Machine Learning, Assignment 1 Report

Seyed Aghigh

Gatech Email: saghigh3@gatech.edu

1 Introduction

In this assignment, we want to explore and compare the outcome of the several supervised classification machine learning algorithms on two dataset. To this end, for each dataset the following steps have been taken.

(i) Preprocessing and feature selection (if needed) on each dataset.

(ii) Naive model for establishing base-line and final model selection after parameter tuning for following models;

(a) **Decision Tree**
(b) **Support Vector Machine** (**SVM**)
(c) **MultiLayered Perceptron** (**MLP**)
(d) **Adaptive Boosting** (**AdaBoost**)
(e) **k-Nearest Neighbor** (**kNN**)

2 Materials and Methods

2.1 Code Accessibility

All implemented functions are available in the following link https://drive.google.com/drive/folders/1jdwMrAzHbmY5tSS8nEPe08Urn7tGbyRf?usp=sharing

2.2 Data Selection

Two dataset has been selected; **Bank data** and **Wine data**. Bank Personal Loan Modeling [1], for simplicity ***bank data***, is a highly imbalanced supervised classification data which includes 5000 datapoints and 14 continuous and categorical variables. This data contains the customer demographic information (age, income, etc.), the customer's association with the bank (mortgage, securities account, etc.) and a dummy variable Personal loan which indicates whether the customer accepts the loan or not. The ***objective* is to come up with a model to make prediction of the likelihood of a liability customer accepting *Personal loan***. This a binary supervised classification problem in which the prediction should be made on the basis of the Personal Loan column.

White Wine Quality, for short ***Wine data*** [2], is a highly imbalanced supervised multi classification data which includes 4898 datapoints and 12 continuous features. This data contains the wine chemical characteristics as the predictive features and the quality column **Quality** which is a multi class categorical variable. The ***objective* is to correctly classify the *Quality* of given white wine**. Although in the reference website is written there are 10 classes, in the actual data, there are only 7 classes. This a multi-labeled classification problem in which the prediction should be made on the bases on the Quality column. To avoid the report complication, the summary of the both data are available in supplementary part 5.1.

2.3 Performance Metric Selection and Learning Curves

For **Bank data**, due to highly imbalance nature of dataset the **F1-Macro** performance metric is selected, which results in bigger penalization, if the model does not perform well with minority class. additionally, it is equally important for any Bank to know whether a customer has a potential to accept a loan or not, this way Bank can gauge its resources accordingly.

For **Wine data** again due its highly imbalance nature of the dataset, and forcing model to do well on minority classes, the **F1-Macro** performance metric is selected. It is also equally important to distinguish poor quality wine versus normal and tasty wine.

[1] Bank Personal Loan Modeling Link
[2] White Wine Quality

As a result, In this report all **Learning and Cross Validation Curves** are based on model performance on F1-Macro score plotted on y-axis against training sample sizes and hyper parameters on x-axis

2.4 Hyper-parameter Tuning

After data preprocessing, the naive algorithm for each classification method with default parameters are used to establish a base line. Next, hyper-parameter tuning step is performed in an attempt to achieve a higher accurate classification algorithm. For each model, the effect of hyper-parameters are explored and visualized individually using cross validation score. To this end, at each run, one of the hyper-parameter is explored with various possible values using **K-Fold Cross Validation** method while the other hyper-parameters are set to their default. This method helps us to see the effect of each hyper-parameter by itself on model performance. At each step, an optimized range/value is establish for each hyper-parameters.

Even though, Many hyper parameters are explored for each algorithm, for **brevity** of the report only few hyper parameters are presented.

2.5 Final Model Selection

At the end, final model is derived by extensive ***Grid Search*** over the derived optimized ranges using **K-Fold Cross Validation** to further investigate the hyper-parameters' trade-off simultaneously.

3 Result and Discussion On Bank Data

3.1 Decision Tree on Bank Data

Decision Tree, for short DT, is a non-linear tree-based classifier model that the split decisions are taken based on information gained (entropy) or impurity (gini) [3]. Using a default parameters a naive DT is firstly obtained. Figure 1 and Table 1 shows naive model is over-fitted the training data since there are no pre/post pruning involved. Hyper-parameter tuning (Figure 3) shows by restricting tree's maximum depth and increasing minimum samples required at each leaf node (pre-pruning) forces the model to generalize better, other hyper parameters such as class weight is set to balance which will prioritize minority class by using the values of target class to automatically adjust weights inversely proportional to class frequencies. And maximum number of features to use at each split helped the model to generalize better and reduced over fitting. The final model performance and metrics are presented in (Figure 2 and Table 2).

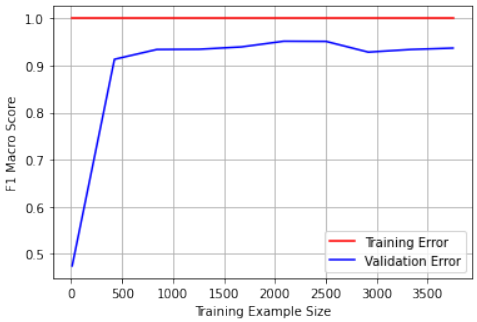


Figure 1: Naive DT model on Bank data.

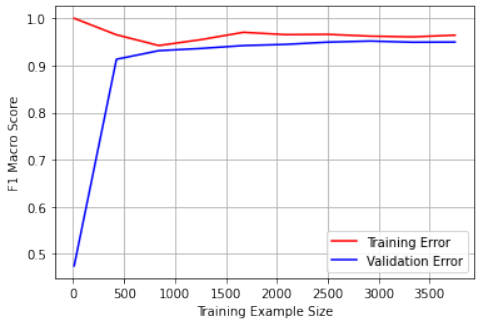


Figure 2: Final DT model on Bank data.

[3] WikiPedia Decision Tree Definition

Naive DT Performance on Train Data			
	precision	**recall**	**f1-score**
class 0	1.00	1.00	1.00
class 1	1.00	1.00	1.00
accuracy			1.00
macro avg	1.00	1.00	1.00
weighted avg	1.00	1.00	1.00

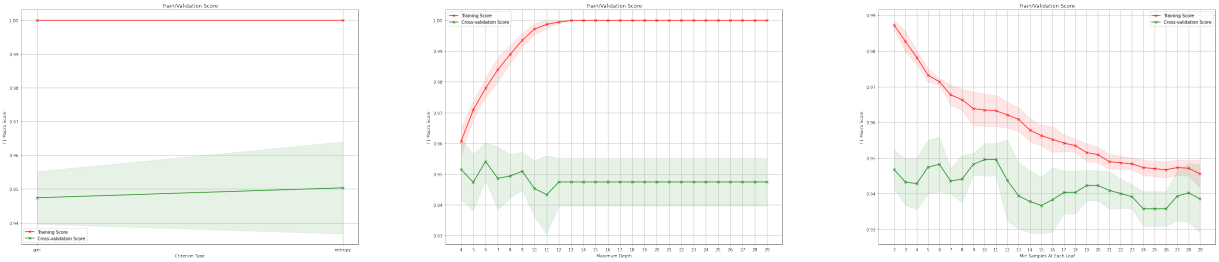
Naive DT Performance on Test Data			
	precision	**recall**	**f1-score**
class 0	0.98	0.99	0.99
class 1	0.91	0.87	0.89
accuracy			0.98
macro avg	0.95	0.93	0.94
weighted avg	0.98	0.98	0.98

Table 1: Train and Test naive DT performance summary on Bank data.

Final DT Performance on Train Data			
	precision	**recall**	**f1-score**
class 0	0.99	0.99	0.99
class 1	0.95	0.92	0.94
accuracy			0.99
macro avg	0.97	0.96	0.96
weighted avg	0.99	0.99	0.99

Final DT Performance on Test Data			
	precision	**recall**	**f1-score**
class 0	0.99	0.99	0.99
class 1	0.90	0.92	0.91
accuracy			0.98
macro avg	0.95	0.95	0.95
weighted avg	0.98	0.98	0.98

Table 2: Train and Test final DT performance summary on Bank data.



