Kunal Shah
CS7641 - Machine Learning

Project 1: Supervised Learners

Introduction:

This project explored a dataset using five supervised learning techniques. Decision trees, K-Nearest Neighbors, boosting, support vector machines and neural networks were tuned and analyzed to gain a better understanding of how the algorithms behave across a range of circumstances. The performance of each of the learners was compared after each was applied to a distinct classification problem.

Dataset Description:

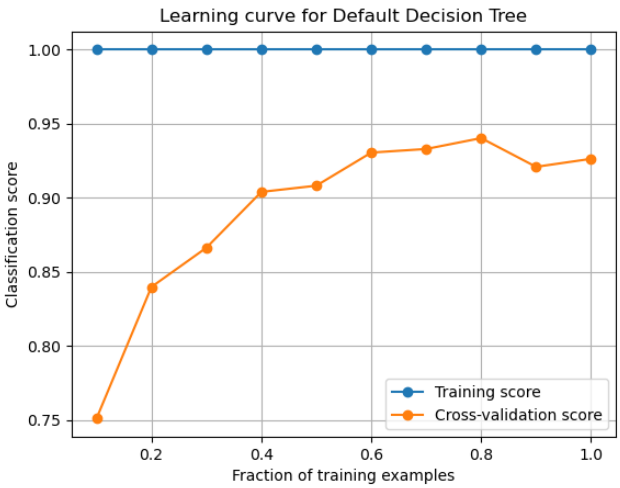
The dataset explored was a collected at the Shirley Ryan AbilityLab using a powered prosthetic leg on 2 different able-bodied users. During ambulation, data were acquired simultaneously from 22 channels from mechanical sensors embedded on the prosthesis. The mechanical sensor information was made up of knee and ankle joint kinematics, 6-Degree of Freedom forces and moments, motor currents and calculated thigh and shank inclination angles. [1] For each channel, 6 features (initial, final, minimum, maximum, mean, standard deviation) were extracted from each frame of data, resulting in a total of 132 features. There were 2386 total examples and the labels for this data consist of 8 possible classes which correspond to different parts of the gait cycle (heel contact, mid-stance, toe off, and mid-swing) and different ambulation modes. [1] These 8 classes are Level Walking(LW), Toe Off(TO), Ramp Descent(RD), Stair Descent (SD), Standing Toe Off (STTO), Standing Heel Contact (STHC), Mid-Swing (MSW), and Mid-Stance (MST). The dataset is imbalanced as the LW and TO classes compose over half of all examples collected and one third of all examples are TO. This is to be expected as TO happens frequently during all modes of ambulation and transitions from walking were collected for all modes to closely mimic normal gait behavior.

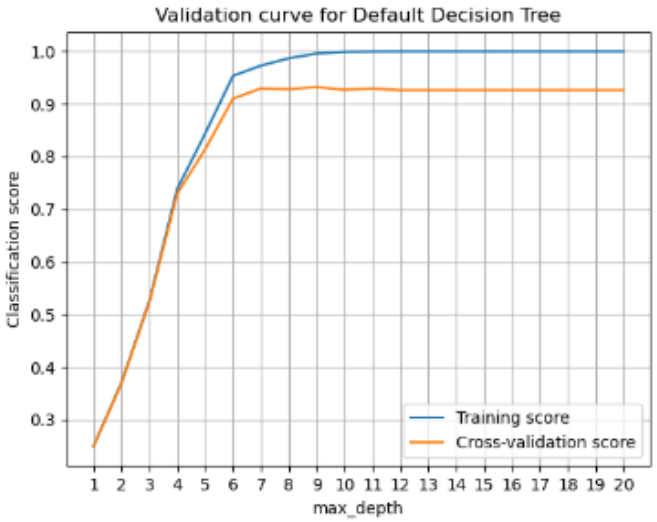
This dataset is interesting because able-bodied users are always the first to test out a prosthesis prior to an amputee for safety concerns. The able-bodied user data is also used to train preliminary models for the intent recognition system to ensure the prosthesis transitions correctly throughout ambulation. This requires a high level of accuracy for all classes because a misprediction can result in severe perturbations or even a fall for amputee subjects.

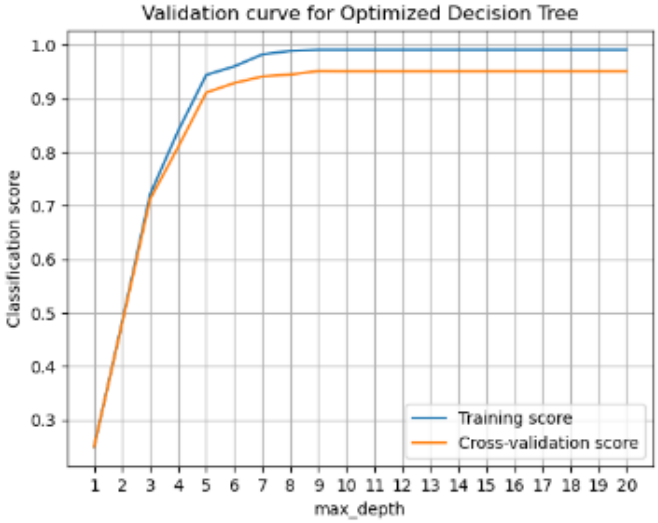
Methods:

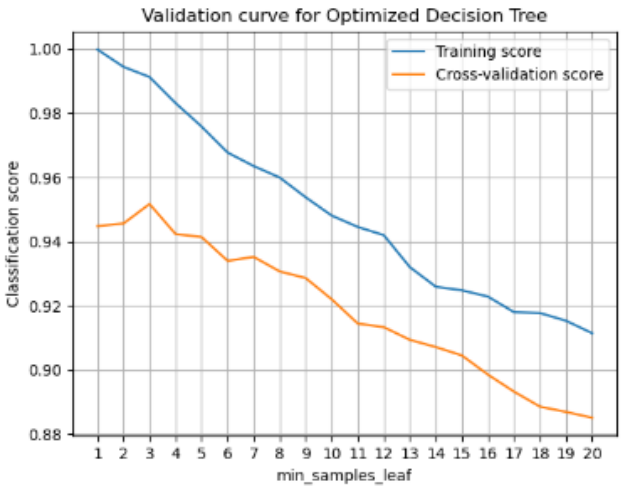
Each dataset was split into training and testing data. 80% of the data made up the training set. K=5 folds were used for each cross validation, as neither dataset had an enormous number of examples. Because the dataset was imbalanced, a stratified cross validation was performed. Also, balanced accuracy was used as the classification score rather than accuracy to adjust for the imbalance in the dataset. Grid searches were run for all algorithms for both datasets to find optimized parameters and create the tuned models. The outputs of these grid searches are summarized in the README.txt submitted with the report.

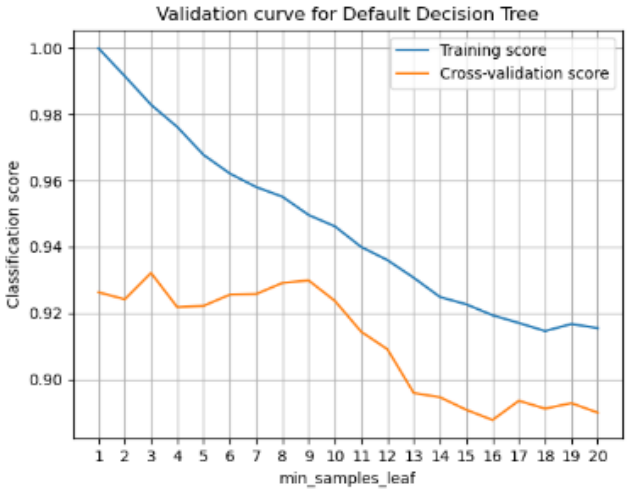
Decision Trees:

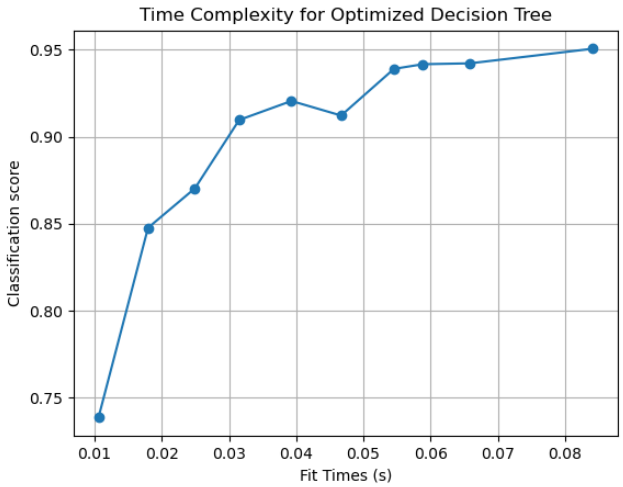












The lower limb dataset was imbalanced, as it was dominated by TO and LW examples. To combat this, the balanced accuracy score was used rather than just accuracy score. Balanced accuracy returns the average accuracy *per class* while accuracy simply returns the percentage of correctly predicted labels.

