A1. Supervised Learning

Jihoon 'Jay' Song
jsong350@gatech.edu

INTRODUCTION

In this section, I introduce the two datasets that I will be referencing throughout the paper. I briefly describe the datasets and discuss what makes them both interesting from a machine learning perspective. Lastly, I state which problems I attempted to solve using the various machine learning algorithms, and how I plan to evaluate their performance.

Dataset #1 – For my first dataset, I picked the "Abalone Data Set", which I sourced from the UCI: Machine Learning Repository. This dataset contained 4176 instances and 8 classes including 'sex', 'length', 'diameter', 'height', 'whole weight', 'shucked weight', 'viscera weight', 'shell weight' and 'rings'. The data type of every class except 'sex' (nominal) and 'rings' (integer) were continuous. The distribution of the dataset in respect to the target feature 'sex' was relatively balanced. From here on, I will refer to this dataset as "dataset #1".

Dataset #2 – For my second dataset, I picked the "Wine Quality Data Set", which I also sourced from the UCI: Machine Learning Repository. This dataset contained 4898 instances and 12 classes including 'fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol', and 'recommend'. The data type of every attribute was continuous except 'recommend' (Boolean). The distribution of the dataset in respect to the target feature, 'recommend' was significantly imbalanced. From here on, I will refer to this dataset as "dataset #2".

Problem Statements – Dataset #1 was used to solve a multi-class classification problem with 3 labels: To predict abalones' 'sex' (Male, Female, Infant) using the 8 remaining attributes. Dataset #2 was used to solve a binary classification problem: To predict whether the wine would be recommended (1 for True, 0 for False) using the 11 remaining attributes.

Justification – I chose the two datasets because both met many of the requirements which make them interesting from a machine learning perspective. During my search, I specifically looked for datasets containing between 2000 and 8000 rows: Both datasets met this requirement. In addition, both sets contained high quality data in that neither of them contained missing nor erroneous values. Additionally, they both contained minimal number of outliers. All 8 attributes for dataset #1 and 12 for dataset #2 seemed to be pertinent and useful for the purposes of solving their respective classification problem.

Lastly, I sought for datasets from two different subject matters, which were also dissimilar from a statistical perspective. This meant looking for datasets with dissimilar data distributions which behave differently to the various algorithms in this assignment. Such datasets would force me to think critically to find the right combination of hyper-parameter values to build a high performing model. Therefore, it was important that the model trained well on the datasets using default parameters but still had room for improvement.

Performance Metric – For both datasets, I selected accuracy as the primary metric to evaluate the performance of my models. For dataset #1, due to the relatively balanced data distribution of the target class 'sex', and given it is a multi-class classification problem, this metric was appropriate. For dataset #2: Although the data distribution was imbalanced with respect to the target class, the data for the majority class made up roughly 66% of the total. My rule of thumb was that if the majority class made up less than 80% of the total, accuracy can be used [1]. In addition, selecting accuracy as the measurement gave me the opportunity to be able to compare the results of both datasets apple-to-apple.

DATA PREPARATION

In this section, I describe the pre-processing steps that were taken to prepare the datasets for training.

Data Split – Both datasets were split into 80% training and 20% testing. During the hyperparameter (HP) tuning phase, training data was further split into two, one for training the model and another for validating the model's performance. The testing data was set aside for the final evaluation of the optimized model's performance.

Standardization – Upon reviewing the statistical summaries of both datasets, I observed that both datasets contained many classes with significant differences in their range of values. In addition, both datasets contained columns which use different units of measurements. Since classifiers such as SVN, KNN and Neural Networks can become adversely affected by the lack of standardization, I standardized both datasets [2]. Standardization was carried out separately on the training and test sets after the split to prevent data leakage.

TRAINING AND TUNING

For training and tuning, I repeated the following process for each supervised learning methods on both datasets. Analysis of each learning algorithm and the effect/result of each HPs are included in this section.

Default Parameters – To establish a baseline, I trained an initial model for each algorithm using only their default hyperparameters (HP). For each model, I plotted learning curves which provided me information about the models' biases, variances, and other tendencies [3].

Hyper Parameter (HP) Tuning – Using intuition, logic and knowledge gained from this course, I hypothesized which HP would improve the models' performance and noted my reasoning. Next, I tested the hypothesis by plotting and analyzing the resulting model's validation curves. I went through multiple iterations of the HP tuning cycle until the models could no longer be improved.

For each algorithm, I used the GridSearchCV optimizer to discover the ideal combination and values of HP to produce the highest performing model. I compared the results of the optimizer against the result of my manual tuning and selected the better of the two taking different factors into account such as training time, testing time, the model's over/underfitting tendencies, among others. Finally, I evaluated the model's ability to generalize by cross validating the model against the test set that was put aside during the initial data split.

1.1 Decision Tree – Dataset #1

The initial model trained using the DecisionTreeClassifier showed that the model showed high variance and extremely low bias – in other words, it was overfitting and struggled to generalize [3]. This was to be expected since the default value for max_depth=None meant all nodes were being expanded until all leaves were pure [4]. Without a max_depth value provided; the tree expanded until there was a leaf per every data point.

max_depth – My goal was to test how the model would perform if I pruned the tree by limiting the maximum depth. Logically, this should reduce the model's overfitting, which I hoped would improve the model's ability to generalize. To test this hypothesis, I plotted a validation curve using max_depth as shown in *Figure 1*.

As expected, the validation curve showed that limiting max_depth prevented the model from overfitting the training data. The model performance was highest at max_depth=5 and afterwards, performance decreased.

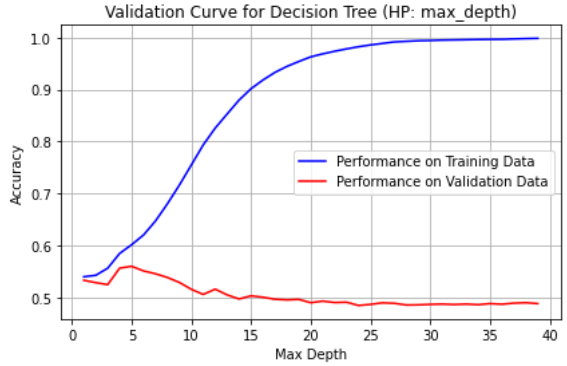

FIGURE 1. *PERFORMANCE INCREASES UNTIL MAX_DEPTH=5*

min_samples_split – Since this HP also prunes the tree, my hypothesis was that like max_depth, it would prevent the model from overfitting and increase the model's performance. Given both are pruning techniques, I tested the model without designating a max_depth to test the effect of min_samples_split on the model independently.

As expected, performance increased slightly, maxing out at min_samples_split=0.1 before decreasing (*Figure 2*). This test indicated that pruning the tree prevents overfitting and improves the model's ability to generalize.

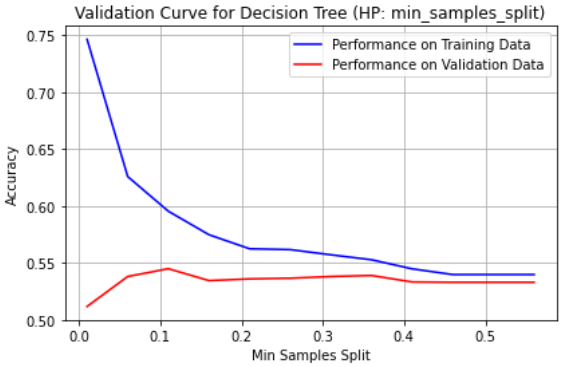

FIGURE 2. *PERFORMANCE INCREASES UNTIL MIN_SAMPLES_SPLIT=0.1*

max_depth and min_samples_split – Next, I chose to tune both max_depth and min_samples_split simultaneously. Although they both affect the decision tree by way of pruning, since the method in which they decide to prune is very different, my hope was that a combination of the two would allow the model to fine tune the model to get slightly more utility out of both HPs [4].

