CS-7641 Machine Learning: Assignment 1, Supervised Learning

Pavel Ponomarev
pavponn@gatech.edu

1 INTRODUCTION

In this project, we [1] study the performance of five supervised learning algorithms on two distinct datasets. The main goal is to better understand how different algorithms work in practice under variety of circumstances (data, hyperparameters, etc). The machine learning methods we explore are: Decision Tree, Boosting, KNN (K-Nearest Neighbours), SVM (Support Vector Machine), and Neural Network.

2 DATASETS

The first dataset, sourced from the "Credit Card Fraud Detection" competition on Kaggle [2], encompasses data on 284,807 transactions. It comprises 29 real-valued features, including 28 anonymized attributes, potentially derived from other interpretable features using Principal Component Analysis (PCA), along with the transaction amount. Each transaction is labeled as either fraudulent (1) or benign (0), with only 492 instances of fraud, resulting in significant class imbalance. Furthermore, the classes in the dataset have varying costs associated with incorrect predictions. This provides us with a valuable opportunity to consider the most appropriate evaluation metric instead of blindly following default metrics like accuracy. To streamline experimentation, we downsized the dataset to 71,571 examples by down-sampling benign transactions while retaining all fraud cases. It still maintains the heavily imbalanced nature of the classification problem, i.e., 0.69% positive and 99.31% negative labeled samples.

Another dataset, sourced from UC Irvine [3], comprises two subsets: one with 1599 samples of red wine and the other with 4898 samples of white wine. The classification task involves predicting wine quality, which falls into 7 classes ranging from 3 to 9. We selected this dataset to explore machine learning techniques for multiclass classification. Notably, class distribution is non-uniform, with more samples falling within the quality range of 5 to 7 than in the ranges of $3-4$ or $8-9$. However, in contrast to the credit card fraud dataset, our emphasis here lies on overall performance (and not optimising for some specific classes), leading to the selection of a different evaluation metric. For analysis, we unified the red and white wine datasets and introduced an additional feature, denoted as *color*.

1 The work described in this report is completed fully independently by one sole author (Pavel Ponomarev). The pronounce "we" is used to follow academic writing style.

2 https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

3 https://archive.ics.uci.edu/dataset/186/wine+quality

3 METHOD

This section begins with a concise overview of the employed technologies, encompassing the programming language and frameworks. Following this, we delineate the standard setup for the experimentation process, applied consistently across two datasets and five supervised learning algorithms, while highlighting key distinctions. Finally, we substantiate the rationale behind the selection of performance metrics.

3.1 Technologies

To implement the described procedures, we utilized Python 3, leveraging its extensive library ecosystem. Specifically, we sourced implementations of all supervised algorithms from the `scikit-learn` module. For visualization purposes, we employed `yellowbrick`, a tool that facilitates the generation of a significant portion of the charts with minimal engineering overhead. Additionally, we made use of other common libraries such as `numpy` and `pandas`.

3.2 Experimental Setup

Each dataset is initially divided into training and testing sets, approximately in an $8:2$ ratio. The testing dataset is exclusively reserved for evaluating the final performance of a model and is never employed in the decision-making process for model selection.

For the wine quality dataset, we conduct one-hot encoding solely for its categorical feature *color*.

Subsequently, for both datasets, we generate duplicates of training and testing data, with features normalized using min-max normalization. The scaling parameters are derived from the training subset and then applied to normalize the testing subset. The need for normalized features arises from the characteristics of certain machine learning algorithms we investigate.

We employ k-fold cross validation consistently throughout our study. Specifically, for the credit card fraud dataset, we set $k = 5$, while for the wine quality dataset, we chose $k = 4$, given that one of the classes had only four instances in the training set.

For each algorithm and dataset, the general approach is as follows:

1. Train a baseline model using the default set of hyperparameters and analyze its performance metrics.
2. Investigate individual hyperparameters in isolation to understand their impact on algorithm performance, employing techniques such as validation curves.
3. Conduct hyperparameter tuning to obtain the best-performing model using cross validation score.
4. Generate learning curves for the best model, analyze its performance metrics, and conclude the final analysis.

3.3 Performance Metrics

For the credit card fraud dataset, our objective is to optimize for recall at a precision 0.9 (for short, R@PR(0.9)): aiming to maximize recall while maintaining a high precision level (> 0.9) as a guardrail. This choice is motivated

by the assumption that the trained model would be deployed in an online setting by a bank or other institution to prevent fraudulent transactions and take appropriate action (such as blocking accounts or initiating manual reviews). While it is crucial to minimize potential harm caused by fraud through transaction blocks, we want to minimize any adverse impact on legitimate users from false positives. Hence, we prioritize recall at a precision threshold. The chosen threshold is informed by practical experience: it is a common practice in automated detection systems for such violations to set a precision threshold around 0.9.

In the wine quality classification problem, we focus on overall performance with respect to individual samples, rather than specific classes. Consequently, we opted for the micro F1 score over the weighted or macro F1 scores.

