Assignment 1 : Supervised Learning (CS7641)

Stella Lai-Hoong Soh

lsoh3@gatech.edu

An Overview of The Datasets

The datasets I have chosen to use are: the wine quality dataset and the star type dataset downloaded from the Kaggle site.

Wine Quality Dataset

This is a 1600 (rows) x 12 (columns) dataset, consisting of input feature attributes such as fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulphur dioxide, total sulphur dioxide, density, pH, sulphates and alcohol based on physicochemical tests. The last target column - the quality is assigned a 'good' or 'bad' string. The additional processing I undertook was assigning a 1 to the 'good' quality wines, and a 0 to the 'bad' quality wines.

Star Type Dataset

This is a 3643 (rows) x 7 (columns) dataset that contains several features of stars. The processing I needed to do was with the SpType (Spectral Type) column. It turns out that there are 584 unique spectral types (in staring) amongst the data collected. I mapped these spectral types in SpType column to integer values spanning from 0 to 583. In the target column, 1 indicates it is a Giant star, and 0 indicates it is a Dwarf star. An alternative way I could have transformed the SpType column is using one-hot encoding. By applying one-hot encoding to the 584 spectral types, I would have added a binary variable for each spectral type. This small change makes the values (vectors) of SpType column friendlier for model training, rescales better, and provides for more nuanced predictions than mere single labels.

Why Are These Interesting Datasets?

Wine Quality Datasets

Winemaking is a long enough process. Running wine tests in wine labs also calls for lab tests and equipment, which often stresses out the smaller wineries that have limited capital resources. If a good machine learning model exists to predict good wines from the bad, it'd save the wine labs from labor-intensive steps and costly equipment essential to the production process.

What defines quality? In this dataset, the wine quality is determined by 11 physical attributes. In the wine industry, however, what constitutes a good wine boils down to factors such as 1) balance, 2) length, 3) depth, 4) complexity, 5) finish and 6) typicity. [1] My hypothesis is: We should make some of these subjective factors into ordinal variables, and introduce them into the original dataset. Having these additional input features may make the model training more accurate.

Star_Type Datasets

Most stars are so far away from the Sun that even if we were equipped with the most powerful telescopes, we can see them only as tiny points of light. How does one measure the distance or attempt to classify Giant or Dwarf stars? The author who published the dataset pointed out that

[1] Refer to *How To Discern Wine Quality*

"different approaches (Neural Network) were used to draw the decision boundary to categorize the stars." Primarily, B-V color and the Absolute Magnitude are used to identify Giant or Dwarf stars. In retrospect, this dataset that I have picked - being a 6 attributes by 3643 examples - offers only a sparse sampling of space. I am curious to know if one obtains better prediction (i.e. greater accuracy) using a neural network model compared to say, a decision tree model.

Experiment Steps

For the two datasets and five ML algorithms, I experimented with the following steps.

1) Scale the feature values and split the dataset into training set (80%) and testing set (20%)
2) Create an unoptimized (baseline) model to get an idea of the model's performance.
3) Generate validation curves for the model with respect to two hyperparameters.
4) Generate a learning curve for the model.
5) Find an optimized model with respect to some hyperparameters using GridSearchCV.

Wine_Quality Dataset

1. Decision Trees

I fitted the decision tree with a max_leaf_nodes=8 and class_weight of 'balanced' to get a baseline idea of the performance. The validation curves were plotted over 2 hyperparameters: (a) max_leaf_nodes, and (b) min_samples_leaf, with 5-fold cross-validation.

Over max_leaf_nodes and min_samples_leaf, the training curve reaches a high accuracy score fairly quickly. While the cross-validation curve (refer Fig.1) for the max_leaf_nodes lags behind, and stays around 0.753, the cross-validation curve (refer Fig.2) increases in accuracy and then drops off. This shows that at high values of max_leaf_nodes, the decision tree model overfits more. For the decision tree model that trains over larger values of min_samples_leaf, the accuracy of the training set gets closer to the accuracy of the training set.

