Supervised Learning - Analysis

Kira Yang
wyang36@gatech.edu

Abstract—Supervised Learning is a subfield of Machine Learning where models are trained using labeled datasets. In this project, 5 different popular supervised learning algorithms are implemented and used to solve two different classification problems. Their performances are then compared in order to understand the pros and cons for each algorithm.

1 CLASSIFICATION PROBLEMS

In order to explore and perform analysis of various supervised learning algorithms in depth, two public datasets well suited for classification tasks are selected from Kaggle.com. Below are brief introductions of the chosen datasets..

1.1 Heart Disease Prediction[1]

The first classification problem is to predict heart disease using the patients' symptoms and demographic information. A brief overview of the raw dataset is shown in Table 1.

Table 1—Snippet of Heart Disease Prediction Dataset

Age	Sex	ChestPain Type	RestingBP	Cholesterol	Fasting BS	RestingECG	MaxHR	Exercise Angina	Oldpeak	ST_Slope	HeartDisease
40	M	ATA	140	289	0	Normal	172	N	0	Up	0
49	F	NAP	160	180	0	Normal	156	N	1	Flat	1
37	M	ATA	130	283	0	ST	98	N	0	Up	0
48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1

This is a binary classification problem with 918 total rows, and 11 attributes of various data types. The data has no missing values and is fairly balanced shown in Figure 1. Because of the cleanliness of the dataset, the only preprocessing needed is nominal encoding for categorical attributes and normalization for the numerical attributes.

[1] "Heart Failure Prediction Dataset." *Heart Failure Prediction Dataset | Kaggle*, /datasets/fedesoriano/heart-failure-prediction. Accessed 31 Jan. 2023.

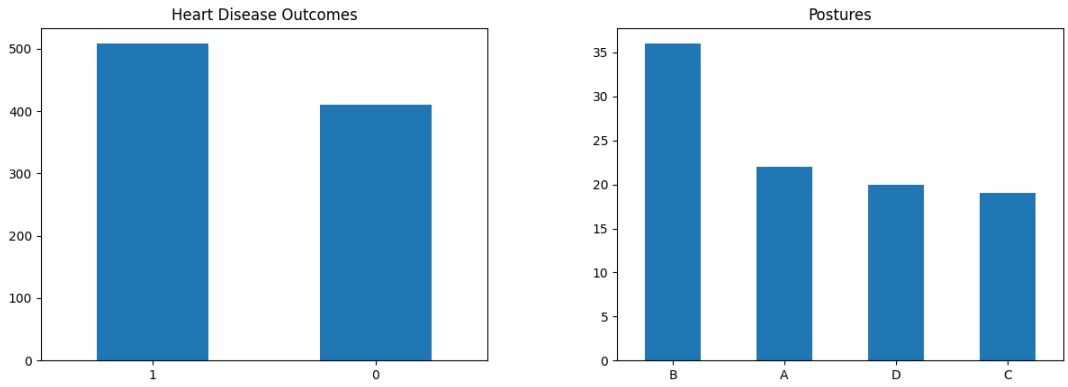


Figure 1—y value distribution of Heart Disease Prediction Dataset (left) and Posture Prediction Dataset (right)

1.2 Posture Prediction using Myers Briggs[2]

The second problem is to predict a person's posture using their Myer Briggs personality profile. A brief overview is shown in Table 2.

Table 2—Snippet of Posture Prediction Dataset

AGE	HEIGHT	WEIGHT	SEX	ACTIVITY LEVEL	PAIN 1	PAIN 2	PAIN 3	PAIN 4	MBTI	E	I	S	N	T	F	J	P	POSTURE
53	62	125	Female	Low	0	0	0	0	ESFJ	18	3	17	9	9	13	18	4	A
52	69	157	Male	High	7	8	5	3	ISTJ	6	15	14	12	21	3	13	9	B
30	69	200	Male	High	0	0	0	0	ESTJ	15	6	16	10	15	9	12	10	A
51	66	175	Male	Moderate	9.5	9.5	9.5	1.5	ISTJ	6	15	21	5	13	11	19	3	D

This is a multi-class classification problem with only 97 rows, and 18 attributes of various types. This is also a fairly clean dataset with no missing values. However, the y values of the data are not balanced with a high occurrence of B. This is a much more challenging problem due to its multi-class nature, small sample size, and imbalance, which are all highly prevalent problems in the real world datasets. Therefore, a 90/10 training and test split is used instead of the standard 80/20 to maximize training data and utilize *balanced_accuracy* as its performance metric for all models.

[2] "Correlation Between Posture and Personality Trait." *Correlation Between Posture & Personality Trait | Kaggle*, /datasets/dhanasekarjaisankar/correlation-between-posture-personality-trait. Accessed 31 Jan. 2023.

2 ALGORITHMS

Similar training and testing procedures are followed for all 5 algorithms on both datasets in order to stay consistent. The `scikit-learn` library in Python is chosen to run all the experiments because of its extensive features and detailed documentation.

First, all the string attributes are encoded in a nominal manner using `pd.get_dummies`, as most classifiers in the `scikit-learn` library do not recognize categorical attributes by default. The numerical attributes are normalized for all the non-tree classifiers. Tree-based models only look at one attribute at a time, so normalization would not be necessary.

After preprocessing, the available data is split into training and test set, using training for all the tuning and learning curve analysis, and reserved the test set only for the final evaluation. The train/test split ratio is dataset dependent.