As I hoped, when I kept max_depth=5, performance increased slightly until min_samples_split=0.05, which equates to 167 samples. There was must have been a node that would have split, but due to having less than 167 samples, it was prevented from doing so, increasing the model's ability to generalize.

GridSearchCV - Before finalizing the model, I used a HP optimizer tool called GridSearchCV to confirm the result of my manual tuning efforts. GridSearch agreed with my assessment that max_depth=5 and min_samples_split=0.05 was optimal.

1.2 Decision Tree – Dataset #2

Interestingly, despite the very high accuracy on training data, the initial model did not seem to be overfitting as accuracy on validation data and test was also high and continued to increase with training size. This indicated to me that the content and distribution of the training data closely reflect those of validation and test data, and presumably the real world very well.

max_depth – I plotted a validation curve using max_depth as shown in *Figure 3*. As expected, it showed that limiting max_depth prevented the model from overfitting. Given the model achieved about the same performance whether the value was 7 or 15, I opted for the former which was less overfitting. I believed this would allow the model to better generalize.

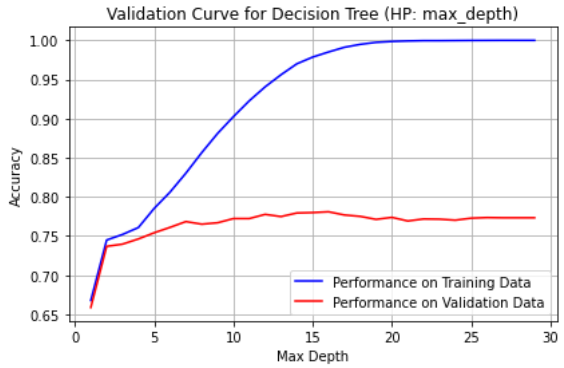

FIGURE 3. PERFORMANCE INCREASES UNTIL MAX_DEPTH=15 BUT RISKS OVERFITTING. MAX_DEPTH=7 IS MOST OPTIMAL

min_samples_split –I tuned min_samples_split while keeping max_depth constant and vice versa. Just as I was able to do with dataset #1, I wanted to use the combination of the two pruning methods to see if I could "fine-tune" my model.

The results were exactly as expected. Through this experiment, I was able to identify a combination of the two HPs which resulted in higher performance overall. As shown in *Figure 4*, this happy medium was achieved using max_depth=7 and min_samples_split=0.02.

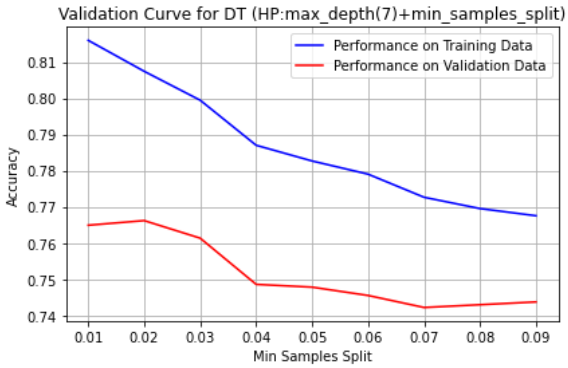

FIGURE 4. PERFORMANCE WAS HIGHEST WHEN MAX_DEPTH=7 AND MIN_SAMPLES_SPLIT=0.02

GridSearchCV – GridSearch results confirmed my assessment. It showed that max_depth=7 and min_samples_split=0.02 is optimal. Just as was the case with dataset #1, a combination of max_depth and min_samples_split had the most positive effect on model performance.

2.1. Neural Network – Dataset #1

The initial model trained using MLPClassifier was underfitting. After 1500 samples, as training size increased, both training and validation performance decreased. To increase its performance, I needed to tune the model to learn from the training data more effectively.

max_iter – Assuming a convergence point exists, I hypothesized that increasing the max_iter would allow the classifier to find the convergence point, which would improve the performance of the resultant model. In other words, by increasing the number of forward and backward passes of each training sample, I believed the learner would be able to find the target function [5].

Interestingly, as shown in *Figure 5*, after max_iter=200, although the model continued to fit better on training data, increasing max iterations did not improve the model's performance on validation data. Therefore, my hypothesis that increasing the max iterations would help the learner better fit the training data was correct. However, it resulted in worse performance on validation data. In other words, given the incremental learning nature of neural networks, after max_iter of 200, the function began to overfit the training data and was unable to generalize.

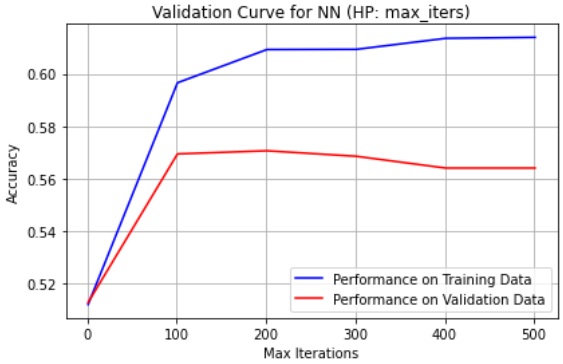

FIGURE 5. *PERFORMANCE WAS AT MAX WHEN MAX_ITER=100*

hidden_layer_sizes (neurons) –When the model was trained with a default value of a single hidden layer containing 100 neurons, it was underfitting. The model had difficulty finding a convergence point, meaning it was unable to arrive at an effective function to classify the abalone's sex. I hypothesized that increasing the number of neurons would help the learner perform better on both training and validation data up to a certain point. However, as was the case with max_iter, after that point, the model would overfit the training data and perform worse on validation data.

Indeed, as shown in *Figure 6*, my hypothesis was correct. The model's performance improved and was at its maximum at hidden_layer_size=(150). Additional neurons led to overfitting. I added up to 800 neurons to confirm and found that the pattern held true throughout.

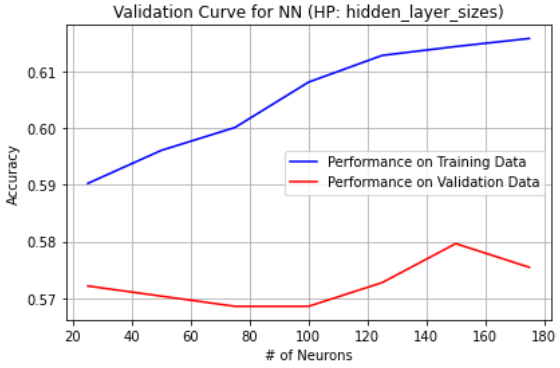

FIGURE 6. *PERFORMANCE WAS HIGHEST AT HIDDEN_LAYER_SIZES=(150)*

hidden_layer_sizes (layers) – Intuitively, it seemed that a single hidden layer would be most optimal to solve this multi-class classification problem. The problem wasn't so simple that every instance was linearly separable. However, it

wasn't so complex that it would necessitate more than a single layer. For example, it was likely that females were larger than males, which in turn were larger than infants. Given most classification problems can be solved with 1 hidden layer or fewer, I hypothesized that dataset #1 was no exception [6].