(a) Criterion hyper-parameters. **(b)** Max depth hyper-parameters. **(c)** Min split leaf hyper-parameter.

Figure 3: Cross validation for hyper-parameter tuning in DT on Bank data.

3.2 Adaptive Boosting on Bank Data

Adaptive Boosting, for short ***AdaBoost***, is an ensemble model that make prediction on the output based on N weaker estimators. Here, the outcome of the AdaBoost on the Bank data will be explored. The naive AdaBoost using Decision tree and support vector machine as a base estimator is explored. SVM failed to classify the minority class correctly therefore, an Adaboost algorithm with Decision tree base estimator is used. Figure 4 and Table 3 shows metric stats and learning curve for naive model. By looking at learning curve one can argue the naive algorithm is doing relatively well, as we have 50 estimators as default, model is not bias nor high variance. In the next step, Figure 5 shows after hyper-parameter tuning of the base estimator the model over fit training data, but still generalizes well. Figure 7 (a,b,c) shows different hyper parameters effect on F1-Macro score. Finally, tuning hyper-parameter of the AdaBoost Model (Figure 7) (d,e,f) help the overall model performance an inch higher. For instance **decreasing learning rate and increasing n estimators** help the model to generalize better. Finally, the final model is derived based on the previous hyper-parameter tuning (Figure 6 and Table 2).

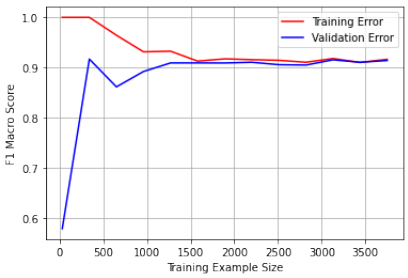


Figure 4: Naive AdaBoost model on Bank data.

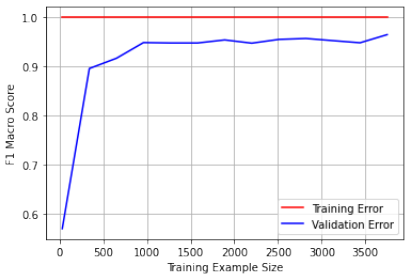


Figure 5: AdaBoost with Optimize Base Estimator model on Bank data.

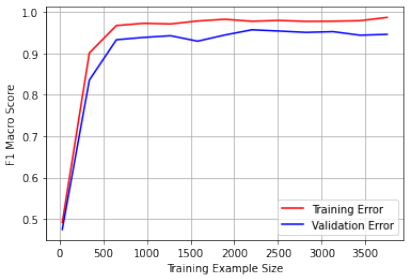


Figure 6: AdaBoost with Optimize Base Estimator and Optimize Hyper-parameters.

Naive AdaBoost Performance on Train Data			
	precision	**recall**	**f1-score**
class 0	0.99	0.98	0.99
class 1	0.79	0.92	0.85
accuracy			0.97
macro avg	0.89	0.95	0.92
weighted avg	0.98	0.97	0.97

Naive AdaBoost Performance on Test Data			
	precision	**recall**	**f1-score**
class 0	0.99	0.98	0.98
class 1	0.78	0.91	0.84
accuracy			0.97
macro avg	0.89	0.94	0.91
weighted avg	0.97	0.97	0.97

Table 3: Train and Test naive AdaBoost performance summary on Bank data.

AdaBoost with Optimize Base on Train Data			
	precision	**recall**	**f1-score**
class 0	1.00	1.00	1.00
class 1	1.00	1.00	1.00
accuracy			1.00
macro avg	1.00	1.00	1.00
weighted avg	1.00	1.00	1.00

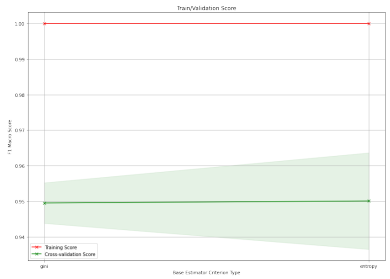
AdaBoost with Optimize Base on Test Data			
	precision	**recall**	**f1-score**
class 0	1.00	0.99	0.99
class 1	0.92	0.96	0.94
accuracy			0.99
macro avg	0.96	0.97	0.96
weighted avg	0.99	0.99	0.99

Table 4: AdaBoost with optimize base estimator on Train and Test on Bank data.

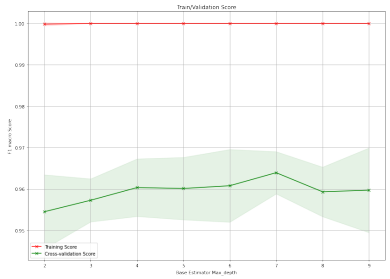
Final AdaBoost Performance on Train Data			
	precision	**recall**	**f1-score**
class 0	1.00	1.00	1.00
class 1	0.96	0.99	0.98
accuracy			1.00
macro avg	0.98	0.99	0.9999
weighted avg	1.00	1.00	1.00

Final AdaBoost Performance on Test Data			
	precision	**recall**	**f1-score**
class 0	0.99	0.99	0.99
class 1	.88	0.92	0.90
accuracy			0.98
macro avg	. 0.94	0.95	0.95
weighted avg	0.98	0.98	0.98

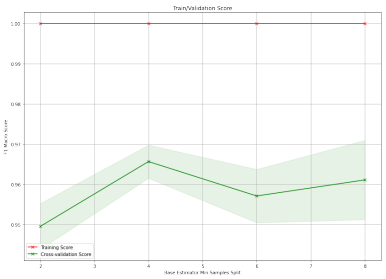
Table 5: Final AdaBoost performance on Train and Test on Bank data.



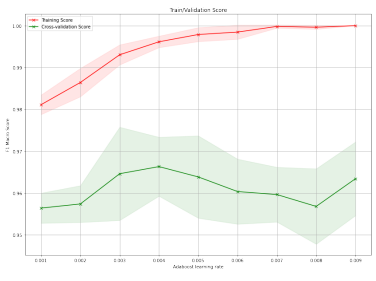
(a) Criterion DT base estimator hyper-parameter.



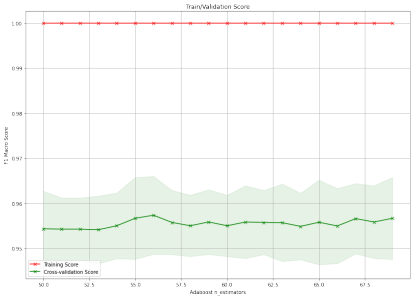
(b) Max depth DT base estimator hyper-parameter.



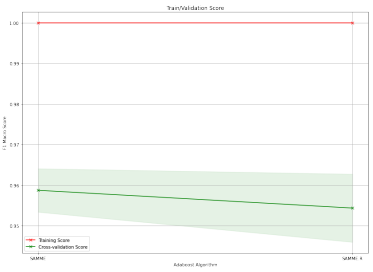
(c) Min sample split DT base estimator hyper-parameter.



(d) Learning rate AdaBoost hyper-parameter.



(e) Number of estimators AdaBoost hyper-parameter.



(f) Algorithm AdaBoost hyper-parameter.

Figure 7: Cross validation for hyper-parameter tuning in both DT base estimator and AdaBoost individually on Bank data.