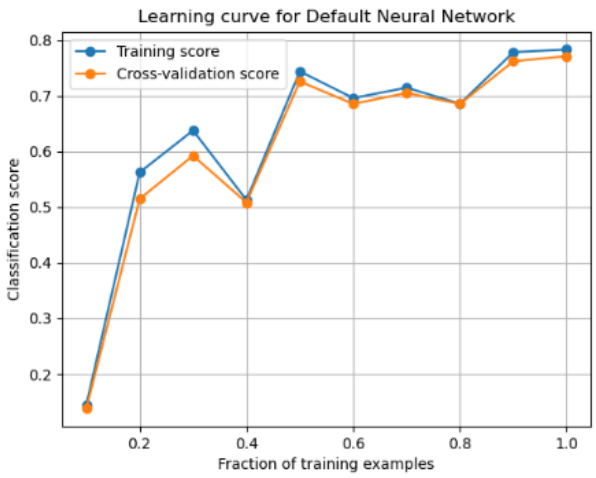
On the lower limb dataset, initially there was low bias and fairly low variance as well. The training data is able to reach a classification score of 1.0 while the cross validation score sat above 0.90 after a max depth of 6. While this is a great classification score, errors need to be reduced as much as possible to minimize changes for perturbations/falls for amputees walking on the prosthesis.

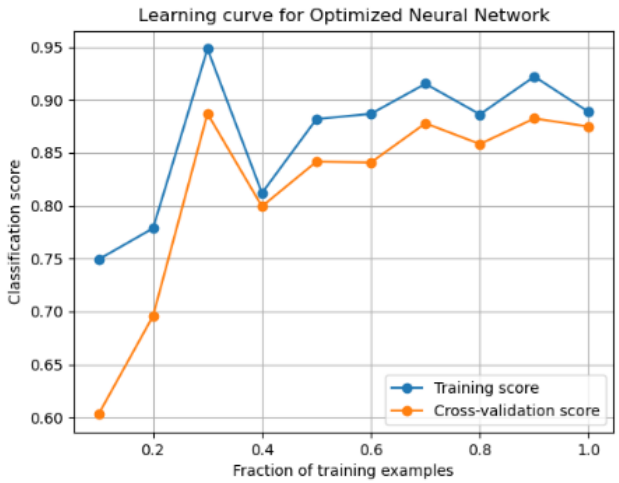
The max depth of the tree was limited to preprune the tree. To explore the effect that pruning has on the model, a validation curve for the max_depth parameter is shown below. Pruning is used to reduce the size of a tree and thus the complexity of a model. Pruning may lower the accuracy of the training set since the optimal parameters will not be fully learned but will reduce the likelihood of the model overfitting, and will perfrom better on the validation set. This can be seen from the learning curves below.

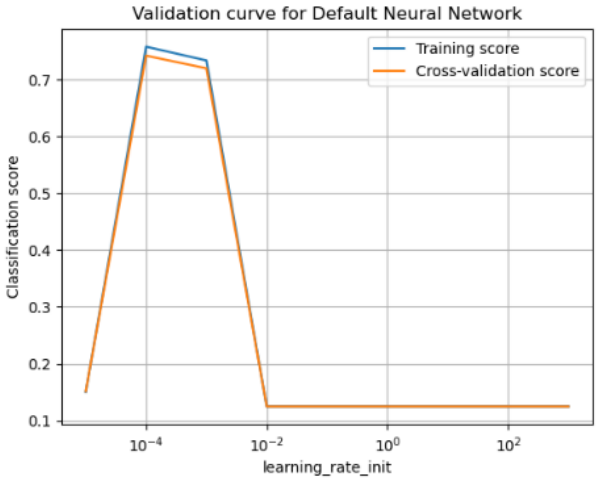
The first is a decision tree with default parameters while the second is the optimized decision tree (with pruning). For the default tree, the training set has a perfect score of 1.0 even using a small subset of the training data. Without the pre pruning, the model can fully learn the optimal parameters. From the learning curve of the tuned tree, the training score is affected by the pruning, but the cross validation score of the tuned tree generally outperforms the default decision tree for all fractions of training examples. This shows that while the training score was a bit lower, the tuned tree is able to better generalize.

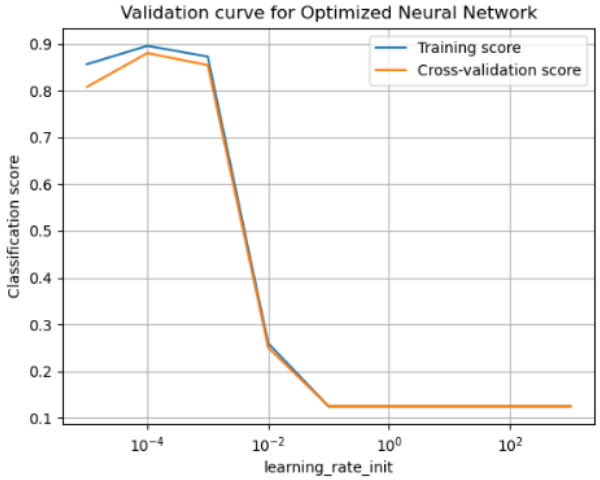
Overall, the tuned decision tree performed quite well on the lower limb dataset. From the validation curve, it can be seen that there is very low bias and somewhat low variance as well. Tuning the decision tree also increased the classification score of the cross validation set to ~0.97.

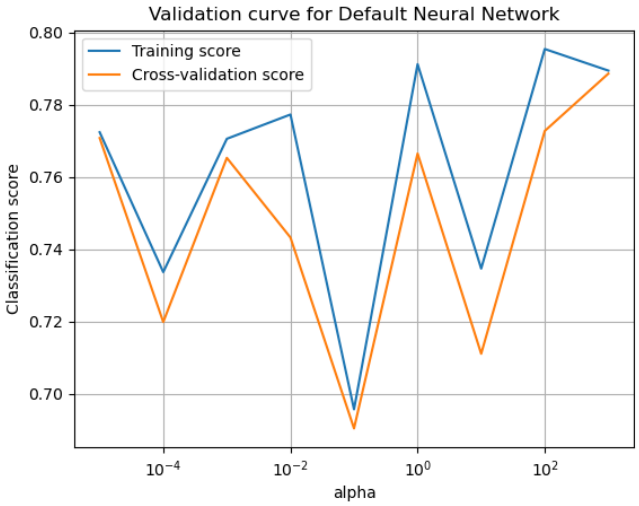
Neural Networks:

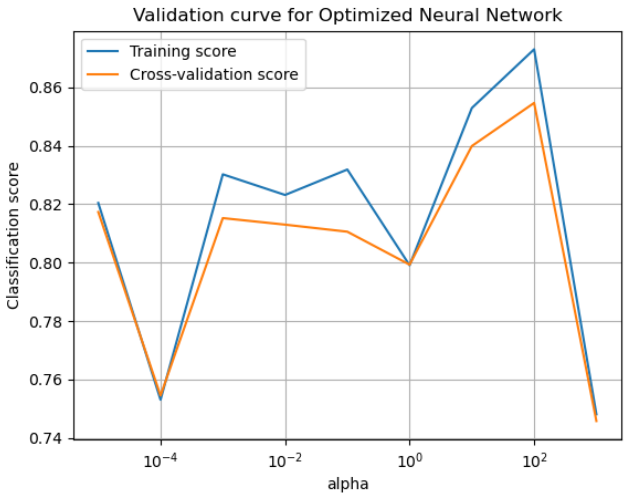


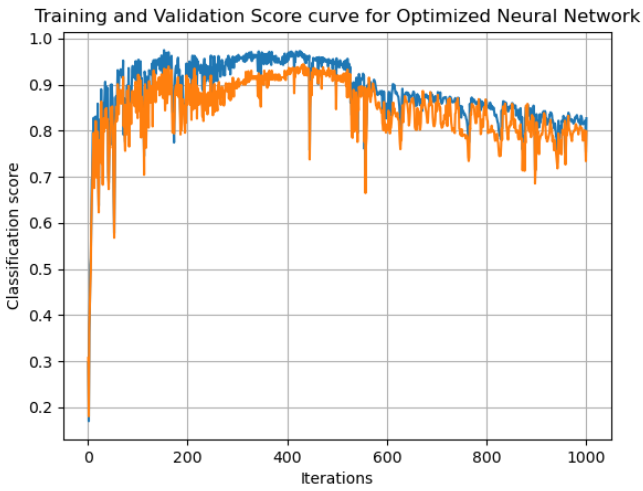


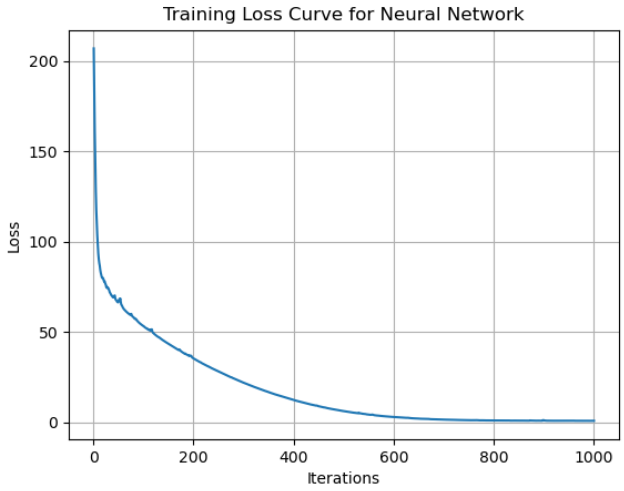


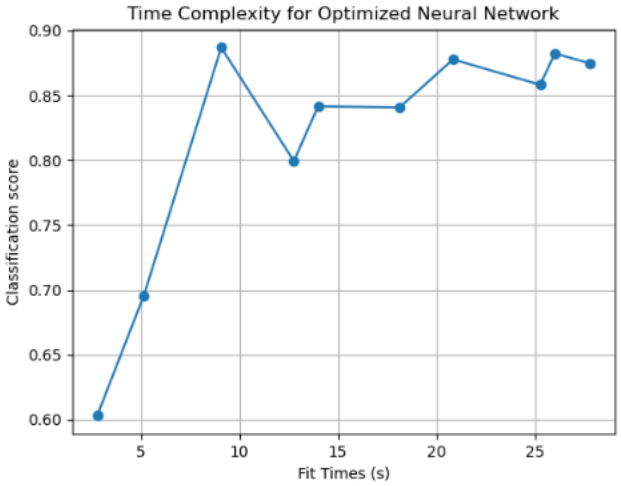












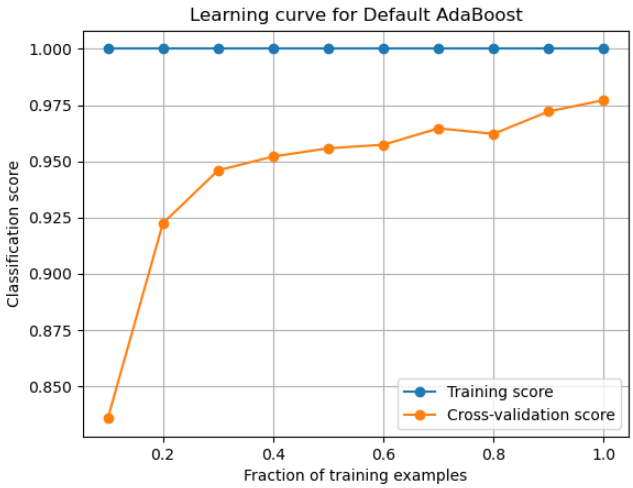
For the lower limb dataset, originally the neural network performed poorly. The variance was very low, but there was a high bias with the best classification score only reaching 0.85. To perform a grid search, a list of possible combinations for hidden layers and neurons was created based on the number of features (N) in the dataset. The number of hidden layers was varied from 1 to 5 and the number of neurons per layer was varied from N/2 to 2*N. Additionally in the grid search, values for alpha and the learning rate were explored.

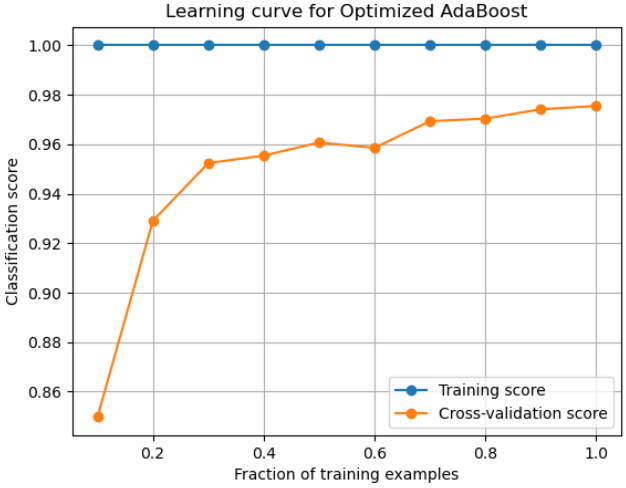
Tuning the parameters of the neural network allowed the model to greatly decrease the bias for both the training and cross validation sets but increased the variance which could signify some overfitting. The validation curve for alpha and learning rate originally had high biases but relatively small variance. After tuning the model, the bias was reduced. Similarly, for the learning rate, there was a high bias before optimization, and almost no variance. After using parameters from the grid search, the bias decreased to achieve an almost perfect classification score, and the variance stayed low.

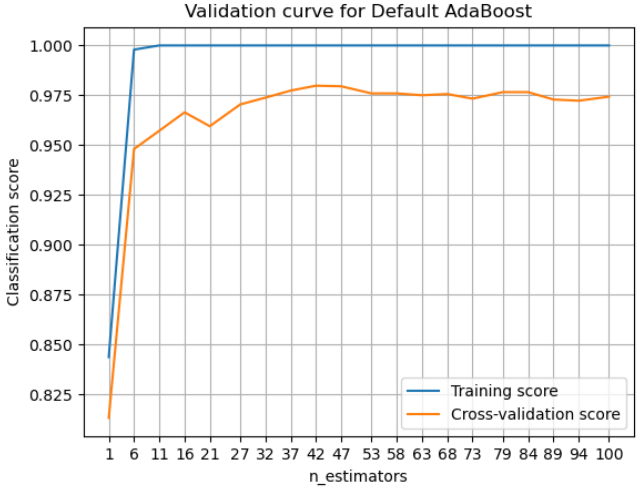
The learning rate controls updating the weights during training of the network. Using the loss curve, the learning rate is sufficient for this problem. Additional experiments showed that any higher the learning rate would be too aggressive, and the loss curve would converge at a higher level of loss, while a lower learning rate would take many more iterations to converge.