As expected, the model trained using a single hidden layer showed the highest performance. As shown in *Table 1*, increasing the number of layers, regardless of whether the number of neurons per layer was constant, increasing or decreasing, resulted in decreased performance. Additional experiments showed me that any more than one-layer resulted in the model overfitting the training data which affected its ability to generalize.

hidden_layer_sizes	Accuracy
(150)	54.3%
(150, 150)	52.8%
(150, 150, 150)	50.6%
(75, 150)	52.9%
(150, 75)	53.9%

TABLE 1. *ADDITIONAL HIDDEN LAYERS REDUCED PERFORMANCE*

GridSearchCV – The optimizer produced the following results: alpha=0.00001, hidden_layer_sizes=(400), max_iter=300. By decreasing the alpha from the default value of 0.0001 to 0.00001, the L2 regularization term had a smaller effect [5]. It reduced the amount of "apportioned error that the weights of the model are updated with each time they are updated" resulting in a "finer-tuned" model [7]. This allowed the learner to make smaller, more incremental changes which resulted in its ability to learn a more optimal combination of weights.

2.2. Neural Network – Dataset #2

max_iter – As expected, accuracy improved until max_iter=300. Further increase helped with the model's performance on training data, but not on validation data. As was the case with dataset #1, the model began overfitting. (*Figure 7).*

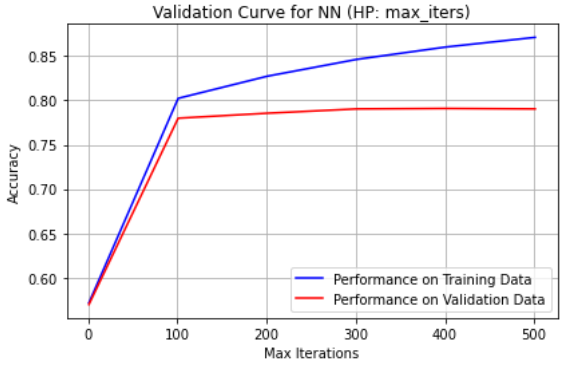

FIGURE 7. *PERFORMANCE HIGHEST WHEN MAX_ITERS=300*

hidden_layer_sizes (neurons) – Given the vast differences in the number, type, frequency, and many other characteristics between the two datasets, I was curious to see how the number of neurons would affect model performance in dataset #2. Due to the subjective nature of this classification problem, my hypothesis was that it would take a significantly higher number of neurons for the model to identify a pattern.

As expected, as shown in *Figure 8*, results showed that 700 neurons had the highest performance. Interestingly, the pattern was consistent in that any additional neurons helped the model fit better to the training data, but at the expense of its ability to generalize.

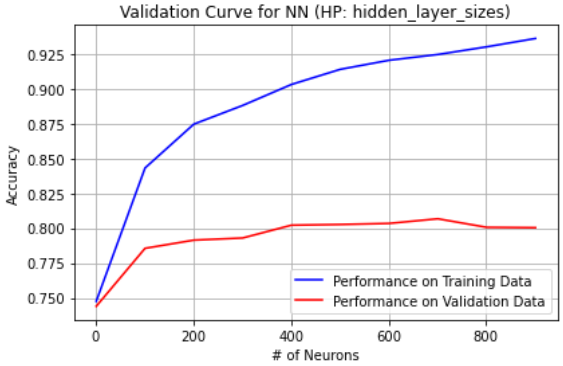

FIGURE 8. *PERFORMANCE HIGHEST AT 700 NEURONS*

hidden_layer_sizes (layers) – Due to the complexity of this problem, I believed multiple layers could improve the model's performance. As predicted, as shown in *Table 2*, the model performed best when there were 3 hidden layers, each containing 700 neurons.

hidden_layer_sizes	Accuracy
(700)	0.807
(700, 700)	0.808
(700, 700, 700)	0.823
(350, 700)	0.815
(700, 350)	0.823
(500)	0.813
(500, 500)	0.821
(500, 500, 500)	0.820
(250, 500)	0.819
(500, 250)	0.819

TABLE 2. *HIDDEN_LAYER_SIZES=(700,700,700) PERFORMED THE BEST*

GridSearchCV –The optimizer produced the following results: alpha=0.0001, hidden_layer_sizes=(500,500), max_iter=200. By utilizing alpha and thereby reducing the learning rate, GridSearch was able to identify a more efficient and higher performing HP combination. The solution required 1 less hidden layer and was able to reduce the number of neurons per layer by 200.

3.1 Boosting – Dataset #1

The initial model was trained using the default parameters of the AdaBoostClassifier on the optimized decision tree model from *Section 1.1*. Unfortunately, this resulted in a poor performing, underfitting model.

n_estimators – Additional weak learners helped the model fit the training data better, but the resulting model could not generalize. My intuition told me I needed to prune the base model more aggressively to gain any benefit from boosting.

My intuition was correct. When the base model was pruned more aggressively (max_depth=3), the resulting ensemble benefited from boosting. As shown in *Figure 9*, when the base model was max_depth=3 and min_samples_split=0.05, it had the highest performance when n_estimators=11.

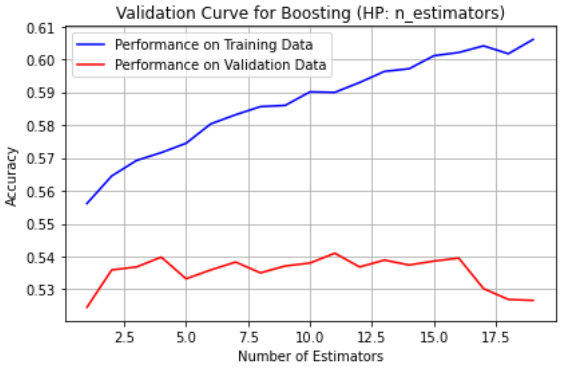

FIGURE 9. *PERFORMANCE WAS HIGHEST AT N_ESTIMATORS=11*

learning_rate– This HP speeds up or slows down the convergence rate of each weak learner [8]. Given the similarities between MLPClassfier's 'alpha' and AdaBoostClassifier's 'learning rate', I hypothesized that the model's accuracy would improve if learning_rate was reduced. To test this, I kept all other HPs constant and plotted the effect of learning rate on the model's performance.

As expected, reducing the learning_rate led to an increase in model's performance, hitting maximum performance at learning_rate=0.30 (*Figure 10*). Interestingly, just as was the case with the 'alpha' value in neural networks, increasing it past this point resulted in the model overfitting the training data, leaving it unable to generalize. Reducing the learning rate prevented the model from overfitting.

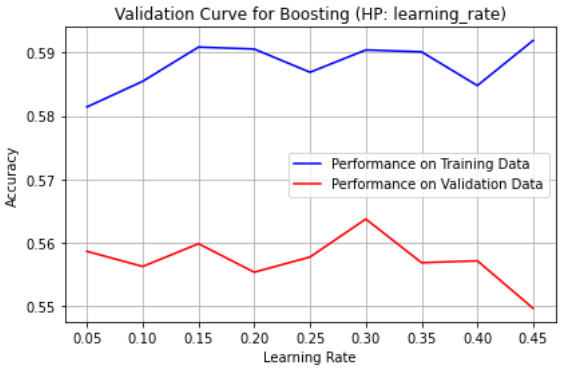

FIGURE 10. *PERFORMANCE WHEN LEARNING_RATE=0.30*

GridSearchCV – Results indicated that the most optimal HP combination was max_depth=3(base), min_samples_split=0.05(base), n_estimators=100, learning_rate=0.10. Although the exact values from my experiments were incorrect, my theory that a combination of learning_rate and n_estimators results in the highest performing model were correct. Given their inverse relationship, by reducing the number of weak learners from 11 to 100 and reducing the learning rate slightly from 0.30 to 0.10, GridSearch was able to find a balance which resulted in the highest performing model.

