Classification Problem Description

Problem 1:

The first classification problem is Spambase. It is a classification dataset containing 4601 emails sent to HP (Hewlett-Packard). Spambase contains numeric 57 features for each email and a binary label, 1 for spam, 0 for non-spam email. The features in this dataset are derived from the textual information in the email. I found this interesting because it presents a primitive yet efficient way for us to build a useful model, without relying on the fancy and advanced natural language processing (NLP) techniques that are commonplace these days. This problem shows us that even without the actual textual information in an email, we can still build a classification model using these extracted features. It would be exciting to see how other real world problems can be abstracted into features and simplified using easier models to achieve similar results as its advanced counterpart. This dataset is normalized to standardize the range of values between the different features.

Problem 2:

The second classification problem I will address is the bank marketing problem. The data is taken from direct marketing campaigns (phone calls) of a Portuguese bank. The objective is to predict if the client will subscribe a term deposit. This is interesting to me because I work in the finance industry and my daily job as a data scientist is to generate leads using machine learning and business rules for my company. Typically, these problems seem easy to solve and a classification model can be built to score each customer's likelihood of conversion. However, in terms of execution, they would need to be accompanied by strong reasoning as determined by the model's selection of feature importance, as well as a decent precision, recall score, F1 and ROC area under curve score. This data suffered from a target imbalance, which led me to perform a downsampling on the 0 (negative) class so that the ratio is approximately 3:1 (12000 negative samples: 3859 positive samples).

Models

Decision Tree

DecisionTreeClassifier[1] from sklearn is used. Parameters that can be set are criterion and cost complexity pruning alpha amongst others not mentioned here for brevity. This criterion determines the probability based on the class frequencies of the training data points that reached a certain leaf 'm'. At each step, in order to branch the decision tree, the criterion is calculated for a feature to split by. If it decreases, the split is validated and will occur with the feature. Otherwise, a split must be attempted with another feature or the Decision Tree can stop at this branch. Entropy[2] is the easiest way to compute using log loss, and hence I chose this due to simplicity. Alpha is used for minimal cost-complexity pruning. As alpha increases, more of the tree is pruned, creating a decision tree that generalizes better. For the Spambase problem, I used the default parameters to build a decision tree. I set the criterion to be "entropy". Using a K-fold cross validation[3] K=5 and a learning curve (Fig 1a),

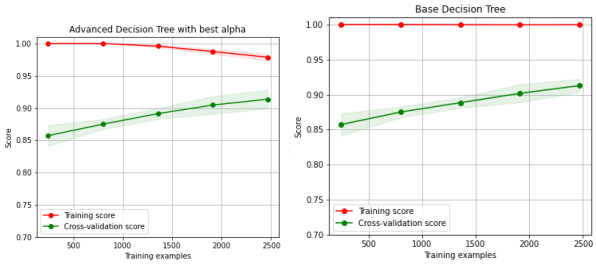

Figure 1a - [Spambase] Decision Tree - Learning Curve

the decision tree shows signs of severe overfitting, where training score is maximum and CV scores increases over time. To combat this overfitting, I used the cost complexity pruning path[4,5] methodology to find the best alpha. Fig 1b and 1c shows this methodology, and the best alpha is determined at 0.00195 with the highest metric scores.

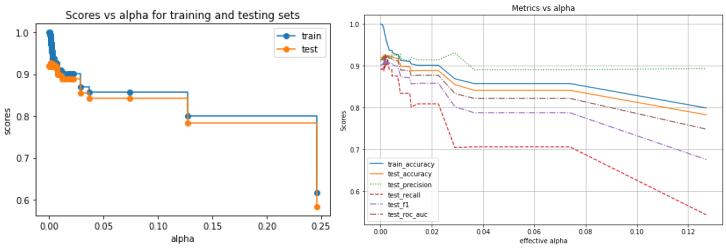

Figure 1b – [Spambase] Decision Tree – Cost Complexity Pruning

Following which, I used this to train a better (advanced) decision tree and compared the learning curve (Fig 1a) and scores (Fig 1c, 1d) from the previous model. The learning curve starts converging slowly after 1400 training examples. This shows we are heading in the right direction of controlling the overfitting, but more can be done to make the training and CV score meet. The final scores on the test set also show an improvement of AUC from 0.926 to 0.916 from adding the alpha value.

Decision Tree	Base	Base	Adv.	Adv.
Metric	**Train**	**Test**	**Train**	**Test**
Accuracy	1.00	0.920	0.969	0.926
Precision	1.00	0.913	0.948	0.901
Recall	0.999	0.893	0.972	0.924
F1	1.00	0.903	0.960	0.913
ROC AUC	1.00	0.916	0.969	0.926

Figure 1d – [Spambase] Decision Tree - Train, Test

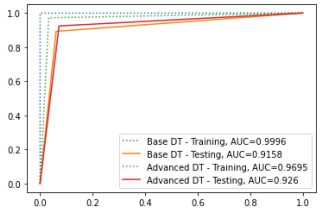

Figure 1d – [Spambase] Decision Tree - AUC ROC

For the Bank Marketing problem, I built a basic decision tree using the default parameters with the criterion to be entropy. A cross validation with K=5 is used to get a more approximate score by taking the mean of the 5 different models. I used the cost complexity pruning path to find the best alpha to prune the model and avoid overfitting. When alpha is set to zero, the tree overfits, leading to a 100% training accuracy and 84% testing accuracy (Fig 1e). In this example, setting alpha=0.00175 (Fig 1f) maximizes the testing accuracy at a point where training accuracy and test accuracy are almost similar (85%)