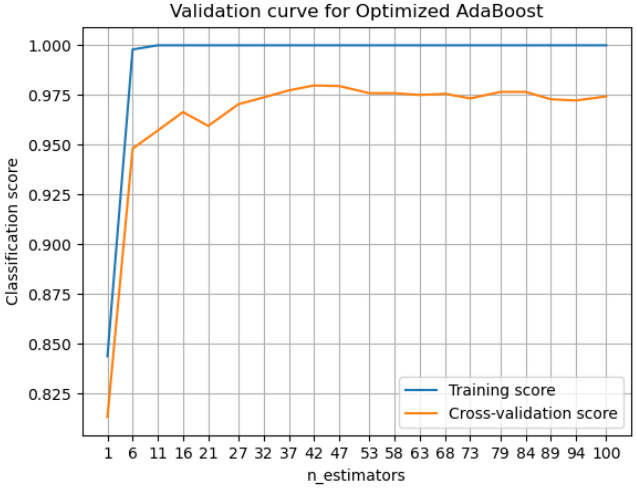
To create the training and validation score curve, the training data split into a smaller training set and validation set. This data was then used to fit a neural network model and the accuracy score of the predictions was calculated and plotted. From the plot, like before, the variance was low, but the bias increased after ~400 iterations of the model and the model does not converge in the given number of iterations. This suggests the model is not performing optimally and would benefit from additional optimization.

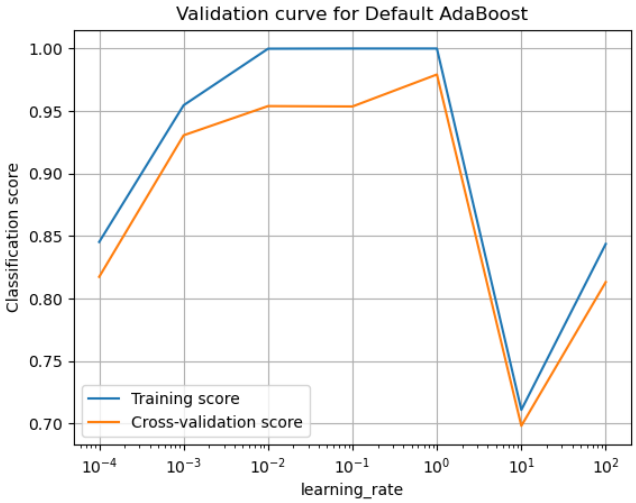
Boosting:

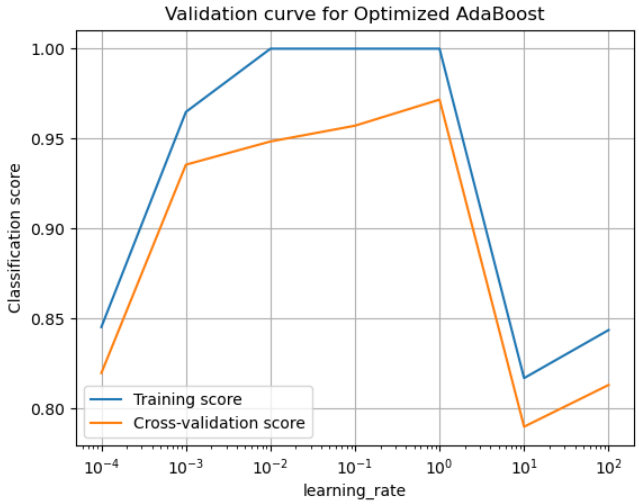

Learning curve for Default AdaBoost
Classification score
Fraction of training examples
Training score
Cross-validation score

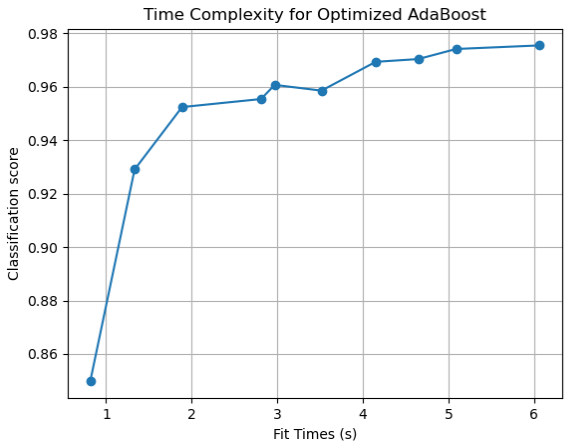

Learning curve for Optimized AdaBoost
Classification score
Fraction of training examples
Training score
Cross-validation score


Validation curve for Default AdaBoost
Classification score
n_estimators
Training score
Cross-validation score


Validation curve for Optimized AdaBoost
Classification score
n_estimators
Training score
Cross-validation score


Validation curve for Default AdaBoost
Classification score
learning_rate
Training score
Cross-validation score


Validation curve for Optimized AdaBoost
Classification score
learning_rate
Training score
Cross-validation score



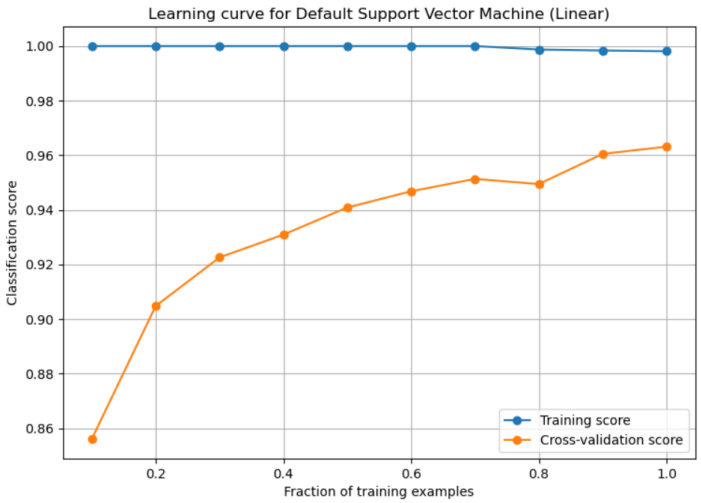
AdaBoost performed well on the dataset. To create the boosted learner, decision trees with different pre pruning (max depth of 1, 5, 10) were applied to the learner and a grid search was performed. A max depth of 5 provided the best performance. While the performance of the model was good, it was very resistant to optimization. Before and after tuning the bias and variance stayed similar (both values were quite small to begin with)**.** Additionally, the validation curves show very little change before and after tuning. One thing of note was the time complexity. Compared to the decision tree, the fit time was much greater, which was to be expected as the optimal number of estimators was 96 for this model.

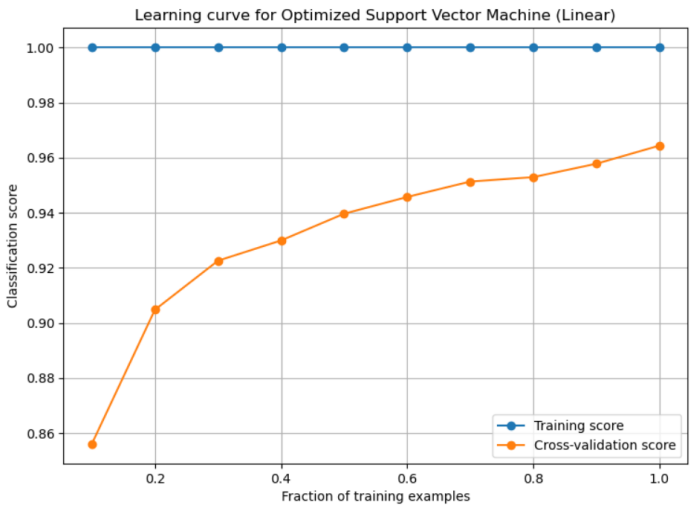
Support Vector Machines:

For SVM, scaled data was used. Because SVMs are scale variant and the dataset had data that varied by magnitudes between features, scaling would allow the SVM to optimize correctly. If the scaling did not occur, the optimal hyperplane would be influenced by the features with larger values.

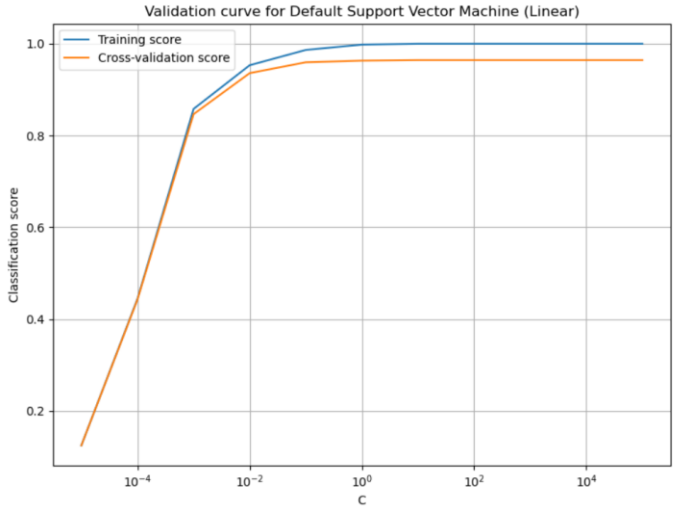
For both datasets, Linear and Poly kernels were investigated.