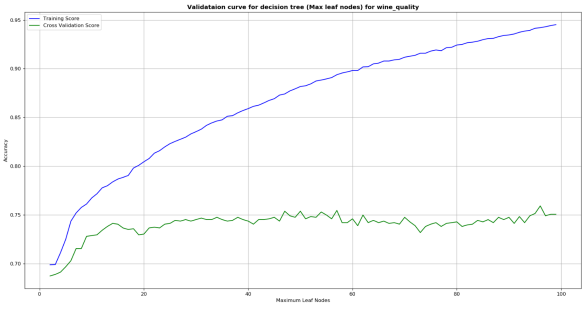

Fig.1 -- Validation curve for decision tree of wine_quality(max_leaf_nodes)

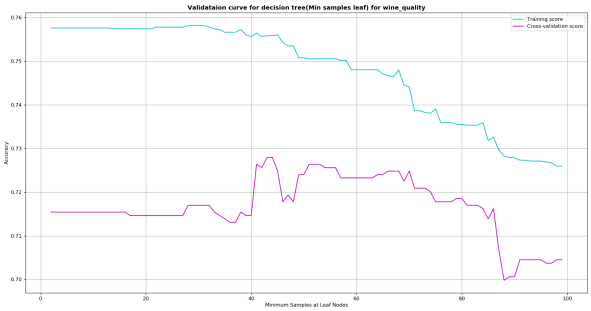

Fig.2 -- Validation curve for decision tree of wine_quality(min_samples_leaf)

For the learning curve, I observe that as the size of the training set increases, the training score and cross-validation score slowly gets closer to 0.75. The training score is slightly more accurate when compared to the cross-validation score. Both training and cross-validation accuracies are reasonably

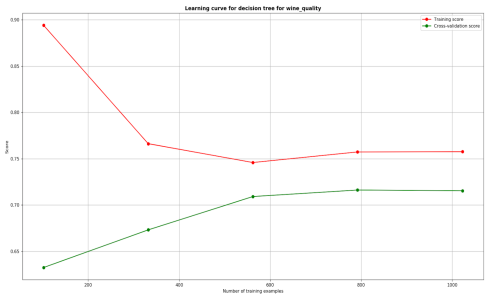

Fig.3 -- Learning curve for decision tree of wine_quality

high while the gap between them is small. The cross-validation's accuracy appears to be increasing when we add more training data.

Hyperparameter Tuning and Performance
The training of wine_quality dataset completed in (1011.2044/60) 16.85 minutes for this decision tree classifier.

To get an idea of which is the "weakest" link[2], and essentially what is the effective ccp_alpha to post-prune, I initially wanted to use DecisionTreeClassifier's cost_complexity_pruning_path(). I came to the realization, however, that one could obtain the best ccp_alpha through GridSerachCV(). The best decision tree parameters found are: {ccp_alpha: 0.0, max_leaf_nodes: 90, min_samples_leaf:2}

From GridSearchCV(), it appears that pruning is not necessary for my datasets. I believe this is due to the relatively small number of features in the dataset.

2. Neural Networks

Using sklearn's MLPClassifier, I trained and fitted a neural network with 2 hidden layers of 5 and 2 nodes. The default ReLU activation function for the hidden layers was used.

Validation curves of the neural network were plotted over 2 hyperparameters: (a) the regularization term, alpha and (b) the learning rate using 5-fold cross-validation.

Evaluating the model based on alpha, I see in Fig. 4, the training curve(in blue) reaches a high accuracy score fairly quickly while the cross-validation curve (in green) lags behind.This means the model is overfitting at lower values of alpha. Evaluating the model based on learning rate as in Fig. 5 gives us the insight that the model captures the underlying patterns of the dataset pretty well as the learning curve and cross-validation curve look similar to each other. The model exhibits low bias, low variance.

[2] Refer to sklearn's documentation on post-pruning decision trees

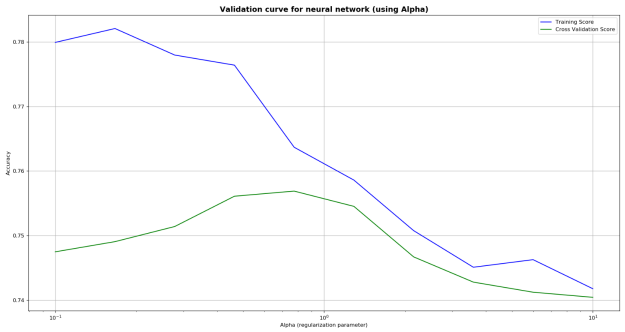

Fig.4 — Validation curve for neural network of wine_quality dataset (alpha)

