Final Project Report

Introduction

The dataset I am using is the red wine quality dataset of the Portuguese "Vinho Verde" wine. The dataset consists of 1,599 data points and 12 features per data point. The features are fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphate, alcohol, and quality. The underlying supervised machine learning problem I am trying to solve is using the 11 features of each wine (fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphate, and alcohol) to predict the quality of wine using classification. All the features are floats, except the wine quality, which is an integer that ranges from 0 - 10. The higher the integer is, the better quality the wine is. This is important to me because I am curious to see what attributes of a wine make a better quality wine. The metric I will use to measure performance is accuracy. Accuracy is defined as the the number of predictions divided by the number of test instances, and it ranges from 0 to 1, where a number closer to 1 is more accurate [1].

Description of Algorithms

The algorithms I will be testing are a random forest with bagging, a support vector machine with a non-linear kernel and a multi-layer perceptron neural network. A random forest classifier with bagging is a type of ensemble learner that consists of a large amount of individual decision trees that work together as an ensemble. Each individual decision tree picks split points randomly (instead of optimally) and randomly samples the data with replacement (bagging). The hyperparameters of a random forest that I will be tuning are the number of trees in the forest and the maximum depth of each tree. As the number of trees in a forest increases, the computational cost and accuracy increase as well, but after a certain number of trees, the model will not see any improvements [2]. The maximum depth of each tree limits the depth each tree can grow, thus makes the ensemble learner converge earlier [2].

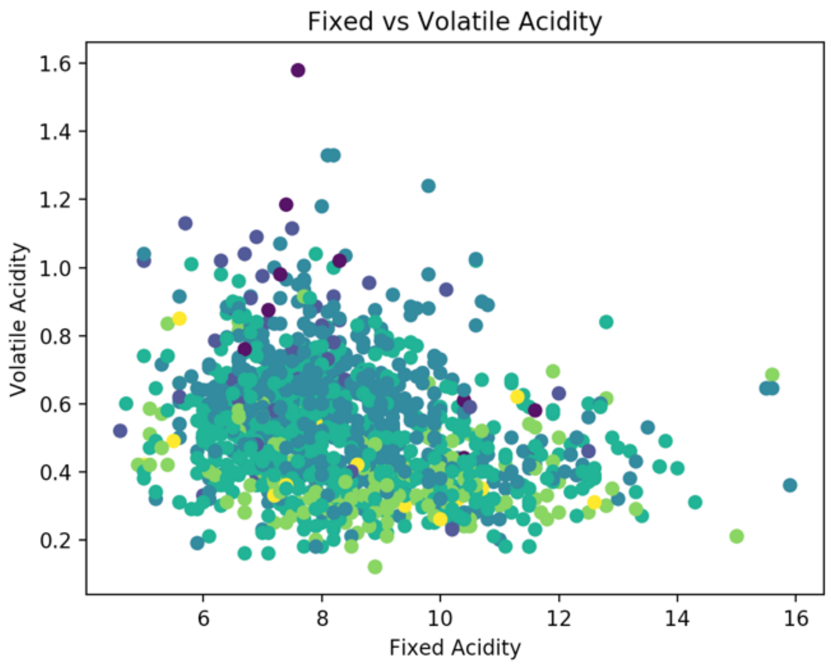
A support vector machine is based upon the maximum merging principle, meaning the best hypothesis is the one that maximizes the distance to the training data. Support vector machines use quadratic programming methods to find the parameters of the model that determine which training samples are necessary for finding support vectors. A support vector machine with a non-linear (Radial Basis Function) kernel increases the dimension of the hyperplane, then finds a new line of classification in the higher dimension. The hyperparameterers of a support vector machine are the kernel coefficient (gamma) and regularization parameter (C). Gamma is how far the influence of a single training example reaches with low and high values [3]. If gamma is too large, only the the support vector itself is in the area of influence and no value of C can prevent overfitting [3]. If gamma is too small, the model will not be able to capture the complexity of the data [3]. C is the tradeoff between "correct classification of training examples" and "maximization of the decision function's margin" [3]. A large value of C will choose a smaller margin if the hyperplane classifies all the training points correctly [3]. A smaller value of C will choose a larger margin and simpler decision function, thereby reducing the training accuracy [3].

A neural network analyzes data by having a parameter or neuron take in an input, then produce an output, and then pass along the output to the next layer of neurons [4]. The output of the previous layer is multiplied by a weight, and then passed onto the next layer of neurons as an input [4]. The final layer outputs the result of the model.The hyperparameters of a neural network are the number of hidden layers, the number of neurons in each hidden layer, the learning rate (alpha), and the number of epochs. The hidden layers are the layers between the input and output layer. More hidden layers and neurons increase the accuracy of the model and a small amount of hidden layers may cause under fitting [4]. The learning rate controls the rate at which a neural network updates its parameters [4]. If the learning rate is small, the learning process is slowed down, and the model converges smoothly [4]. If the learning rate is large, the learning process is sped up, but the model may not converge [4]. The number of epochs is the number of times all the training data is shown to the model while training. The number of epochs should be increased until the validation accuracy begins decreasing [4].

