Supervised Learning

Georgia Institute of Technology CS7641: Machine Learning Assignment 1

Weixu Chen

GT ID: wchen607

Abstract

This report summarizes the application of different supervised learning algorithms to analyze on two datasets which are loan pack back and wine quality.

In methodology, I have used Decision Trees, Neural Networks, Boosting, Support Vector Machines and k-Nearest Neighbors. Performance of different models are also compared in this analysis. The programming and analysis are done using Python scikit-learn.

Datasets Introduction

Lending Club connects people who need money (borrowers) with people who have money (investors). As an investor you would want to invest in people who showed a profile of having a high probability of paying you back. I will try to create a model that will help predict this. I use lending data from 2007-2010 and be trying to classify and predict whether or not the borrower paid back their loan in full. The class distribution is “not fully paid” (1) 16% and “fully paid” (0) 84%. Thus, the dataset is unbalanced. It is labeled as Case I.

Wine quality justification are based on given physicochemical properties and sensory data of the Portuguese "Vinho Verde" wine. The goal is to predict wine quality. The dataset related to the white variant of the wine was chosen for experiments since it contains a larger amount of data. The labels in the original dataset are the wine quality measured on a scale from 0 to 10. To facilitate analysis, a cut-off of 6 was chosen. Values above it were relabeled as ‘1’ (good) and those below it as ‘0’ (bad). The dataset contains 66.52% positive samples and 33.48% negative samples. It is labeled as Case II.

Both datasets are from real-world and are classification problems. They are non-trival as well since involving hundreds of samples.

Data Preprocessing

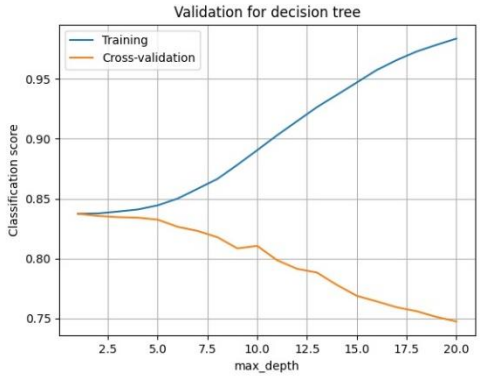
Each data set is 70-30 split into tow datasets. The portion of 70% is training set. The remaining 30% dataset is testing set. Various supervised learning algorithms are applied on the training set first and then on the testing dataset. Hyperparameters are tuned through 5 fold cross validation on the training set in the model selection. The model is selected based on the best testing score. Testing dataset will not be touched during model selection process and will only be used for testing the model. Taking the parameters for each model, the values are first run

using sklearn default values, and a detailed analysis and the learning curve will be given in each of learning algorithms. The performance will be summarized and compared.

Decision Trees

The decision tree was pre-pruned by limiting its maximum depth. Pruning is often used to reduce the size of tree. However, pruning would decrease the accuracy of the training set, but in general increases the accuracy of testing data. It is used to mitigate overfitting, in case the model achieves too good accuracy for training data, but at the same time too specific and performs poorly on test data. To beat overfitting, the decision tree was pre-pruned by limiting the parameter of maximum depth. Generally, a tree with a large depth will try to fit the training data exactly and is likely to result in overfitting. Next consideration was how to split a node? The criterion used to split a node is Gini index in the algorithm, a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset.

At low values of max_depth, the tree is very pruned and could suffer from underfitting as both the training and testing scores are low for Case II. In the following the training scores increase as max_depth increasing. And the testing accompanying increases and then plateaus. This means at high values of max_depth, the tree starts to overfit. However, the observations are different in Case I. The tree is not underfitting at low values of max_depth and quickly overfitting since the testing score decreasing when max_depth increasing. So the excessively pruned tree is fit for this dataset.