3.2 Boosting – Dataset #2

The initial model was trained using the default parameters of the AdaBoostClassifier on the optimized decision tree model from *Section 1.2*. Interestingly, the boosted model fit the training data completely and outperformed the base model on validation data without any HP tuning.

n_estimators – Interestingly, in dataset #2, even without more aggressive pruning, the optimized decision tree benefited from boosting. As shown in *Figure 11*, the model showed highest performance when n_estimators=300. As was the case when I experimented with the base decision tree and neural network classifier, due to the unique characteristic of dataset #2, completely fitting the training data helpe`d improve the model's ability to generalize. Therefore, even without more aggressive pruning, the model benefited from the use of additional weak learners.

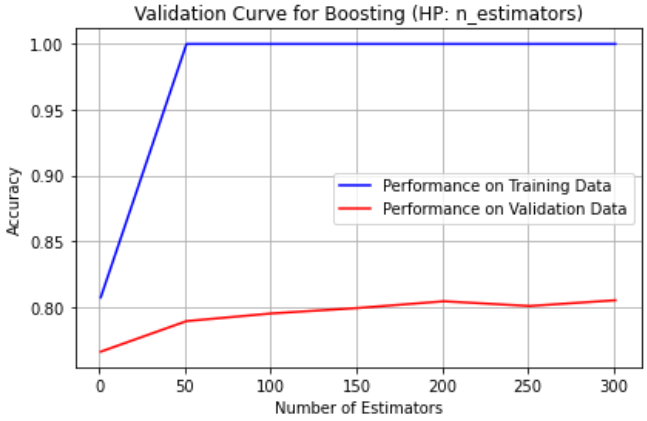

FIGURE 11. *PERFORMANCE WAS HIGHEST AT N_ESTIMATORS=300*

learning_rate – Previous experiments have shown that reducing the learning rate can often improve model performance. It seemed to work by allowing the model the option to tune other HPs to levels which results in an overfitting model otherwise. My hypothesis was that reducing the learning_rate would allow me to increase the number of weak learners which would improve the model performance.

Unfortunately, the model trained using the default learning rate of 1 was already overfitting the training model. Therefore, as shown in *Figure 12*, reducing the learning rate had no effect on performance.

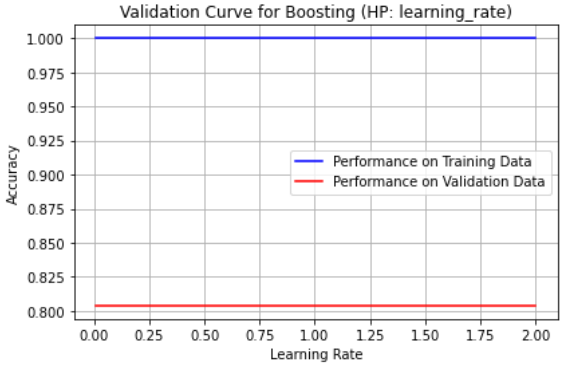

FIGURE 12. *TUNING LEARNING_RATE HAD NO EFFECT ON PERFORMANCE*

GridSearchCV – Results showed that the most optimal HP combination was max_depth=8(base), min_samples_split=0.05(base), n_estimators=700, and learning_rate=0.10. As expected, the model's performance was able to be increased by reducing the learning_rate which allowed the number of weak learners to be increased. Given the increased performance, I trained the final model using these HP values.

4.1. Support Vector Machines – Dataset #1

The initial model trained using the default HP values of SVC was balanced in that it was neither underfitting nor overfitting. The model's performance on validation data continued to increase with training size.

kernel – Per its definition, linear SVMs was meant to be used to solve linear problems, and Radial Basis Function (rbf) to solve non-linear problems [9]. I hypothesized that rbf SVMs would result in the highest performing model since I believed the samples could not be separated linearly on all features. To test this, I plotted a validation curve using 4 types of kernels: linear, poly, rbf and sigmoid. Although most sources recommended that only linear and rbf be considered, for experimental purposes, I included both poly and sigmoid [10].

Interestingly, as shown in *Figure 13*, linear kernel-based model performed slightly better than others, followed closely by the rbf kernel-based model.

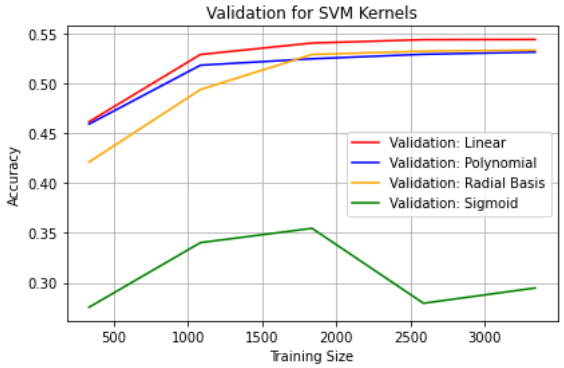

FIGURE 13. LINEAR KERNEL BASED MODEL SHOWED THE HIGHEST PERFORMANCE

C – In dataset #1 numerous features contained overlaps between all 3 target class values making it difficult for every sample to be linearly separated. For example, although on average, female abalones are larger than their male counterparts, there exists a range which contain both female and male abalones. I wanted to see how C would affect the performance of both linear and rbf kernel-based models.

My hypothesis was that linear kernel-based model would benefit from a lower value of C which allows the model to be more flexible [11]. In addition, I predicted that rbf-based model would benefit from a higher value of C, as it would force the learner to look for a more complex function by taking advantage of rbf kernel-based model's ability to create functions spanning multi-dimensions.

Interestingly, as I hypothesized, rbf kernel-based model performed better than the linear kernel-based models after C was tuned. As shown in *Figure 14*, at C=10, the rbf kernel-based model was the highest performing. The learner was able to identify a complex, multi-dimensional function which better suited the problem. Increasing the C value over 10 resulted in the model overfitting.

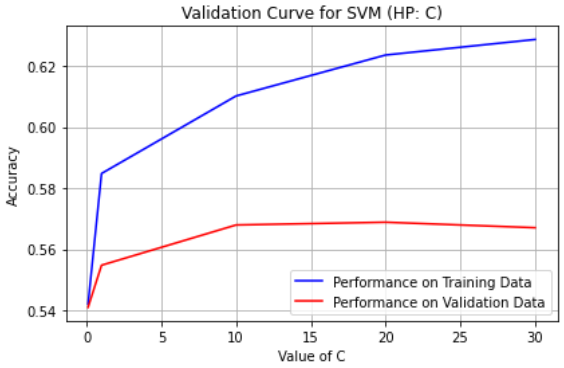

FIGURE 14. RBF KERNEL BASED MODEL SHOWED THE HIGHEST PERFORMANCE AT C=10

gamma – I believed I could increase the rbf model's performance by adjusting the weight of samples which weren't as close to the hyperplane. Given the complexity of the function on the hypersphere, I hypothesized that a lower value of gamma would increase the model's performance.

As shown in *Figure 15*, the model had the highest performance on validation data at gamma=0.1. This meant that the model's complex function benefited from giving more weight to the sample points which were further away from the hyperplane.

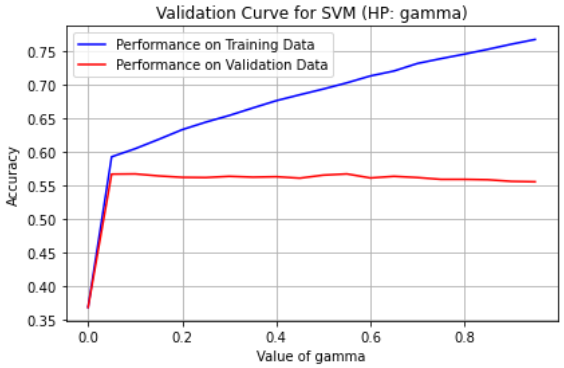

FIGURE 15. RBF KERNEL BASED MODEL SHOWED THE HIGHEST PERFORMANCE AT GAMMA=0.1

GridSearchCV – The optimizer output the following: kernel='rbf', C=10, gamma=0.10. Therefore, I was able to confirm that the results of my manual tuning of the 3 HPs were indeed optimal.