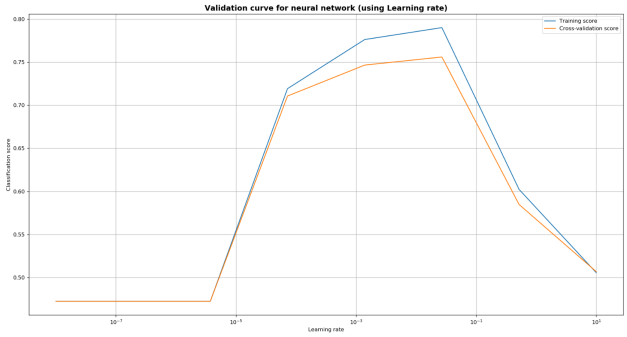

Fig.5 — Validation curve for neural network of wine_quality dataset (learning rate)

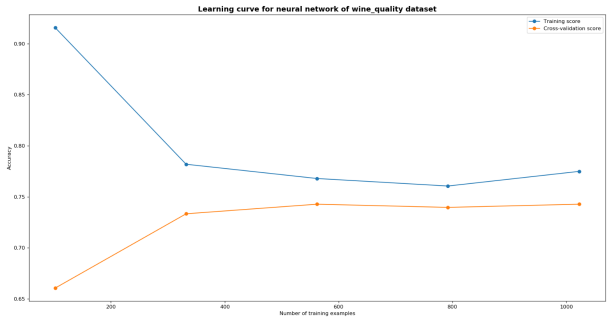

Fig. 6 — Learning curve for neural network

For the learning curve in Fig. 6, at higher numbers of training examples, the training accuracy increases. The cross-validation does not get more accurate as we add more training data. Again, this confirms that we have a low bias, low variance model.

Loss curve of the neural network was plotted over the 3 hyperparameters: (a) alpha, (b) hidden layer sizes and (c) learning rate. I see that the training loss decreases as the number of epochs increases and then converges to a low value (meaning training loss is maintained at a minimum). This indicates that the neural network is performing well for the wine_quality dataset.

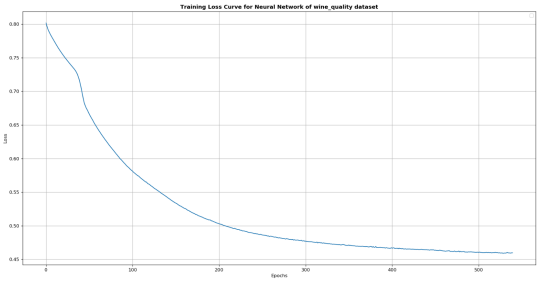

Fig. 7 -- Training loss-curve

Performance and Accuracy

From GridSearchCV(), the best neural network parameters found for wine_quality dataset are:{'alpha': 0.46415888336127786, 'hidden_layer_sizes': (10, 5, 2), 'learning_rate_init': 0.001389495494373139}

The training completed in (408.4268/60) 6.81 minutes.

3. Boosting (AdaBoost Classifier)

I used AdaBoostClassifier from sklearn.ensemble, which, by default, uses the decision tree classifier as a base estimator initialized with max_depth=1. Therefore the same decision tree code base has been used.

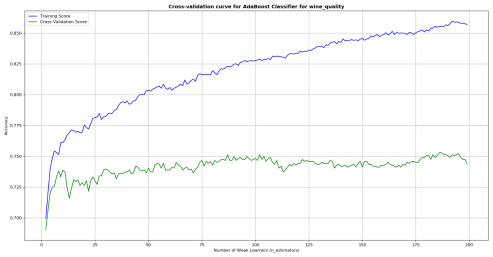

Fig. 8 -- Validation curve of AdaBoost (n_estimators)

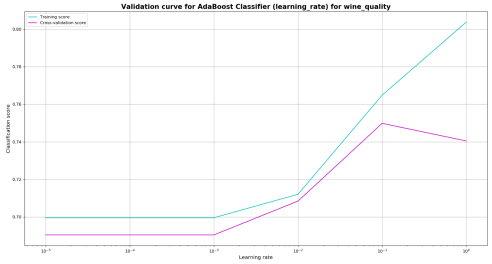

Fig. 9 -- Validation curve of AdaBoostClassifier (learning rate)

Refer to Fig. 8. Using n_estimators, the cross-validation's (in green) accuracy reaches a maximum of 0.75. Interestingly, using learning rate, the cross-validation 's (in magenta) accuracy starts out much lower at 0.695 and stays constant at this value for learning rates of 0.00001 to 0.001, and then increases to 0.75 and drops off in Fig. 9.