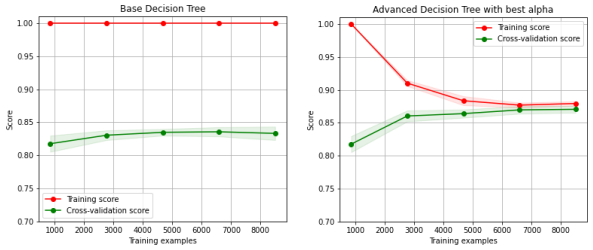

Figure 1e – [Bank] Decision Tree – Learning Curve

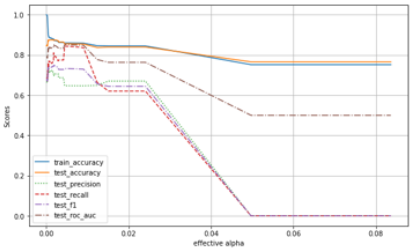

Figure 1f – [Bank] Decision Tree – Learning Curve

Subsequently, I built an advanced decision tree model using the best alpha value found and compared the learning curves and CV scores with the basic model. The training and cross-validation scores converge together as more data is added, showing that the model will not benefit from more data and the model is sufficiently trained to perform well. The basic Decision Tree had a training score much greater than the validation score, that the model probably requires more training examples in order to generalize more effectively. The effect of changing the alpha value helped to speed up the training of the model where a decrease in test accuracy after 7000 training examples, telling us that the model is overfitted. Adjusting the alpha therefore helped to control the overfitting of the Decision Tree. Finally, looking at the metrics of the 2 models, the advanced Decision Tree scored a test F1 of 0.716 in contrast to the test F1 of the basic model that scored 0.652. Similar scenario is seen across the board for the other metrics, specifically ROC area under curve which improved from 0.775 to 0.818 (Fig 1g, 1h)

Model	Base DT	Base DT	Adv. DT	Adv. DT
Metric	Train	Test	Train	Test
Accuracy	1.00	0.846	0.875	0.873
Precision	1.00	0.666	0.714	0.696
Recall	1.00	0.686	0.829	0.811
F1	1.00	0.676	0.768	0.749
ROC AUC	1.00	0.790	0.860	0.851

Figure 1g – [Bank] Decision Tree - Train, Test metrics

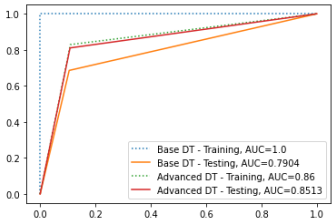

Figure 1h – [Bank] Decision Tree – AUC ROC Curve

K-Nearest Neighbours (KNN)

For KNN, I used the KNeighborsClassifier[6] in sklearn. In KNN, A query point is assigned to the data class that has the most representatives among its nearest neighbours after classification is calculated using a simple majority vote of each point's nearest neighbours. A small value of k indicates that noise will have a greater impact on the outcome. A large value makes it computationally expensive and somewhat contradicts the fundamental tenets of KNN. Distance calculation[7] is also important, where Euclidean distance penalises small values and exponentiates the larger values, causing the dimensional space to be further apart hence lowering the accuracy rates of points that are supposed to be near to each other. Manhattan distance on the other hand promotes sparsity of the data for high dimensional problems. In the Spambase problem, I first used an arbitrary k=3 to train a model. The default distance is set at Euclidean. Following which, I looked at the learning curve (Fig 2a) and noticed that the training and CV scores are not converging. There may be potential overfitting, which leads me to want to find an optimal k and p to use.

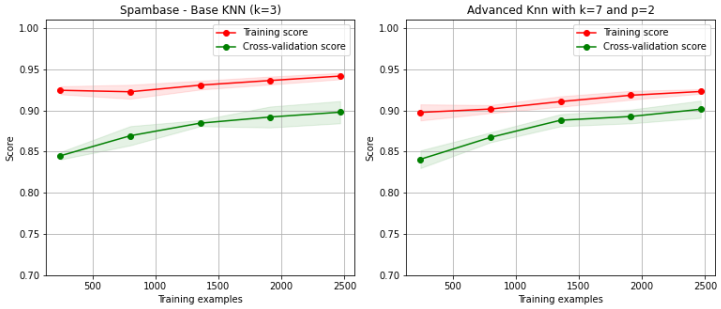

Figure 2a – [Spambase] KNN – Learning Curve

Iterating through the space[8] of k=[1 to 45], p = [1, 2] (where 1 is Manhattan, 2 is Euclidean), I plotted the following curves (Fig 2b) to determine the best k and p. I trained another (advanced) model using k=7 and p=2 as it gave the ROC score and lowest error. The following results showed an improvement of ROC from 0.890 to 0.899 after using the best

parameters (Fig 2c, 2d). Comparing the learning curves (Fig 2a), a smaller gap is also seen in the training and CV score, showing a possibility of convergence on further training.