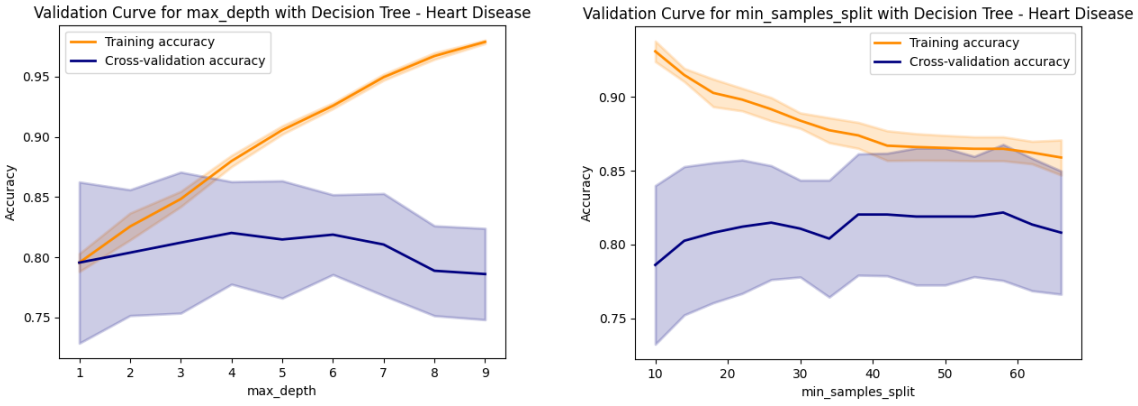
With the training set, 2 hyperparameters are chosen and tuned for each classifier and we plot the validation curve for each dataset, all using 10-fold cross validation. The optimized hyperparameters are then found using `GridSearchCV`. With the optimal parameters, we plot the learning curve after experimenting with different training data sizes in order to observe whether we have sufficient data for training.

Finally, a new model is re-train with the optimal hyperparameters found earlier, and the final training/test accuracy metrics and training/query time is recorded using the reserved test set. The performance of each classifier on each dataset is then compared and analyzed.

2.1 Decision Tree

Decision trees are a type of highly popular supervised machine learning algorithms due to its intuitive nature and its insensitivity to missing data. However, they're also known for their potential long training time and their tendency to overfit. In our experiments, pre-pruning is utilized to mitigate overfitting.

With the training set, we pre-pruned the trees by tuning the hyperparameters `max_depth` and `min_sample_split`. Pre-pruning is chosen instead of post-pruning in order to avoid long training time caused by overgrowing the tree. Validation curves for the 2 hyperparameters for both datasets are shown in Figure 2.



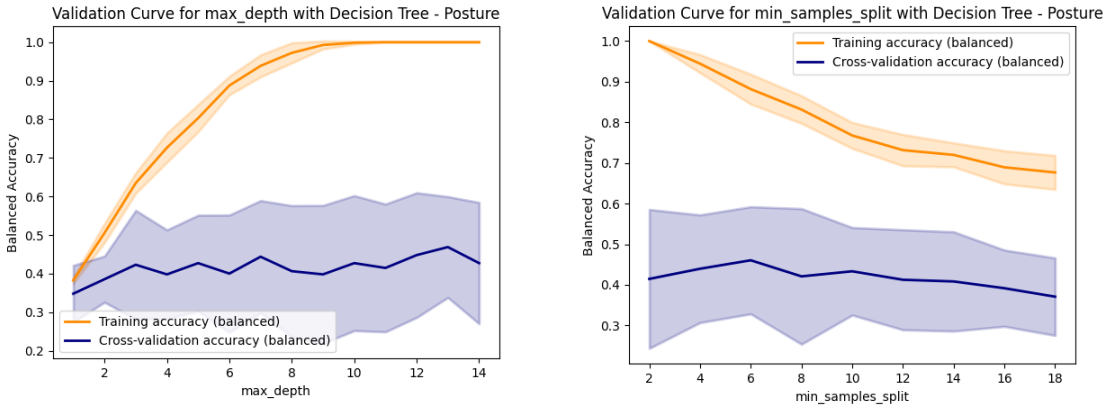


Figure 2—Validation curves of Decision Tree

As shown in Figure 2, a smaller tree with a `max_depth` 4 and `min_sample_split` of around 50 is performing the best for the validation sets of the Heart Disease Prediction dataset; any larger tree results in overfitting. In contrast, the small Posture Prediction dataset is favoring a much deeper tree while the optimal `min_sample_split` is still very low. This suggests a dataset this small is still struggling to find a generalized pattern even after reaching the depth where it would most likely overfit with larger datasets.

The data size issue can be further explored by plotting the learner curves shown in Figure 3. As we can see, for both datasets, validation accuracy is rising continuously as we increase the number of samples in the training set. In the Heart Disease Prediction dataset, the validation accuracy started to taper off and stabilize close to the training accuracy towards the right end of the chart. In the Posture Prediction dataset however, the validation accuracy is still increasing steadily even as we reach the far right of the graph where we were using all available data. This indicates that the Posture Prediction model has vast potential for improvement if more data is available for training.