From the logfile that I collected, the difference in accuracy between the training and testing data before

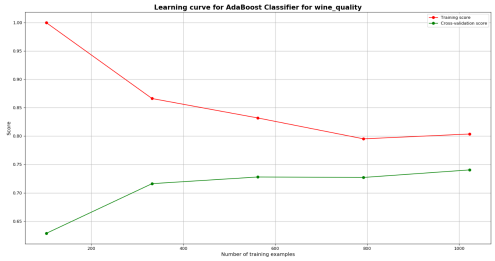

Figure 10 - Learning curve

optimization is 0.0576 (0.8014 - 0.7438) . The difference in accuracy using the optimized AdaBoost classifier is 0.0835 (0.8210 - 0.7375). This is a jump of 2.59% in accuracy. Further, the AUC score for training data in the optimized model also shows a higher score - 0.8207 vs. 0.8003. It appears that the optimized model overfits more compared to the non-optimized model. One reason could be the addition of more weak learners and the weight-assigning technique after every iteration ensures that the AdaBoost classifier increases the model's accuracy.

For the learning curve, the accuracy of the training score starts off pretty high at 1.0 around 150 training examples. The training score's accuracy decreases further and then goes up and stays around 0.80. At higher numbers of training examples, the decrease of training score points to this model being high bias.

The cross-validation scores of both datasets do not get more accurate as we add more training as we add more training data. This Adaboost model exhibits low variance and high bias.

Training of AdaBoost Classifier took (18.7023 / 60) 0.3117 minutes.

4. Support Vector Machines

For Support Vector Machines, I experimented with a linear kernel and a polynomial kernel. I found that the AUC for training and testing data is slightly higher for SVM with a polynomial kernel. Therefore, the rest of the experiments have been run with SVM with a polynomial kernel.

The training score and cross-validation score of SVM with linear kernel (refer Fig.11) increases at a faster rate than those of SVM with polynomial kernel (refer Fig.12). The difference in accuracy between training and testing data is 0.0201. The AUC for training data is 0.7499 and for testing data is 0.7336.

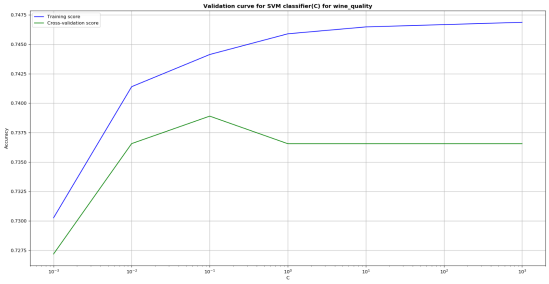

Fig. 11— Validation Curve of SVM with linear kernel

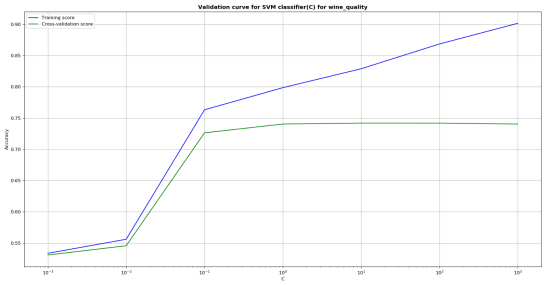

Fig.12 -- Validation Curve of SVM with polynomial kernel

The training time for SVM with a polynomial kernel is a mere 10.8502 seconds (!) - the fastest amongst all supervised learning algorithms.

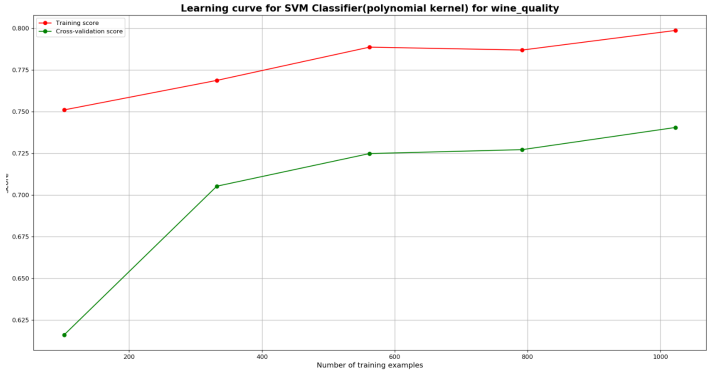

Fig.13 -- Learning Curve of SVM with polynomial kernel

The learning curve of SVM (refer Fig.13) shows that the training score and cross-validation score increases as more training examples are added. But they have not converged even at 1000 training examples. So this model would probably benefit from having more training data.