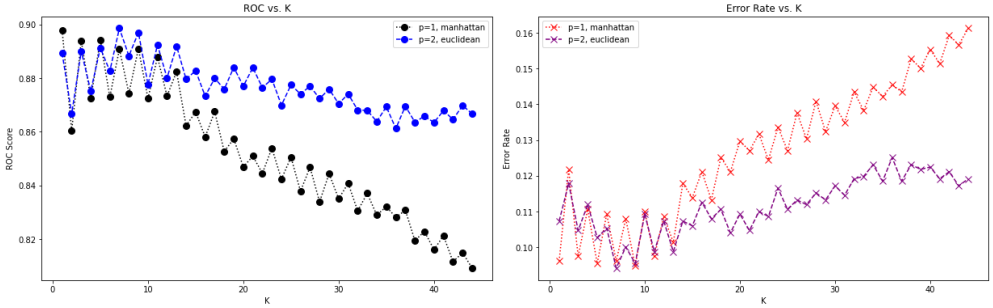

Figure 2b – [Spambase] KNN – Metrics vs k for different p

KNN	Base	Base	Adv.	Adv.
Metric	**Train**	**Test**	**Train**	**Test**
Accuracy	0.943	0.895	0.926	0.906
Precision	0.929	0.887	0.923	0.914
Recall	0.921	0.858	0.879	0.855
F1	0.925	0.872	0.901	0.883
ROC AUC	0.939	0.890	0.917	0.899

Figure 2c – [Spambase] KNN - Train, Test metrics

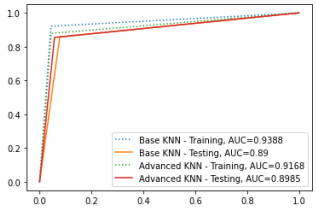

Figure 2d – [Spambase] KNN – AUC ROC Curve

In the Bank Marketing problem, I built a basic KNN using the default parameters where k=5 and the distance metric p=2 to use the Euclidean distance (L2). Subsequently, I tried a variation of k and p to find the best combination that would lead to the lowest error rate and best ROC score using the same approach as before. A cross validation with K=5 is used to get a more approximate score. I found the best scores at k=37 using p=1 (Fig 2e)

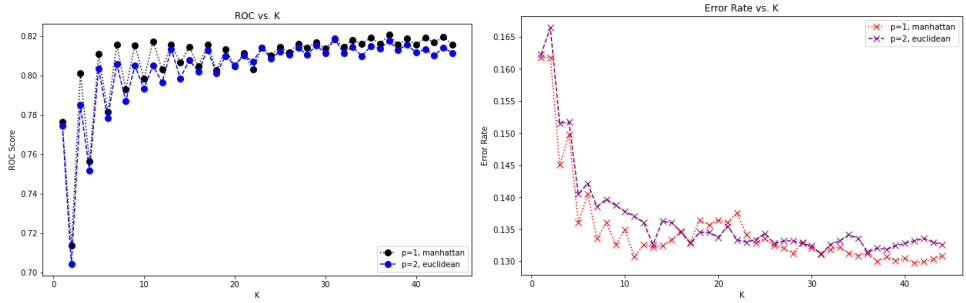

Figure 2e – [Bank] KNN – Metrics vs k for different p

The Manhattan (L1) (p=1) distance achieved a better score than Euclidean because the dataset contains discrete and binary attributes and having a relatively high dimension. Next, using the best parameters, an advanced KNN was trained following the same methods as the base KNN such as using 5 fold cross validation. Looking at the learning curves (Fig 2f), in the base KNN, the training score is higher than the validation score, signalling that the model probably requires more training examples in order to generalize more effectively. The advanced model showed the training and cross-validation scores converge together as more data is added. This tells us that the model will not benefit from more data and the model is sufficiently trained to perform well.

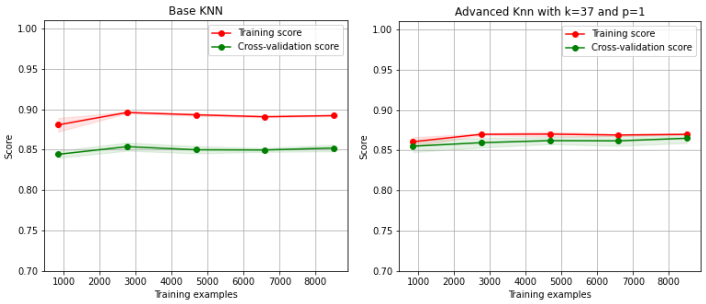

Figure 2f – [Bank] KNN – AUC ROC Curve

The effect of changing the k and p value helped to speed up the training of the model. Finally, comparing the metrics of the 2 models, the advanced KNN scored a test F1 of 0.724 in contrast to that of the basic model that scored 0.699 and ROC area under curve which improved from 0.803 to 0.821 (Fig 2g)

KNN	Base	Base	Adv.	Adv.
Metric	**Train**	**Test**	**Train**	**Test**
Accuracy	0.893	0.860	0.871	0.870
Precision	0.786	0.701	0.737	0.720
Recall	0.782	0.698	0.745	0.727
F1	0.784	**0.699**	0.741	**0.724**
ROC AUC	0.856	**0.803**	0.829	**0.821**

