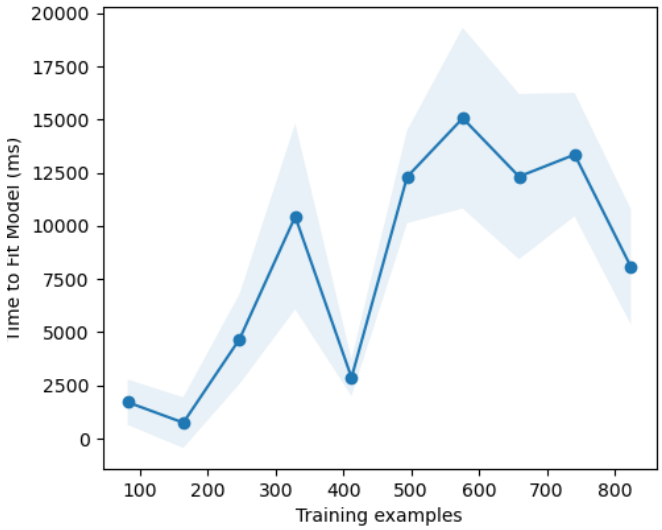

Fig. 20: Plot of the Scalability of a Neural Network learner with hyperparameters of *n_layers* = 3, *n_nodes* = 30, *alpha* = 1e-5, *solver* = 'Stochastic Gradient Descent' on the Diabetic Retinopathy Dataset.

B. Red Wine Dataset

The loss curve is generated by running 10 separate runs of using a Stratified Shuffle Split. The mean and standard

deviation of the loss at each iteration is plotted in Figure 21.

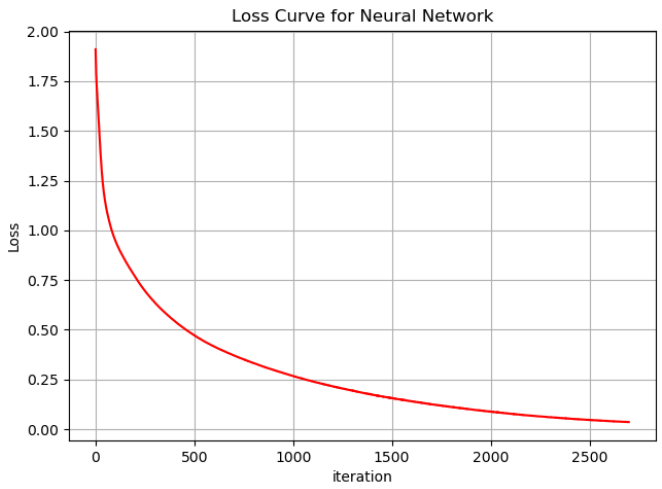

Fig. 21: Loss curve plot of a Neural Network learner with hyperparameters of *n_layers* = 3, *n_nodes* = 30, *alpha* = 1e-5, *solver* = 'Stochastic Gradient Descent' on the Red Wine Dataset.

For a neural network learning curve, I expected an initially overfit model that would decrease bias with an increase in training samples. Unlike the Diabetic dataset, Figure 22 shows what I predicted. The model is initially overfit and as bias reduces as the number of training samples increases.

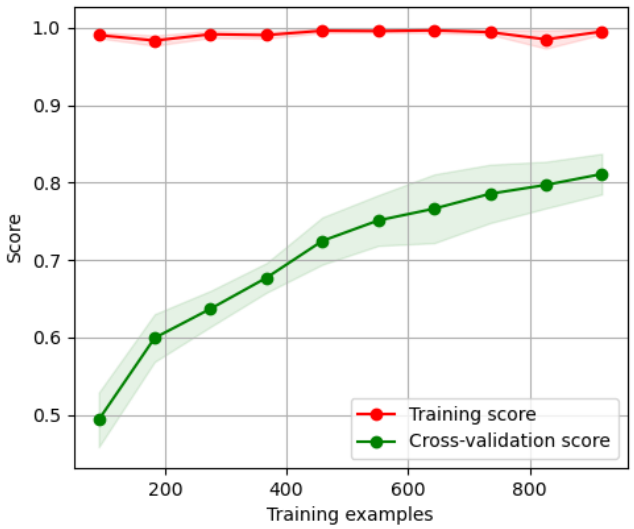

Fig. 22: Plot of the Learning Curve of a Neural Network learner with hyperparameters of *n_layers* = 3, *n_nodes* = 30, *alpha* = 1e-5, *solver* = 'Stochastic Gradient Descent' on the Red Wine Dataset.