Case I

Case II

After conducting a grid search, the optimal value of max_depth and min_sample_split have been found

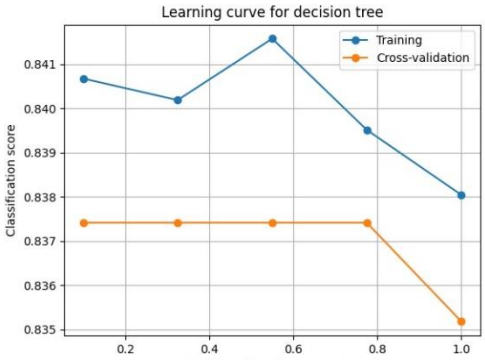
Case I: {'max_depth': 1, 'min_samples_split': 2}

Case II: {'max_depth': 21, 'min_samples_split': 2}

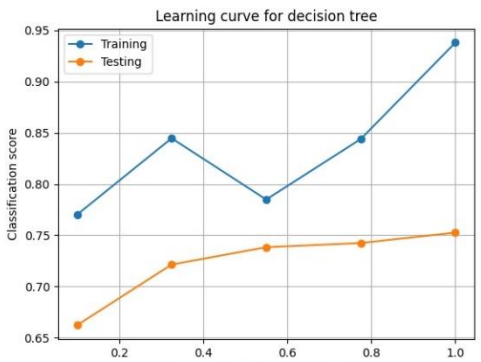
Case I: Decision Tress ---> outputs				
	precision	recall	f1-score	support
0	0.85	1	0.92	2431
1	0.19	0	0	443
accuracy			0.74	2874
macro avg	0.49	0.5	0.46	2874
weighted avg	0.73	0.84	0.77	2874
Case II: Decision Tress ---> outputs				
	precision	recall	f1-score	support
0	0.64	0.66	0.65	460
1	0.84	0.83	0.84	1010
accuracy			0.78	1470
macro avg	0.74	0.74	0.74	1470
weighted avg	0.78	0.78	0.78	1470

For Case I, we can see from learning curves that the accuracy decreases with the number of training samples becoming over 50%, this occurs on both training and test curves. It indicates that unbalanced dataset worsening the algorithm performances.

For Case II, there is a big difference between the training and testing scores at the maximum data size. This suggests that the classifier is suffering from high variance and will likely benefit from more training data. Both the scores are low indicating that it is also suffering from high bias.