Figure 2g – [Bank] KNN - Train, Test metrics

SVM

I used the SVC[9] package in sklearn for Support Vector Machines. In the default parameters, C is set to 1 with an rbf[10] kernel used. C is a regularization parameter and the strength of regularization is inversely proportional to C. The C parameter controls the SVM optimization to avoid misclassifying each training example. For larger values of C, the optimization will choose a smaller-margin hyperplane, provided that hyperplane does a better job of getting all the training points classified correctly. Kernels help to transform the training data so that a non-linear decision surface is able to transform to a linear equation in a higher number of dimension spaces. Polynomial and RBF are especially useful when the data-points are not linearly separable. For both classification problems, I built a basic SVC model using the default parameters (C=1, kernel = rbf) and used K-fold cross validation of K=5. Subsequently, I searched for the best hyperparameters by tuning both the kernels and C. The options explored were: kernels = [linear, poly, rbf, sigmoid] and C = [0.0001, 0.001, 0.01, 0.1, 1]. In the Spambase problem, the best hyperparameters were found at rbf (radial basis function) kernel and C = 1 (Fig 3a). This was the default parameters. This kernel allows us to find a non-linear classifier or regression line in the data given that the data is high dimensional and a linear classifier may not find the best separation between the classes. Given the close results between the linear and rbf classifiers at C=1, I built another model to compare them. Looking at the learning curves (Fig 3b) and metrics (Fig 3c) they are both highly similar and show convergence to a similar value, but the default model of rbf kernel still edges out slightly over the linear model in terms of AUC. This suggests that even though the data has fairly smaller values, in the higher dimensional space it is still better to map is using a non-linear approach.

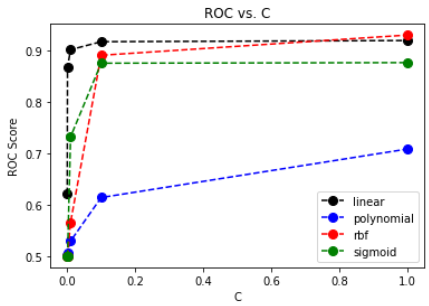

Figure 3a – [Spambase] SVM - ROC vs C & kernels

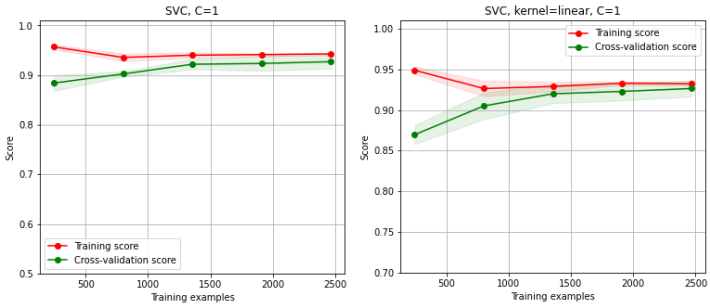

Figure 3b – [Spambase] SVM – Learning Curve

SVM	RBF	RBF	Linear	Linear
Metric	**Train**	**Test**	**Train**	**Test**
Accuracy	0.942	0.935	0.932	0.924
Precision	0.944	0.945	0.927	0.930
Recall	0.903	0.896	0.892	0.885
F1	0.923	0.920	0.909	0.907
ROC AUC	0.935	0.929	0.924	0.919

Figure 3c – [Spambase] SVM - Train, Test Metrics

For Bank Marketing problem, the learning curve of the base SVC model showed that both the training and CV score converged around 2800 training examples and trailed off with a gradual increase to max score at about 0.83 (Fig 3e). This shows that the model still can be tuned, but required a lot more examples. The linear kernel and C=0.001 provided the best ROC score so I built an (advanced) model utilising these parameters.

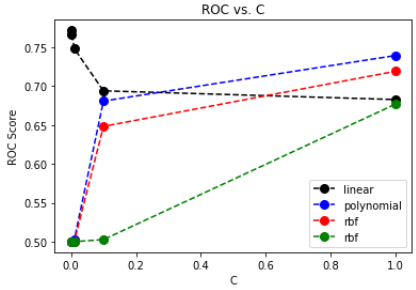

Figure 3d – [Bank] SVM - ROC vs C & kernels

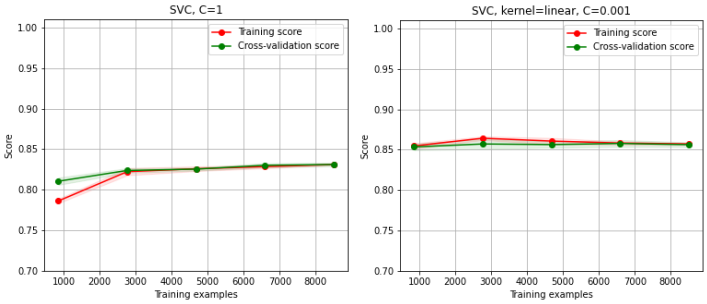

Figure 3e – [Bank] SVM – Learning Curve

This hyperparameter search result is understandable because the values in the dataset are relatively small, with some binary values and smaller values that can be easily fit into a linear kernel. A small value of C suggests a small penalization on the classifier. This is because the classifier can maximize the margin using a majority of the points while misclassifying a few points. Looking at the new learning curve (Fig 3e), the training and CV scores converged around 6500 training examples, reaching a score slightly above 0.85, signalling that the model has been sufficiently tuned. The final result of the advanced SVC model performed significantly better than the base model of C=1, kernel=rbf where the AUC improved from 0.719 to 0.772 in the test set (Fig 3f)