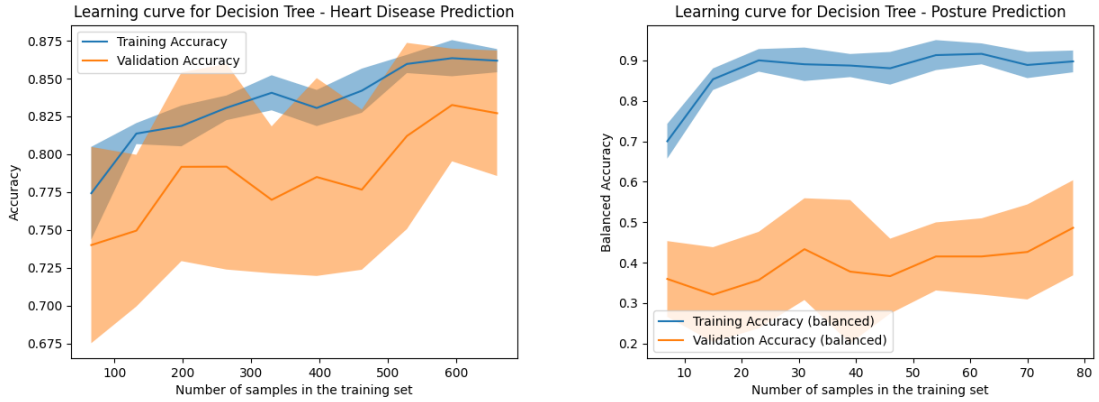


Figure 3—Learning curves of Decision Tree

The final metrics of the optimized decision trees are shown in Table 3. To summarize, the optimized decision tree performed nicely for the Heart Disease dataset with consistently over 80% accuracy. However, it performed only marginally better than chance for the Posture Prediction set. The tree is fast to train and even faster to query for both datasets, which is expected due to the data size and the fact we utilized pre-pruning.

Table 3—Final performance metrics - Decision Tree

	Training Time	Query Time	Training Accuracy	Test Accuracy
Heart Disease	0.002571s	0.001808s	0.862398	0.875
Posture	0.002107s	0.001718s	0.851634	0.3125

2.2 Neural Network

Artificial Neural Networks (ANN) are a type of deep learning algorithm inspired by the biological structure of the brain. In this project, we are utilizing the `MLPClassifier` from the `scikit-learn` library. The hyperparameters chosen for the ANNs are `hidden_layer_sizes` and `activation`. Validation curves are shown in Figure 4.

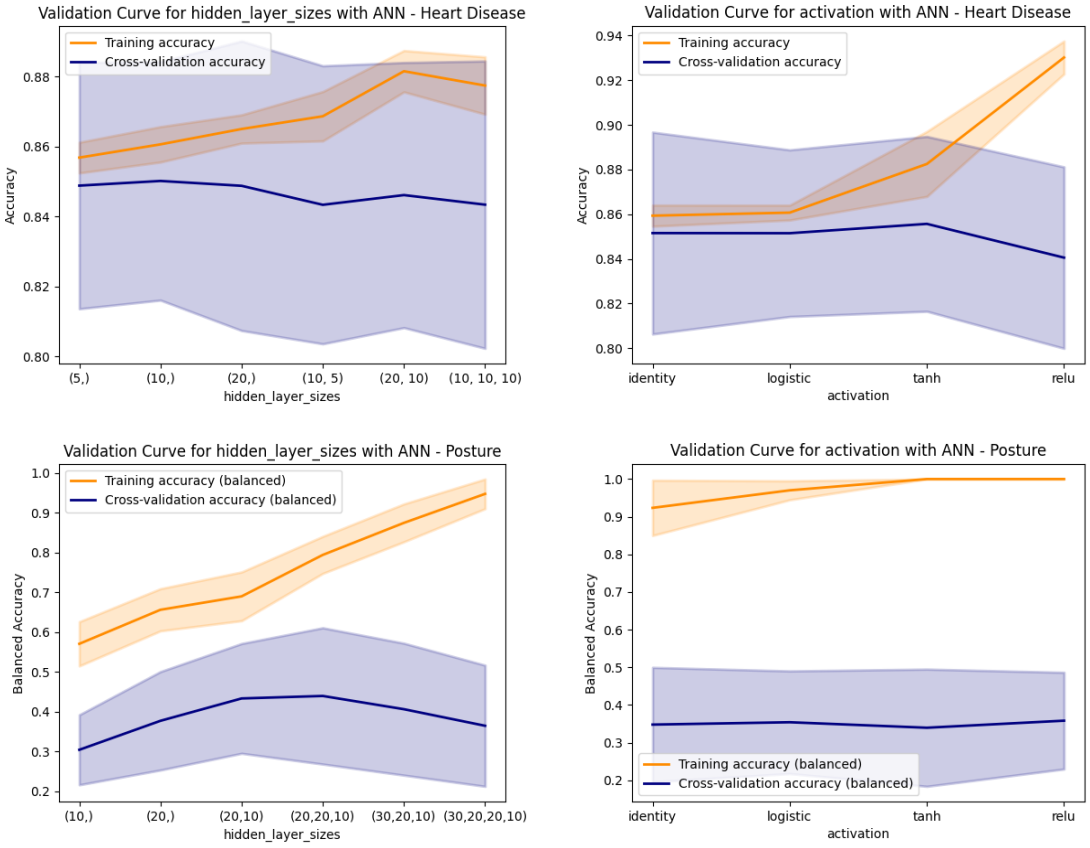


Figure 4—Validation curves of Neural Network

As shown in Figure 4, a larger network is not always better. For the Heart Disease dataset, 10 perceptrons in 1 hidden layer appear to be sufficient, while the Posture Prediction dataset appears to perform better with a larger and deeper network, suggesting smaller datasets to be less vulnerable to overfitting. As for `max_iter`, 200 iterations appears to be sufficient for both datasets, any further only improves training accuracy.