3.3 k-Nearest Neighbor on Bank Data

k-Nearest Neighbors, for short kNN, is a non-parametric non-linear classifier in which the prediction is based on the k closest point. The naive kNN firstly is obtained Figure 8 and Table 6 shows the model is neither bias nor high variance, however the overall performance of model using default parameters on training and test set is low and train and test score are diverging. During hyper parameters tuning, Figure 10 (a) shows by decreasing number of k neighbors from default 5 to 3 the performance of model increases drastically on training and validation set. Also changing the weights parameter from uniform to distance helps to reduce the effect of points which are at located at further distance and closer neighbors of a query point will have a greater impact. Figure 9 and Table 7 shows after Hyper-parameter tuning the final model performance boosted from 68% to 88% and training and test score are converging nicely as sample size increases.

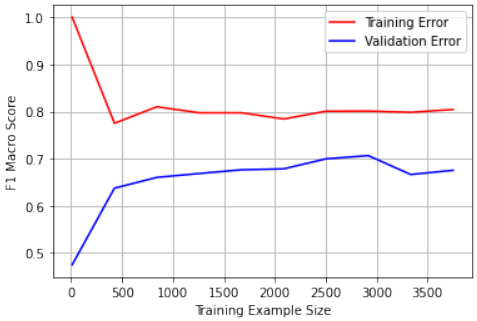


Figure 8: Naive kNN model on Bank data.

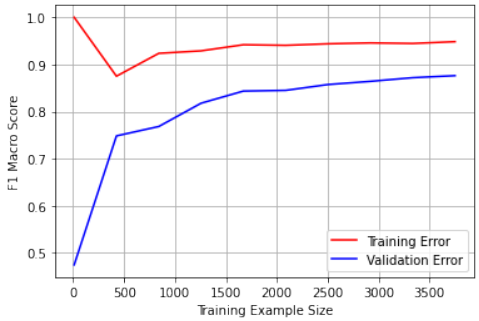


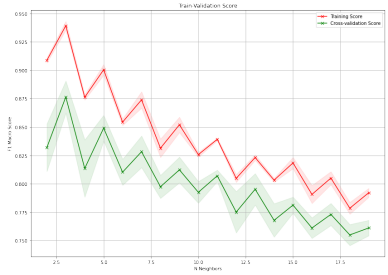
Figure 9: Final kNN model on Bank data.

Naive kNN Performance on Train Data			
	precision	**recall**	**f1-score**
class 0	0.99	0.95	0.97
class 1	0.53	0.81	0.64
accuracy			0.94
macro avg	0.76	0.88	0.80
weighted avg	0.96	0.94	0.95
Naive kNN Performance on Test Data			
	precision	**recall**	**f1-score**
class 0	0.97	0.93	0.95
class 1	0.32	0.54	0.40
accuracy			0.91
macro avg	0.64	0.74	0.68
weighted avg	0.9	0.91	0.92

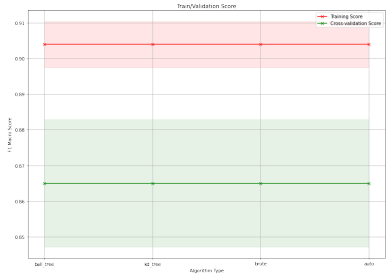
Table 6: Train and Test naive kNN performance summary on Bank data.

Final kNN Performance on Train Data			
	precision	**recall**	**f1-score**
class 0	0.98	1.00	0.99
class 1	1.00	0.83	0.91
accuracy			0.98
macro avg	0.99	0.91	0.95
weighted avg	0.98	0.98	0.98
Final kNN Performance on Test Data			
	precision	**recall**	**f1-score**
class 0	0.97	0.99	0.98
class 1	0.92	0.67	0.77
accuracy			0.96
macro avg	0.94	0.94	0.88
weighted avg	0.96	0.96	0.96

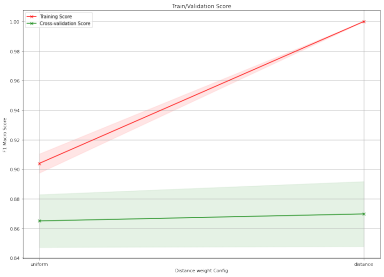
Table 7: Train and Test final kNN performance summary on Bank data.



(a) Number of n-neighbors hyper-parameter.



(b) Algorithm hyper-parameter



(c) Weights hyper-parameter.

Figure 10: Cross validation for hyper-parameter tuning in kNN on Bank data.

3.4 Support Vector Machine on Bank Data

Support vector Machine, for short SVM, is a linear supervised model that can be used for classification and regression. The naive SVM firstly is obtained Figure 11 and Table 8 shows SVM with default parameters are not able to train and generalize well as sample size increases the overall train and test score is around 55% which a little bit better than random guess. As far as, selecting the best kernel, after running Grid Search over different Kernels, poly with degree 5 and Gaussian Kernel outperformed Linear and Sigmoid kernels, with Gaussian kernel generalizing better than Poly (Figure 13a). After exploring hyper-parameters, it turns out C parameter which controls the width of margin of safety between the the decision boundary and classes, or in other words, it controls the degree of model regularization, have the highest impact on performance of model. The default value is 1 which is by looking at Figure 11 learning curve, one can argue the margin of safety is too wide to let algorithm learn and generalize properly. By increasing C to 5.4 the model train and generalizes much more effectively Figure 13d. After tuning Class Weight and Gamma parameters Figure 13 the final model is derived. Performance of model boosted by more than 40%. (Figure 12 and Table 9).

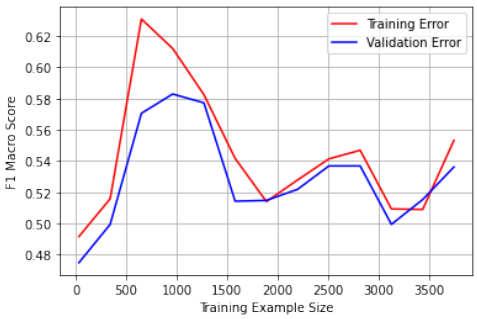


Figure 11: Naive SVM model on Bank data.

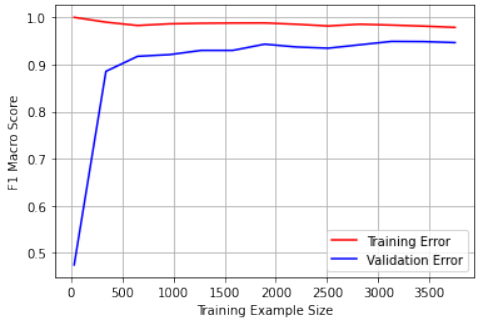


Figure 12: Final SVM model on Bank data.

Naive SVM Performance on Train Data			
	precision	**recall**	**f1-score**
class 0	1.00	0.91	0.95
class 1	0.09	0.74	0.15
accuracy			0.91
macro avg	0.54	0.82	0.55
weighted avg	0.99	0.91	0.94

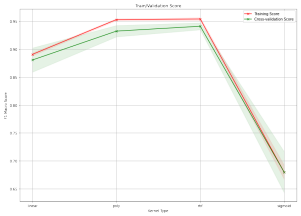
Naive SVM Performance on Test Data			
	precision	**recall**	**f1-score**
class 0	1.00	0.91	0.95
class 1	0.07	0.67	0.12
accuracy			0.91
macro avg	0.53	0.79	0.54
weighted avg	0.99	0.91	0.94

Table 8: Train and Test naive SVM performance summary on Bank data.

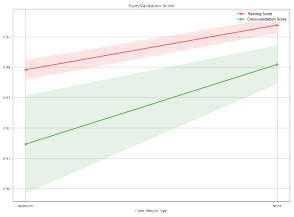
Final SVM Performance on Train Data			
	precision	**recall**	**f1-score**
class 0	0.99	1.00	1.00
class 1	0.99	0.93	0.96
accuracy			0.99
macro avg	0.99	0.96	0.98
weighted avg	0.99	0.99	0.99

Final SVM Performance on Test Data			
	precision	**recall**	**f1-score**
class 0	0.99	0.99	0.99
class 1			
accuracy			0.98
macro avg	0.95	0.94	0.95
weighted avg	0.98	0.98	0.98

Table 9: Train and Test final SVM performance summary on Bank data.