SVM	Base	Base	Adv.	Adv.
Metric	**Train**	**Test**	**Train**	**Test**
Accuracy	0.833	0.846	0.857	0.861
Precision	0.763	0.773	0.763	0.751
Recall	0.475	0.482	0.615	0.605
F1	0.585	0.594	0.681	0.670
ROC AUC	0.713	0.719	0.776	0.772

Figure 3f – [Bank] SVM - Train, Test Metrics

Gradient Boosting

In Gradient Boosting, I utilized the GradientBoostingClassifier[11] (GBC) in sklearn. In order to create a powerful learner, the boosting strategy combines a number of weak learners (predictors with low accuracy) to create a model with high accuracy. Each model learns from the flaws of the one before it and improves it by reducing the errors. In this case, each learner is 1 decision tree. The learning rate is a hyperparameter that is used to scale each trees contribution, sacrificing bias for better variance. In other words, we multiply this number by the predicted value so that we do not overfit the data. The max depth parameter controls the maximum depth of each tree, meaning the construction of a tree will stop growing once the tree exceeds a certain depth. This is a form of pruning the tree to avoid overfitting. The number of estimators simply refer to the number of trees being built or the number of boosting stages to perform. The model learns the data better with more trees built, but this may also lead to overfitting. In the Spambase problem, I used default parameters to build a boosting model. For the default model, the parameters were set at learning rate = 0.1, max depth = 3, number of estimators = 100. The learning curve looked promising, showing both the training and CV scores converging, albeit slowly (Fig 4a). Subsequently, I attempted a GridSearchCV to find the best parameters[12] for this problem. The search space was max_depth = [1,2,3,4], learning_rate = [0.001, 0.01, 0.1, 1] and n_estimators = [25,50,75,100,125, 150]. The best hyperparameters were found at learning rate at 0.1, max depth at 4, n estimators at 150. A new (advanced) model is trained with this parameter set. Comparing the new learning curve with the previous (Fig 4a), the gap between the training and CV score is noticeably larger. This suggests that the newer advanced model may be overfitted and will generalize worse than the base model. This shows that the hyperparameter search optimized the results but in doing so, led to overfitting by using more estimators (50 more than default) and using deeper trees (4 instead of 3). Given this understanding, I attempted 1 more model (advanced 2), limiting the n estimators to 50 and max depth to 2 to prune the model more aggressively. The learning curve (Fig 4a) showed a narrower gap and a faster convergence compared to the previous models.

Figure 4a – [Spambase] GBC – Learning Curve

Overall, the results for the advanced model seems to perform the best on the test set at 0.952 AUC (Fig 4b). However, in future unseen data, the advanced 2 model will outperform the other models even though it has a lower AUC at 0.929 because it is less overfit compared to the first 2 models.

GBC	Base	Base	Adv.	Adv.	Adv. 2	Adv. 2
Metric	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Accuracy	0.963	0.951	0.982	0.956	0.939	0.936
Precision	0.961	0.954	0.985	0.961	0.949	0.956
Recall	0.942	0.926	0.968	0.931	0.887	0.888
F1	0.952	0.940	0.976	0.946	0.917	0.921
ROC AUC	0.959	**0.947**	0.979	**0.952**	0.929	**0.929**

Figure 4b – [Spambase] GBC - Train, Test Metrics

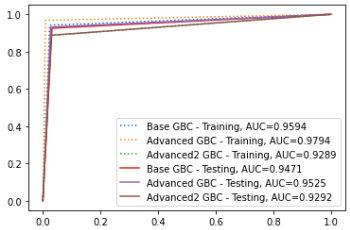

Figure 4c – [Spambase] GBC – AUC ROC Curve

For the Bank Marketing problem, I used a basic boosting model using random parameters set at learning_rate = 0.1, n_estimators = 200, max_depth = 4. Using these parameters, I first ran the model with cross validation of K=5. The learning curve (Fig 4d) displays a slow convergence of the training and CV scores, even after 8000 training examples they are still not converged, indicating that the model still requires some training to generalize well.