To explore the effect of data size on the model's performance, learning curves for both datasets are shown in Figure 5. As we can see, the gap between training and validation accuracy at the right end of the chart for the Heart Disease dataset is quite small, while the same gap for Posture Prediction dataset remains large. This suggests the ANN can also benefit from more training data from the Posture Prediction dataset.

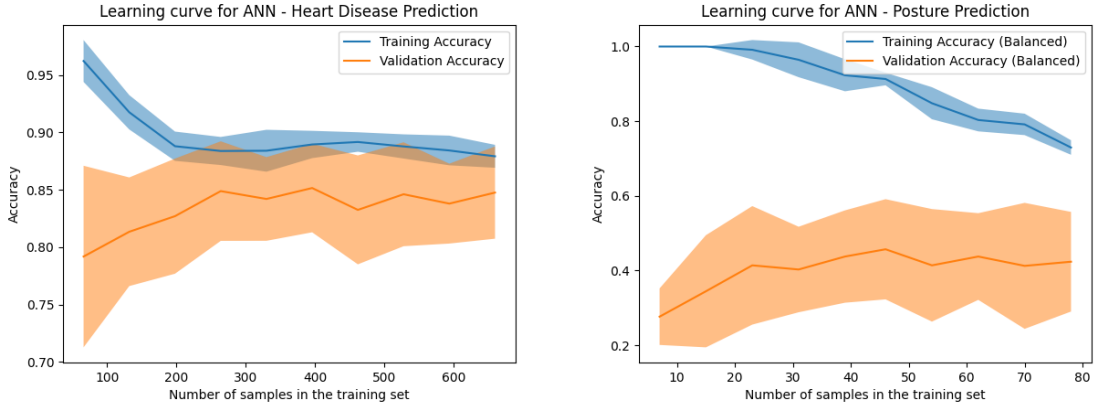


Figure 5—Learning curves of Neural Network

Next, we examine the loss curves of the neural networks, shown in Figure 6. As we can see, both models have converged to the fairly low final loss. The curve for the Posture Prediction dataset looks more linear than the Heart Disease Prediction set, signifying that we may be able to use a slightly higher learning rate with success.

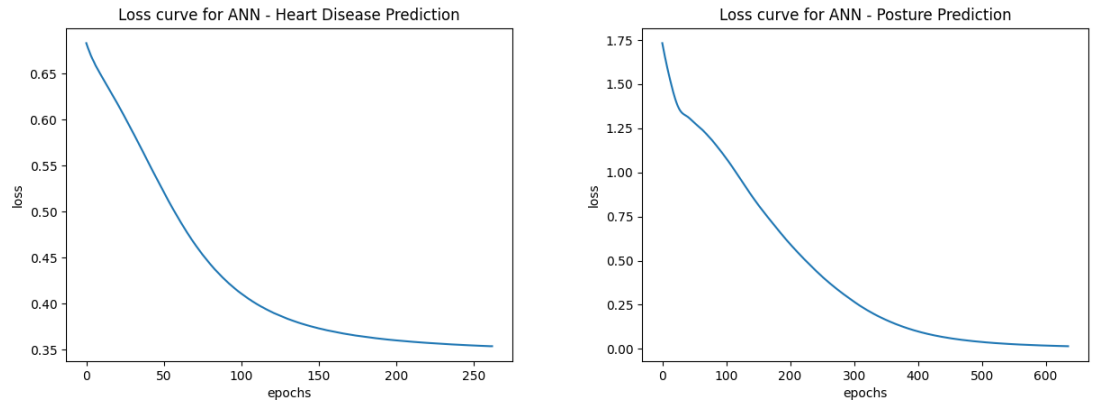


Figure 6—Loss curves of Neural Network

Finally, the performance of the finalized optimal neural network is recorded in Table 4. Compared to the Decision Tree, the ANNs are slower to train, but quicker to query. Furthermore, while the accuracy for the Heart Disease dataset is comparable, the balanced accuracy for the Posture Prediction dataset is noticeably higher, indicating that the ANN can be a good option for multi-class classification problems with limited data.

Table 4—Final performance metrics - Neural Network

	Training Time	**Query Time**	**Training Accuracy**	**Test Accuracy**
Heart Disease	0.372967s	0.000705s	0.867847	0.896068

Posture	0.374703s	0.000655s	1.0	0.541666

2.3 Boosting

Boosting is a supervised learning method that creates a strong learner using an ensemble of weak learners. In this project, we experiment with using the `AdaBoostClassifier` from `scikit-learn` with `DecisionTreeClassifier` as its estimator. The hyperparameters chosen for boosting are `n_estimators` and `max_depth` of each decision tree, as it would be interesting to explore if an ensemble would improve the performance of a single tree produced in Section 2.1.