Tuning Hyperparemters

To tune the hyperparameters, I used nested cross validation and grid search on the data to search over all possible settings and pick the setting with the highest accuracy. To do this, I split the dataset into training and testing data (60/40 split). I then split the training set into k = 5 folds and ran cross validation to tune the hyperparamters. Once the optimal hyperparameters were found in the training set, I split the testing set into k = 5 folds and ran cross validation with the optimal hyper parameters. For each algorithm, I returned the averaged test score, confidence intervals, the best estimator, and best parameters. I began each experiment with a very large range for each hyperparameter, then narrowed my search space by looking at the best parameters grid search returned and using the one standard error rule. All the tables below show how I narrowed my search space for each parameter. The numbers in red color are the optimal hyperparameters for that test.

This is a non-trivial dataset and is not linearly separable, due to the 11 classifications the wines can fall into. If the dataset had a very high training error, I would know that the the dataset is trivial because there would only be a very small number of ways to classify the data. For example, if I had a dataset that predicted the quality of wine using either 1 = good quality or 0 = bad quality, the dataset would be trivial as I could only have 2 class labels, but in my case, I have 10 class labels. The plot below plots 2 features of the dataset, fixed and volatile acidity, which each color of a point representing the class label it is in.



Random Forest with Bagging

Average Test Score	**Number of Trees**	**Max Depth**	**Confidence Intervals**	**Computation Time (in seconds)**
0.658	1-101, every 9: 82	2,4,6,8,10,12,14,16,18	(0.636, 0.680)	176.61
0.662	76-90: 88	8,9,10,11,12,13,14,15,16	(0.634, 0.690)	432.24
0.659	100, 110, 120,130,140,150,160	10,11,12,13,14,15,16,17,18,19	(0.638, 0.680)	351.33
0.652	88-99: 94	11,12,13,14,15,16	(0.633, 0.673)	283.77
0.661	108-121: 108	11,12,13,14,15,16	(0.635, 0.687)	526.39
0.662	96-108: 104	11,12,13,14,15,16	(0.637, 0.687)	397.3

Optimal Hyperparameters

Number of trees: 82

Max depth: 14

*Chose because small confidence interval and less number of trees

Support Vector Machine

Average Test Score	**C Range**	**Gamma Range**	**Confidence Intervals**	**Computation Time (in seconds)**
0.586	[0.001, 0.01, 0.1, 1, 10, 100, 1000, 1050, 3000]	[0.0001, 0.001, 0.01, 0.1, 1.0, 10]	(0.555, 0.618)	53.778
0.580	[500, 700, 900, 1000, 1200, 1500]	[0.001, 0.0015, 0.01, 0.1, 1.0]	(0.564, 0.598)	49.57
0.582	900-1100, every 10: 900	[0.0001, 0.001, 0.01]	(0.548, 0.617)	124.03
0.571	890-910: 893	[0.001, 0.01]	(0.571, 0.609)	86.92
0.585	[100, 200, 300, 400]	[0.001, 0.01]	(0.561, 0.609)	9.351

Optimal Hyperparameters

C: 300

gamma: 0.001

Chosen due to smaller C and confidence interval and second highest average test score

Neural Network

Average Test Score	**Hidden Layer Sizes/Neurons per layer**	**Alpha**	**Number of Epochs**	**Confidence Intervals**	**Computation Time (in seconds)**
0.583	(15,15,15), (10,10,10)	[0.0001, 0.001, 0.01, 0.1, 0.5]	[1000, 2000 3000]	(0.555, 0.613)	1147.90
0.570	(15,15,15,15), (20,18,15,12)	[0.001, 0.01, 0.1]	[1000, 1500, 1800]	(0.547, 0.594)	804.549
0.592	(15,15,15,15), (15,15,15,15,15)	[0.001, 0.01, 0.1]	[800,900,1000, 1110]	(0.569, 0.615)	699.79