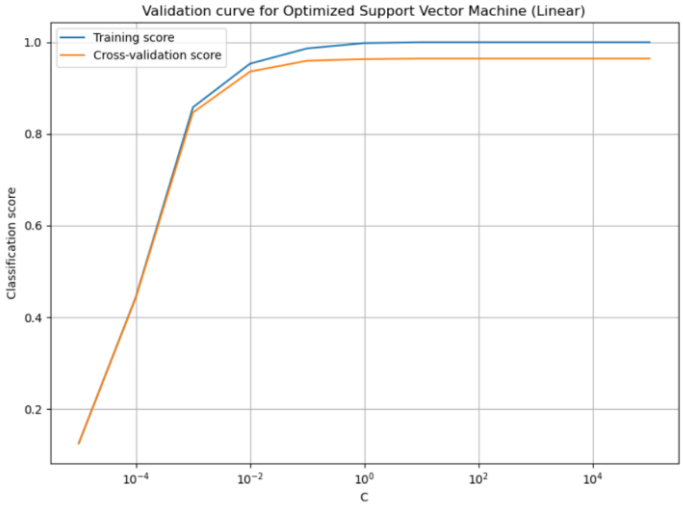
Linear Kernel

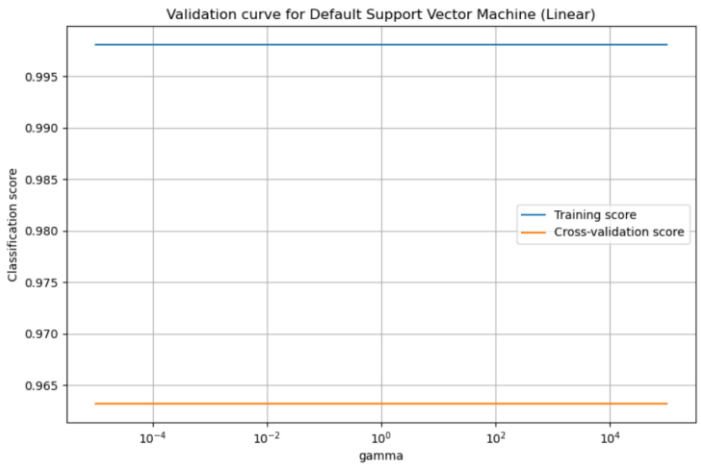


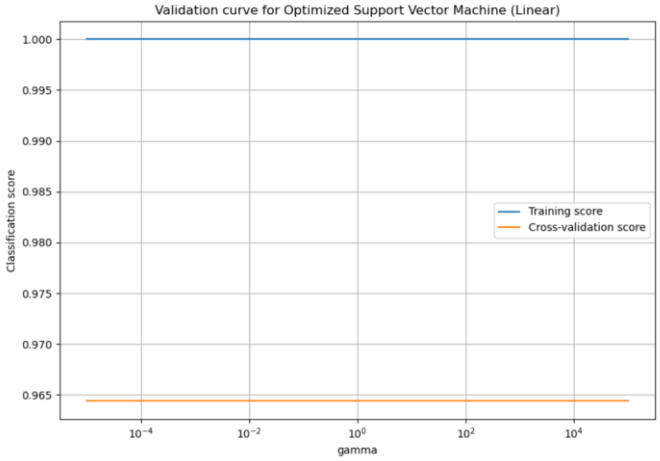


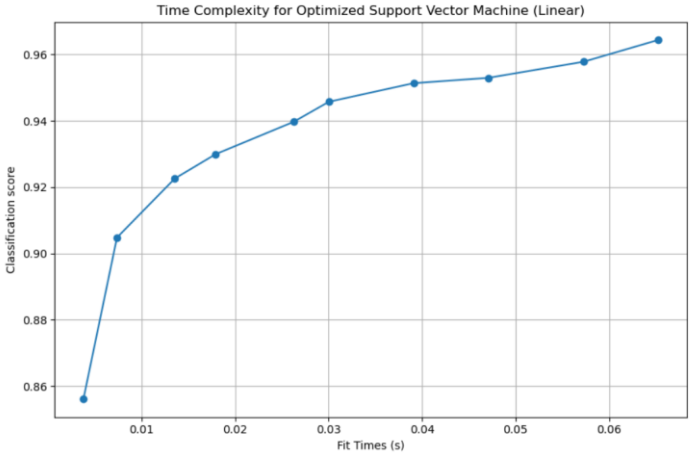
The linear SVM initially fit the data well. Low bias and minimal variance characterized the model. After tuning, the bias decreased slightly and the variance decreased for certain fractions of the training data as well. Looking at the validation curves for C and gamma, there is very low bias and variance, which are both decreased after tuning. This model seems to fit the data well, which is reflected by the high classification scores.