The validation curves for both hyperparameters are shown in Figure 7. As we can see, for the Heart Disease Prediction dataset, a small ensemble of deeper trees appears to be performing the best, while the Posture Prediction dataset favors a large ensemble of very small trees (`max_depth`=1). However, it's also interesting to note that the Posture Prediction dataset is not showing an overfitting pattern even as we grow each tree to as deep as `max_depth`=15, unlike the steep dropoff in accuracy in the Heart Disease dataset plot. This is a good illustration that a boosting algorithm will only overfit when each estimator is already overfitting themselves.

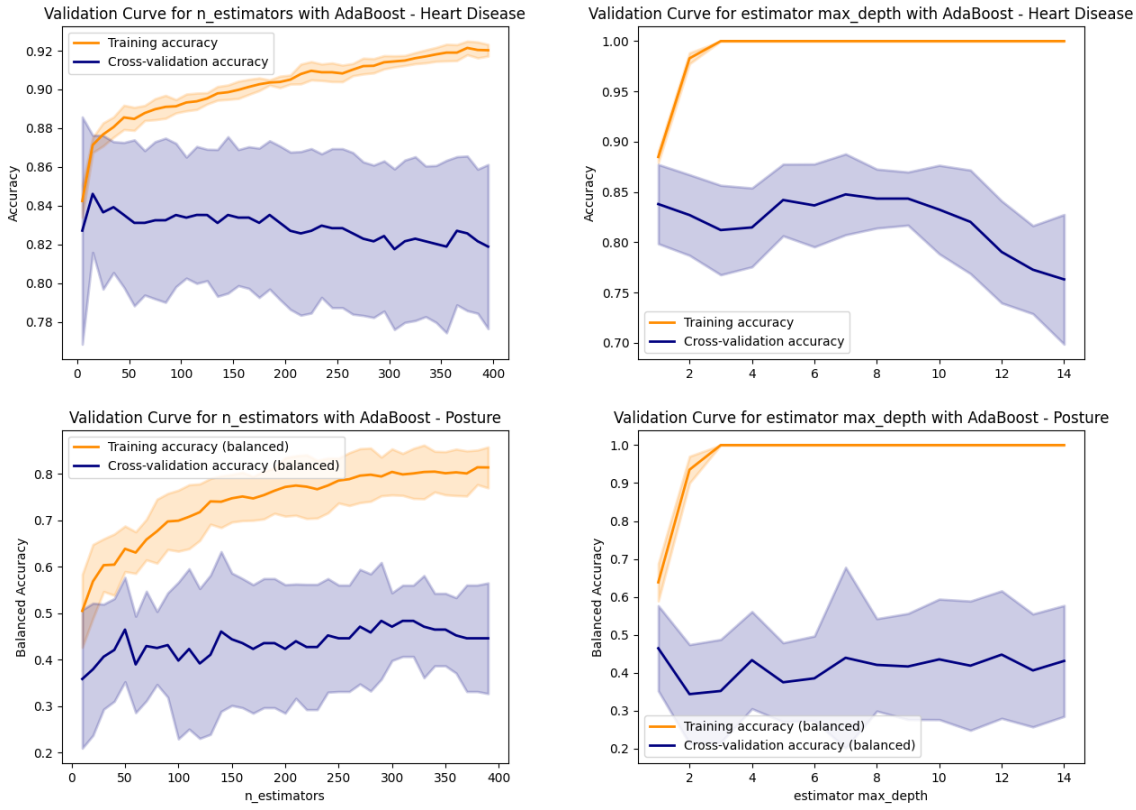


Figure 7—Validation curves of AdaBoost

To understand the effect of training data size, the learning curves for both datasets are shown in Figure 8. The Heart Disease dataset's training accuracy in the learning curves are alway at 1.0 because the plot is generated using the optimal hyperparameters found by `GridSearchCV`, which happened to be a deep tree

(`max_depth`=6). The validation accuracy for the Posture Prediction plot is increasing continuously as we increase training data size, while the Heart Disease plot staggers when we get past 350 rows, indicating the Posture Prediction models can benefit from more training data.