Algorithm	**Recall @ Precision 0.9**				**Train Time**	**Inference Time**	**Best Hyperparameters**
	Base		**Best**				
	Train	**Test**	**Train**	**Test**			
Decision Tree	1.0000	0.0000	0.8421	0.7419	1.3173 sec	0.0036 sec	`ccp_alpha=0.0, min_samples_leaf=21, max_depth=7`
Boosting	0.8471	0.7527	0.8571	0.7419	39.9236 sec	0.8013 sec	`lr=0.0464, n_estimators=200`
KNN	0.8622	0.7634	0.8697	0.7634	0.0034 sec	8.2721 sec	`metric='cityblock', n_neighbors=5`
SVM	0.8596	0.7527	0.9148	0.7527	14.1791 sec	2.2159 sec	`kernel='rbf', C=71.9685`
Neural Network	0.8371	0.7634	0.8396	0.7527	27.8913 sec	0.0118 sec	`height=50, width=2, batch_size=16, alpha=0.001, lr_init=0.01`

Table 1—Credit Card Fraud Dataset: Results Overview

4 RESULTS

The experimental results are consolidated in Table 1 for the credit card fraud dataset and Table 2 for the wine quality dataset. These tables present the performance metrics for each supervised learning algorithm on both training and testing subsets. The metrics are provided for both the baseline version (without hyperparameter tuning) and the best-performing version of each algorithm. We put the total training and inference times, both computed using on the training sets. In the final columns, we detail the hyperparameters of the best-performing algorithm version, with all other parameters set to their default values.

For the credit card fraud dataset, K-Nearest Neighbors (KNN) demonstrated the best testing performance. However, it's crucial to consider trade-offs. KNN had the longest inference time, exceeding 8 seconds, followed by SVM at approximately 2.2 seconds. In contrast, KNN showed the fastest training time due to its deferred computations. Boosting required the lengthiest training, followed by the Neural Network and SVM.

Furthermore, it's worth noting that although hyperparameter tuning improved the performance of the best classifiers (KNN, SVM, and NN) on the training dataset, this enhancement did not extend to the testing dataset, with the performance metrics remaining unchanged.

In the wine quality dataset, SVM demonstrated the highest performance on the testing set with a micro F1 score of 0.54, followed by K-Nearest Neighbors (KNN). Notably, in this instance, SVM required the most time for inference, surpassing even the time needed by KNN. The longest training time was attributed to the Neural

Network (approximately 20 seconds), followed by SVM (around 12 seconds), and then Boosting, which took only 0.5 seconds.

An intriguing observation is that, with this metric, nearly every algorithm exhibited improved performance on the testing set after hyperparameter tuning, except for the Neural Network, which experienced a drop. This discrepancy, alongside the significant improvement on the training set, may suggest overfitting in the case of the Neural Network.

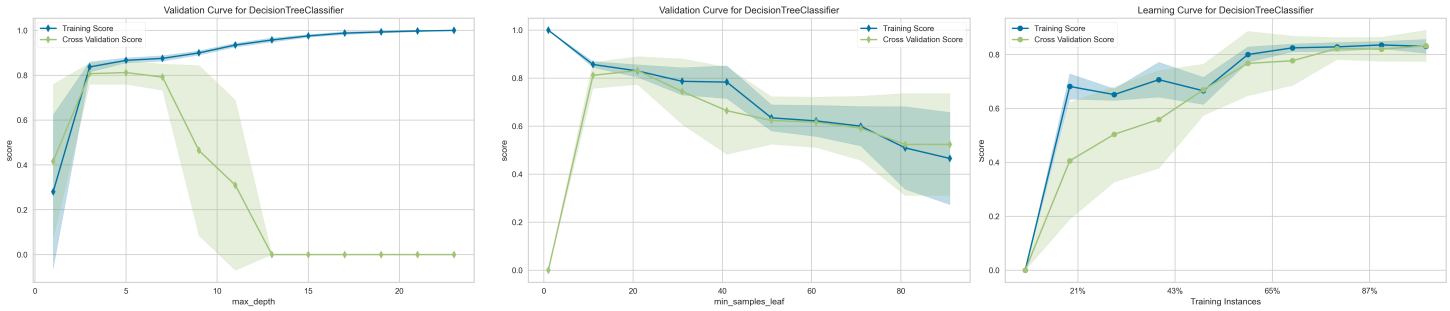
We proceed with a more thorough examination of experiment results in Section 5.