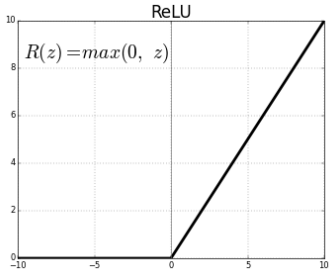
Case I



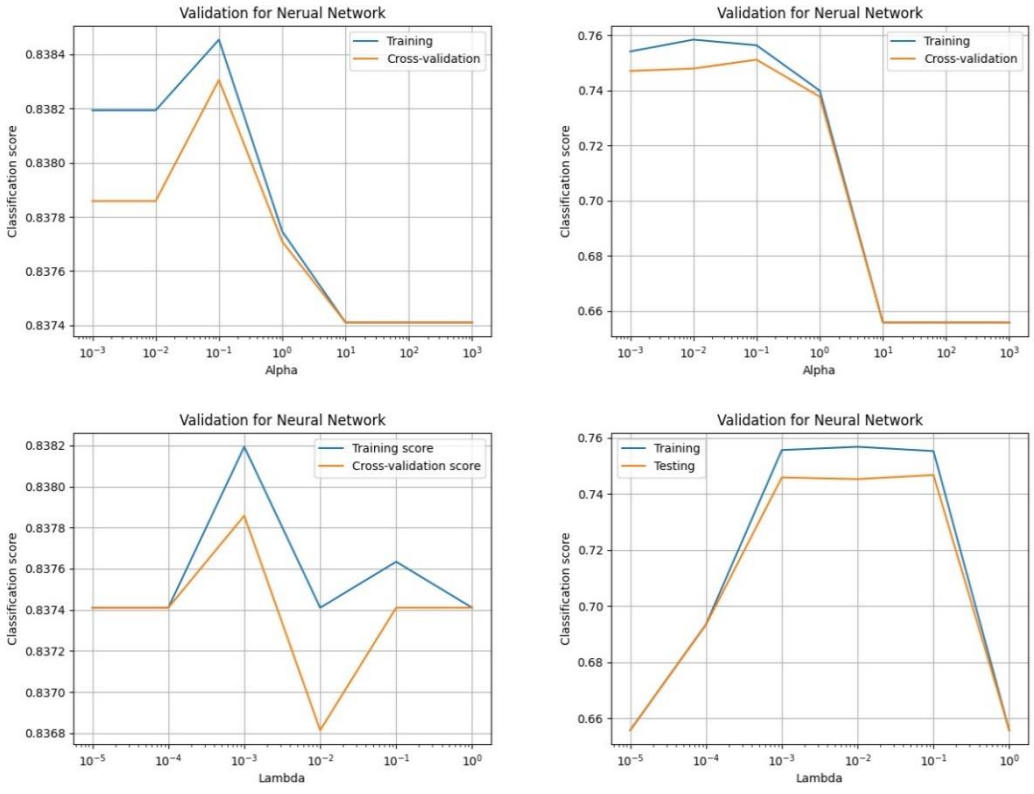
Case II

Neural Network

The neural network has been set up with 2 hidden layers which including 7 and 2 nodes and their activation functions was ReLU. The function plot is given as the following. It is default activation function of sklearn neural network module as well.



As learned from lectures, regulation parameter, alpha and learning rate, lambda are two key parameters which need to a better selection and tune. So I have run the validation curve to visual both parameters. The plots show below.



Case I Case II

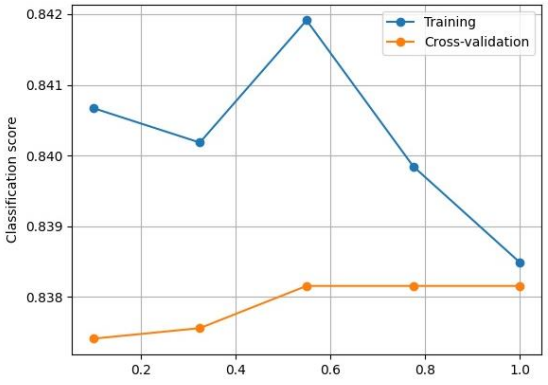
As we can capture from the plots, the performance of model is going to be degraded when alpha is too high, and it is due to network faces the high bias. Comparingly, when lambda it too high or too low, model gives down performance because of the training/testing can not converge. In Case I, the situation is even worse, the classification scores are wavy and hardly

locate the good value of lambda. Even that I still performed grid search to try to find the optimal hyperparameters.

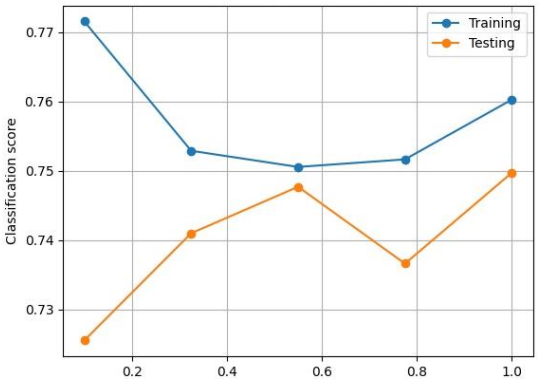
Case I: {'alpha': 0, 'learning_rate_init': 10^-3}

Case II: {'alpha': 0.1, 'learning_rate_init': 0.1}

Case I: Neural Network ---> outputs				
	precision	recall	f1-score	support
0	0.85	0.99	0.92	2431
1	0.48	0.03	0.05	443
accuracy			0.85	2874
macro avg	0.66	0.51	0.48	2874
weighted avg	0.79	0.85	0.78	2874
Case II : Neural Network ---> outputs				
	precision	recall	f1-score	support
0	0.78	0.3	0.43	460
1	0.75	0.96	0.84	1010
accuracy			0.75	1470
macro	0.76	0.63	0.64	1470
weighted	0.76	0.75	0.71	1470