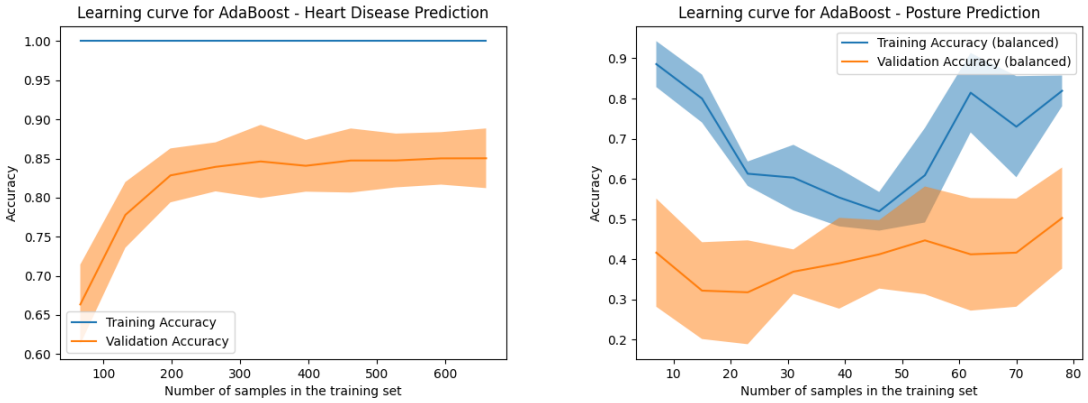


Figure 8—Learning curves of AdaBoost

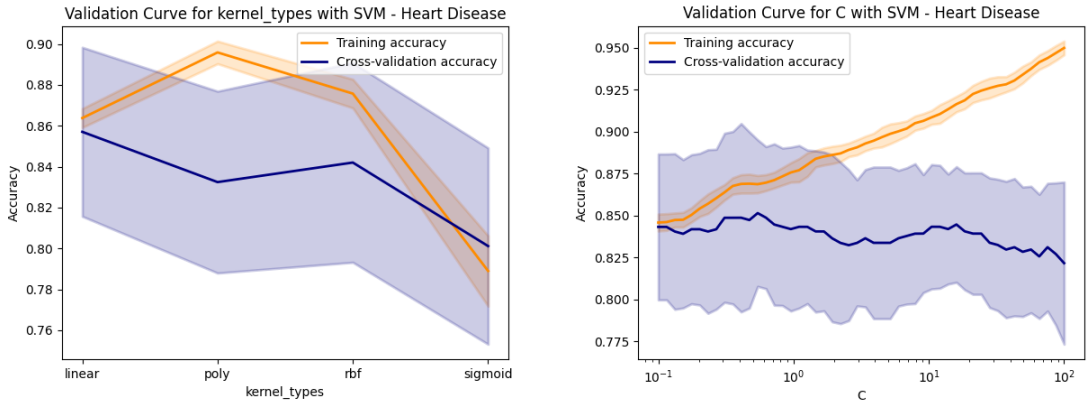
Finally, the final performance metrics of the optimal ensemble are recorded in Table 5. Compared to the single decision tree, the ensemble is significantly slower to train and slow to query, especially for the Posture Prediction dataset since the optimal `n_estimators` is over 300. However, the ensemble also produced significantly higher test accuracy for the Posture Prediction problem, suggesting that a boosting algorithm can be a good candidate for a problem with very little data.

Table 5—Final performance metrics - Boosting (AdaBoost)

	Training Time	Query Time	Training Accuracy	Test Accuracy
Heart Disease	0.667345s	0.030412s	1.0	0.880434
Posture	0.650539s	0.078547s	0.791507	0.5

2.4 Support Vector Machine

Support Vector Machines (SVMs) is a type of supervised learning model that classifies data with the goal of maximizing separation between different classes. For this project, we are utilizing the `sklearn.svm.SVC` class and experimenting with hyperparameters `kernel`, which is the type of kernel used, and `C`, which is the regularization parameter. Validation curves are shown in Figure 9.



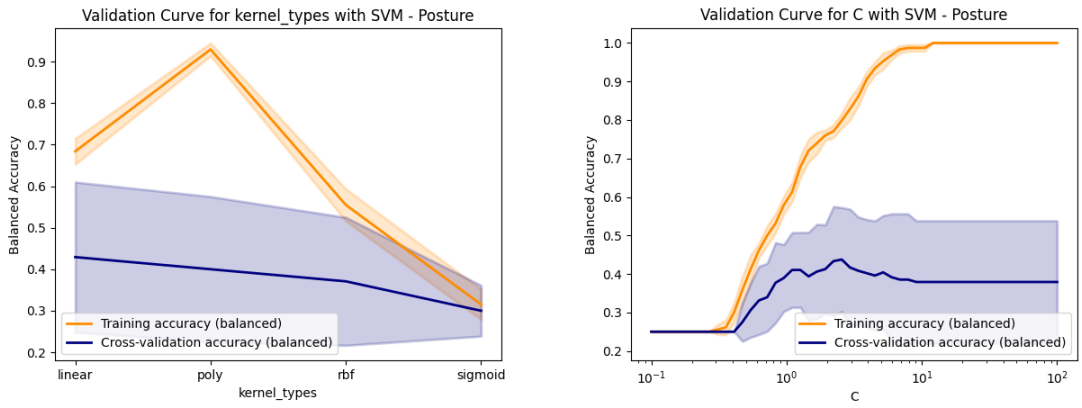


Figure 9—Validation curves of Support Vector Machine

As we can see, the SVM favors a linear kernel with a lower C for both datasets. This is expected as more complicated kernel functions or higher C values are more likely to overfit. However, it is also worth noting that the model is already choosing simpler hyperparameters for very limited training data, suggesting it is generalizing much quicker than other algorithms. This can be further explored by plotting the learning curves, shown in Figure 10.

As expected, the gap between training and validation accuracy is quite small towards the right end of the plot for both datasets, indicating we're approaching the optimal performance with the training data we have, which is impressive especially for the Posture Prediction dataset with only about 80 samples in our training set. With the optimal hyperparameters, we train a new model for each dataset and record their final performance in Table 6.