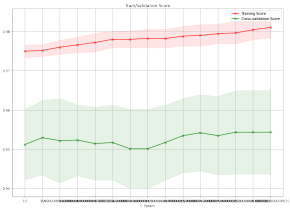


(a) Kernel hyper-parameter; "linear", "rbf", "poly", "sigmoide"



(b) Class weight hyper-parameter

(c) Gamma hyper-parameter.

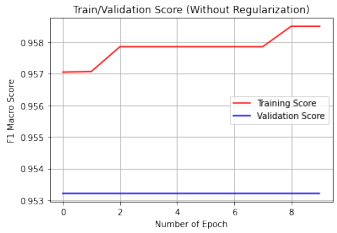


(d) Parameter C hyper-parameter.

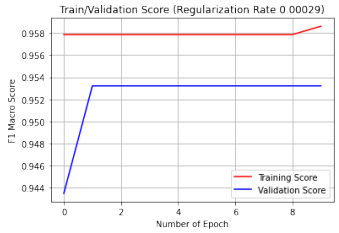
Figure 13: Cross validation for hyper-parameter tuning in SVM on Bank data.

3.5 Multilayer Percptron Bank Data

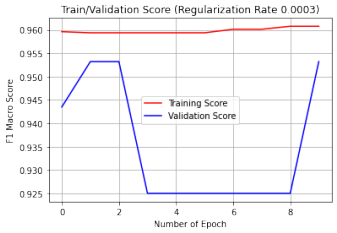
Multilayer Perceptron, for short MLP, is neural network classification model that can be used for linear and non-linear model. In this experiment, a simple MLP with one hidden layer consisting of 10 nodes with Sigmoid activation functions is considered. And effect of different hyper-parameters such as Learning Rate, Regularization Rate with different Optimizer (Adam and SGD) are explored. And result are visualized based on F1 Macro score against number of Epoch Figure 14. is applied to achieve the most out-performing model(Figure 14). During Hyper-Parameters tuning, Adam optimizer with learning rate set 0.003 and with regularization tend to outperform other setting with having average of 95% F1 Macro score on Test set. By looking at Figure 14a without regularization the model tends to over fit and there is gap between train and test curve, by adding regularization Figure 14b, The gap between training and test curve closes.



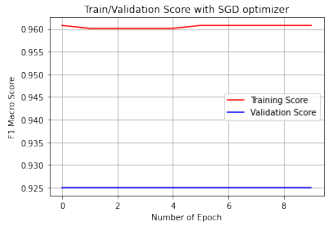
(a) MLP with no regularization



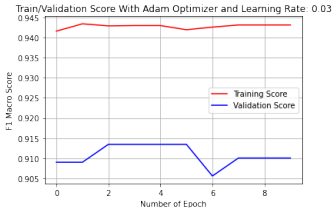
(b) MLP with regularization set to 0.0029



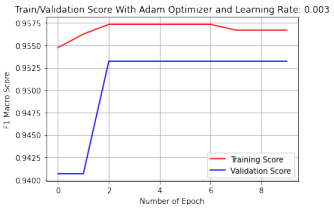
(c) MLP with regularization set to 0.003



(d) MLP with stochastic gradient descent



(e) MLP with Adam and learning rate of 0.03



(f) MLP with Adam and learning rate of 0.003

Figure 14: Cross validation for hyper-parameter tuning in MLP on Bank data.

4 Result and Discussion on Wine Data

4.1 Decision Tree on Wine Data

For ***Decision Tree*** (DT) on wine data a naive algorithm is developed. Figure 15 and Table 10 shows the model is unable to predict class 3 and 9 at all. Often This could cause by having redundant features in our dataset. Therefore, an analysis of variances (ANOVA) is used to capture the importance of features based on p-value metric. Figure 20 shows -log10 of p values of each feature with respect to target variable. Out of 11 features the most 6 significant features are selected for further analysis. Selected features are as follow; "volatile acidity", "residual sugar", "chlorides","total sulfur dioxide", "density", "alcohol". Another decision tree is fitted on **Selected Features** and result are shown in Figure 20 and table Table 11. Now, the model predict all classes and average F1 macro increased by 3% and average precision score increased by 13%. Few hyper-parameters tuning did not help the model performance and graphs are skipped for brevity of report. Figure 19 shows the model is over-fitted to training data, since the test curve trend is upward as a last resort, SMOTE technique is utilized for over sampling, Figure 17 shows the over sampling did help the learning curve noticeably. As a final step, post pruning technique is used to explore the possibility of closing gap between training and test curves. Figure 21 and Figure 23 depicts the level of impurity and number of nodes/tree depth with respect to hyper-parameter ccpalpha. As ccpalpha increases more nodes and tree's length are pruned leading to higher impurity at leaves. Figure 22 shows train/test score based on ccpalpha parameter from this graph, it can be concluded the current decision tree is almost at its maximum capacity for post pruning. Regardless to explore the effect of post pruning the alpha value of 0.0002 is selected. The selection leads to decrease tree's depth from 29 to 27 and total number of nodes decreases from 1186 to 901. The overall F1 Macro score decreased by 3% on Test set. (Table 12)

Naive DT on Test Data.			
	precision	**recall**	**f1-score**
class 3	0.00	0.00	0.00
class 4	0.26	0.27	0.26
class 5	0.61	0.61	0.61
class 6	0.65	0.66	0.65
class 7	0.60	0.56	0.58
class 8	0.39	0.43	0.41
class 9	0.00	0.00	0.00
accuracy			0.60
macro avg	0.36	0.36	0.36
weighted avg	0.60	0.60	0.60

Table 10: Naive DT without feature selection.

DT with Selected Features on Test on Bank data.			
	precision	**recall**	**f1-score**
class 3	0.17	0.20	0.18
class 4	0.25	0.34	0.29
class 5	0.62	0.61	0.62
class 6	0.63	0.59	0.61
class 7	0.49	0.53	0.51
class 8	0.37	0.41	0.39
class 9	1.00	0.00	0.00
accuracy			0.57
macro avg	0.50	0.38	0.37
weighted avg	0.58	0.57	.057

Table 11: DT with selected features on Test on Bank data.

Post-pruned DT with Selected Features on Test Data.			
	precision	**recall**	**f1-score**
class 3	0.20	0.06	0.09
class 4	0.27	0.23	0.25
class 5	0.60	0.59	0.60
class 6	0.58	0.62	0.60
class 7	0.50	0.49	0.50
class 8	0.34	0.35	0.34
class 9	0.00	1.00	0.00
accuracy			0.55
macro avg	0.36	0.48	0.34
weighted avg	0.55	0.55	0.55

Table 12: DT with selected features with post-pruning on Bank data.

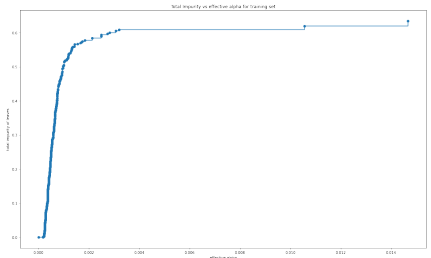


Figure 21: Impurity - ccpalpha ratio for DT on Wine data.

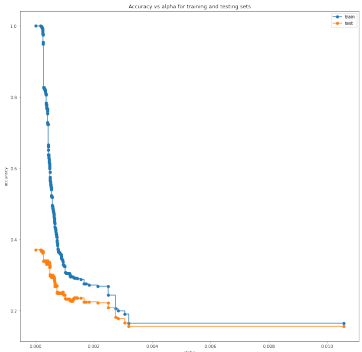


Figure 22: Train/Test Score vs ccpalpha for DT on Wine data.