4.2. Support Vector Machines – Dataset #2

The initial model using the default parameters showed relatively high performance on both training and validation data. In addition, the model's performance on validation data continued to increase with training size.

kernel – I hypothesized that rbf-kernel based model would be the most appropriate for dataset #2. Compared to dataset #1, the ranges and units between the feature columns

of dataset #2 varied much more widely. Therefore, linear separation would be very difficult. Furthermore, given that the target class "recommend" is subjective in nature, I believed the target function would likely span multiple dimensions.

Unsurprisingly, like dataset #1, when the value of C was 1, linear kernel outperformed that of other kernels (*Figure 16*). Given my experience with dataset #1, I planned to run both linear and rbf-kernel models through my next experiment with the HP: C.

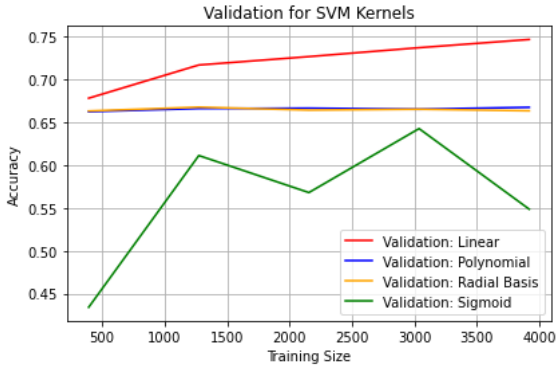

FIGURE 16. LINEAR KERNEL BASED MODEL SHOWED THE HIGHEST PERFORMANCE WHEN C=1

C – Due to the subjective nature of the target class, the problem statement associated with dataset #2 was much more complex compared to that of dataset #1. For example, it was highly likely that the taster's personal preference and bias affected whether he/she recommended a wine. Therefore, my hypothesis was that increasing the value of C would result in a higher performing rbf-kernel model [9].

As expected, with the increase in the value of C, the rbf-kernel model outperformed the linear-kernel model. Its performance was highest at C=20 as shown in *Figure 17* below. As was the case in dataset #1, further increasing the value of C resulted in an overfit model.

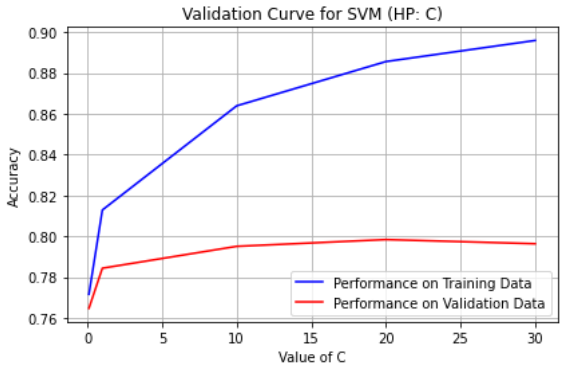

FIGURE 17. RBF KERNEL BASED MODEL SHOWED THE HIGHEST PERFORMANCE AT C=20

gamma – My hypothesis was that as was the case in dataset #1, reducing the value of gamma to increase the weight of outliers would help improve the model's performance. And as expected, the model showed the highest performance on validation data when gamma=0.1 (*Figure 18*). Again, this indicated that taking the outliers into greater consideration helped the model better estimate the target function [9].

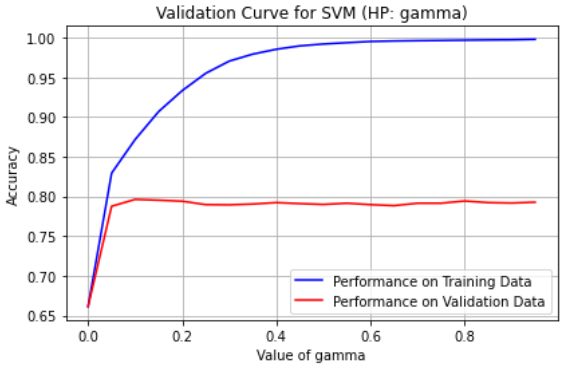

FIGURE 18. RBF KERNEL BASED MODEL SHOWED THE HIGHEST PERFORMANCE AT GAMMA=0.1

GridSearchCV – The optimizer output the following result: kernel='rbf', C=10, gamma=0.10. Therefore, I was able to confirm that the results of my manual tuning of the 3 HPs were optimal.

5.1. K-Nearest Neighbor – Dataset #1

The initial model trained using the default HP values of KNeighborsClassifier was underfitting and performance on both training and validation data continued to decrease with additional training size. It was clear that the HPs needed to be tuned to help the model better learn the training data.

n_neighbors – Performance improved significantly when this HP was increased. As shown in *Figure 19*, the model was highest performing at n_neighbors=34. Interestingly, at higher n_neighbors, the performance on both training data and validation data suffered. This made sense because intuitively, very small number of n_neighbors would lead to overfitting, and very large numbers of n_neighbors would lead to underfitting.

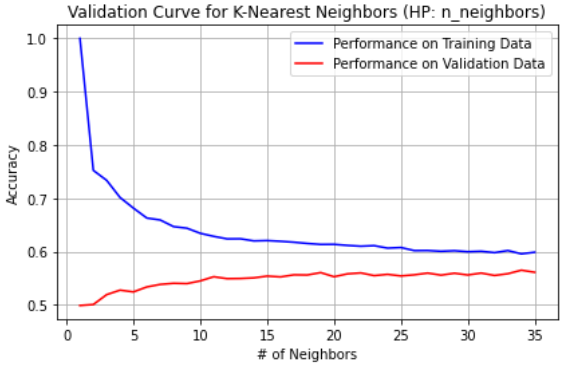

FIGURE 19. KNN MODEL SHOWED THE HIGHEST PERFORMANCE AT N_NEIGHBORS=34

weights – After seeing how significantly the performance of SVM models were affected by the value of gamma, I was confident that HP: weights would affect the performance of KNN models. A fair comparison would be that KNN's weight=distance would be equivalent to the baseline SVM models. In SVM, decreasing the gamma value to 0.1 improved performance by giving samples which are more distant from the hyperplane more weight. In KNN, this

can be done by manipulating the HP: weights [12].

Given the similarities I identified between SVM and KNN, I hypothesized that the KNN model would perform better when weights='uniform', as it more closely resembles the smaller, fractional gamma in SVM. To test this hypothesis across a spectrum of specificity, I plotted the validation curve for HP: n_neighbors just as I did in the previous experiment, but with the weights='distance'. I also noted the best accuracy over all n_neighbors values for both weights values.

As shown in *Figure 20* and *Table 3*, at n_neighbors=34, my hypothesis was correct, but at 20, it was not.

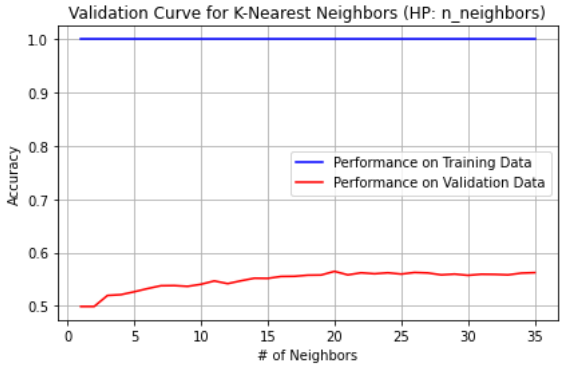

FIGURE 20. THE KNN MODEL SHOWED THE HIGHEST PERFORMANCE AT N_NEIGHBORS=34 AND WEIGHT='UNIFORM'

	(weights='uniform', n_neighbors=34)	**(weights='distance', n_neighbors=20)**
Accuracy	56.53%	56.50%

TABLE 3. PERFORMANCE METRICS FOR KNN DATASET #1

GridSearchCV – GridSearch output the following result: weights='uniform', n_neighbors=34. Again, through the GridSearch results, I was able to confirm that my HP combination was optimal.