Algorithm	**Micro F1 Score**				**Train Time**	**Inference Time**	**Best Hyperparameters**
	Base		**Best**				
	Train	**Test**	**Train**	**Test**			
Decision Tree	1.0000	0.4568	0.5780	0.5066	0.0134 sec	0.0017 sec	`ccp_alpha=0.0, min_samples_leaf=41, max_depth=7`
Boosting	0.3569	0.3665	0.5094	0.5094	0.5150 sec	0.0737 sec	`lr=0.0003, n_estimators=150`
KNN	0.6692	0.5047	0.6217	0.5291	0.0011 sec	0.3106 sec	`metric='cosine', n_neighbors=9`
SVM	0.5456	0.5291	0.6302	0.5404	12.3855 sec	0.5302 sec	`kernel='poly', degree=5, C=17.7827`
Neural Network	0.5616	0.5620	0.7357	0.5105	20.6169 sec	0.0079 sec	`height=200, width=2, batch_size=64, alpha=0.001, lr_init=0.01`

Table 2—Wine Quality Dataset: Results Overview

5 RESULTS DISCUSSION

5.1 Decision Trees



(a) Validation curves for `max_depth` ***(b)*** Validation curves for `min_samples_leaf` ***(c)*** Learning curves of the best model

Figure 1—Validation & Learning Curves for Decision Tree on Credit Card Fraud Dataset

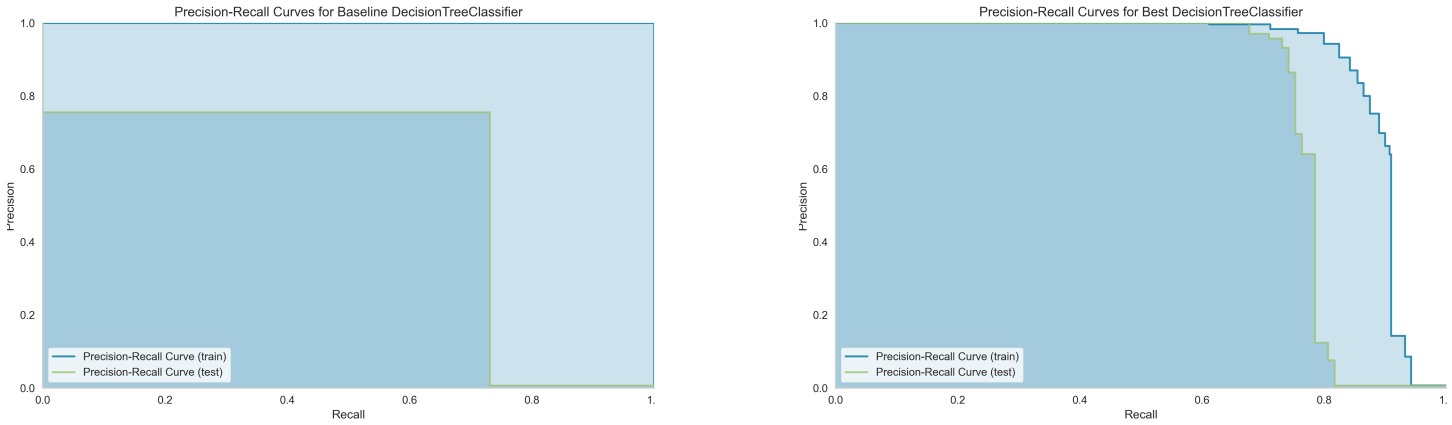
First, we overview behavior of decision trees on credit card fraud dataset. From Table 1 one could see that baseline Decision Tree achieved R@PR(0.9) equal one on training set, while absolutely failing on testing dataset showing zero-valued performance. This happened due to two main reasons: (i) overfitting to training data; (ii) choice of the metric.

Let us start with (i). The baseline decision tree had its parameters `max_depth` and `min_samples_leaf` set to ∞ and 1 respectively, effectively allowing the tree to grow without any pruning constraints. Given the highly

imbalanced dataset, it is likely that the tree may have isolated each positive sample into its own individual leaf node. This hypothesis is supported by the validation curves for these parameters shown in Figures 1a and 1b. It is evident from the charts that the tree requires a non-trivial depth to effectively learn, but excessive depth leads to overfitting. Regarding the number of minimum samples in a leaf, larger values lead to smaller variance but higher bias in the experiment.

As pointed out, another reason (ii) we get such a big discrepancy between training and testing metric value is metrics itself. Indeed, R@PR(0.9) is a very strict metric due to its high precision threshold. We confirmed it after looking at precision-recall curves on training and testing datasets produced by the baseline classifier (Figure 2a). One could see that if we chose a lower precision threshold we would not notice the overfitting problem that easy. For example, judging from the chart R@PR(0.75) of a baseline model is 1 on training and greater than 0.7 on testing set.

In light of these observations, we proceeded to fine-tune the pruning parameters mentioned earlier, along with the additional parameter `ccp_alpha`. This parameter is utilized in a post-pruning technique known as *cost complexity pruning*, where it facilitates the removal of nodes rather than inhibiting their creation. However, in isolation, `ccp_alpha` exhibited minimal impact. Moreover, when combined with the tuning of the other two parameters, it did not yield any noticeable difference in the model's performance. The classifier then demonstrates very health training behaviour maintaining low variance and minimising bias reaching a plateau after using around 70% of the data (see Figure 1c). Also, precision-recall curves look much more sound: Figure 2b is another confirmation that tuned classifier generalises much better.