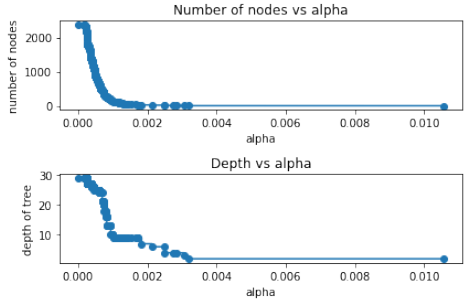


Figure 23: Number of Nodes & Depth - ccpalpha DT on Wine data.

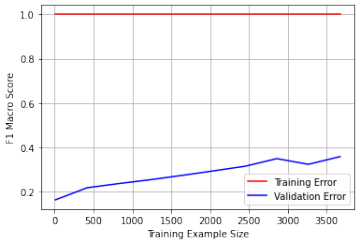


Figure 15: Naive DT with default Hyper-parameters on Wine data.

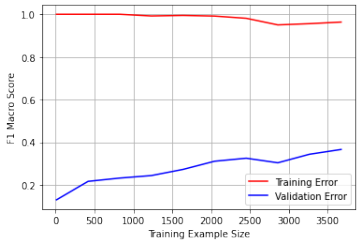


Figure 16: DT after hyper-parameter tuning on Wine data.

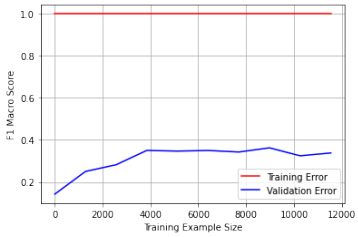


Figure 17: DT fitted on over-sampled data using SMOTE method on Wine data.

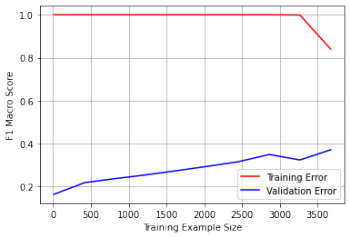


Figure 18: DT after post-pruning on Wine data

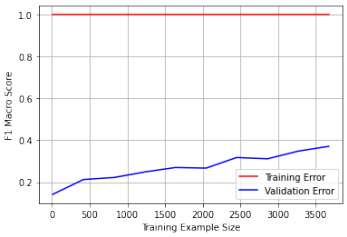


Figure 19: DT After Feature Selection.

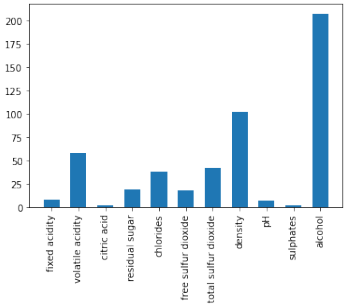


Figure 20: -log() of P values for all features on Wine data.

4.2 Adaptive Boosting on Wine Data

A naive ***Adaptive Boosting*** (AdaBoost) algorithm with Decision tree base estimator is fitted to selected features. Figure 24 and Table 13 shows how poorly the naive algorithm learns and generalizes. We'll begin with base estimator hyper-parameters tuning. Figure 26 shows series of Cross Validation hyper parameters tuning. With maximum depth being the most performance booster of the model on training and test set. (Figure 26b). Further more, minimum samples split required at each split help performance metrics. AdaBoost learning rate and number of estimators hyper-parameters are explored individually first and through another **Grid Search** the both parameters are explored together to find the perfect trade off. Figure 25 and Table 14 shows the final model learning curve and performance metrics respectively. The average F1 Macro score increased from 15% to 40%. Test error curve trend is increasing as sample size increases. This suggests if we could collect more quality data we can improve the generalization of the model.

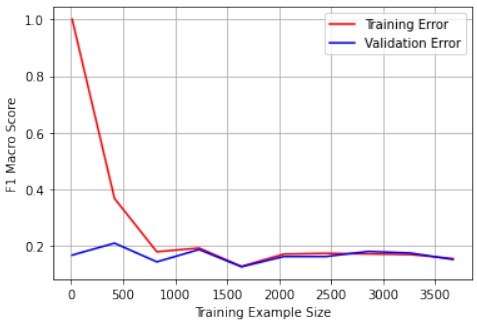


Figure 24: Naive AdaBoost model on Test for Wine data.

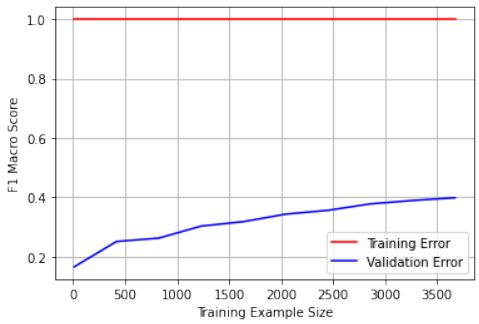


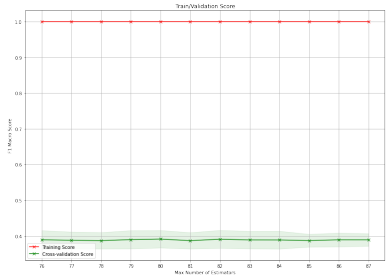
Figure 25: Final AdaBoost model on Test for Wine data.

Naive AdaBoost Performance on Test Data			
	precision	**recall**	**f1-score**
class 3	0.40	0.02	0.04
class 4	0.00	1.00	0.00
class 5	0.55	0.46	0.50
class 6	0.60	0.48	0.53
class 7	0.00	1.00	0.00
class 8	0.00	1.00	0.00
class 9	0.00	1.00	0.00
accuracy			0.44
macro avg	0.22	0.71	0.15
weighted avg	0.57	0.44	0.48

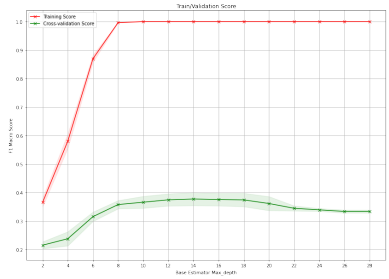
Table 13: Test naive AdaBoost performance on Wine data.

Final AdaBoost Test Data.			
	precision	**recall**	**f1-score**
class 3	0.00	0.00	0.00
class 4	0.62	0.20	0.30
class 5	0.69	0.68	0.69
class 6	0.64	0.81	0.72
class 7	0.73	0.47	0.57
class 8	0.89	0.36	0.52
class 9	1.00	0.00	0.00
accuracy			0.67
macro avg	0.65	0.36	0.40
weighted avg	0.68	0.67	0.66

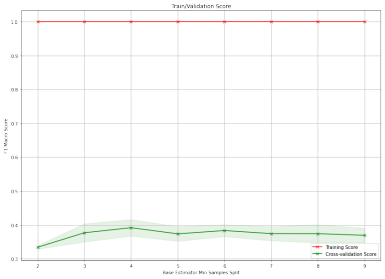
Table 14: Test final AdaBoost performance on Wine data.



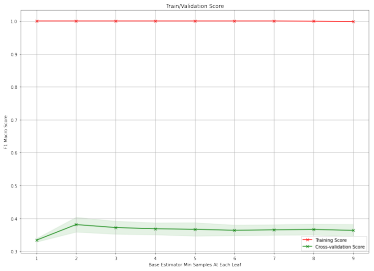
(a) Max number of estimator DT base estimator hyper-parameter.



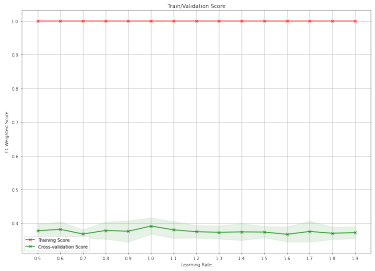
(b) Max depth of base estimator hyper-parameter.