5.2. K-Nearest Neighbor – Dataset #2

The initial model trained using default values of KNN performed much better for dataset #2 than it did for dataset #1. Performance on training data began to plateau at 2500 training samples, while performance on validation data continued to increase. This indicated the model was not overfitting and was able to generalize well.

n_neighbors – As was the case with dataset #1, I hypothesized that there existed an optimal value greater than 5 which resulted in a higher performing model.

My hypothesis was incorrect as n_neighbors=1 resulted in the highest performing model. As shown in *Figure 21*, when n_neighbors=1, the model completely fit the training data. In other words, the learner produced a function which describes every sample in the training data perfectly. Interestingly, the model still performed very well on validation data.

I remembered that this phenomenon was also seen during my decision tree experiment on dataset #2. When max_depth was set to have no limit, the DecisionTreeClassifier produced a model which fit the training data completely. Despite this, the model was still able to generalize well.

It was worth noting that in my Decision Tree experiment, when other HPs were introduced in combination with max_depth, the resulting models performed better without fitting the training data completely. Therefore, I hypothesized that despite the initial results, the most optimal KNN model would not have n_neighbors=1.

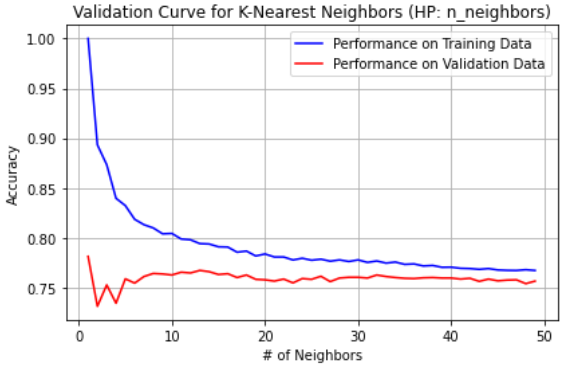

FIGURE 21. THE KNN MODEL SHOWED THE HIGHEST PERFORMANCE AT N_NEIGHBORS=1

weights – For most datasets, a model which is fit completely on training data would result in poor ability to generalize, indicated by its poor performance on out-of-sample data. However, due to the unique makeup of dataset #2, there seemed to be no penalty. Not only that, as shown in *Figure 21*, it seems to give models an edge.

Intuitively, I believed models trained using the distance weight function would serve a similar purpose. Since each sample's unique distance from the query point would be taken into consideration by the learner, it would help the model fit the training data completely [12]. Therefore, my hypothesis was that for dataset #2, weights='distance' would result in a higher performing model compared to weights='uniform'. To test this theory, just as I did for dataset #2, I plotted another validation curve for HP: n_neighbors using weights='distance'.

As shown in *Table 4*, I compared the performance of the two models: One trained using the optimal n_neighbors with weights='uniform' and another using weights='distance'. As shown in *Figure 22*, I was able to confirm that by changing the value of HP: weights to 'distance', the model fit the training data completely, regardless of n_neighbors. This "advantage" in combination with n_neighbors=34 resulted in the highest performing KNN model up to this point.

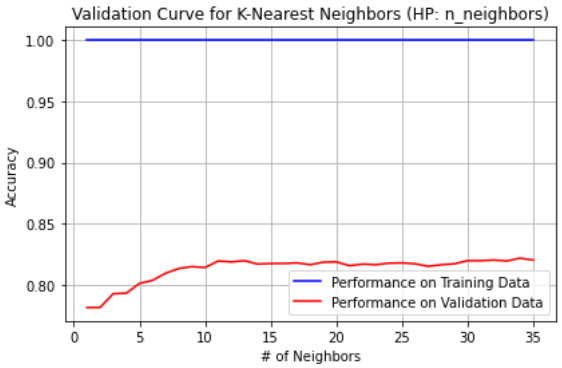

FIGURE 22. PERFORMANCE OF THE KNN MODEL USING WEIGHT='DISTANCE' SHOWED VERY HIGH PERFORMANCE

	(weights='uniform', n_neighbors=1)	**(weights='distance', n_neighbors=34)**
Accuracy	78.15%	82.18%

TABLE 4. PERFORMANCE METRICS FOR KNN DATASET #2

GridSearchCV – GridSearch output the following result: weights='distance', n_neighbors=32 and p=1. Therefore, through the optimizer, in addition to confirming the results of my manual tuning effort, I was able to determine that the model benefits from Manhattan distance opposed to the default Euclidian to calculate distance.

RESULTS

This section contains the final learning curves and performance metrics for each algorithm and dataset. Analysis of the results are provided in the upcoming **Discussion** section.