Case I



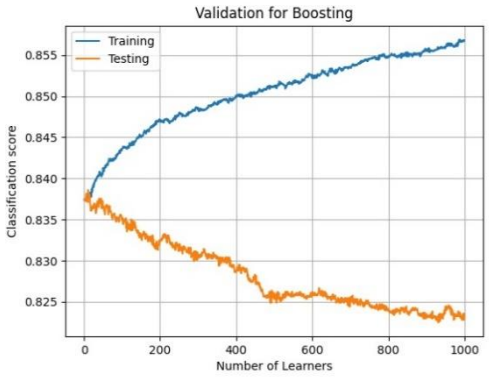
Case II

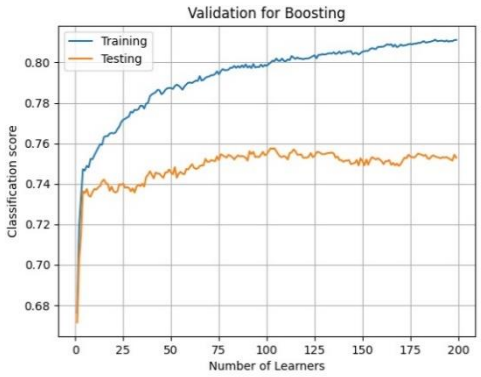
For Case I, the training/testing converge to the lower values, it suggests that the network is underfitting. The solution could be adding more layers or increasing the size of layers is driving to make network more powerful and learn better.

For Case II, training and testing converge to the high value at the maximum training data size. It suggests the model learn well from data.

Boosting

For boosting, I have implemented AdaBoost classifier. AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases. The base estimator I have kept the determined model from the first part Decision Trees which 'max_depth': 1, 'min_samples_split': 2 for Case I and 'max_depth': 21, 'min_samples_split': 2 for Case II. The tuned parameters are focus on number of learners. As the same method conducted in previous portions, the validation curves are plots to show the influents of performances of classifier.





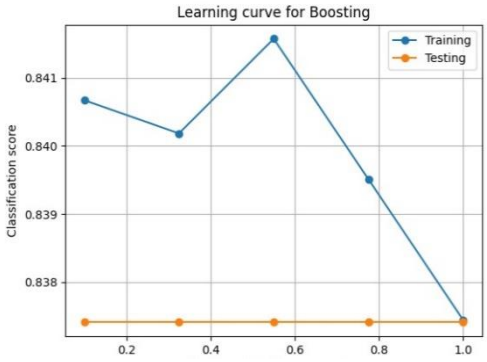
Case I Case II

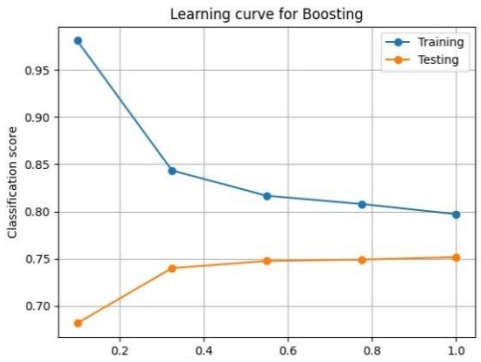
For Case I dataset, the training classification score/accuracy increases with an increase in the number of learners. It could be higher if adding more learners. However, the testing score oppositely decreases with an increase in the number of learners. This might be due to the excessive unbalanced presence of dataset in Case I. The more learners, the worse AdaBoost classifier is going to predict.

For Case II dataset, both training score and testing score increase when increasing the number of learners. The testing score goes to plateaus when learners reaching about 100. And some level of noise presenting in Case 2 datasets.

Case I: n_estimators = 1

Case II: n_estimators = 100





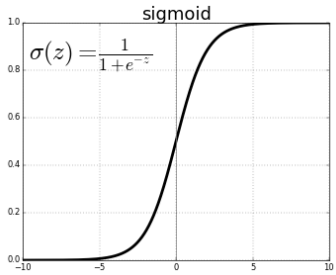
Case I dataset, training score drops excessively in the large training samples while testing accuracy does not change any. This tells us that Ada classifier failing to classify the datasets of Case I, and this is due to the extreme unbalanced data from Case I. As we can see from Case I outputs, precision, recall, f1-score were all zeros. Further feature engineering would have to be conducted in order to apply SVM module again.

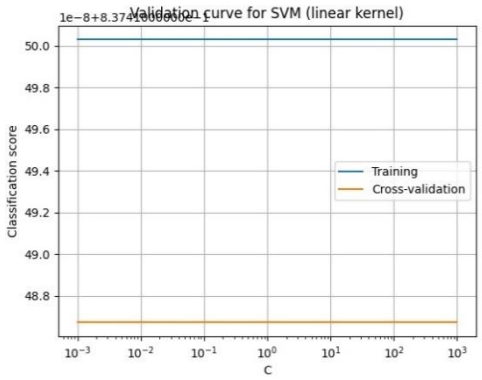
For Case II, training and testing converge to the high value at the maximum training data size. It suggests the model learn well from data.