(a) Baseline Classifier

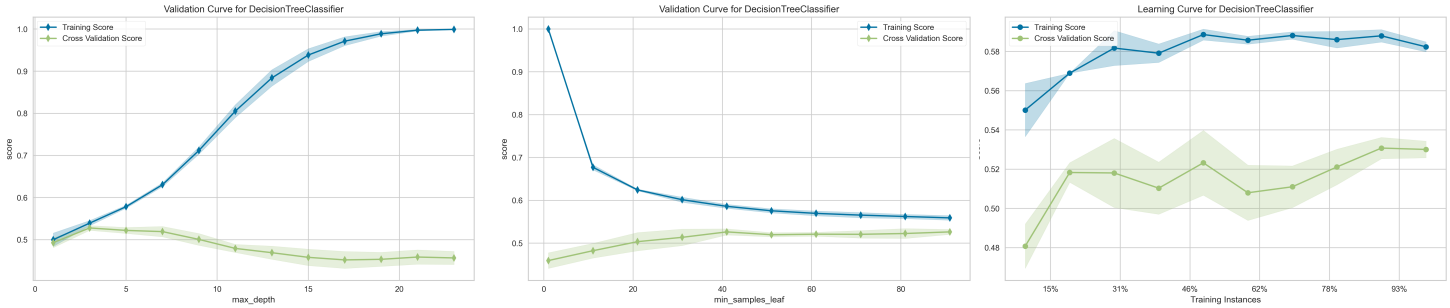
(b) Best Classifier

Figure 2—Precision-Recall Curves for Decision Tree on Credit Card Fraud Dataset

With the wine quality dataset, we observe a similar pattern in the behavior of decision trees concerning the `max_depth` parameter, as depicted in Figure 3a. However, it's important to note that even with an increasing maximum depth in the tree, we do not observe the cross-validation score reaching zero as in the previous case. This discrepancy is attributed to the fact that we are focusing on a different metric here (micro F1 score), and the dataset is considerably more balanced compared to the credit card fraud dataset.

Similarly to the first dataset, increasing the `min_samples_leaf` parameter leads to a better generalization of the classifier, resulting in a higher cross-validation score. In this case, we observe that the training score decreases more rapidly in the initial stages compared to its behavior on the credit card fraud problem.

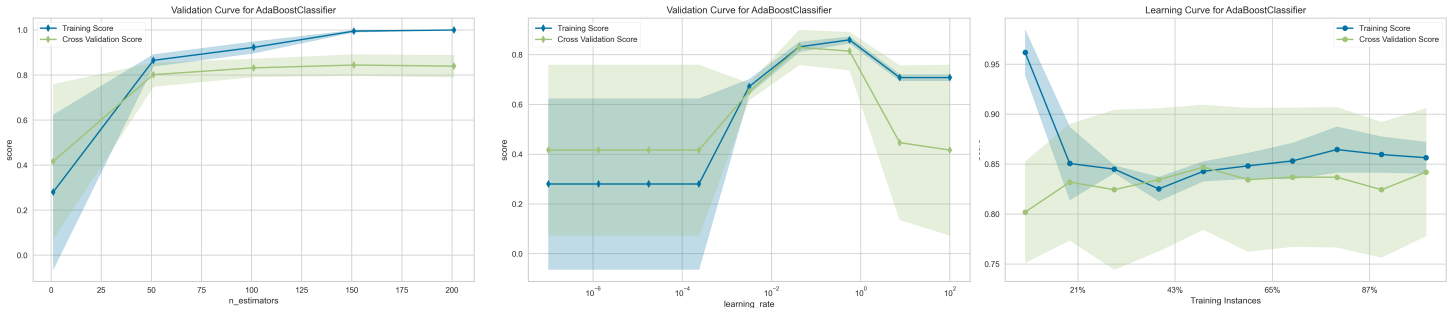
Upon tuning the hyperparameters `max_depth`, `min_samples_leaf`, and `ccp_alpha` and examining the learning curve in Figure 3c, it becomes apparent that the model has reached its limit. The training score levels off, indicating that further complexity may be needed, possibly necessitating a more sophisticated classifier than a decision tree to achieve higher performance.