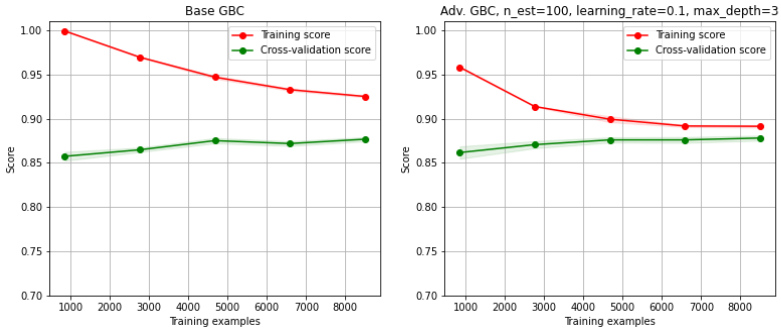

Figure 4d – [Bank] GBC – Learning Curve

Next, I used a Gridsearch CV to find the best hyperparameter within a set space. These included max_depth = [1,2,3], learning_rate = [0.001, 0.01, 0.1, 1] and n_estimators = [25,50,75,100,125]. The result of the hyperparameter tuning showed that the default model can be tuned to achieve a robust model. The suggested hyperparameters were learning rate at 0.01, max depth at 3, n estimators at 100 (Fig 4e). A possible reason why these parameters were selected is that the basic model overfits using a high number of estimators, a deeper tree of depth 4 and also with a higher learning rate of 0.1.

Param_learning_rate	Param_max_depth	Param_n_estimators	Mean_test_score	Std_test_score
0.1	**3**	**100**	**0.878024**	**0.002876**
0.1	**3**	**125**	**0.878024**	**0.002407**
0.1	**3**	**75**	**0.877929**	**0.003295**

Figure 4e – [Bank] GBC – Top 3 results GridsearchCV hyperparameter tuning

The suggested hyperparameters are more conservative, using less estimators that are more shallow with a smaller learning rate to shrinks the contribution of each tree. A secondary (advanced) boosting model is trained using these parameters. In this advanced model, the learning curve (Fig 4d) indicates that the training and CV gap is narrower compared to the basic boosting model. This suggests that the new advanced model is able to generalize better to new data. The testing and training learning curves also now converge at similar values. The overall score on the test set is also higher at 0.838 compared to 0.832 in the basic boosting model. Whilst this is a slight improvement, I would be more confident to use this advanced boosting model for future iterations of the same problem as I am certain it will generalize better to unseen data compared to the initial boosting model.

SVM	Base	Base	Adv.	Adv.
Metric	Train	Test	Train	Test
Accuracy	0.918	0.877	0.890	0.878
Precision	0.830	0.734	0.766	0.730
Recall	0.841	0.747	0.803	0.762
F1	0.835	0.740	0.784	0.746
ROC AUC	0.892	0.832	0.861	0.838

Figure 4f – [Bank] GBC - Train, Test Metrics

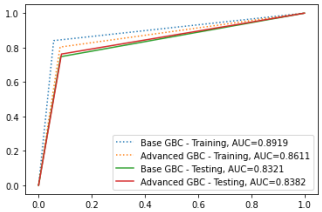

Figure 4g – [Bank] GBC – AUC ROC Curve

Neural Networks

The Neural Networks implementation is accomplished using the MLPClassifier[13] package from sklearn. For both classification problems, I start with the default parameters from MLPClassifier. This includes 1 hidden layer with 100 neurons, relu (rectified linear unit) for activation function, solver using adam, max iterations at 200, alpha at 0.0001 and learning rate to be constant. Most of these parameters by default are already best in class, such as using Relu as an activation function, because it does not cause saturation of its gradient and propagates back gradients effectively, we avoid the vanishing/exploding gradient problem, which greatly accelerates the convergence of the stochastic gradient descent. Other parameters like alpha is used as a regularization or penalty term that combats overfitting by constraining the size of the weights. Increasing alpha may fix high variance (to control overfitting) by encouraging smaller weights. This will result in a decision boundary plot with lesser non-linearity. Decreasing alpha may fix high bias (to improve underfitting) by encouraging larger weights and result in a more complicated decision boundary*. The hidden layers help to construct and solve the complexity of the non-linearity in the data. The max_iter parameter controls the maximum number of iterations (or epochs) that the solver iterates until convergence. In the following experiments, I noticed that most of these parameters do not change and will attempt to change those that matter to improve the performance of the model. I will leverage a GridSearchCV with the following parameter space: max_iter: [100, 200, 300], hidden_layer_sizes: [(50,50,50), (50,100,50)], activation: [tanh, relu], solver: [sgd, adam], alpha: [0.0001, 0.05], learning_rate: [constant,adaptive]. The aim of doing GridSearchCV is to optimize and find the best parameters for the dataset, assuming that we do not have any understanding of the data at all and instead let the Neural Network find the best ways to accurately run the prediction to solve the problem. In the Spambase problem, the learning curve for the default base model showed that the training and CV scores have not even began to converge. The gap in the 2 scores indicates a weak generalization of data (Fig 5a). Despite this, the scores of the test set is relatively high at 0.948 (Fig 5). An advanced model is built after doing the hyperparameter search to find a model that can generalize better. The best parameters were reported with {'activation': ‘tanh’, 'alpha': 0.0001, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'constant', 'max_iter': 300, 'solver': 'sgd’}. Tanh as an activation function because it is centred around 0 and has a larger range (-1 to 1), creating larger derivatives which usually leads to faster convergence of the model to the minimum. Activation function is suggested to be sgd over adam because it is relatively more locally unstable compared to adam. Therefore, it is more likely to converge to the minima at the flat or asymmetric basins/valleys (or local minima) which often leads to better generalization. Max iteration has been suggested at 300 because in the base model, convergence has not been reached yet. After training the advanced neural network model, the new learning curve (Fig 5a) indicates a smaller gap between the training and CV score, suggesting a possible convergence in more training examples.