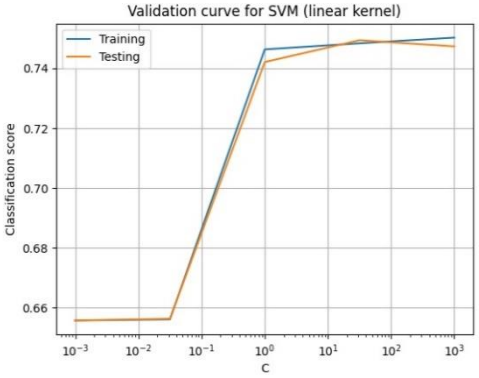
Case I: Ada Boosting [n_estimators = 1]---> outputs				
	precision	recall	f1-score	support
0	0.85	1.00	0.92	2431
1	0.00	0.00	0.00	443
accuracy			0.75	2874
macro avg	0.42	0.50	0.46	2874
weighted avg	0.72	0.85	0.78	2874
Ada Bo0sting [n_estimators = 100]---> outputs				
	precision	recall	f1-score	support
0	0.67	0.61	0.63	460
1	0.83	0.86	0.84	1010
accuracy			0.78	1470
macro	0.75	0.73	0.74	1470
weighted	0.78	0.78	0.78	1470

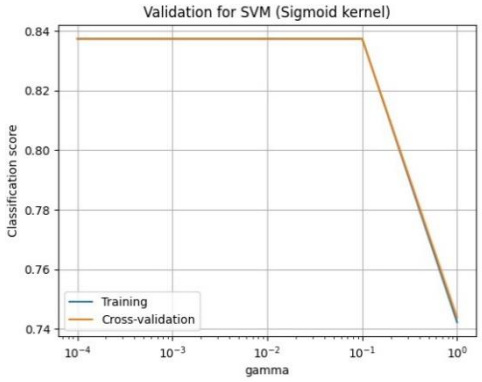
Support Vector Machines

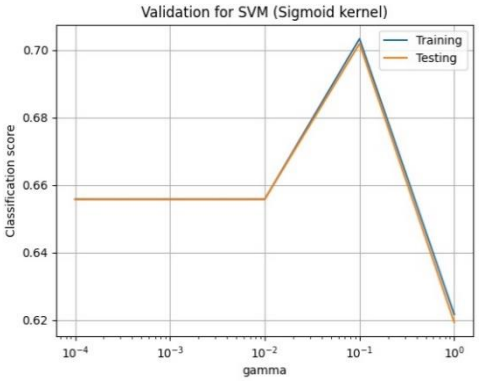
In Support Vector Machines (SVM) module, I have experimented two kernels which are linear, and sigmoid. The sigmoid function plot is given as the following.











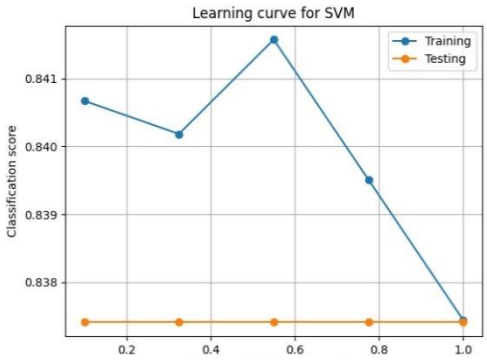
Case I Case II

The regularization parameter C, the strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. It has been checked to see how it affects the performance of SVM model. Kernel coefficient gamma has been checked as well in sigmoid kernel using. The output validation curves provided several observations.

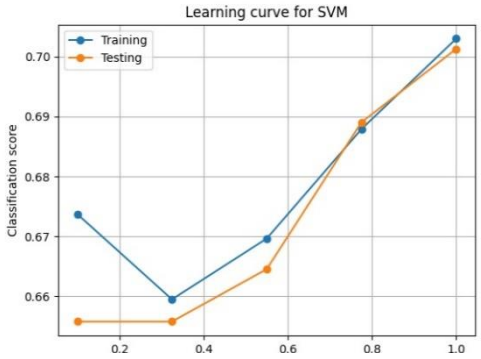
When taking linear kernel in Case I, both training and testing scores are flat at low values and does not converge. This suggests SVM was not able to learn well in using linear kernel. Meanwhile, in Case II, the training and testing curves increase as C increasing and converging to

a low value. It suggests that even with weak regularization, the SVM was not able to learn well as well. SVM was not fit for both Case I and Case II dataset when using linear kernel.