1.1 Decision Tree – Dataset #1: Learning Curve

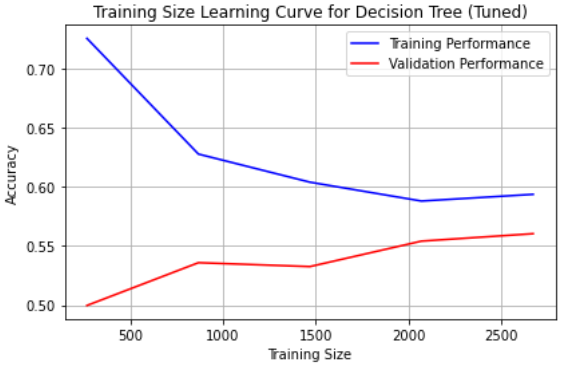

FIGURE 23. FINAL LEARNING CURVE FOR DT DATASET #1

1.2 Decision Tree – Dataset #2: Learning Curve

FIGURE 24. FINAL LEARNING CURVE FOR DT DATASET #2

2.1 Neural Network – Dataset #1: Learning Curve

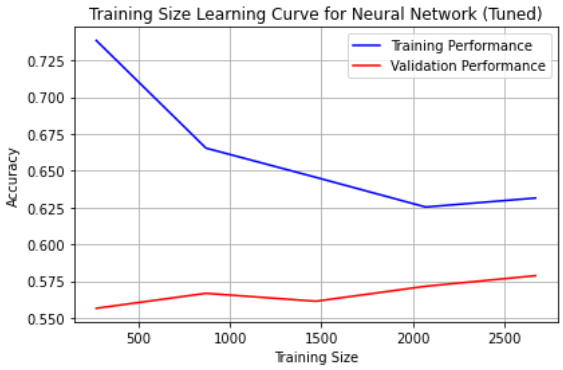

FIGURE 25. FINAL LEARNING CURVE FOR NN DATASET #1

2.2 Neural Network – Dataset #1: Loss Curve

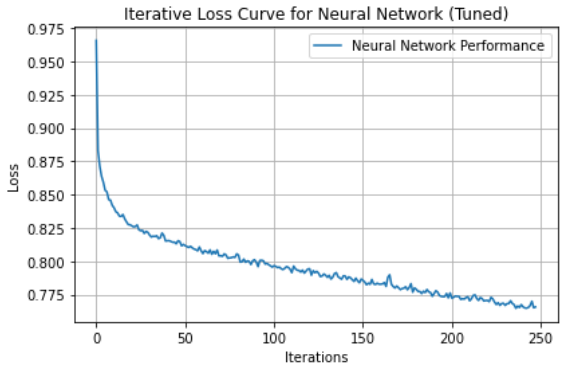

FIGURE 26. FINAL LOSS CURVE FOR NN DATASET #1

2.3 Neural Network – Dataset #2: Learning Curve

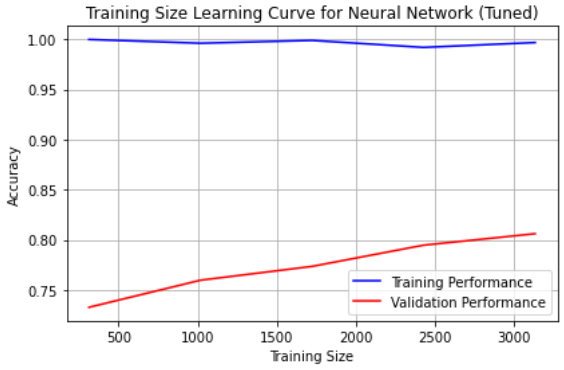

FIGURE 27. FINAL LEARNING CURVE FOR NN DATASET #2

2.4 Neural Network – Dataset #2: Loss Curve

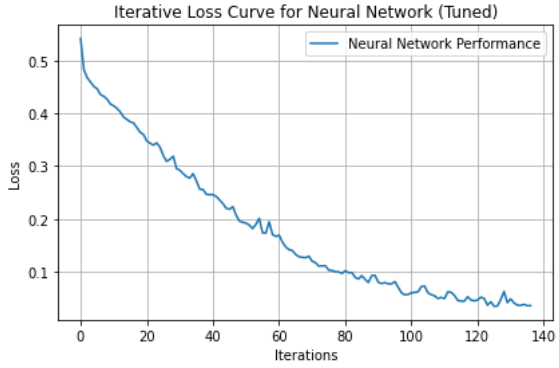

FIGURE 28. FINAL LOSS CURVE FOR NN DATASET #2

3.1 Boosting – Dataset #1: Learning Curve

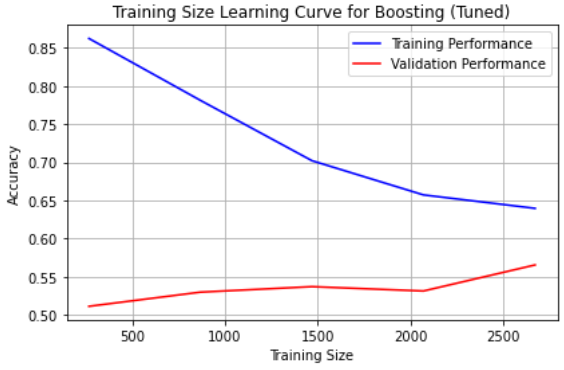

FIGURE 29. FINAL LEARNING CURVE FOR BOOSTING DATASET #1

3.2 Boosting – Dataset #2: Learning Curve

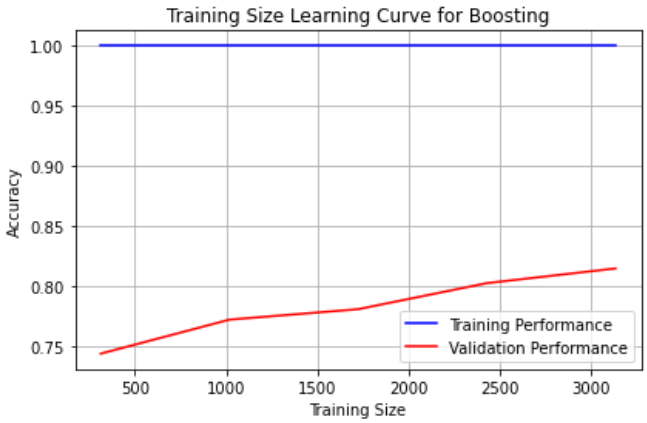

FIGURE 30. FINAL LEARNING CURVE FOR BOOSTING DATASET #2

4.1. SVM – Dataset #1: Learning Curve

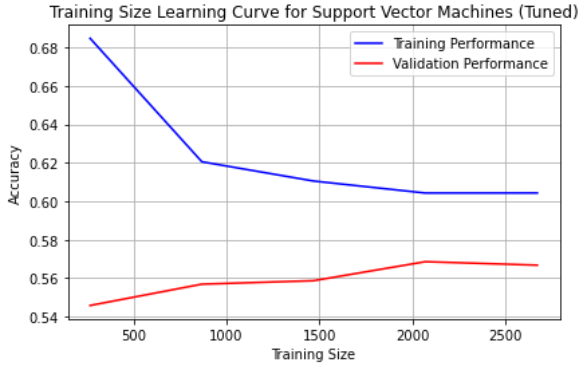

FIGURE 31. FINAL LEARNING CURVE FOR SVM DATASET #1

4.2. SVM – Dataset #2: Learning Curve

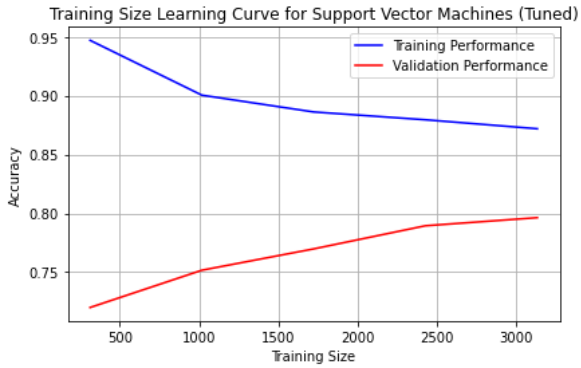

FIGURE 32. FINAL LEARNING CURVE FOR SVM DATASET #2

5.1. K-Nearest Neighbor – Dataset #1: Learning Curve

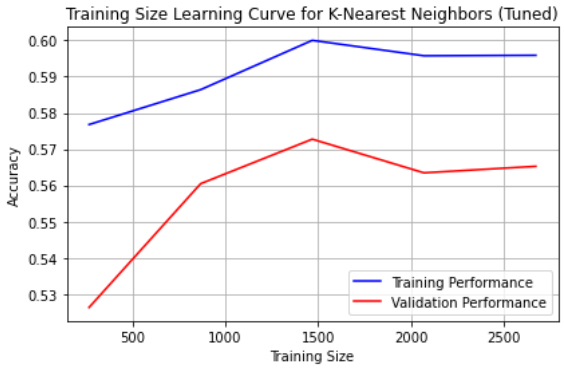

FIGURE 33. FINAL LEARNING CURVE FOR KNN DATASET #1

5.2. K-Nearest Neighbor – Dataset #2: Learning Curve

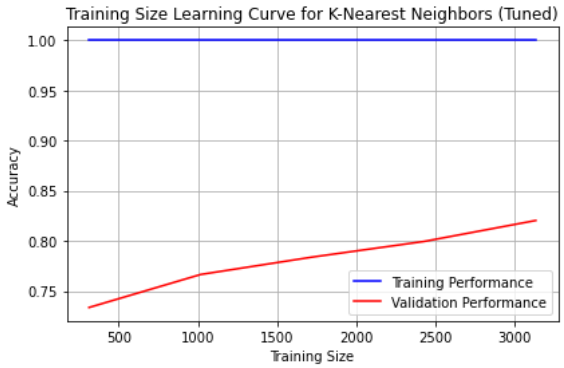

FIGURE 34. FINAL LEARNING CURVE FOR KNN DATASET #2

6.1 Dataset #1: Performance Metrics

Algorithm	Train (s)	Test (s)	Accuracy
Decision Tree	0.007	0.001	56.0%
Neural Network	5.068	0.002	57.9%
Boosting	0.431	0.010	56.6%
SVM	0.224	0.095	56.7%
KNN	0.008	0.049	56.5%

TABLE 5. FINAL PERFORMANCE METRICS FOR DATASET #1

6.2 Dataset #2: Performance Metrics

Algorithm	Train (s)	Test (s)	Accuracy
Decision Tree	0.011	0.000	76.6%
Neural Network	29.22	0.008	81.0%
Boosting	9.52	0.156	81.8%
SVM	0.278	0.086	79.6%
KNN	0.011	0.025	77.9%

TABLE 6. FINAL PERFORMANCE METRICS FOR DATASET #2

DISCUSSION

This section contains the analysis of results of the various learning algorithm on my problems. Analysis on each HP can be found in the **Training and Tuning** section.