Like previously stated, I expected the fit times of the neural network would increase with the number of training samples. Figure 23, shows that the model does indeed increase linearly with the increase in sample size as I predicted. The variance also increases with sample size as with the Diabetic dataset. This could be because the neural network is optimizing a local minimum instead of the global minimum.

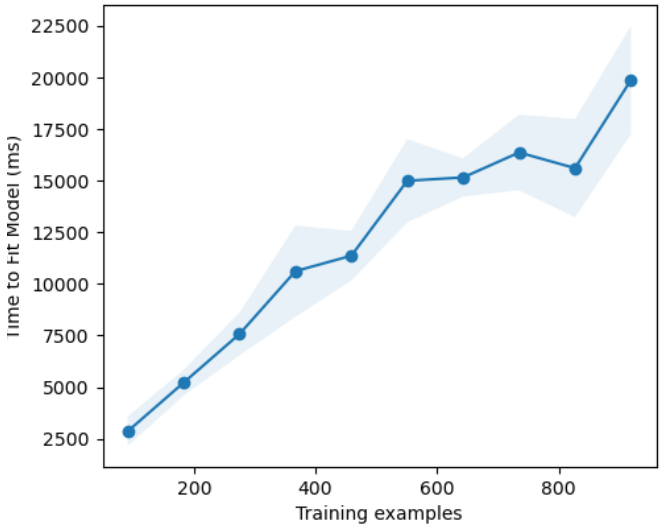

Fig. 23: Plot of the Scalability of a Neural Network learner with hyperparameters of *n_layers* = 3, *n_nodes* = 30, *alpha* = 1e-5, *solver* = 'Stochastic Gradient Descent' on the Red Wine Dataset.

I expected that increasing the number of hidden layers in the neural network would continue to increase the score of the model, but provide diminishing returns. As seen in Figure 24, the performance of the neural network model peaks at 3 layers and then flattens out. I did expect this to continue to increase, but the model could instead be optimizing to local minima instead. This means that 3 should be selected because the added layers don't add any performance and instead will add a large amount of overhead during training with each additional layer. This is because each additional layer brings 30 new weights that need to be optimized with back propagation over 1000s of runs.

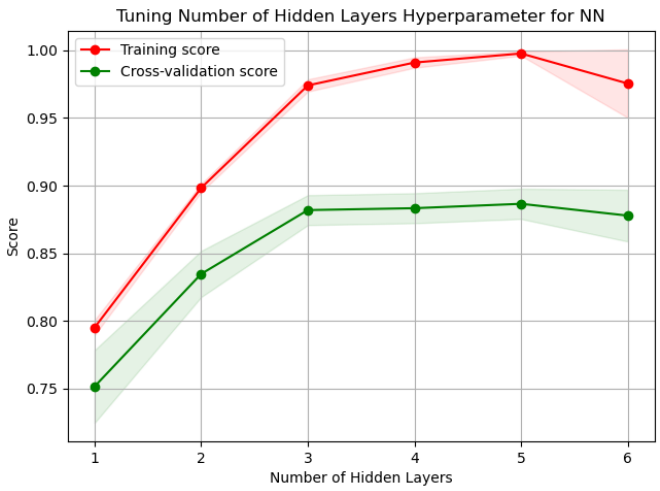

Fig. 24: Plot of the hyperparameter validation curve of the number of layers of a Neural Network learner with hyperparameters of *n_nodes* = 30, *alpha* = 1e-5, *solver* = 'Stochastic Gradient Descent' on the Red Wine Dataset.

Similar to my prediction for the number of layers, I expected that as the number of nodes increased, the performance of the model would increase, but would eventually plateau as the

model fully approximated the underlying function. In Figure 25, the validation score continues to increase until it flattens out at about 50 nodes. This is the diminishing returns that I predicted.