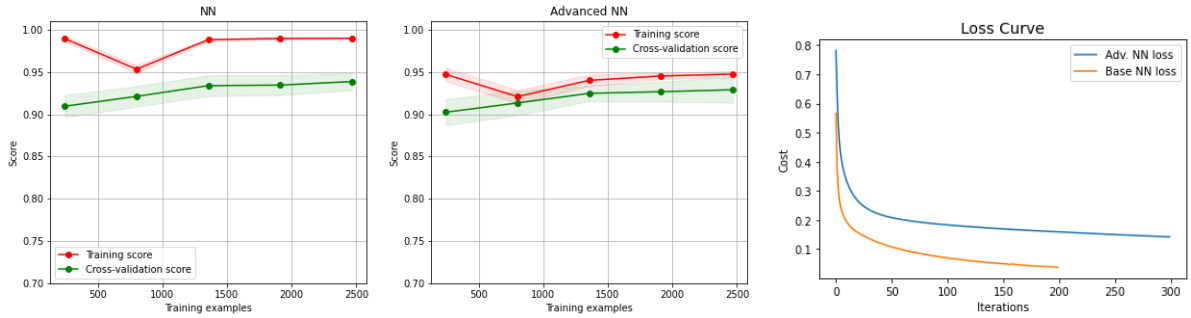

Figure 5a – [Spambase] Neural Networks – Learning Curve, Loss Curve

This also means a more generalizable model compared to the initial base neural network. Looking at a comparison of the loss[14] curves (Fig 5a), even though the base model has lower loss than the advanced model, it represents a greater overfitting when we cross examine with the learning curves. Finally, looking at the metrics comparison, the newer model shows a slightly lower AUC score at 0.942 compared to 0.948 (Fig 5b, 5c).

NN	Base	Base	Adv.	Adv.
Metric	**Train**	**Test**	**Train**	**Test**
Accuracy	0.991	0.951	0.946	0.945
Precision	0.993	0.952	0.939	0.944
Recall	0.983	0.930	0.920	0.924
F1	0.988	0.941	0.929	0.934
ROC AUC	0.989	**0.948**	0.942	**0.942**

Figure 5b – [Spambase] Neural Networks - Train, Test Metrics

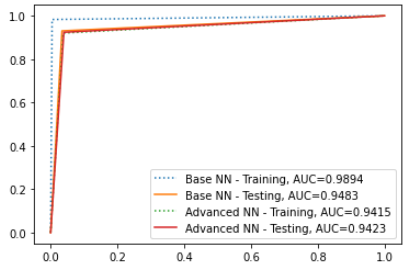

Figure 5c – [Spambase] Neural Networks – AUC ROC Curve

For the Bank Marketing problem, after training the default model, the learning curve (Fig 5d) showed that the training and CV scores were converging. However, the scores are decreasing with more training samples added. This suggests a potential overfitting. I use GridSearchCV to find ways to tune and lower the overfitting such as controlling the regularization parameter. After the hyperparameter tuning, the best parameters were reported with {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'max_iter': 100, 'solver': 'adam'}. Whilst most of these hyperparameters are similar to the default, hidden layer size and max iteration are changed. The best hidden layer is at (50, 50, 50), meaning 3 layers with 50 neurons each. One reason this is suggested as a good hyperparameter is the non-linearity in the data that requires a few hidden layers to map out in the high dimensional space. The max iteration at 100 suggests that the model does not need too many iterations to reach convergence. Training the Neural Network again with this advanced implementation, the learning curve (Fig 5d) showed a similar pattern, but does not seem to degrade as fast as the initial model. Comparing the loss curves (Fig 5d) between the 2 models, the advanced model seems to be relatively more stable compared to the base model. Looking at the scores on the test set, it is visibly apparent that the newer advanced model greatly outperforms the base model, with a huge improvement of AUC from 0.584 to 0.810 (Fig 5e, 5f).

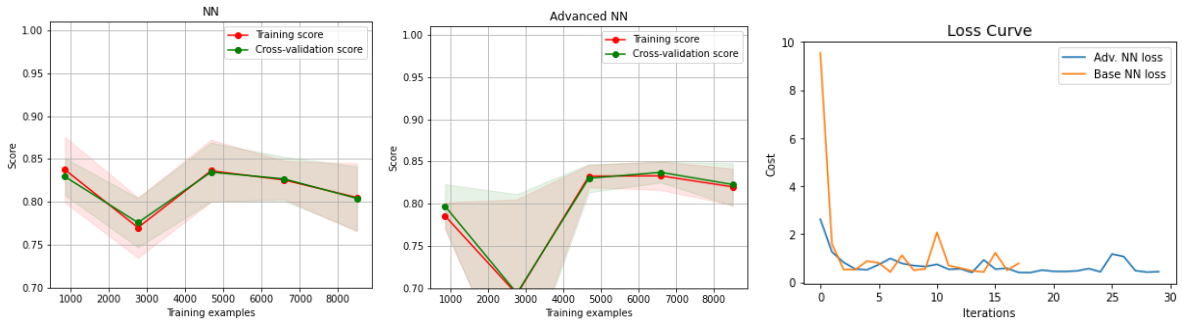

Figure 5d – [Bank] Neural Networks – Learning Curve, Loss Curve

NN	Base	Base	Adv.	Adv.
Metric	**Train**	**Test**	**Train**	**Test**
Accuracy	0.791	0.799	0.851	0.845
Precision	0.834	0.825	0.675	0.646
Recall	0.196	0.180	0.768	0.745
F1	0.317	0.296	0.718	0.692
ROC AUC	0.592	**0.584**	0.823	**0.810**