When taking sigmoid kernel, both training and testing curves converge and drop to low values at 10^-1. This occurs on both Case I and Case II datasets. So the optimal gamma 10^-1 has been chosen to perform learning.



Case I



Case II

For Case II dataset, we can observe that SVM works pretty well when fraction of training example getting larger and larger. The testing score/accuracy has been converged to the training score as well. SVM did a good learn for Case II dataset.

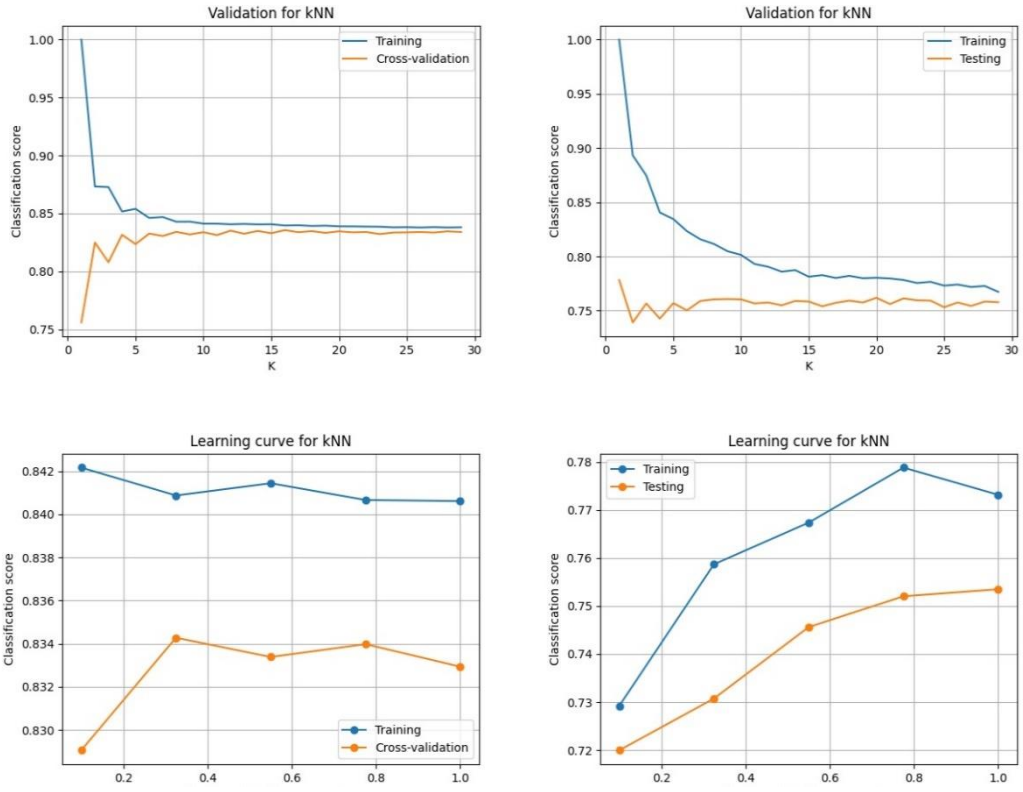
However for Case I dataset, training score drops excessively in the large training samples while testing accuracy does not change any. This tells us that SVM failing to classify the datasets of Case I, and this is due to the extreme unbalanced data from Case I. As we can see from Case I outputs, precision, recall, f1-score were all zeros. This indicates that SVM failed classifying. Further feature engineering would have to be conducted in order to apply SVM module again.

Case I: Support Vector Machines[sigmoid] ---> outputs				
	precision	recall	f1-score	support
0	0.85	1.00	0.92	2431
1	0.00	0.00	0.00	443
accuracy			0.75	2874
macro avg	0.42	0.50	0.46	2874
weighted avg	0.72	0.85	0.78	2874
Case II: Support Vector Machines[sigmoid] ---> outputs				
	precision	recall	f1-score	support
0	0.77	0.27	0.4	460

1	0.74	0.96	0.84	1010
accuracy			0.75	1470
macro avg	0.76	0.62	0.62	1470
weighted avg	0.75	0.75	0.7	1470

KNN

Neighbors-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point. The k-neighbors classification in KNeighborsClassifier is the most commonly used technique. The optimal choice of the value is highly data-dependent: in general a larger k suppresses the effects of noise, but makes the classification boundaries less distinct.