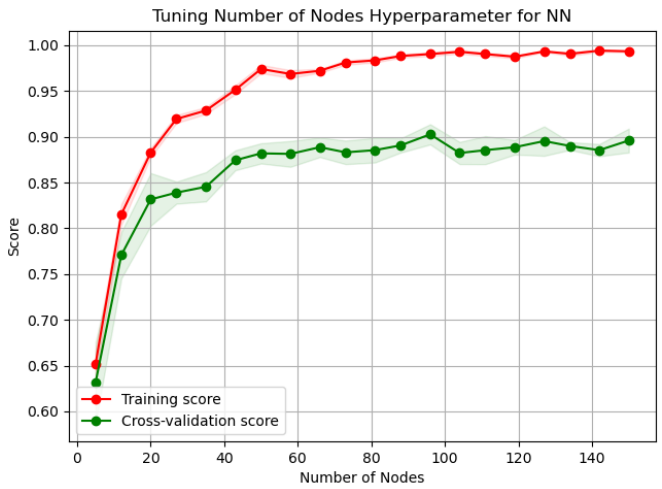

Fig. 25: Plot of the hyperparameter validation curve of the number of neurons of a Neural Network learner with hyperparameters of *n_layers* = 3, *alpha* = 1e-5, *solver* = 'Stochastic Gradient Descent' on the Red Wine Dataset.

VIII. Support Vector Machines

A. Diabetic Retinopathy Dataset

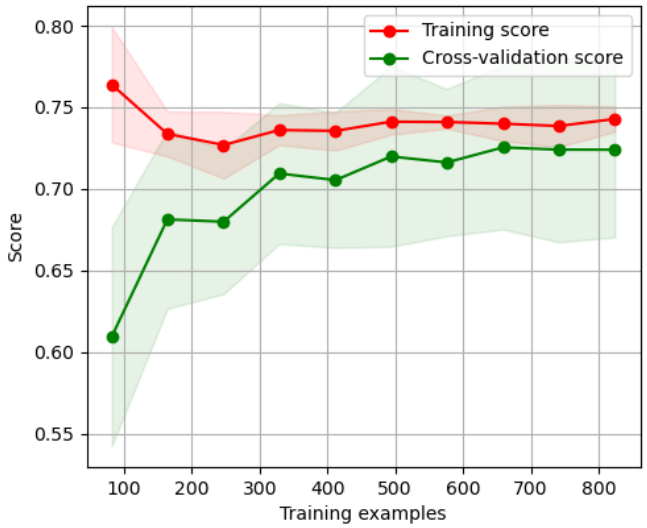

Fig. 26: Plot of the Learning Curve of a Support Vector Machine learner with hyperparameters of C = 1, *kernel* = *rbf* on the Diabetic Retinopathy Dataset.

For SVM, I expect the fit times to increase linearly if not exponentially with additional sample data points. This is because adding more points means that that the learner has more points to calculated distances from the hyper plane which would increase computation time. As I expected, Figure 27, shows a roughly exponential increase in time to fit with the increase in training examples.

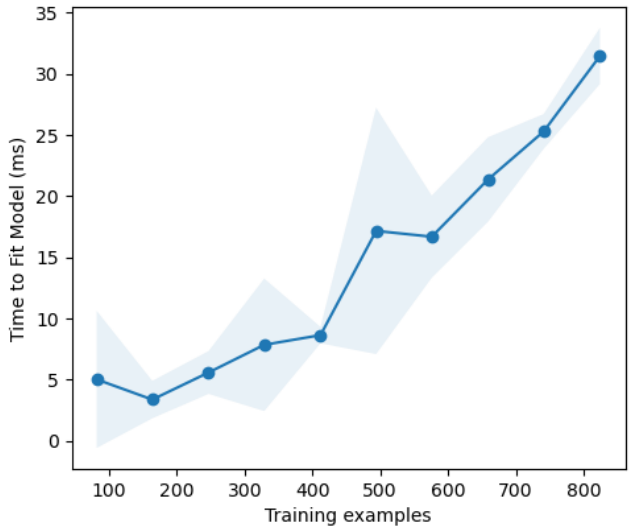

Fig. 27: Plot of the Scalability of a Support Vector Machine learner with hyperparameters of C = 1, *kernel* = *rbf*, on the Diabetic Retinopathy Dataset.