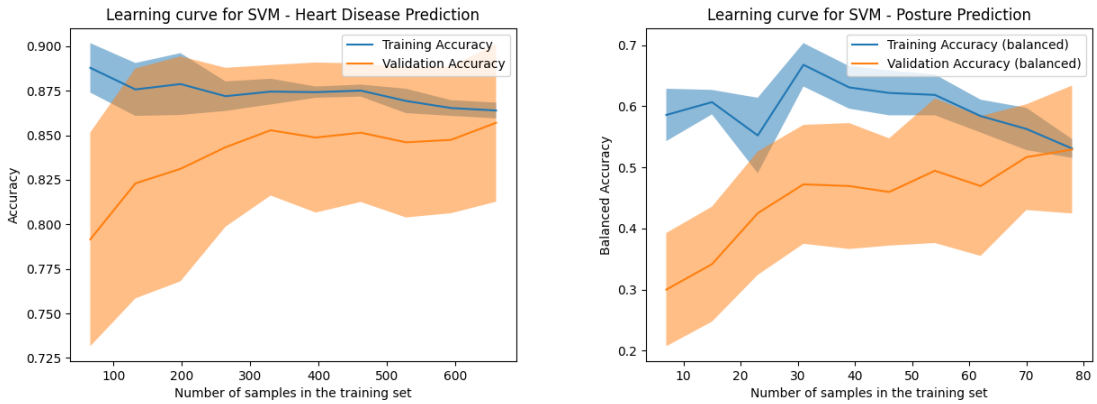


Figure 10—Learning curves of Support Vector Machine

Compared to all the other models, the SVM is remarkably fast to train and fast to query. The model also achieved the highest test accuracy for the Heart Disease dataset so far. However, the Posture prediction dataset only tested 29% balanced accuracy with the reserved test data. This may be due to the high variance in accuracy which can be seen in the learning curve, or the low number of samples in the test set not able to give a good representation of the metric. Therefore, I experimented with training the final model with a newly

shuffled 80/20 split, and was able to achieve around 50% balanced accuracy. For this reason, the test accuracy score for the Posture dataset is marked with an asterisk(*).

Table 6—Final performance metrics - Support Vector Machine

	Training Time	Query Time	Training Accuracy	Test Accuracy
Heart Disease	0.009671s	0.004720s	0.865122	0.913043
Posture	0.001260s	0.000653s	0.486124	0.29167*

2.5 k-Nearest Neighbors

K-Nearest Neighbors (KNN) is a type of lazy, non-parametric algorithm that classifies data based on similar data points in the training set. With the training set, we chose to tune the hyperparameters `n_neighbors` and `p` (power parameter for the Minkowski metric). The parameter `p` is chosen so that we can optimize our distance function in a quantifiable way, as `p=1` is equivalent to using manhattan distance, and `p=2` is equivalent to using euclidean distance, and so on. Validation curves are shown in Figure 11.

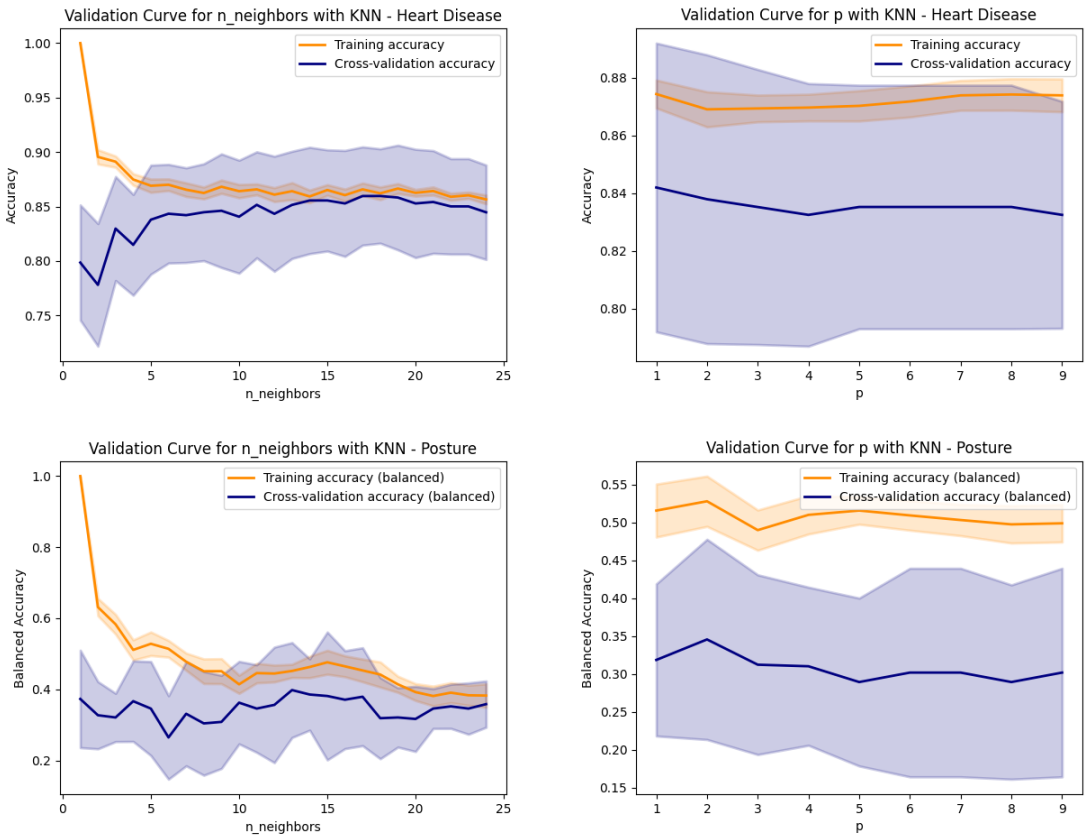


Figure 11—Validation curves of k-Nearest Neighbors

As shown in the plots, both datasets do well with a high `n_neighbors` value in the teens, since a high `n_neighbors` value places lower weight in each individual training sample and avoids overfitting. The Heart

Disease Prediction dataset does best with `p`=1 (manhattan distance), while Posture Prediction dataset does better with `p`=2 (euclidean distance).