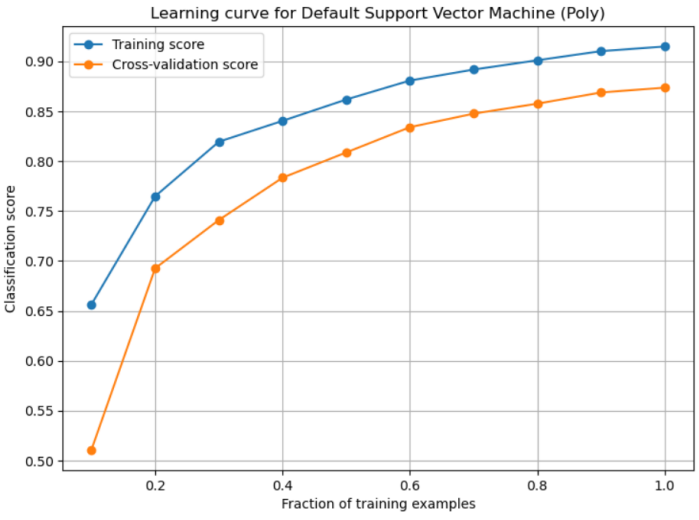


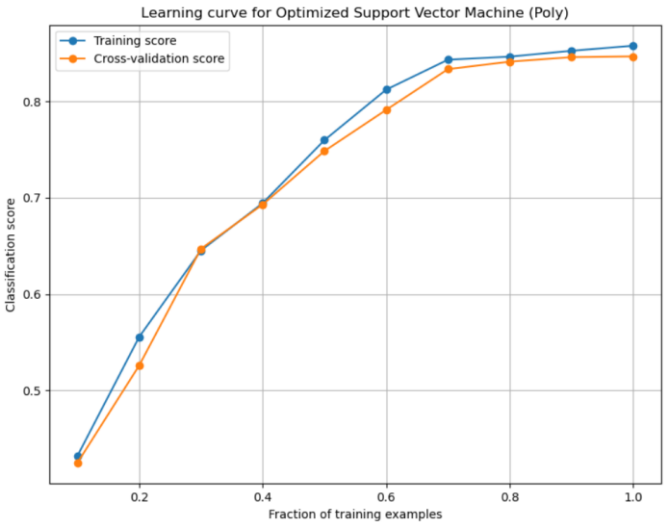




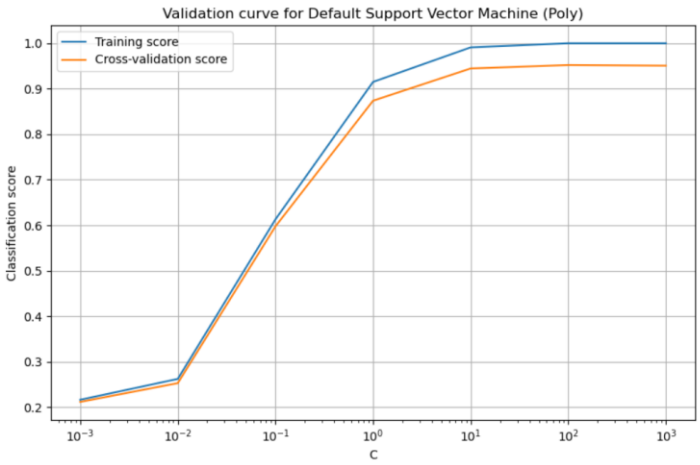


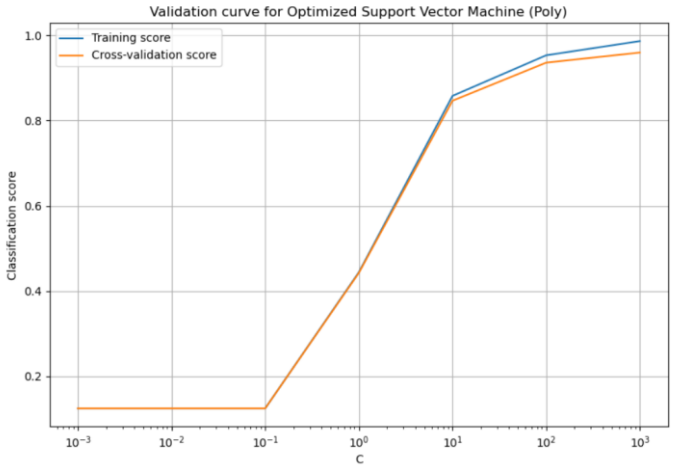
Polynomial Kernel

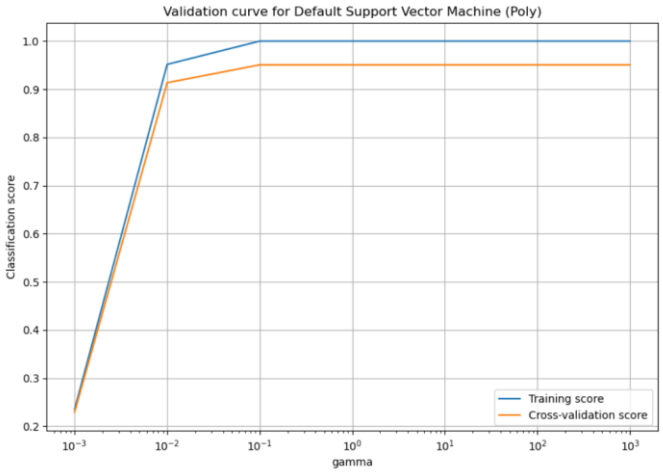


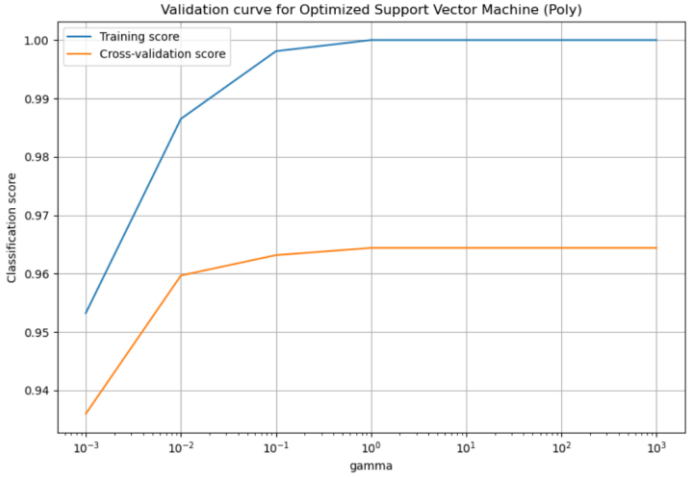


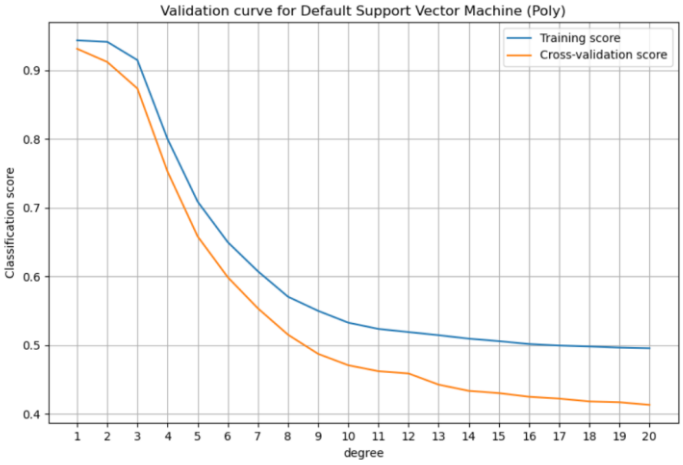
For poly, the initial model had low bias and relatively high variance, but after optimization the bias increased and variance decreased, indicating underfitting. Tuning C allowed the model to decrease the variance but increased the bias for some values of C. Tuning the poly SVM seemed to make the performance worse. The fit time increased while the classification score decreased and the optimal degree for the model was 1. This dataset may be linearly seperable which is why the linear kernel performed so well and the poly kernel is trying to act as 'linear' as possible.

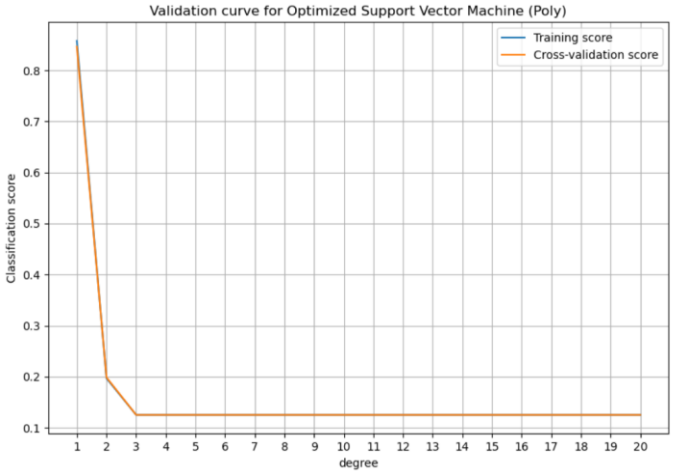

Validation curve for Default Support Vector Machine (Poly)
Training score
Cross-validation score
Classification score
C

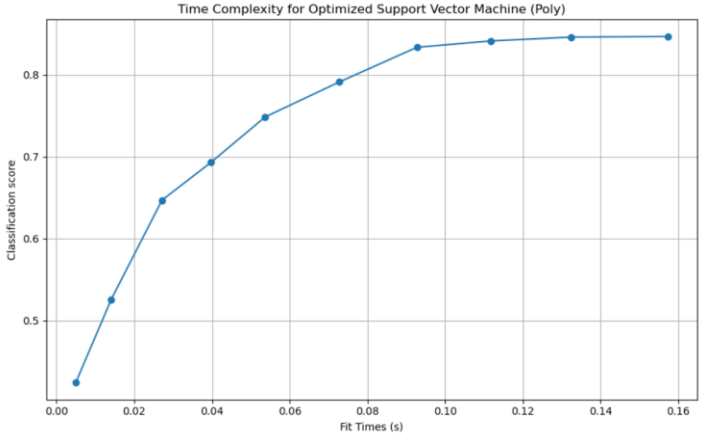

Validation curve for Optimized Support Vector Machine (Poly)
Training score
Cross-validation score
Classification score
C


Validation curve for Default Support Vector Machine (Poly)
Training score
Cross-validation score
Classification score
gamma

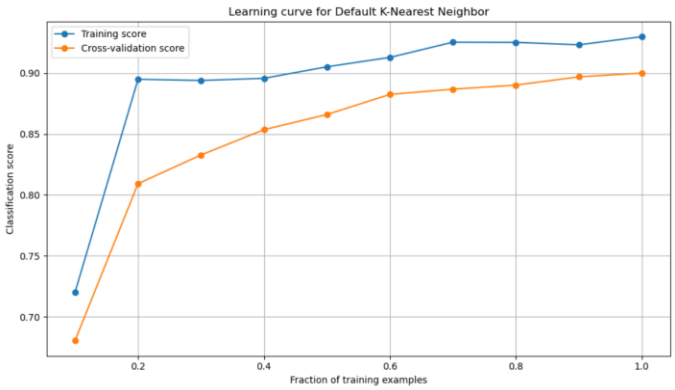

Validation curve for Optimized Support Vector Machine (Poly)
Training score
Cross-validation score
Classification score
gamma