5. k-Nearest Neighbors

For the wine_quality dataset, the training score remains at a constant 1.0 (refer Fig.14) vs. the validation score which oscillates between 0.75 and 0.80 for all values of n_neighbors. This accords with what I am seeing in the classification report for k-NN where precision, recall and F-1 scores are 1.0 for k-NN classifier's training and testing data. Refer to Tables 1 and 2 on Page 12. The training time for both datasets is the shortest under k-NN. But, why does the training score remain at a constant 1.0? It piqued my interest to find out.

k-NN is non-parametric and uses a lazy learning algorithm[3]. I have come to understand that unlike eager learners, k-NN first stores all instances of the training points during the training phase. Let's say in the training data, there is a point P1 whose label we need to predict. k-NN algorithm calls for finding k closest points to P1, and then by majority voting, vote for the label and thus classify P1. During the predicting phase, since the instances of all the training points are already loaded into the model, when k-nearest neighbors of P1 are found, its own instance is amongst these neighbors as well. By majority voting, it's highly likely that the label of P1 will be voted. This probability of the algorithm finding and matching P1's own label is very high. Therefore the accuracy of the training score as reflected in Fig. 14 is 1.0.

On the other hand, instances of the testing data are not loaded into the model. Let's say we have a point P2 in the testing data. Due to majority voting, since P2's instance is not in the model, the label that's voted to classify P2 may not always match P2' own label. Thus the accuracy of cross-validation is less than 1 as reflected in Fig.14.

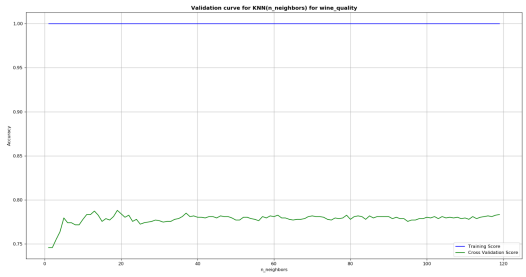

Fig. 14 -- Validation Curve for k-NN (n_neighbors)

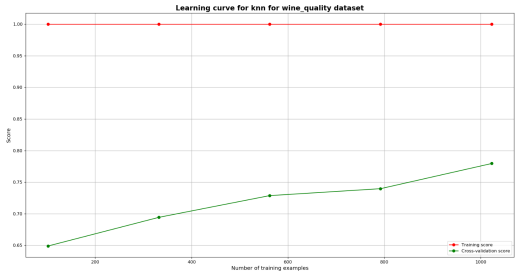

Fig. 15 — Learning Curve for k-NN classifier

For the learning curve, the accuracy of cross-validation increases with an increasing number of neighbors, up to a certain point. From running my knn.py code, I found that the best knn score is 0.788, and the best knn parameter found is n_neighbors=19. Looking at the validation curve in Fig.14, it confirms the same. The training score accuracy (as explained above) stays at 1.0.

Training completed in 7.0562 seconds for KNN classifier for wine_quality dataset.

Star_type Dataset

1. Decision Trees

The validation curves were plotted over 2 hyperparameters: (a) max_leaf_nodes, and (b) min_samples_leaf, with 5-fold cross-validation.

[3] Refer to *KNN Classification using Scikit-learn*

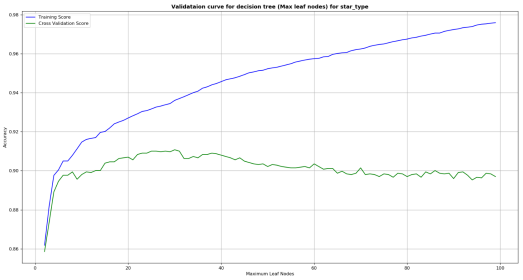

Fig.16 -- Validation Curve over max_leaf_nodes

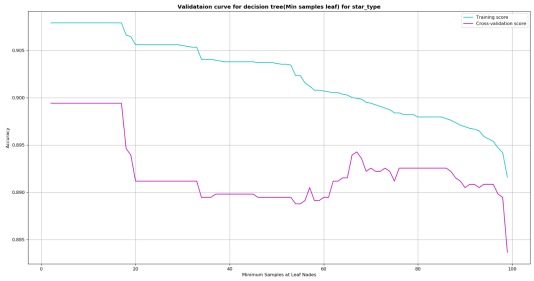

Fig.17 --Validation Curve over min_samples_leaf

The AUC before optimization is 0.9069 for training data, and 0.8864 for testing data. The AUC after optimization is 0.9375 for training data, and 0.9053 for testing data. There is a significant increase in AUC score after optimization. One reason could be the combination of best parameters : {'ccp_alpha': 0.0, 'max_leaf_nodes': 30, 'min_samples_leaf': 2} when applied makes for a model that's better at distinguishing the 'good' wines and 'bad' wines much more than before optimization.