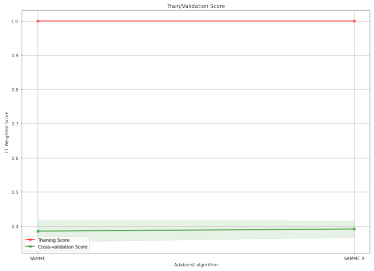
(c) Min sample base estimator hyper-parameter.



(d) Min sample leaf base estimator hyper-parameter.



(e) Learning rate AdaBoost hyper-parameter.



(f) Algorithm AdaBoost hyper-parameter.

Figure 26: Cross validation for hyper-parameter tuning in both DT as the base estimator and the AdaBoost by itself on Wine data.

4.3 k-Nearest Neighbors on Wine Data

For ***k-Nearest Neighbors*** (kNN) initially three individual naive model are developed over unprocessed data, standardized data and features selected/standardized data and learning curve (Figure 27, Figure 28, Figure 29) plus classification reports are generated. It was established that feature selection negatively impact the overall performance metrics, therefore standardized version of data is selected for further optimization. Among hyper-parameters decreasing k neighbors from default value 5 to 3 and changing class weight from uniform to distance

help the generalization the most. This could be due to high noise in original data, by giving weights to the distance of k nearest neighbors and decreasing the neighbors the model can learn and generalize better by distinguishing noise from real data. With optimized kNN model, our model achieved **45%** F1 Macro score (Figure 31, Table 15).

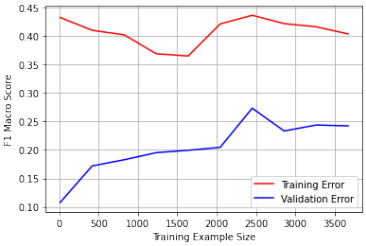


Figure 27: Learning curve for kNN on unprocessed data on Wine data.

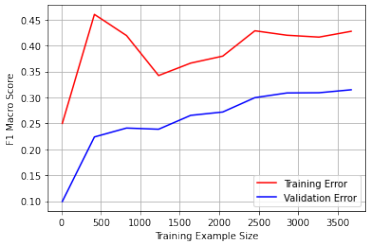


Figure 28: Learning curve for kNN on processed data on Wine data.

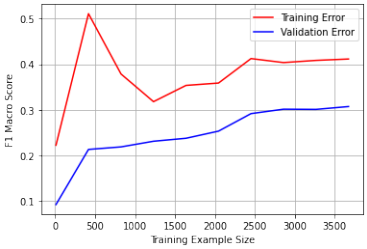
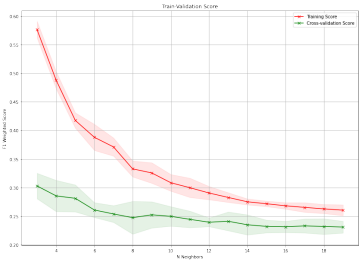
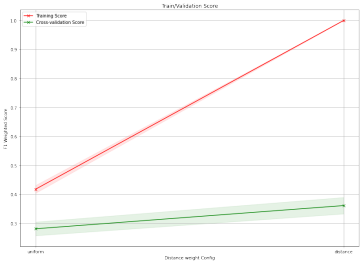


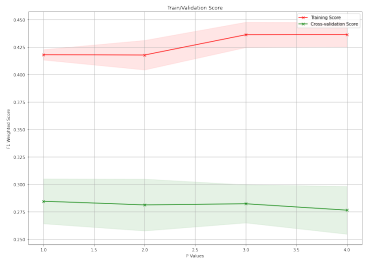
Figure 29: Learning curve on the process data on the selected features on Wine data.



(a) Number of neighbors hyper-parameters.



(b) Distance weight hyper-parameter.



(c) P metric hyper-parameter.

Figure 30: Cross validation for hyper-parameter tuning in kNN on Wine data.

Final kNN on Test Data.			
	precision	**recall**	**f1-score**
class 3	1.00	0.20	0.33
class 4	0.64	0.17	0.27
class 5	0.66	0.68	0.67
class 6	0.67	0.74	0.70
class 7	0.63	0.60	0.62
class 8	0.75	0.41	0.53
class 9	1.00	0.00	0.00
accuracy			0.66
macro avg	0.76	0.40	0.45
weighted avg	0.66	0.66	0.65

Table 15: Final kNN on Test set for Wine data.

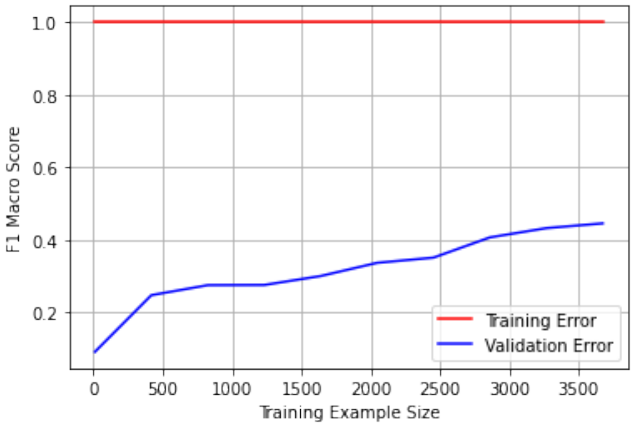


Figure 31: Final kNN learning curve on Test set for Wine data.

4.4 Support Vector Machine on Wine Data

For Support Vector Machine (SVM) the initial approach was very similar to kNN, developed three naive algorithm using unprocessed, Standardized and selected feature, the learning curve (Figure 32, Figure 33, Figure 34) and classification report are generated. From reports, standardized data is selected for further investigation. When exploring for the best kernel, K-Fold Grid Search result showed Poly Kernel with degree 9 is outperforming other kernels, while default RBF (Gaussian) Kernel carries higher score but misses three of classes completely (3, 8, 9). however further investigation revealed using Poly kernel with degree 9 would not miss any of the labels but result in overall lower performance. Further tuning was perform in C and gamma parameter with resulted in modest improvement. Final model was derived by doing an extensive grid search over all hyper parameters, interestingly the RBF kernel was selected along with other hyper parameters tuning, the final model misses two classes (3, 9) the most underrepresented classes in Wine dataset. However resulted in relatively higher performance (Table 16, Table 36).

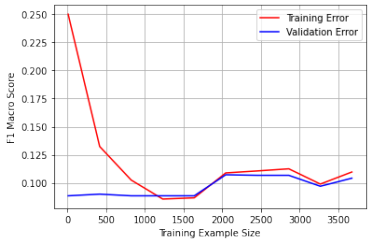


Figure 32: Learning curve for SVM on unprocessed data. Fail for 5 classes.

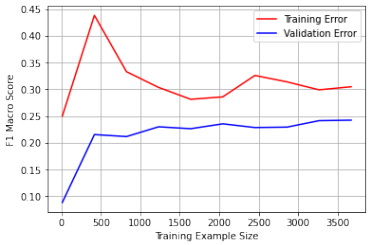


Figure 33: Learning curve for SVM on processed data. Fail for 3 classes.