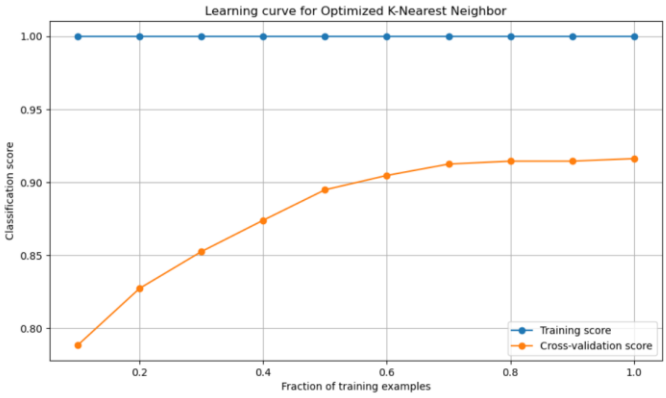

Validation curve for Default Support Vector Machine (Poly)
Training score
Cross-validation score
Classification score
degree


Validation curve for Optimized Support Vector Machine (Poly)
Training score
Cross-validation score
Classification score
degree


Time Complexity for Optimized Support Vector Machine (Poly)
Classification score
Fit Times (s)

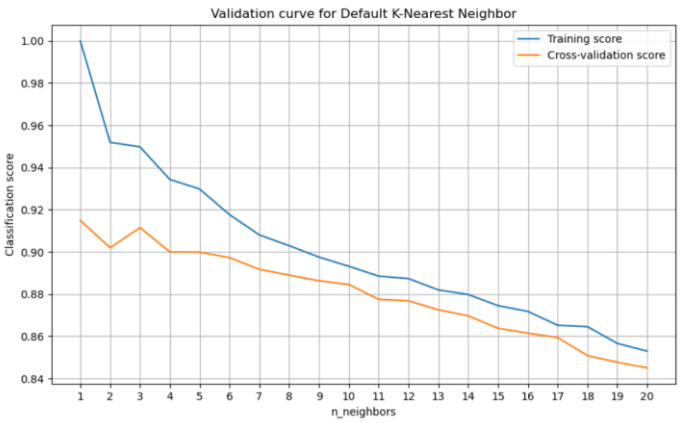
K-Nearest Neighbors:

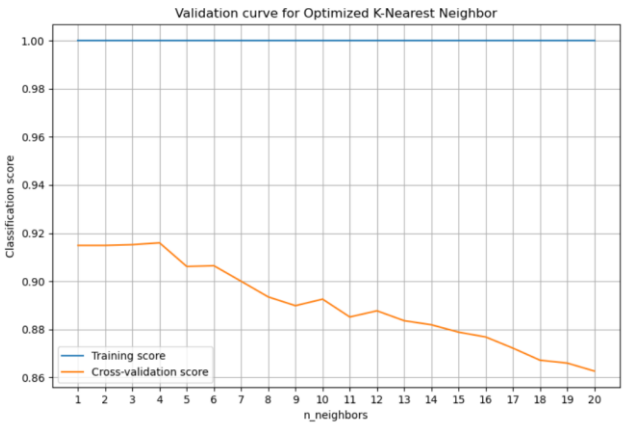


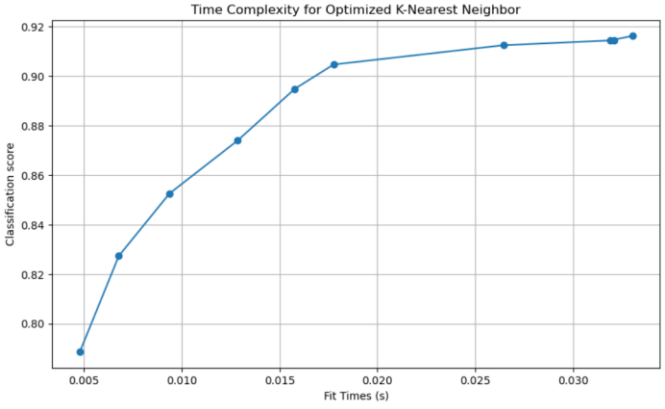


For the lower limb dataset, the KNN originally had a small bias and small variance, as the cross-validation set had a score of 0.90 when using the entire training set. The original parameters for the KNN were k=5 with a 'uniform' weight function used in prediction. Uniform weights mean that "all points in each neighborhood were weighted equally." [3] After running a grid search, the best parameters found were k=4 and 'distance' as the weight function. This does seem to somewhat overfit the data but looking at the validation curve for the number of neighbors, the accuracy of the model quickly degrades as the number of neighbors increases. Adding additional data could help to combat the overfitting.

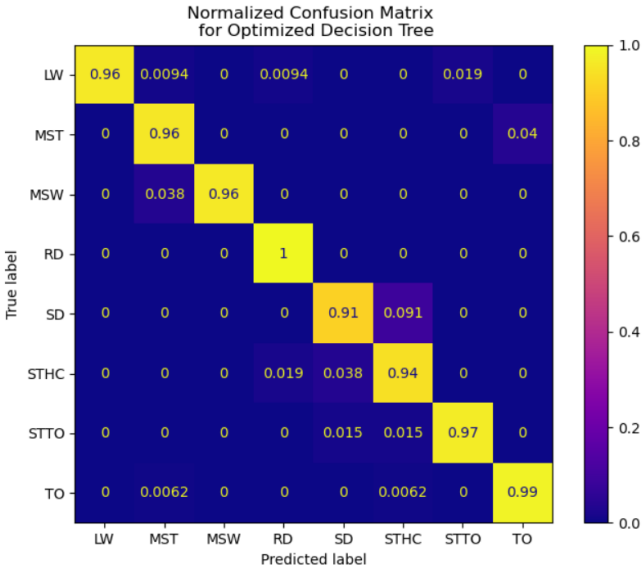
Looking at the learning curve for the optimized model, the training set has a perfect classification score. This change was due to the change in the weight function. Using the distance weighting, each point is weighted by the inverse of their distance. [3] Because the training curve is created using the training data, the points will have a 0 distance to themselves. Taking the inverse of this gives the points infinite weights and thus perfect predictions. The optimized model