Optimal Hyper Parameters

Number of hidden layers: 4

Number of neurons per layer: 15

Alpha: 0.001

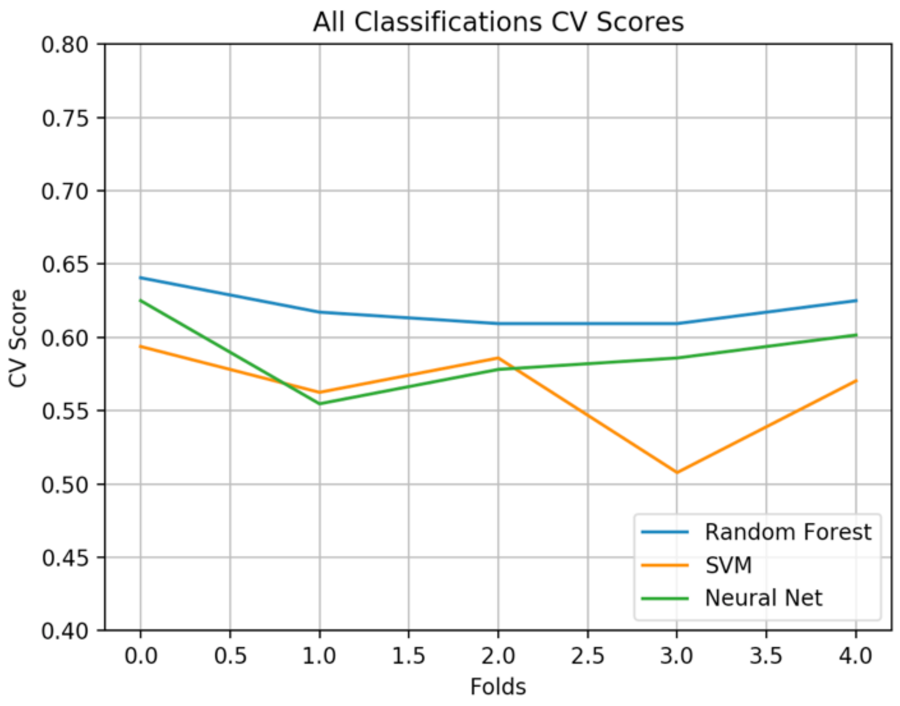
Number of Epochs: 1000

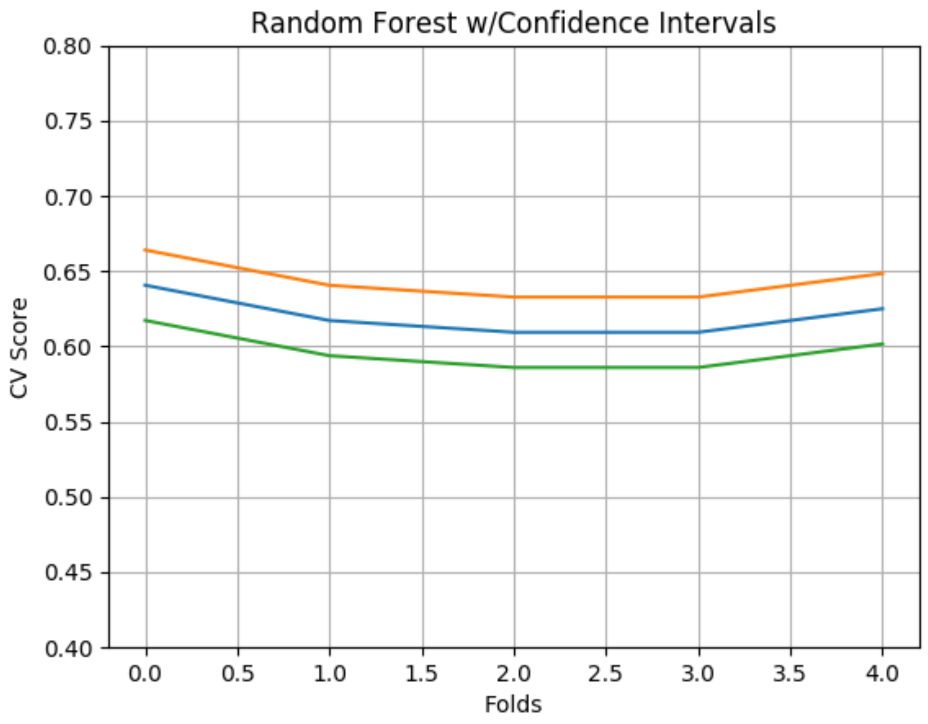
Algorithm	**Average Test Score**
Random Forest	0.662
Support Vector Machine	0.619
Neural Network	0.592