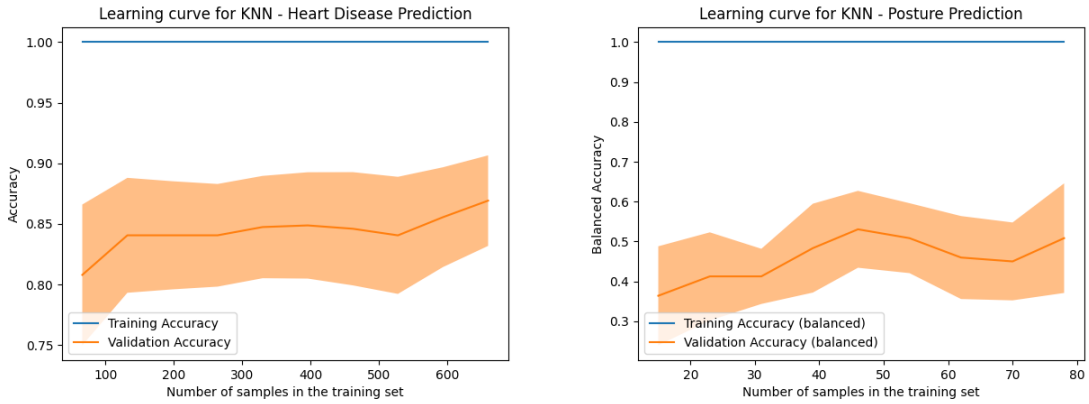


Figure 12—Learning curves of k-Nearest Neighbors

Using the optimized hyperparameter, we plot the learning curves shown in Figure 12. Interestingly, the training accuracy for both datasets resulted as 1 for any number of training samples, even though the optimal `n_neighbors` determined by `GridSearchCV` are both in the teens. This remained true even after experimenting with freshly shuffled and split datasets with different split ratios. This indicates the KNN is able to find a pattern and precisely classify data with very little training samples. This makes cross validation especially valuable during tuning since we'll need another metric to measure performance after all training samples have been correctly classified.

After training and recording the final performance of the optimized models in Table 7, we can see that the KNN has the lowest training time out of all the classifiers by far. This is expected as the KNN is a lazy algorithm. However, the KNNs trained are also fairly quick to query. This is due to the size of both of our datasets being fairly small. The KNNs also achieved fairly decent accuracy for both datasets, making it a good choice if we're looking for a simple, fast-to-train model with a small dataset.

Table 7—Final performance metrics - k-Nearest Neighbors

	Training Time	Query Time	Training Accuracy	Test Accuracy
Heart Disease	0.000843s	0.006245s	1.0	0.891304
Posture	0.000395s	0.003156s	1.0	0.354166

3 SUMMARY

After training an optimized model for each algorithm, we put their final metrics side by side against one another for each dataset, shown in Table 8 and Table 9.

For the Heart Disease Prediction dataset, all 5 algorithms performed decently, achieving accuracy in the high 80% range. In this scenario, the best choice would be the model with the lowest wall clock time, the SVM in our case. The SVM we trained has the second fastest training time, only losing to the KNN which defers all the

computation to the query stage, and second fastest query time, losing to the ANN which is very slow to train. Furthermore, since the KNN stores all the training data, it would not scale too well if we obtain more data in the future, while the SVM only saves the decision boundary, the query time will remain low as we add more data.

Table 8—Final performance metrics Heart Disease Dataset

	Training Time	Query Time	Training Accuracy	Test Accuracy
Decision Tree	0.002571s	0.001808s	0.862398	0.875000
Neural Network (ANN)	0.372967s	0.000705s	0.867847	0.896068
Boosting (AdaBoost)	0.667345s	0.030412s	1.0	0.880434
Support Vector Machine (SVM)	0.009671s	0.004720s	0.865122	0.913043
k-Nearest Neighbor (KNN)	0.000843s	0.006245s	1.0	0.891304

In the case of the Posture Prediction dataset, all 5 algorithms produced weak learners as it was a much more challenging classification problem. Both the ANN and AdaBoost achieved accuracy twice as high as random chance, however they are also the slowest out of the 5 models to train. Between ANN and Adaboost, ANN would be a better choice for this problem because it's very fast to query, and therefore much more usable once it's trained.

Furthermore, the small training sample size was a big challenge for all 5 algorithms. The SVM has achieved good cross validation results but low final test accuracy due to high variance. For this reason, the SVM would be a good candidate to revisit if we ever obtain more data.

Table 9—Final performance metrics Posture Prediction Dataset

	Training Time	Query Time	Training Accuracy	Test Accuracy
Decision Tree	0.002107s	0.001718s	0.851634	0.312500
Neural Network (ANN)	0.374703s	0.000655s	1.0	0.541666
Boosting (AdaBoost)	0.650539s	0.078547s	0.791507	0.5
Support Vector Machine (SVM)	0.001260s	0.000653s	0.486124	0.29167[3]
k-Nearest Neighbor (KNN)	0.000395s	0.003156s	1.0	0.354166

[3] Low value due to high variance, higher accuracy have been achieved with different shuffles. See Section 2.4 for details.