Datasets and Problems – Results show that regardless of learning algorithm, the datasets and problems chosen most heavily influence the models' performance. As shown in the Accuracy column of both *Table 5* and *Table 6*, most algorithms performed similarly when solving the same problem using the same dataset.

For example, compared to dataset #1, all learning algorithms found dataset #2 much easier to learn. As I noted during the training and tuning section of every learning algorithm, when learning from the training data for dataset #2, the target function could fit the training data perfectly without sacrificing its ability to generalize.

There are several reasons for this behavior in dataset #2. First, there were 937 duplicate rows in the dataset. It's likely that during the collection of this data, several wine tasters tasted the same wine, and rated them similarly. In addition, since I consolidated the original column, 'quality', into a binary one, 'recommend', it created more duplicates. Secondly, as evidenced by several features which show very high correlation with the target class, several features appeared to be highly predictive. Lastly, since it was a binary classification problem, the probability of the model guessing correctly was 50%. These factors inflated the accuracy of the models' predictions related to dataset #2.

To account for the factors above in addition to both dataset's imbalanced data distribution, I also measured and evaluated the final models' F1 scores. The F1 scores were used to alert me if the accuracy scores of any models were unusable. Fortunately, the final models' F1 scores were not significantly different from their accuracy scores.

Decision Tree – As shown in *Figure 23* and 24, both learning curves show that neither final DT models are overfit. The model's accuracy on training data decreases with training size while accuracy on validation data increases, making full use of the entire training data.

I found it interesting that unlike other algorithms, DT models didn't require data to be standardized. Intuitively, this makes sense since the learner looks through each feature one at a time to determine splits.

DT models came in last place on both datasets in respect to performance. They did, however, have the fastest training/testing time (*Table 5-6*). Given DT's fast training and testing time, in addition to its ability to perform relatively well with minimal data pre-processing, it is perfect for users constrained on time.

Neural Network – As shown in *Figure 25* and *27*, both learning curves show that accuracy on validation data continues to increase with training size. Although dataset #2 NN model appears to be completely fitted to the training data, as discussed previously, this is due to the unique characteristics of the dataset. I found it interesting that a model's tendency to overfit the training data does not always mean it is unable to generalize.

As shown in *Figure 26* and *28*, both loss curves show that loss decreases over iterations. Dataset #1's loss curve shows that loss decreases exponentially at first, but then slows down, eventually settling at about 77% loss. To

account for this, in following assignments, I will explore additional HP combinations using an even lower learning rate (alpha). In contrast, while dataset #2's loss curve also shows that loss decreases exponentially at first, it continues to trend down, settling at about 5% loss. This is an ideal rate of loss over iterations.

NN models came in 1st place on dataset #1 and 2nd place on dataset #2 in respect to performance. However, they took significantly longer to train compared to other learning algorithms. But once trained, their testing time was very fast (*Table 5-6*). Given NN's significant upfront time and computing resource requirement, it is best for users willing to make the upfront investment to create a high performing model.

Boosting – As shown in *Figure 29* and *30*, both learning curves show that neither of the final boosted DT models are overfit. Like the NN and KNN model for dataset #2, the boosted model is completely fitting the training data. However, just like the other two, the boosted model has no issues generalizing. Just as the base DT models did, both boosted models' accuracies on validation data increases with data size, making full use of the entire training data.

Interestingly, the boosted DT models scored 3rd place on dataset #1 and 1st on dataset #2 in respect to performance, compared to the base models' 5th on both datasets. It took significantly longer to train, and test compared to its base model (*Table 5-6*). Despite its exceptional performance, due to the long training and test time, boosting is best suited for users who already have a tuned base model and isn't under a time and resource constraint.

SVM – As shown in *Figure 31*, dataset #1 SVM model's performance on validation data needs to be monitored as there appears to be a slightly negative slope towards the tail end of the curve. This indicates possibility of overfitting which would negatively impact model's ability to generalize. As shown in *Figure 32*, however, dataset #2 SVM model is not overfitting.

In contrast to the DT algorithm, I found that SVM was very heavily affected by data standardization. This makes sense because the rbf-kernel model utilizes the Euclidean/Manhattan distance of the samples. Therefore, especially in dataset #2, due to the different units of measurements used between features, the SVM model performed very poorly without standardization.

SVM models came in 2nd place on dataset #1 and 3rd place on dataset #2 in respect to performance. They performed relatively well on training time, scoring 3rd on both datasets, but was by far the slowest in respect to testing time (*Table 5-6*). Therefore, this algorithm is best suited for users with more time and resources.

KNN – As shown in *Figure 33 and 34*, both learning curves show that neither KNN models are overfit. In dataset #1, the model's accuracy on training data decreases with training size while accuracy on validation data increases. In dataset #2, the model is completely fitting the training data. However, like NN model for dataset #2, it is still able to generalize, as evidenced by the model's high performance on validation data.

KNN models came in 4th place on both datasets in respect to performance. Like DT models, they had one of the fastest training times. However, like SVM models, they required much more time for testing. Therefore, KNN is best suited for users whose data changes frequently since no upfront cost is required to train the model.

REFERENCES

[1] Brownlee, J. (2021, April 30). Tour of evaluation metrics for imbalanced classification. Machine Learning Mastery. Retrieved September 25, 2022, from https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification

[2] Jaadi, Z. (2019, September 4). When and why to standardize your data. Built In. Retrieved September 25, 2022, from https://builtin.com/data-science/when-and-why-standardize-your-data

[3] Brownlee, J. (2019, August 6). How to use learning curves to diagnose machine learning model performance. Machine Learning Mastery. Retrieved September 25, 2022, from https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

[4] Sklearn.tree.decisiontreeclassifier. scikit-learn. (n.d.). Retrieved September 25, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[5] Sklearn.neural_network.MLPClassifier. scikit-learn. (n.d.). Retrieved September 25, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[6] Uzair, M., & Jamil, N. (2020). Effects of hidden layers on the efficiency of Neural Networks. 2020 IEEE 23rd International Multitopic Conference (INMIC). https://doi.org/10.1109/inmic50486.2020.9318195

[7] Brownlee, J. (2019, August 6). How to configure the learning rate when training deep learning neural networks. Machine Learning Mastery. Retrieved September 25, 2022, from https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

[8] Sklearn.ensemble.adaboostclassifier. scikit-learn. (n.d.). Retrieved September 25, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[9] Sklearn.svm.SVC. scikit-learn. (n.d.). Retrieved September 25, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[10] Pupale, R. (2019, February 11). Support vector machines(svm) - an overview. Medium. Retrieved September 25, 2022, from https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989

[11] SVM hyperparameter tuning using GRIDSEARCHCV. Velocity Business Solutions Limited. (2020, February 19). Retrieved September 25, 2022, from https://www.vebuso.com/2020/03/svm-hyperparameter-tuning-using-gridsearchcv/

[12] Sklearn.neighbors.kneighborsclassifier. scikit-learn. (n.d.). Retrieved September 25, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html