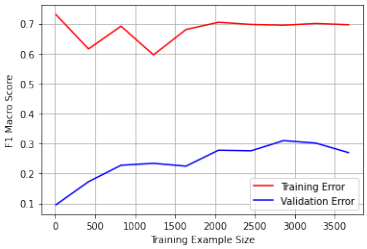
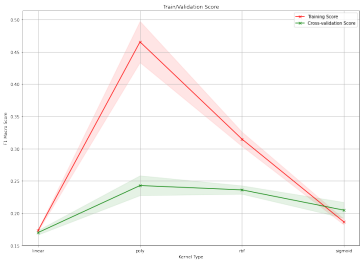
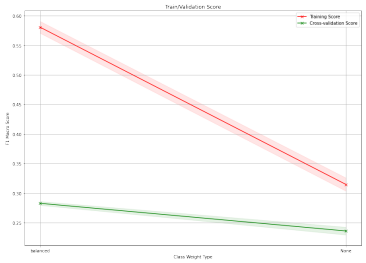


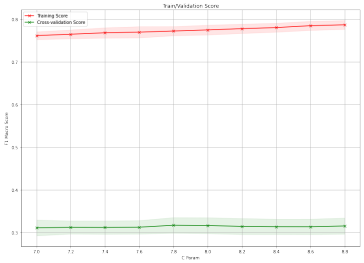
Figure 34: Learning curve on the process data on the selected kernel of ploy 9.No failing.



(a) Kernel hyper-parameters.



(b) Weight class hyper-parameter.



(c) C metric hyper-parameter.

Figure 35: Cross validation for hyper-parameter tuning in SVM.

Final SVM on Test Data.			
	precision	**recall**	**f1-score**
Class 3	0.00	0.00	0.00
Class 4	0.20	0.37	0.26
Class 5	0.55	0.60	0.57
Class 6	0.66	0.47	0.55
Class 7	0.46	0.58	0.51
Class 8	0.30	0.55	0.38
class 9	0.00	0.00	0.00
accuracy			0.52
macro avg	0.31	0.36	0.32
weighted avg	0.56	0.52	0.53

Table 16: Final SVM on the Test data.

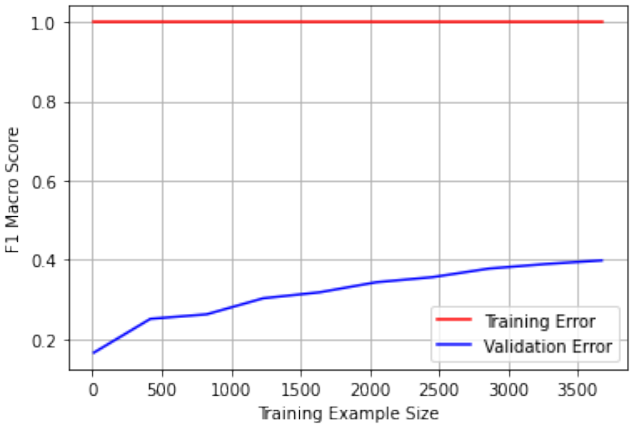
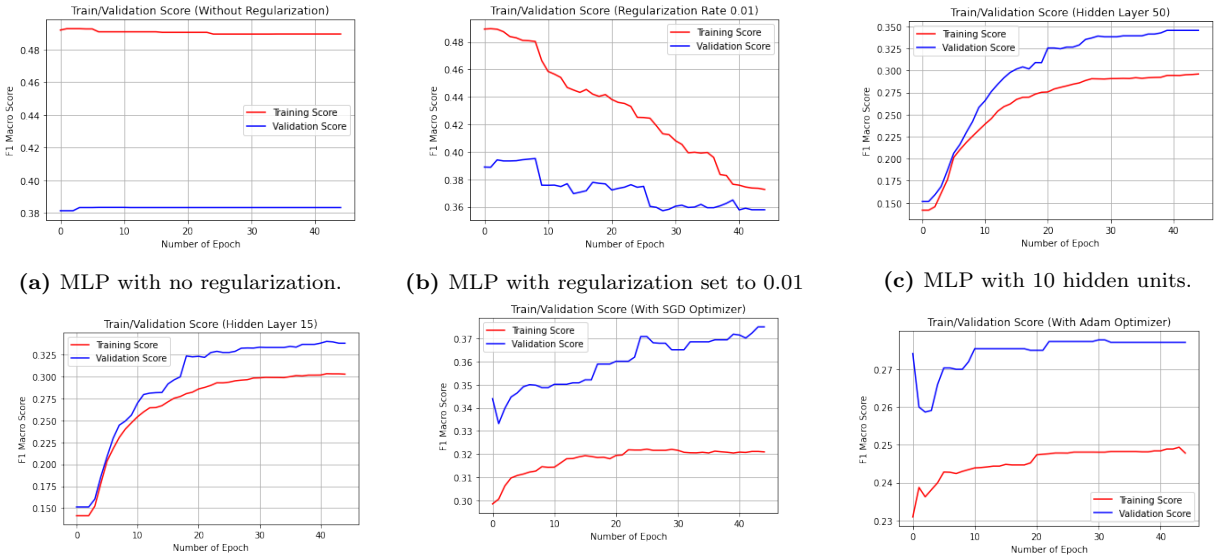


Figure 36: Final SVM learning curve on the Test data.

4.5 Multilayer Perceptron on Wine Data

In ***Multi-Layers Perceptron*** (MLP) section, a simple MLP algorithm with one hidden layer consisting of 20 nodes is generated, essentially the reason for picking simple model is the limited data and highly imbalance classes. The larger models carry more learning parameters which demands more labeled data. An interesting observation was in almost all learning curves the Validation curve was higher than training curve, meaning the model generalizes better than training. One possible reason is when model is in training mode it uses all the regularization techniques such as weight decay, dropouts, while when model is in evaluation mode this parameters are inactive leading to slightly higher validation score. Even though the model is weaker in learning process but it generalizes almost at same level as the other learning algorithms, F1 macro score averaged about 38%. Different hyper-parameters tuning are considered. Starting with no regularization rate which resulted in huge gap between, training and validation curve (Figure 37a). After applying regularization rate, the gap between two curves started to decreased as number of epoch increased (Figure 37b). However regularization term had negative impact on validation curve. Figure 37c and Figure 37d shows increasing hidden layers width (nodes), improve the learning capacity slightly, however due to data limitation the improvement is modest. Finally Figure 37e and Figure 37f shows Neural Net (NN) with Stochastic Gradient Descent (SGD) optimizer is performing better than Adam Optimizer.



(a) MLP with no regularization.

(b) MLP with regularization set to 0.01

(c) MLP with 10 hidden units.

(d) MLP with 15 hidden units.

(e) MLP with stochastic gradient descent and optimizer.

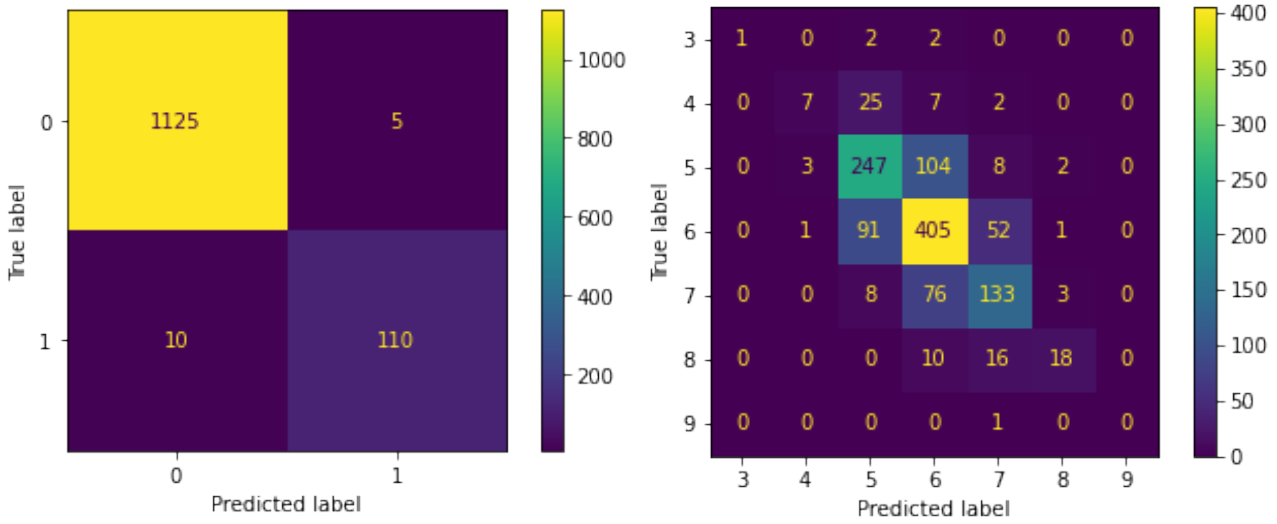
(f) MLP with Adam and optimizer.