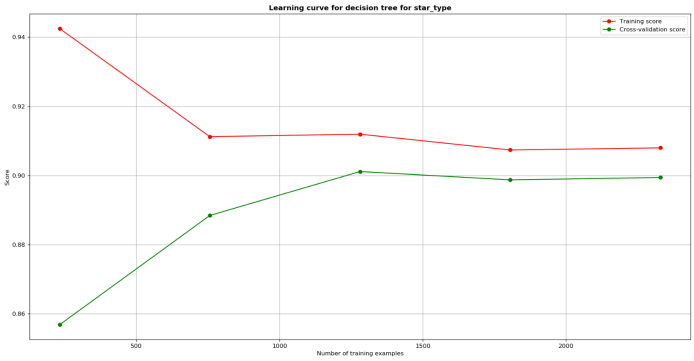

Fig. 18 -- Learning curve of star_type dataset

Training completed in (1675.0565 / 60) 27.917 minutes for star_type dataset.

2. Neural Networks

Validation curves for both alpha and learning rate are plotted and shown below:

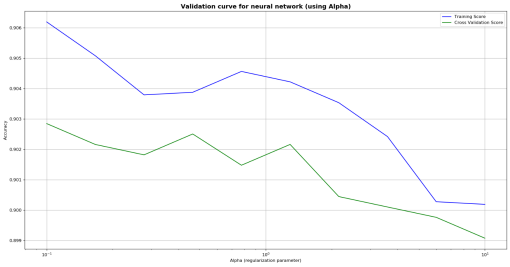

Fig. 19 — Validation curve for star_type dataset(alpha)

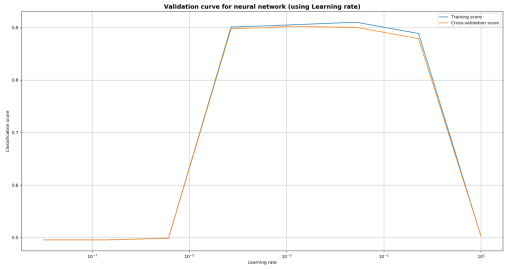

Fig. 20 — Validation curve for star_type dataset(learning rate)

Using alpha, we can see that the difference between the training and cross-validation scores is large in Fig. 19. On the other hand, using the learning rate, the difference between the training and

cross-validation is much smaller in Fig.20. The high bias that we are seeing in the alpha case is not there in the learning rate case.

The learning curve in Fig. 21 shows that with a higher number of training examples, the decrease in training score's accuracy points to the model being high bias. The cross-validation score's accuracy does increase as we add more training data. We can conclude that this neural network model exhibits low variance and high bias.

The loss curve in Fig.22 was plotted over the 3 hyperparameters of alpha, hidden layer sizes and learning rate. I observe that the training loss decreases as the number of epochs increases and then it converges to a low value. Again, this indicates that the neural network model is performing well for the star_type dataset.

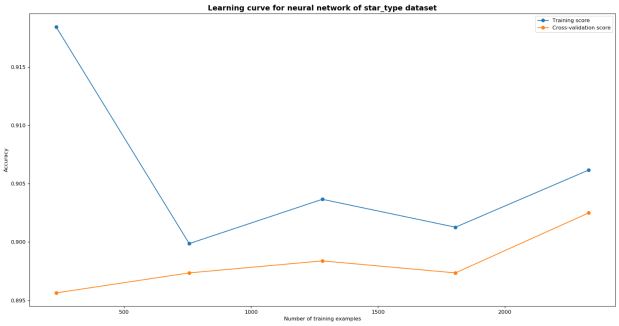

Fig. 21 - Learning curve of nn_star_type

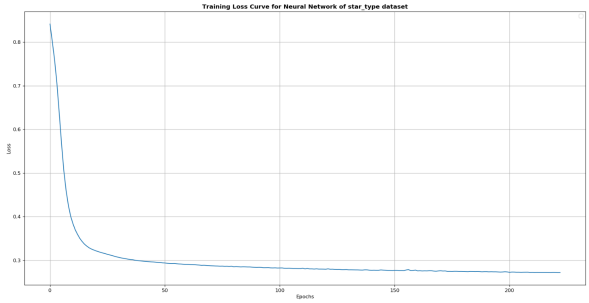

Fig.22 - Loss curve of nn_star_type

Training for the star_type dataset completed in (860.9914/60) 14.35 minutes.