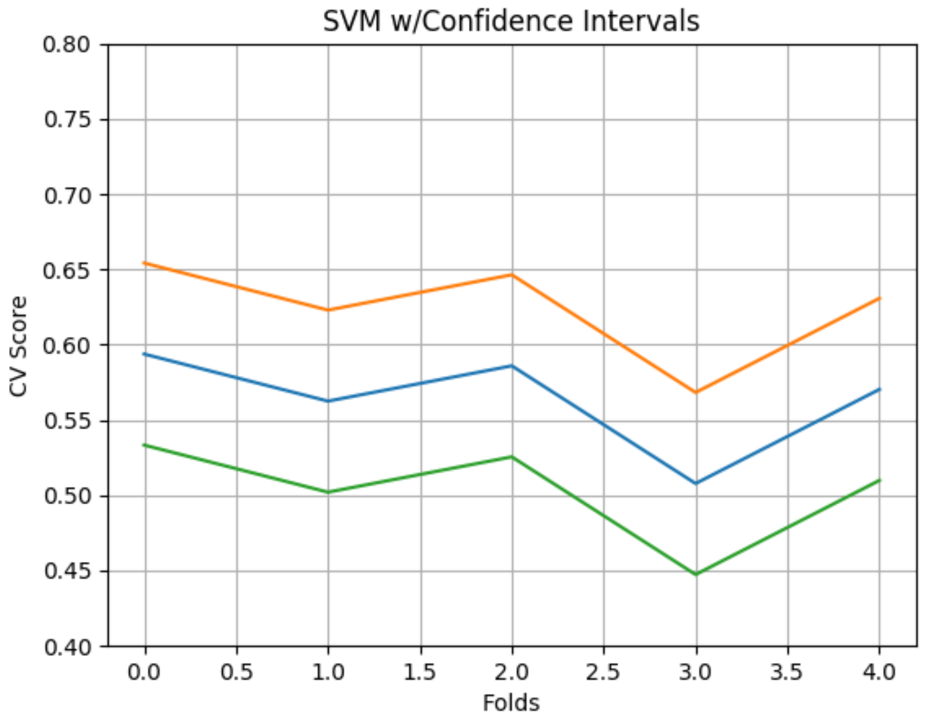
Comparing Algorithm Performance

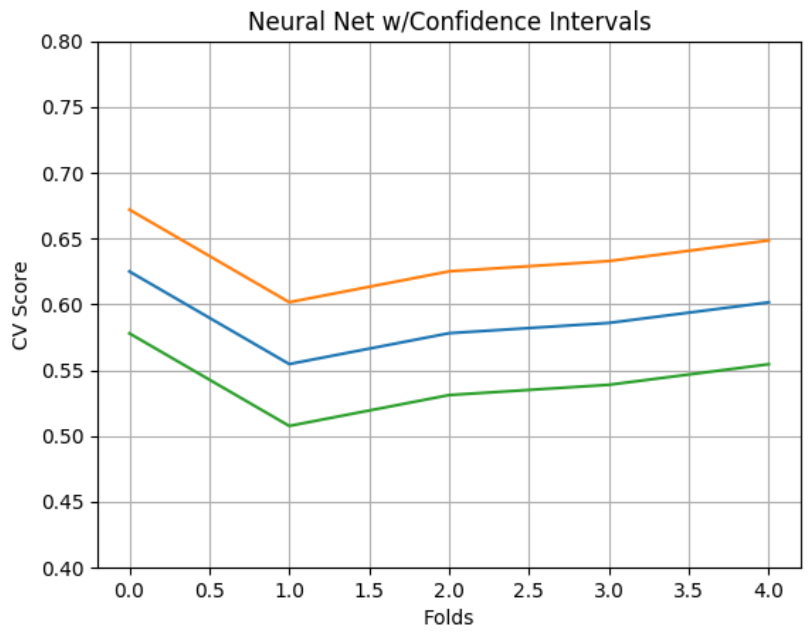
To compare the performance of the classification models, I used k = 5 folds cross validation to compute 95% confidence intervals and the accuracy for each algorithm with the optimal parameters on the testing data. The results are shown in the table and graphs below.

Algorithm	**Accuracy**	**Confidence Intervals**
Random Forest	0.620	(0.610, 0.631)
Support Vector Machine	0.5640625	(0.538, 0.591)
Neural Netowk	0.589	(0.568, 0.610)




Random Forest w/Confidence Intervals
CV Score
Folds
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


SVM w/Confidence Intervals
CV Score
Folds
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



The random forest algorithm works best on this data set because not only does it have the best performance (highest accuracy), but also the smallest confidence intervals.

Conclusion

Based on the computation time and number of hyperparameters, I would still select the random forest classifier as the best one for this data set. The support vector machine is faster, but the random forest classifier is more accurate even and does not take too long. I would not chose the neural network due to the number of hyperparameters it has, and the amount of computation time it took to run each experiment for tuning the hyperparamters. With real world data, I would have a lot more class labels since the quality of wine is not solely ranked with integers. Depending on how many class labels I had with the real world data, I would choose either the support vector machine or random forest. If I had a lot of class labels, I would choose the support vector machine because it is much faster, but if I had a manageable amount of class labels, I would still choose the random forest classifier.

Acknowledgments

[1] https://makingnoiseandhearingthings.com/2017/12/30/how-to-be-wrong-measuring-error-in-machine-learning-models/

[2] https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[3] https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

[4] https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a

[5] https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html

Software Libraries

Scikit-Learn: http://scikit-learn.github.io/stable