Case I Case II

For Case I dataset, the training score is looked as flat over the whole faction of training samples comparing with the cross-validation/testing score increasing. But the scores is still lower than

the training one at the maximum data size. This suggests that the classifier will need more training data to generalize the model.

For Case II dataset, bot training and testing score increase with increasing the number of training samples. As the maximum training data size, the training score is still larger than the testing one, the solution will be likely adding more training data to upgrade the generalization of classifier.

Through the tuning hyperparameters process, the optimal K values for Case I and Case II are 15 and 25. The outputs have been tabulated in the following.

Case I: KNN [K=15]---> outputs				
	precision	recall	f1-score	support
0	0.85	1	0.92	2431
1	0.67	0	0.01	443
accuracy			0.85	2874
macro avg	0.76	0.5	0.46	2874
weighted avg	0.82	0.85	0.78	2874
Case II: KNN[K=25]---> outputs				
	precision	recall	f1-score	support
0	0.66	0.57	0.61	460
1	0.82	0.87	0.84	1010
accuracy			0.77	1470
macro avg	0.74	0.72	0.73	1470
Weighted avg	0.77	0.77	0.77	1470

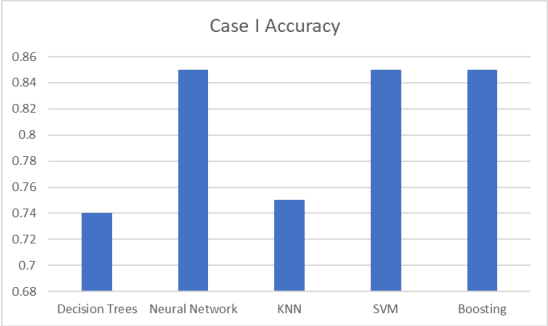
Summary of Classifiers

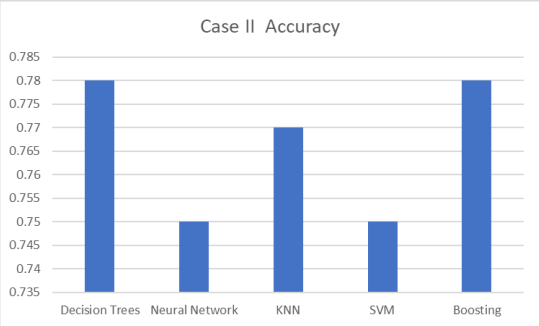
In Cass I: Decision Trees has the lowest accuracy due to it is the simplest and least expressive model among others. Neural network, SVM and Boosting has provided the improved accuracy. However, the computation time is more expensive than the base model decision trees. KNN did not give the good accuracy from Case I dataset.

In Cass II: Decision Trees and AdaBoost show the best accuracy followed by KNN. Decision Trees, the least expensive algorithm could provide the accept accuracy. This may suggest that the Case II dataset are well balanced and easily to be classified.

Looking at another optimized target precision, KNN gives the best output in Case I and Boosting gives the best output in Case II. Depends on the targeted parameter, the further feature

engineering will be needed as accuracy and precision are set as target optimized parameters in this study.





Case I: Precision			Case II: Precision		
Decision Trees	0	0.85	Decision Trees	0	0.64
	1	0.19		1	0.84
Neural Network	0	0.85	Neural Network	0	0.78
	1	0.48		1	0.75
KNN	0	0.85	KNN	0	0.66
	1	0.67		1	0.82
SVM	0	0.85	SVM	0	0.77
	1	1		1	0.74
Boosting	0	0.85	Boosting	0	0.67
	1	1		1	0.83

References

[1] https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

[2] https://www.quora.com/What-is-the-ReLU-layer-in-CNN

[3] Cortez, P., et.al, (2009). Modeling wine preferences by data mining from physicochemical properties. Decision Support Systems, 47(4), 547-553.