3. Boosting (AdaBoost Classifier)

The validation curve in Fig.(24) for the star_type starts off at a low accuracy score and increases with increasing learning rate, and doesn't drop off, as in the case for wine_quality in Fig. 9.

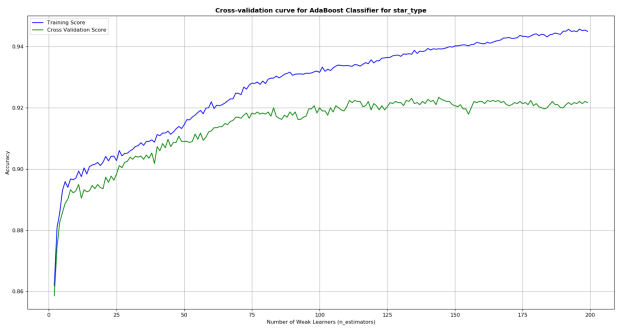

Fig. 23 - Validation curve for star_type (n_estimators)

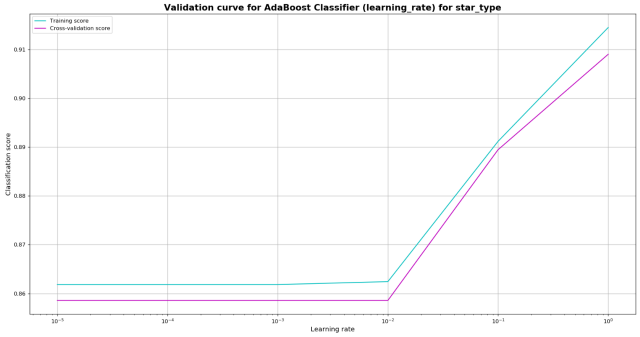

Fig. 24 - Validation curve for star_type (learning_rate)

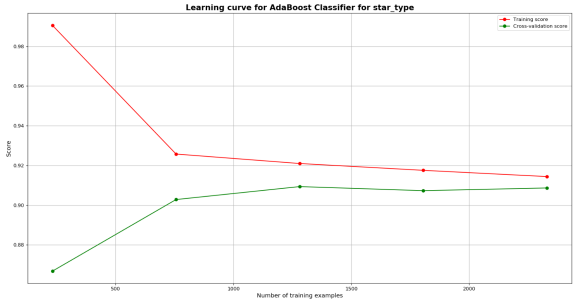

Fig. 25 -- Learning curve for star_type

The learning curve for star_type dataset, Fig. 25 (above) illustrates that at higher numbers of training examples, there is a decrease in accuracy of training score. The cross-validation score gets more accurate as we add more training data. This is good news, as it points to neither high bias nor high variance. We can conclude that the AdaBoost classifier is performing well for star_type dataset. Furthermore, the AUC score for training and testing data in the optimized model also shows an increase - 0.9201 vs. 0.9149 (training) and 0.905 vs. 0.9013 (testing), which confirms our conclusion.

Training completed in (27.2954/60) 0.4549 minutes.

4. Support Vector Machines

Fig. 26, the training and cross-validation curve for SVM with linear kernel looks interesting. The regularization C parameter conveys to the SVM optimizer how much one wants to avoid misclassification of data. At C=10 and beyond, we see the cross-validation score's accuracy gets to be higher than that of the training score. This scenario defies comprehension.

From researching on the web, I learned that at higher values of C, the optimizer "encourages" a smaller-margin hyperplane. A smaller-margin hyperplane translates to lower misclassification rate for the training data. We see therefore on Fig.26, the training accuracy starts increasing at C=10. But why would the cross-validation score's accuracy get to be higher? I still don't have an explanation. But I see that eventually the training score catches up and converges to the same level of accuracy as the cross-validation score.

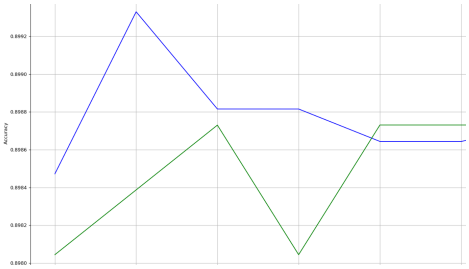

Fig. 26 - Validation curve of SVM (linear kernel) for star_type

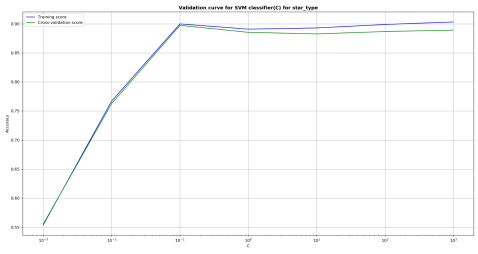

Fig.27 - Validation curve of SVM (polynomial kernel) for star_type

Refer to Fig. 27. With a polynomial kernel, the training and cross-validation score curves only start to diverge at C=0.1 and beyond.