Figure 5e – [Bank] Neural Networks - Train, Test Metrics

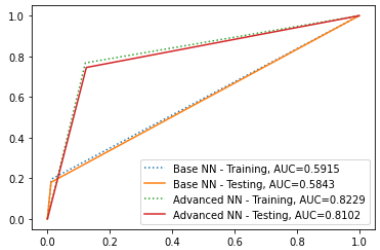

Figure 5f – [Bank] Neural Networks – AUC ROC Curve

Conclusion

For the Spambase problem, most of the models seem to achieve a high score of at least 0.9 in AUC (Fig 6a) . I suspect this is because of the simplicity of the data and the problem. Neural networks achieved the best result due to the complexity of the hidden layers that allows the model to identify extreme non-linearity in the higher dimensions. This is also the case as seen in SVM using the rbf kernel. Boosting and decision trees achieved relatively similar scores, with boosting slightly edging the decision trees, an understandable situation given the methodology of building multiple decision trees. I would have expected this difference to be higher. KNN performed the worst, given that we were using linear distances of either Manhattan or Euclidean and was less capable in drawing out the non-linearity in the data. Looking at the results of all the models, it might also be wise to consider upsampling or resampling the data to create more samples to train on, given that most models have not converged using the limited samples (3000+ in the training set).

For the Bank Marketing problem, most of the models achieve approximately 0.8 in AUC (Fig 6b). Decision trees proved to be the best model here, articulating the need to not build over complicated models to solve simple problems. Trailing behind would be the boosting model and judging by the drop in AUC (0.838 from 0.851), it seems that the boosting model would have been too overfitted to solve the problem given the additional number of trees being built. Other models - KNN, Neural Networks and SVM are not performing as well but this can be attributed to the data. In the bank marketing data set, the data has various binary features, discrete and continuous features. It would be easier to build step-wise decisions such as Decision trees to solve the problem, compared to creating artificial classifiers in the high dimensional space in all the other algorithms (except boosting). On hindsight, it may have helped if I could experiment with the downsampling of the negative class to different values or even upsampling the positive class. As mentioned at the start, the ratio of negative to positive class is about 3:1. This could have been used as a “hyperparameter” to be tuned to find the ratio that gives the best results.

Spambase	Decision Tree	KNN	SVM	Boosting	Neural Networks
Accuracy	0.926	0.906	0.935	0.936	0.945
Precision	0.901	0.914	0.945	0.956	0.944
Recall	0.924	0.855	0.896	0.888	0.924
F1	0.913	0.883	0.920	0.921	0.934
ROC AUC	0.926	0.899	0.929	0.929	0.942

Figure 6a – [Spambase] Models Test Set Comparison

Bank	Decision Tree	KNN	SVM	Boosting	Neural Networks
Accuracy	0.873	0.870	0.861	0.878	0.845
Precision	0.696	0.720	0.751	0.730	0.646
Recall	0.811	0.727	0.605	0.762	0.745
F1	0.749	0.724	0.670	0.746	0.692
ROC AUC	0.851	0.821	0.772	0.838	0.810

Figure 6b – [Bank] Models Test Set Comparison

References

[1]https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html?highlight=decisiontree#sklearn.tree.DecisionTreeClassifier

[2]https://bricaud.github.io/personal-blog/entropy-in-decision-trees/

[3]https://www.kdnuggets.com/2022/07/kfold-cross-validation.html

[4]https://scikit-learn.org/stable/modules/tree.html#minimal-cost-complexity-pruning

[5]https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html#sphx-glr-auto-examples-tree-plot-cost-complexity-pruning-py

[6]https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html?highlight=nearest

[7]https://medium.com/@kunal_gohrani/different-types-of-distance-metrics-used-in-machine-learning-e9928c5e26c7#:~:text=Manhattan%20distance%20is%20usually%20preferred,similarity%20between%20two%20data%20points.

[8]https://ai.plainenglish.io/how-to-find-best-fit-k-value-in-knn-4f8a31ba64f2

[9]https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html?highlight=svc#sklearn.svm.SVC

[10]https://www.geeksforgeeks.org/radial-basis-function-kernel-machine-learning/

[11]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

[12]https://medium.com/all-things-ai/in-depth-parameter-tuning-for-gradient-boosting-3363992e9bae

[13]https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html?highlight=mlpclassifier#sklearn.neural_network.MLPClassifier

[14]https://developers.google.com/machine-learning/testing-debugging/metrics/interpretic

Learning curve code:
https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html#sphx-glr-auto-examples-model-selection-plot-learning-curve-py

Timer code:
https://realpython.com/python-timer/