B. Red Wine Dataset

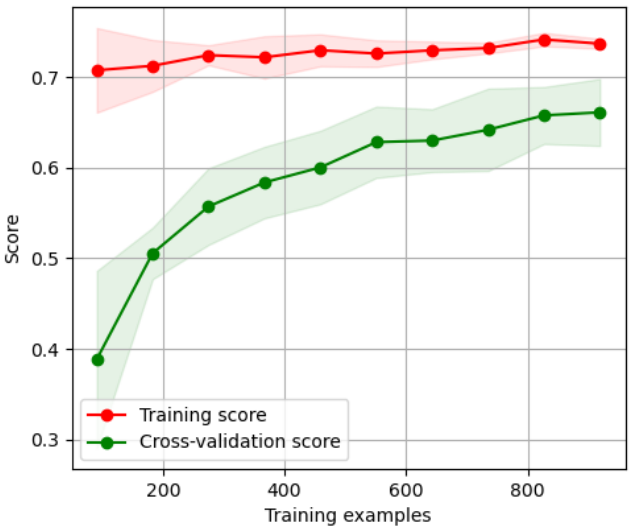

Fig. 28: Plot of the Learning Curve of a Support Vector Machine learner with hyperparameters of C = 1, *kernel* = *rbf* on the Red Wine Dataset.

As with the Diabetic dataset, I also expect the same behavior with the Red Wine dataset and SVM. I expect an exponential increase in model fit time with an increase in training samples. Figure 29 shows that my prediction was correct and shows an exponential increase in fit time as training samples increases. Contrasting to the Diabetic dataset, the variance in scalability with the Red Wine dataset increases as the training samples increases.

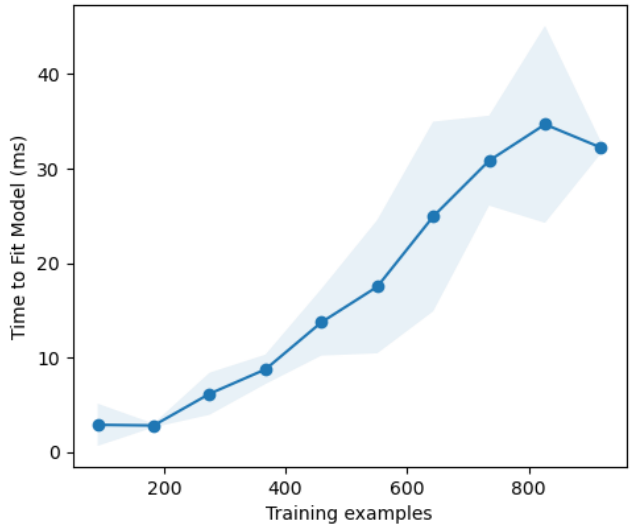

Fig. 29: Plot of the Scalability of a Support Vector Machine learner with hyperparameters of $C = 1$, *kernel* = *rbf*, on the Red Wine Dataset.

I expected that the non-linear kernel functions would perform the best on this dataset. In Figure 30, we can see that the rfb kernel performs the best and the performs terribly. This is because the rbf kernel can better approximate non-linear functions and create more interesting and flexible hyperplanes. This implies that the Red Wine dataset has non-linear relationships between the features and the target class.

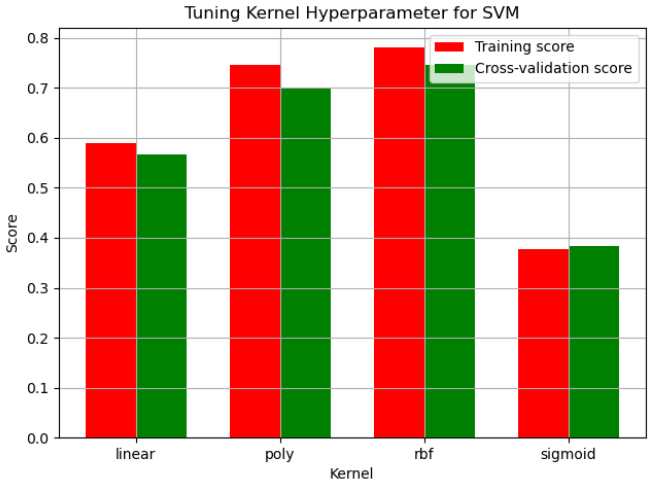

Fig. 30: Plot of the hyperparameter validation curve of the kernel function of a Support Vector Machine learner with hyperparameters of $C = 1$, on the Red Wine Dataset.