Figure 37: Cross validation for hyper-parameter tuning in MLP on Wine data.

4.6 Conclusion

The winner model on Bank data is AdaBoost algorithm with optimized base estimator of decision tree. The Average F1 macro score is 96%. Figure 38a shows the confusion matrix for this model. The bank data consist of continuous and categorical variables, decision tree base models are best to try on these type dataset for classification purposes. They require the minimum preprocessing on these type of dataset. AdaBoost uses n weaker decision trees to predict on new queries and combine the answers of theses estimator to get more accurate estimates.

The winner model for Wine dataset is kNN algorithm with overall F1 macro score of 45%. This was very challenging dataset as it had much more normal quality wines than good and bad ones. After ANOVA analysis, it was found almost half of features are insignificant and they are just adding redundancy. They were very few amount of data for minority class and almost all over sampling techniques failed to generate quality data. But overall it came with its own perks. I learned many over/under sampling techniques, tried many different hyper tuning techniques and got very excited to move the performance metric up to 45%. In particular, due to noisy data This kNN achieved higher score by decreasing the k neighbors, plus giving weights to the distance of k nearest neighbors that leads to distinguishing noise from real data. Figure 38b shows confusion matrix for generated using the kNN model. One can argue model is performing much better on classes with higher amount of data in compare to minority classes.



(a) Confusion matrix for Bank data.

(b) Confusion matrix for Wine data.

Figure 38: Confusion matrix for both Bank and Wine data.

5 Supplementary

5.1 Bank Data Description

The summary of the features in the data is illustrated in the Table 17.

Features	count	mean	std	min	max
ID	5000.0	2500.50	1443.52	1.0	5000.0
Age	5000.0	45.34	11.47	23.0	67.0
Experience	5000.0	20.11	11.47	-3.0	43.0
Income	5000.0	73.78	46.04	8.0	224.0
ZIP Code	5000.0	93152.50	2121.85	9307.0	96651.0
Family	5000.0	2.40	1.15	1.0	4.0
CCAvg	5000.0	1.94	1.75	0.0	10.0
Education	5000.0	1.89	0.84	1.0	3.0
Mortgage	5000.0	56.50	101.71	0.0	635.0
Personal Loan	5000.0	0.096	0.29	0.0	1.0
Securities Account	5000.0	0.10	0.36	0.0	1.0
CD Account	5000.0	0.06	0.24	0.0	1.0
Online	5000.0	0.60	0.50	0.0	1.0
CreditCard	5000.0	0.30	0.46	0.0	1.0

Table 17: Bank data; 5000 customers and 14 features. For each feature, statistics count, mean, standard deviation (std), min, and max is calculated.

In the preprocessing step, following tasks have been explored.

- There is no missing value in the dataset.
- Feature **ID** is dropped; it does not give us any information for the prediction.
- Feature **ZIP Code** is dropped; it is a categorical variable and one-hot coding will lead to a very high dimensional data, and therefore is not efficient.
- Feature **Age** is dropped; it is highly correlated with the **Experience** feature (0.9939), and thus does not give us more information (redundant feature).
- Try to make the data balance using either **up-sampling** or **down-sampling**; neither of those methods resulted in a better prediction model in any aforementioned algorithms.

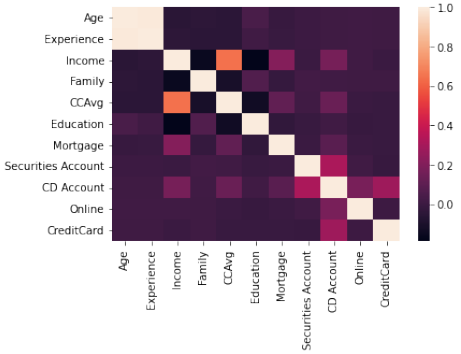


Figure 39: Correlation heatmap of the features in Bank data.

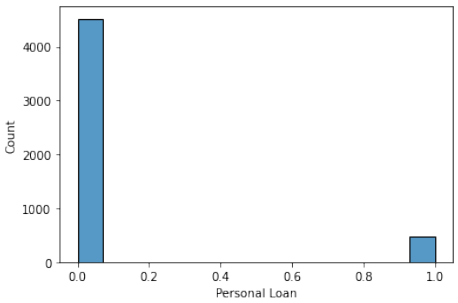


Figure 40: Histogram of the Personal loan feature in Bank data.

As it seen from the Figure 40, there are only 9.6% of the customers accepted the personal load that was offered to them which means that the data is so unbalanced. Thus, in order to come up with a good model, one of the early try was to make it more balance via either **up-sampling** or/and **down-sampling**. However, neither of the efforts leaded to a better prediction model. To avoid of the complication of the assignment report, those result has not explained here.

5.2 Wine Data Description

The summary of the features in the data is illustrated in the Table 18. This data is highly imbalance meaning that the number of datapoints fall in each class is not equal.

Features	count	mean	std	min	max
fixed acidity	4898.00	6.85	0.84	3.80	14.20
volatile acidity	4898.00	0.28	0.10	0.08	1.10
citric acid	4898.00	0.33	0.12	0.00	1.66
residual sugar	4898.00	6.39	5.07	0.60	65.80
chlorides	4898.00	0.05	0.2	0.009	0.35
free sulfur dioxide	4898.00	35.31	17.01	2.00	289.00
total sulfur dioxide	4898.00	138.36	45.50	9.00	440.00
density	4898.00	0.99	0.003	0.99	1.04
pH	4898.00	3.19	0.15	2.72	3.82
sulphates	4898.00	0.49	0.11	0.22	1.08
alcohol	4898.00	1051	1.23	8.00	14.20
quality	4898.00	5.88	0.88	3.00	9.00

Table 18: Wine data; 4898 wines and 12 features. For each feature, statistics count, mean, standard deviation (std), min, and max is calculated.

Figure 41 shows the heatmap of the features and Figure 42 shows the histogram of the classes. As it seen, this data is highly imbalance, and class 6 and 9 has the most and the least number of datapoints

- There is no missing value in the dataset.
- Try to make the data balance using either **up-sampling** or **down-sampling**; neither of those methods resulted in a better prediction model in any aforementioned algorithms.
- Try to perform feature selection and if it leads to better model performance.

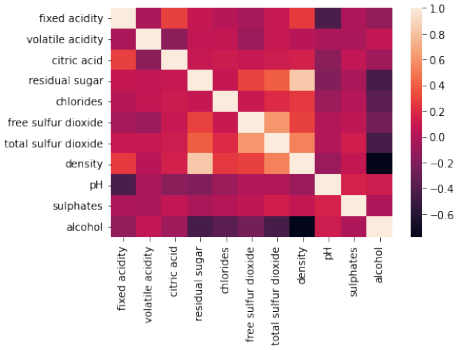


Figure 41: Correlation heatmap of the features in Wine data.

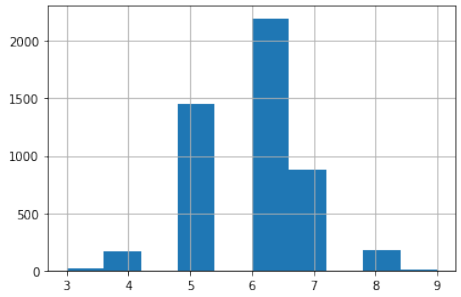


Figure 42: Histogram of the wine Quality feature in Wine data.