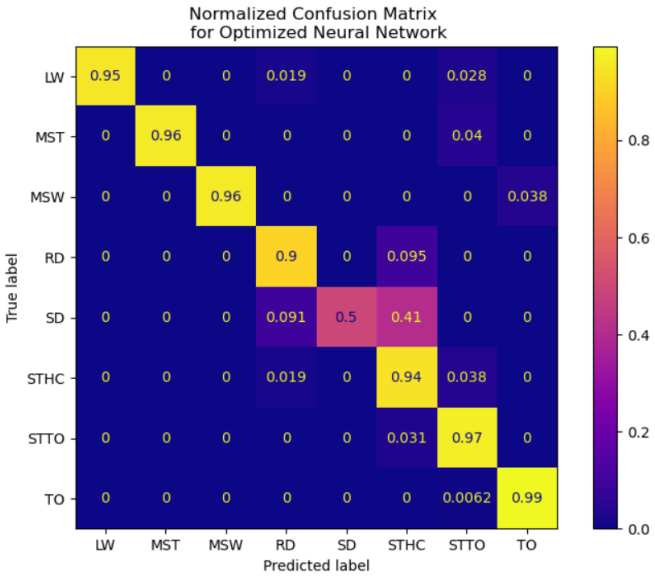


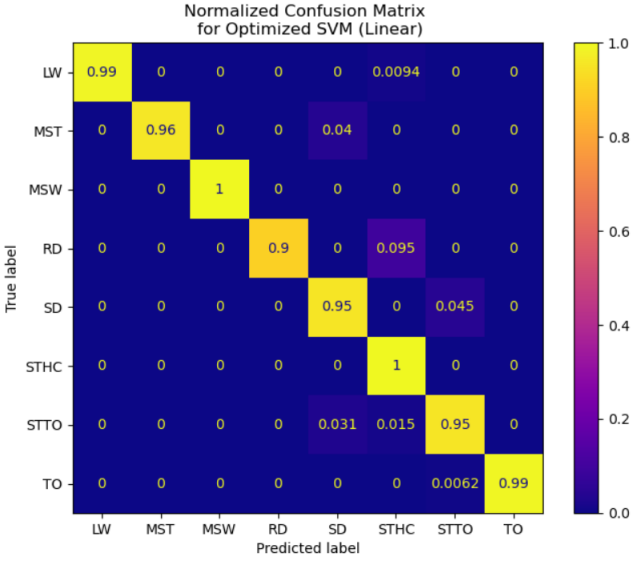


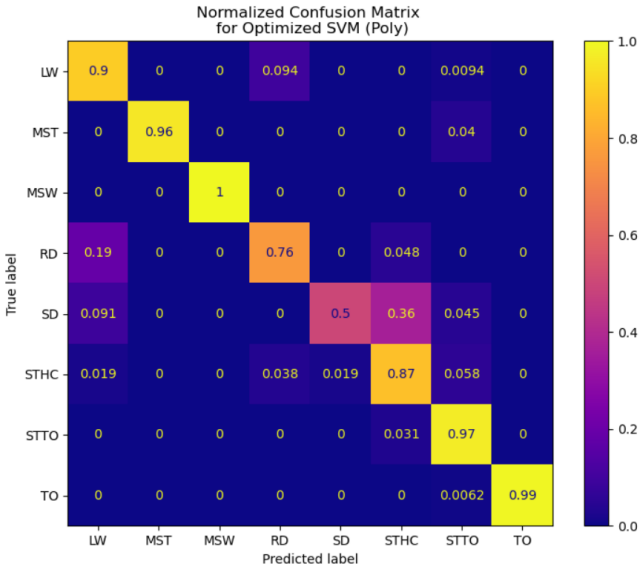


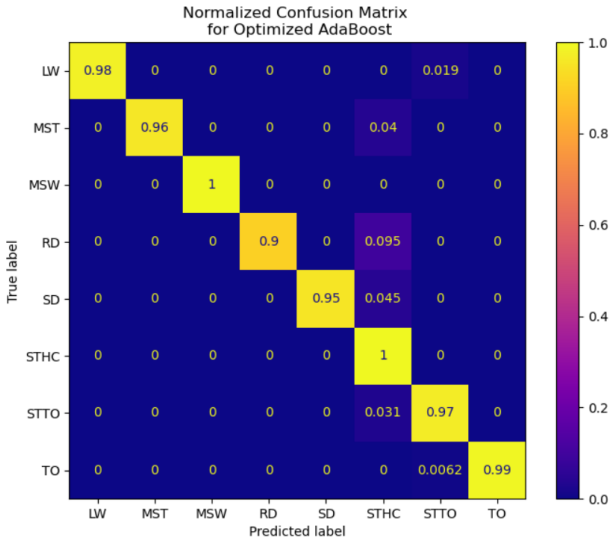
Discussion

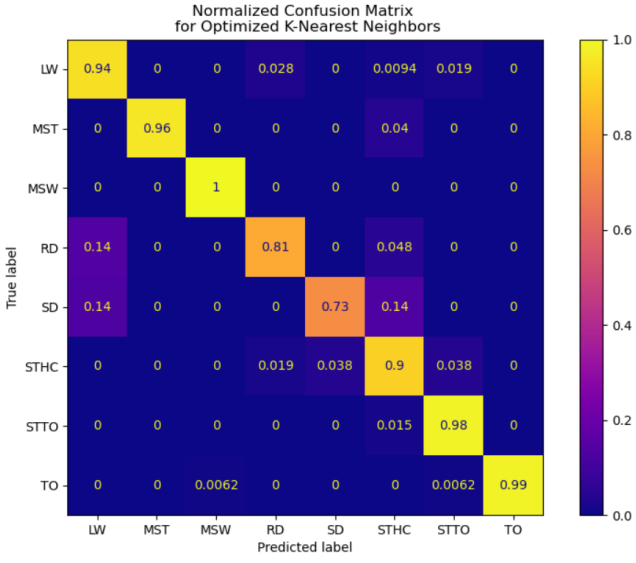

Normalized Confusion Matrix
for Optimized Decision Tree
True label
Predicted label
LW MST MSW RD SD STHC STTO TO
0.96 0.0094 0 0.0094 0 0 0.019 0
0 0.96 0 0 0 0 0 0.04
0 0.038 0.96 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0.91 0.091 0 0
0 0 0 0.019 0.038 0.94 0 0
0 0 0 0 0.015 0.015 0.97 0
0 0.0062 0 0 0 0.0062 0 0.99
1.0 0.8 0.6 0.4 0.2 0.0


Normalized Confusion Matrix
for Optimized Neural Network
True label
Predicted label
LW MST MSW RD SD STHC STTO TO
0.95 0 0 0.019 0 0 0.028 0
0 0.96 0 0 0 0 0.04 0
0 0 0.96 0 0 0 0 0.038
0 0 0 0.9 0 0.095 0 0
0 0 0 0.091 0.5 0.41 0 0
0 0 0 0.019 0 0.94 0.038 0
0 0 0 0 0 0.031 0.97 0
0 0 0 0 0 0 0.0062 0.99
1.0 0.8 0.6 0.4 0.2 0.0


Normalized Confusion Matrix
for Optimized SVM (Linear)
True label
Predicted label
LW MST MSW RD SD STHC STTO TO
0.99 0 0 0 0 0.0094 0 0
0 0.96 0 0 0.04 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0.9 0 0.095 0 0
0 0 0 0 0.95 0 0.045 0
0 0 0 0 0 1 0 0
0 0 0 0 0.031 0.015 0.95 0
0 0 0 0 0 0 0.0062 0.99
1.0 0.8 0.6 0.4 0.2 0.0


Normalized Confusion Matrix
for Optimized SVM (Poly)
True label
Predicted label
LW MST MSW RD SD STHC STTO TO
0.9 0 0 0.094 0 0 0.0094 0
0 0.96 0 0 0 0 0.04 0
0 0 1 0 0 0 0 0
0.19 0 0 0.76 0 0.048 0 0
0.091 0 0 0 0.5 0.36 0.045 0
0.019 0 0 0.038 0.019 0.87 0.058 0
0 0 0 0 0 0.031 0.97 0
0 0 0 0 0 0 0.0062 0.99
1.0 0.8 0.6 0.4 0.2 0.0


Normalized Confusion Matrix
for Optimized AdaBoost
True label
Predicted label
LW MST MSW RD SD STHC STTO TO
0.98 0 0 0 0 0 0.019 0
0 0.96 0 0 0 0.04 0 0
0 0 1 0 0 0 0 0
0 0 0 0.9 0 0.095 0 0
0 0 0 0 0.95 0.045 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0.031 0.97 0
0 0 0 0 0 0 0.0062 0.99
1.0 0.8 0.6 0.4 0.2 0.0


Normalized Confusion Matrix
for Optimized K-Nearest Neighbors
True label
Predicted label
LW MST MSW RD SD STHC STTO TO
0.94 0 0 0.028 0 0.0094 0.019 0
0 0.96 0 0 0 0.04 0 0
0 0 1 0 0 0 0 0
0.14 0 0 0.81 0 0.048 0 0
0.14 0 0 0 0.73 0.14 0 0
0 0 0 0.019 0.038 0.9 0.038 0
0 0 0 0 0 0.015 0.98 0
0 0 0.0062 0 0 0 0.0062 0.99
1.0 0.8 0.6 0.4 0.2 0.0

Overall, the models performed quite well on the dataset. Decision Trees, Boosting, and Linear SVM performed the best while Neural Networks and Poly SVM performed poorly comparatively. The classes they performed the worst in were also the classes which had the least number of examples, which intuitively makes sense.

References:

[1] Spanias, J., Simon, A., Finucane, S., Perreault, E., & Hargrove, L. (2018, February). Online adaptive neural control of a robotic lower limb prosthesis. Retrieved September 17, 2020, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5802866/

[2] https://cs231n.github.io/neural-networks-3/

[3] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html