I would expect that as C as see is lower, we would have lower accuracy because the SVM algorithm is trying to create a higher margin. In Figure 31, we see this exact trade off between accuracy.

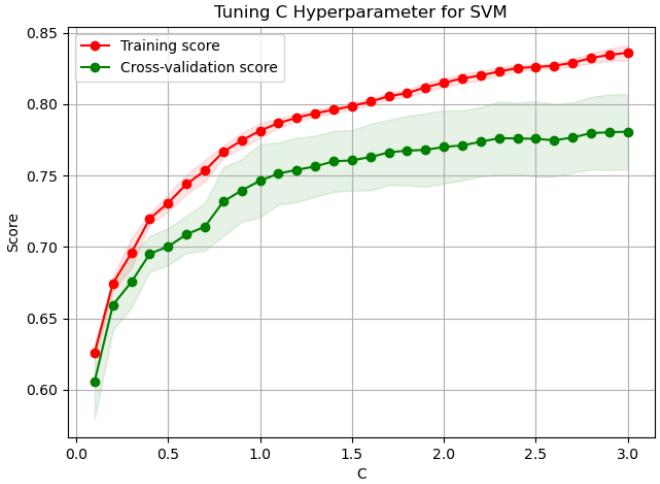

Fig. 31: Plot of the hyperparameter validation curve of C of a Support Vector Machine learner with hyperparameters of *kernel* = *rbf*, on the Red Wine Dataset.

IX. Overall Performance

A. Performance

In the final performance for the Diabetic dataset we see that the neural network and boosted take incredibly long to train compared to the rest but they does come out with the overall best performance. The high performance of SVM, and the neural network on the Red Wine Dataset imply that there could be a non-linear underlying function. This is easier to approximate with neural networks and SVM. The low training and test time for the decision tree and Knn is not to be under estimated as they still provided very high performance.

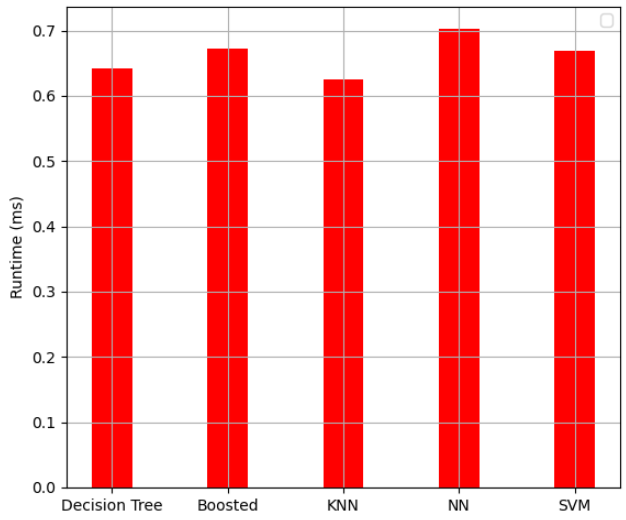

Fig. 32: Plot of the Final Testing F1 Scores of Various Algorithms, on the Diabetic Retinopathy Dataset.

In the final performance of the Red Wine dataset (Figure 33 we see that the neural network take incredibly long to train and test, but it does come out with the overall best performance. The high performance of SVM, and the neural network on

the Red Wine Dataset imply that there could be a non-linear underlying function. This is easier to approximate with neural networks and SVM.