(a) Validation curves for `max_depth` *(b)* Validation curves for `min_samples_leaf` *(c)* Learning curves of the best model

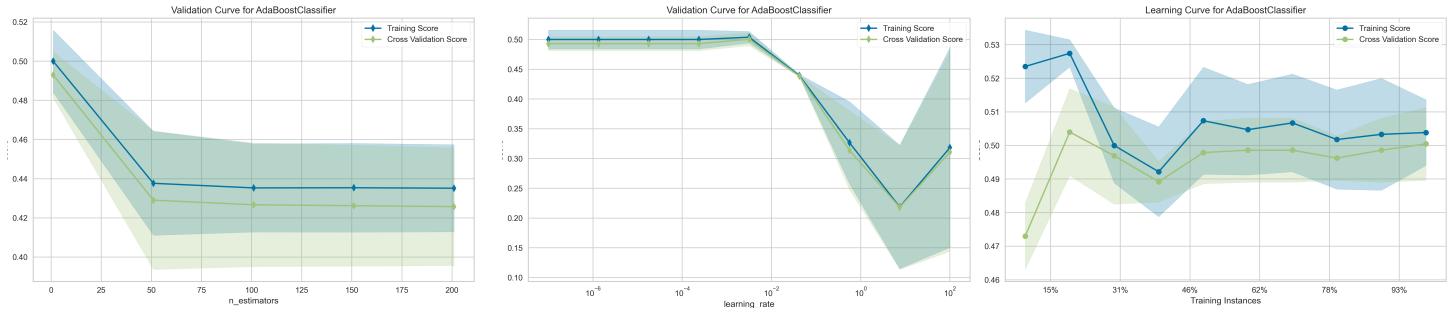
Figure 3—Validation & Learning Curves for Decision Tree on Wine Quality Dataset

5.2 Boosting



(a) Validation curves for `n_estimators` *(b)* Validation curves for `learning_rate` *(c)* Learning curves of the best model

Figure 4—Validation & Learning Curves for Boosting on Credit Card Fraud Dataset



(a) Validation curves for `n_estimators` *(b)* Validation curves for `learning_rate` *(c)* Learning curves of the best model

Figure 5—Validation & Learning Curves for Boosting on Wine Quality Dataset

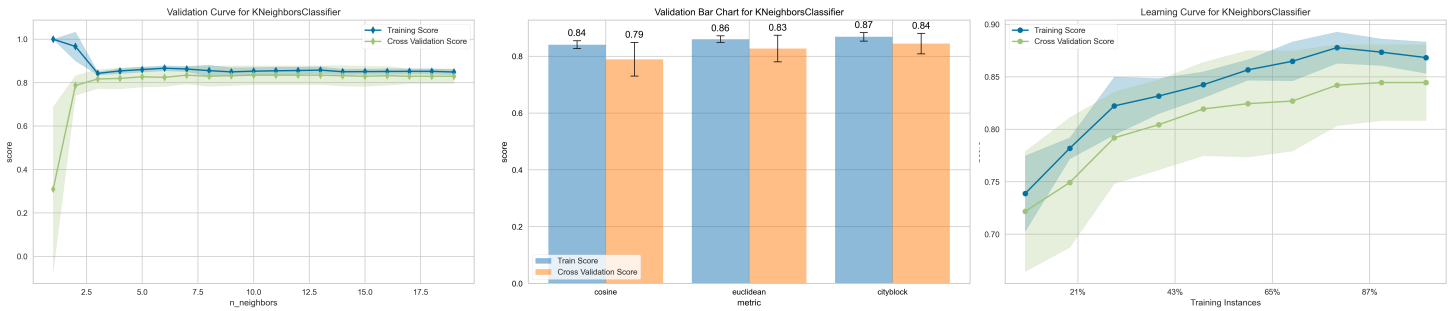
To analyse approach with Boosting, we used `AdaBoostClassifier` from `scikit-learn`. For the sake of simplicity, we decided to use default weak learner for all experiments, which is Decision Tree with maximum depth 1.

The two hyperparameters we chose to explore were `n_estimators` and `learning_rate`. Initially, our general expectation was that a higher number of weak learners would inevitably lead to better performance. This held partly true for the first dataset, where increasing the number of weak learners led to higher training and validation scores, reaching 1.0 and approximately 0.82, as depicted in Figure 4a. However, this trend did not hold for the wine quality dataset. As shown in Figure 5a, increasing the number of weak learners did not result in an improvement in either training or cross-validation micro F1 score. We hypothesized that this might be attributed to the default learning rate being ill-suited for this specific problem. Upon exploring the `learning_rate` hyperparameter, we observed that we could achieve a micro F1 score of at least 0.5 with a much lower learning rate than the default one (i.e., less than 10^{-2}, Figure 5b). Indeed, this proved to be the case: as indicated in Table 2, utilizing 150 weak learners with a learning rate of 0.0003 resulted in a performance slightly exceeding 0.50 on both the training and testing datasets.

On both datasets, the improvement achieved using Boosting compared to a simple Decision Tree is debatable. For the credit card fraud problem, although we were able to enhance the training score by approximately 2

Additionally, upon reviewing the learning curves (Figure 4c and Figure 5c) for both problems, it appears that we struggled to fit the data well with the Boosting algorithm. This observation raises the possibility of attempting more complex weak learners, such as Decision Trees with larger maximum depths[4].