The AUC scores for training and testing data (after optimization) also shows an increase - 0.9076 vs. 0.8913 (training) and 0.8862 vs. 0.8721 (testing).

Training completed in (10.0438 / 60) 0.1674 minutes.

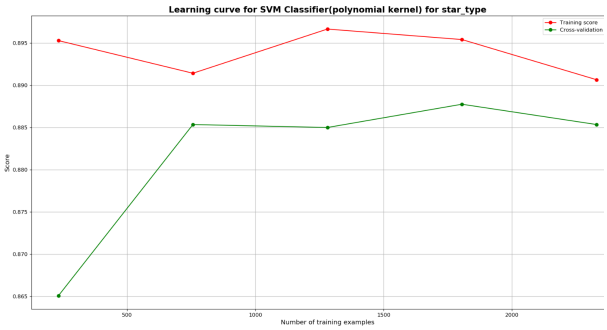

Fig.28 - Learning curve of SVM with polynomial kernel

On Fig.28, there is a difference of 0.03 in accuracy between the training and cross-validation score in the beginning of training. Here, the training and cross-validation accuracies show trends of converging at some point with larger training examples, which is always a good sign.

5. k-Nearest Neighbors

Refer to Fig. 29. The training score similarly remains at a constant 1.0 while the validation score oscillates between 0.893 and 0.91 for all values of n_neighbors. The same analysis (on Page 7) of why the training score is always 1.0 applies here.

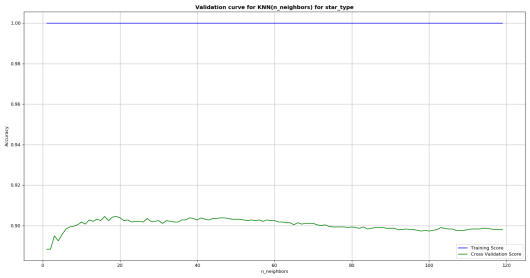

Fig.29 - Validation curve (kNN) for star_type dataset

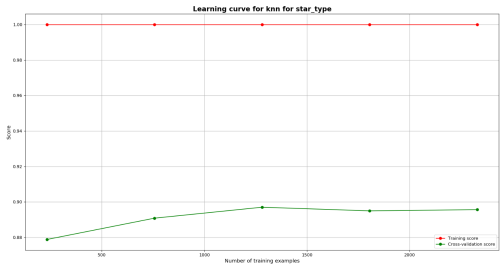

Fig.30 - Learning curve (kNN) for star_type dataset

Training completed in 5.0256 seconds (!) for the star_type dataset to be trained with kNN classifier. This is the shortest time for the training of any dataset amongst all the supervised learning models.

CONCLUSION

For both datasets, k-NN takes the prize for being the one with the shortest training time out of all the supervised learning models. Like most lazy algorithms, k-NN's faster training time is at the expense of a

slower and costlier testing phase. k-NN requires more time to scan data points, and scanning data points translates to requiring more memory for storing these data points.

Wine_Quality

	Decision Tree	**Neural Network**	**AdaBoost**	**SVM**	**kNN**
Training time	16.85 mins.	6.81 mins.	0.32 mins.	0.18 mins.	0.12 mins.
Testing time	0.0003 secs.	0.0012 secs.	0.1776 secs.	0.0062 secs.	0.0003 secs.
Accuracy (testing)	0.7219	0.7781	0.7375	0.7656	0.7844
AUC_testing (optimized)	0.7235	0.7768	0.7368	0.7649	0.7817

Table 1— Training times, Accuracy and AUC scores for wine_quality dataset

Star_type

	Decision Tree	**Neural Network**	**AdaBoost**	**SVM**	**kNN**
Training Time	27.92 mins.	14.35 mins.	0.45 mins.	0.17 mins.	0.084 mins.
Testing Time	0.0005 secs.	0.0038 secs.	0.2217 secs.	0.012 secs.	0.005 secs.
Accuracy (testing)	0.9053	0.8848	0.9053	0.8861	0.8875
AUC_testing(optimized)	0.9053	0.8847	0.905	0.8862	0.8874

Table 2 — Training times, Accuracy and AUC scores for star_type dataset