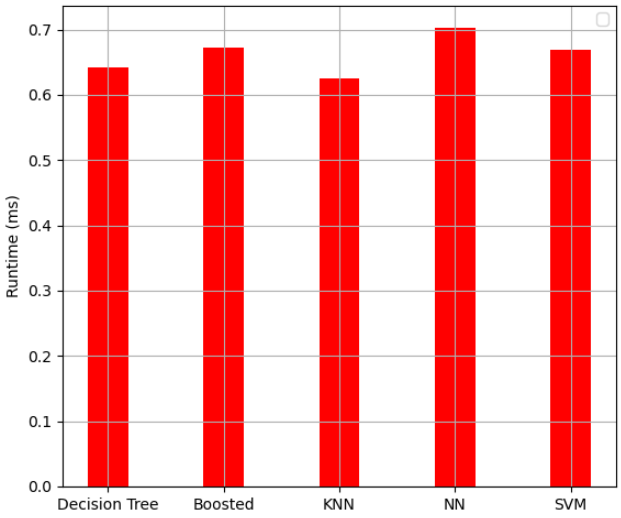

Fig. 33: Plot of the Final Testing F1 Scores of Various Algorithms, on the Red Wine Dataset.

B. Wall Clock Time

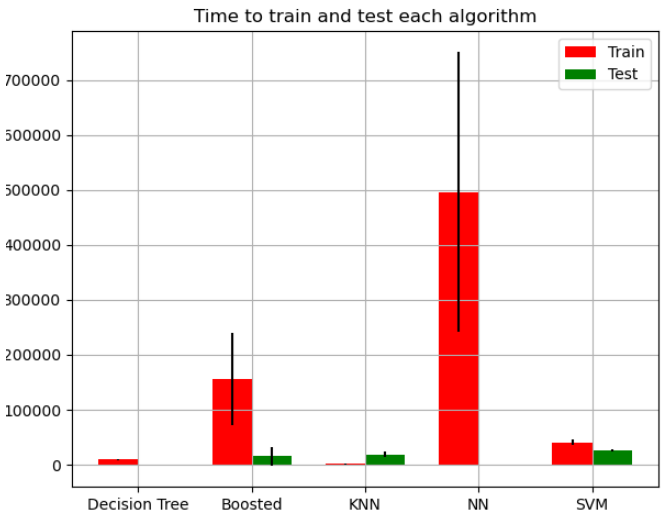

Fig. 34: Plot of the Wall Clock Time to Train and Query of Various Algorithms, on the Diabetic Retinopathy Dataset.

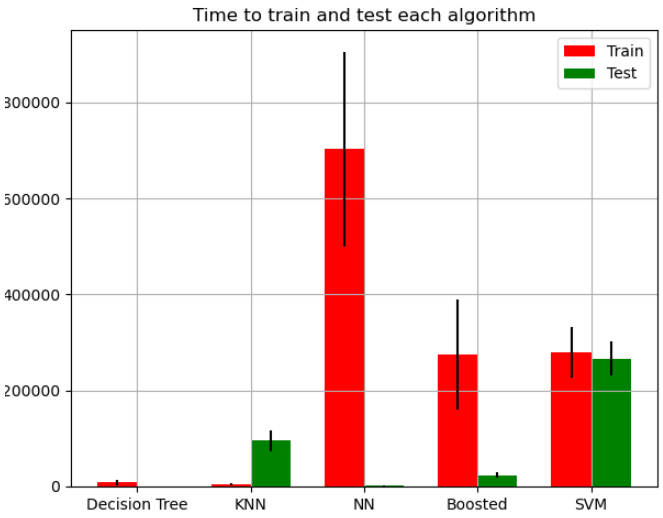

Fig. 35: Plot of the Wall Clock Time to Train and Query of Various Algorithms, on the Diabetic Retinopathy Dataset.

X. Appendix

References

[1] B. Antal and A. Hajdu, "An ensemble-based system for automatic screening of diabetic retinopathy," *CoRR*, vol. abs/1410.8576, 2014. [Online]. Available: http://arxiv.org/abs/1410.8576

[2] P. Cortez, A. L. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decis. Support Syst.*, vol. 47, pp. 547–553, 2009.

[3] "Metrics and scoring: quantifying the quality of predictions," https://scikit-learn.org/stable/modules/model$_evaluation.html precision - recall - and - f - measures, 2022, accessed : 2022 - 09 - 23.$