5.3 KNN



(a) Validation curves for `n_neighbors`

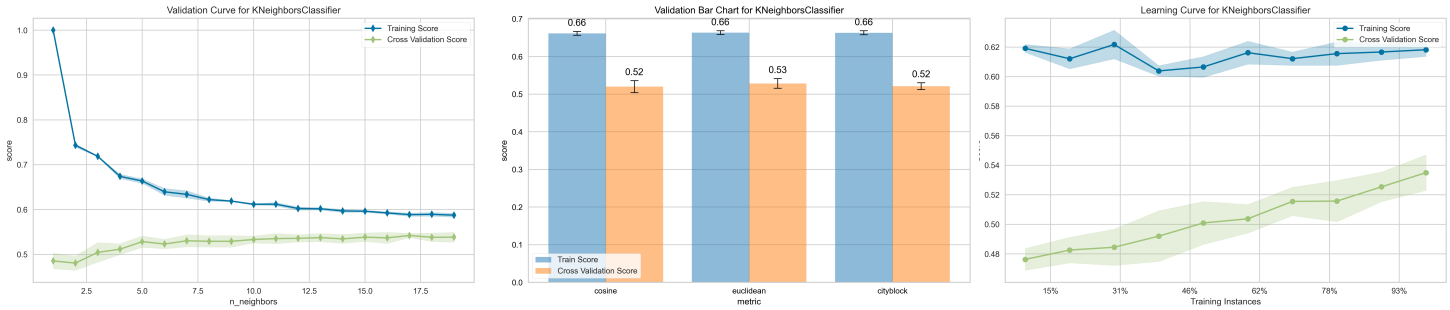
(b) Validation bar chart for `metric`

(c) Learning curves of the best model

Figure 6—Validation & Learning Curves for KNN on Credit Card Fraud Dataset

Before delving into the specifics of hyperparameters, it is noteworthy that even without any optimization, the baseline version of K-Nearest Neighbors (KNN) outperformed the baseline versions of Boosting for both problems. It surpassed the tuned version of Boosting in terms of recall at precision threshold of 0.9 on both the training and testing sets in the credit card fraud dataset. Additionally, it exhibited a better micro F1 score on the

4 The author of this report later verified that this idea indeed improved performance compared to regular Decision Trees, but unfortunately, there wasn't enough time to incorporate these changes into the report.



(a) Validation curves for `n_neighbors`

(b) Validation bar chart for `metric`

(c) Learning curves of the best model

Figure 7—Validation & Learning Curves for KNN on Wine Quality Dataset

training set (compared to the tuned Boosting version) in the wine quality dataset, and achieved a very similar testing score.

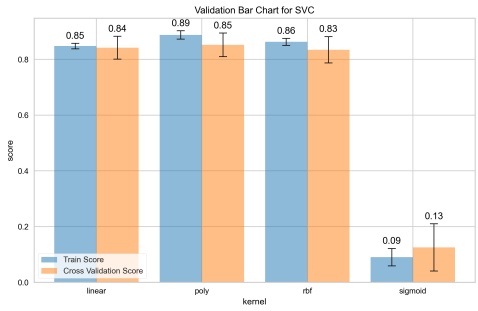
For KNN algorithm, we explored two hyperparameters: number of neighbors `n_neighbors` (essentially `k`) to base the classification result on and `metric` to calculate the distance.

For both datasets we noted similar pattern in training and validation scores with increase of `k` (see Figure 6a and Figure 7a); training score and decreases while cross validation scores increases and they converge. The decrease of training score for both datasets is reasonably justified by the fact that when we consider one closest neighbor in a set S (S is training set) to a sample A and $A \in S$, then A itself will be the closest sample to itself. With increase of `k/n_neighbours`, more additional data points are considered and more likely we get neighbours that have different label. At the same time, when evaluating dataset is not the one we trained the model on, increasing `k` allows us to generalise model better.

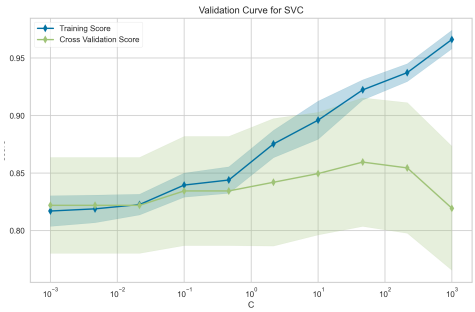
In both problems, the choice of a metric does not affect training or validation scores heavily (see Figures 6b and 7b).

In Figure 6c and Figure 7c, we can observe that the validation score improves with the addition of new samples on both problems. However, there is a notable difference between the two datasets. While adding more samples is unlikely to significantly benefit the credit card fraud problem, as both training and cross-validation scores have reached a plateau and converged, it is likely to be beneficial for the wine quality problem. In the latter case, the cross-validation learning curve has not yet converged.

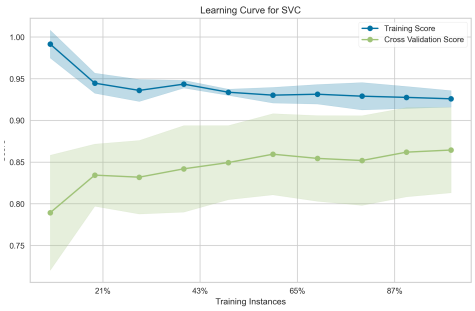
Another interesting distinction between the two datasets and their chosen metrics is that the training score for the second dataset does not show substantial movement along the training score curve. This suggests that it is challenging to achieve a higher micro F1 score with a KNN-based classifier on this dataset, implying that the model has likely reached its maximum performance potential.



(a) Validation bar chart for `kernel`

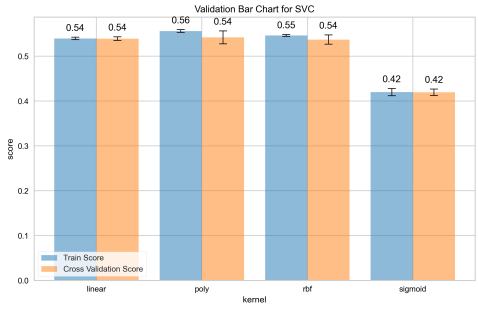


(b) Validation curves for `C`

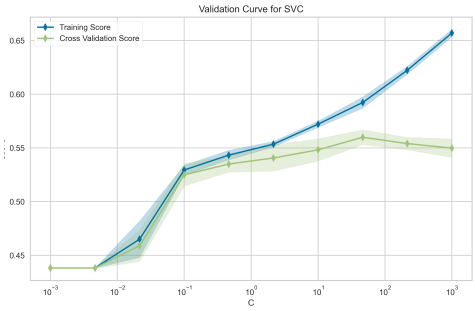


(c) Learning curves of the best model

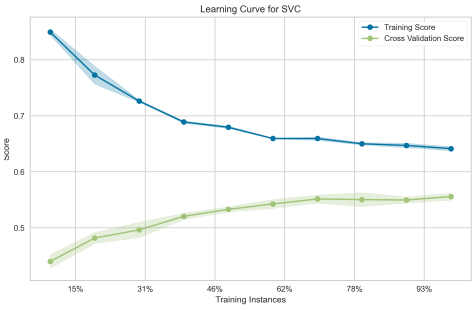
Figure 8—Validation & Learning Curves for SVM on Credit Card Fraud Dataset



(a) Validation bar chart for `kernel`



(b) Validation curves for `C`



(c) Learning curves of the best model

Figure 9—Validation & Learning Curves for SVM on Wine Quality Dataset

5.4 SVM

In the case of Support Vector Machine (SVM) based classifier, we considered two hyperparameters: type of `kernel` (how to convert the data to another dimension), `degree` specifying degree of polynomial used in polynomial kernel (if it is chosen) and C, parameter specifying penalty for misclassification,.

For both problems, sigmoid kernel shows the worst performance metrics, while others are all comparable between each other (Figure 8a and Figure 9a).

When exploring the hyperparameter C, we observed that with an increase in this parameter up to a certain point, both the training and cross-validation scores increase, along with the difference between them. However, beyond a certain threshold, the cross-validation score starts to decline while the training score continues to rise. This behavior suggests the potential occurrence of overfitting, particularly when the values of C become excessively high. According to Figure 8b and Figure 9b, overfitting becomes evident when $C \approx 10^2$ and higher.

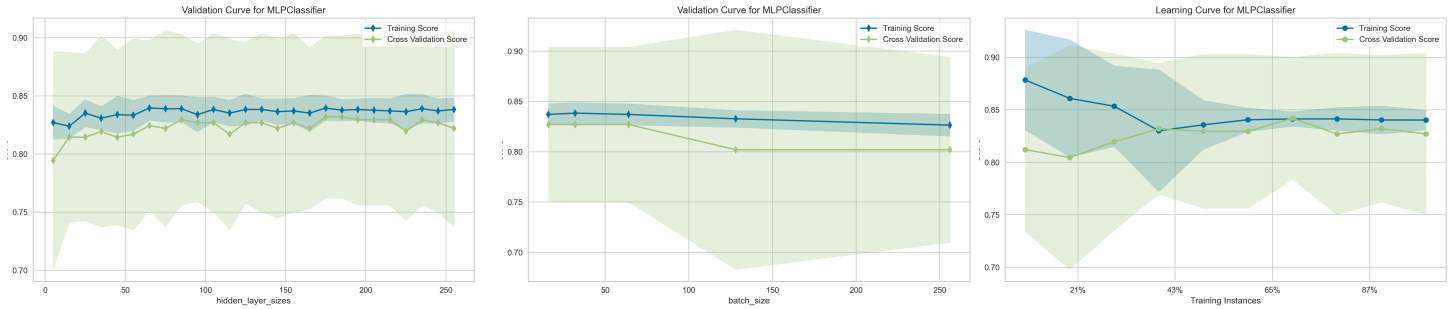
After more thorough experimentation, we also noticed that increasing C beyond this point on both datasets result in a big (almost exponential) increase in training time. This suggests that it is harder for algorithm to solve the problem and find appropriate separating hyperplane.

For the sake of space, we avoid going deep into analysis of degree of polynomial kernel. However we did notice that SVM show signs of overfitting on cards fraud and wine quality datasets with high values of `degree`. It is

worth noting though that for the first problem we can use polynomials of at most degree 3 without degrading in generalisation, while for wine quality we can set `degree` as high as 5.

Learning curves (Figures 8c and 9c) for both problems demonstrate similar behaviors: training score is slowly decreasing while cross validation is increasing and they both seem to converge. However, note that results are different. For the credit card fraud dataset, best SVM model has a much higher training R@PR(0.9) score than its baseline version, while testing scores are identical (see Table 1). This is an early sign of overfitting. Looking closer at the hyperparameters that were chosen this makes sense, since the value of C is high ($C = 71.9685$), which is close to a turning point ($C \approx 10^2$) discussed earlier. At the same time, on the wine quality dataset, SVM demonstrates best testing score, reaching micro F1 Score of 0.54 (see Table 2).

5.5 Neural Network

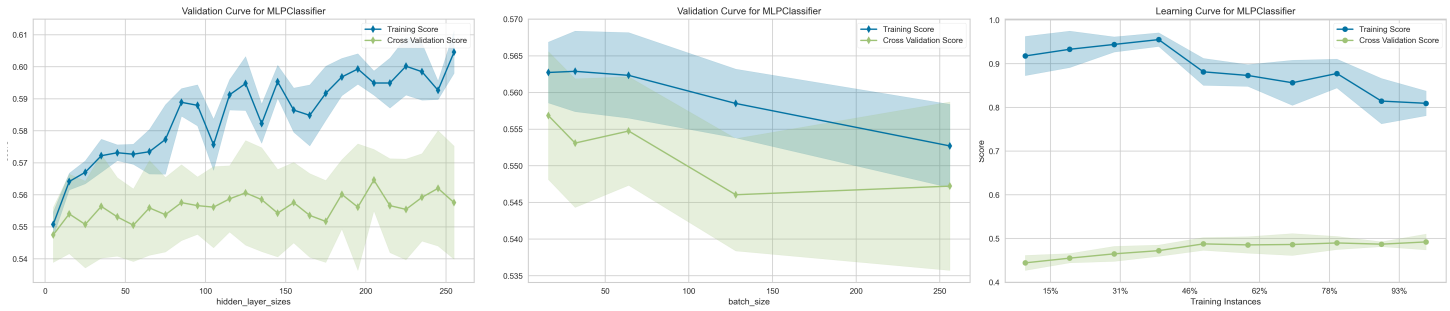


(a) Validation curves for `height`

(b) Validation curves for `batch_size`

(c) Learning curves of the best model

Figure 10—Validation & Learning Curves for Neural Network on Credit Card Fraud Dataset



(a) Validation curves for `height`

(b) Validation curves for `batch_size`

(c) Learning curves of the best model

Figure 11—Validation & Learning Curves for Neural Network on Wine Quality Dataset

Neural Network (`MLPClassifier` from `scikit-learn`) was by far the most difficult part of this project. Having a lot of hyperparameters that are possible to tune it is challenging to determine which ones make more sense to try. During the work, we overview multiple parameters including but not limited to `height` and `width` of hidden layers, initial learning rate, learning rate scheduling, regularization parameter `alpha`, `batch_size` and optimisation tolerance `tol`. We will overview only `height` and `batch_size` for the sake of space.

On the first dataset (Figure 10a), we can see that height barely affects the metric we are goaling on R@PR(0.9): both training and cross-validation score remain relatively stable. At the same time, for the second dataset

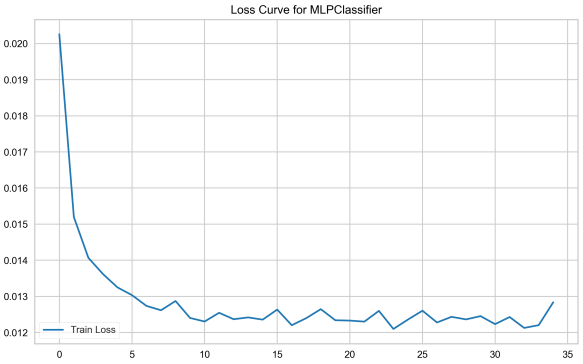
(Figure 11a), we can clearly see that when the number of neurons in a hidden layer increases, training score grows very fast as well. At the same time, cross-validation score almost remains in place showing only minimal signs of going up, suggesting that more complex architectures might lead to overfitting.

For both datasets we can notice from validation curves (Figures 10b and 11b) for `batch_size` that smaller batches might be more preferable to use.

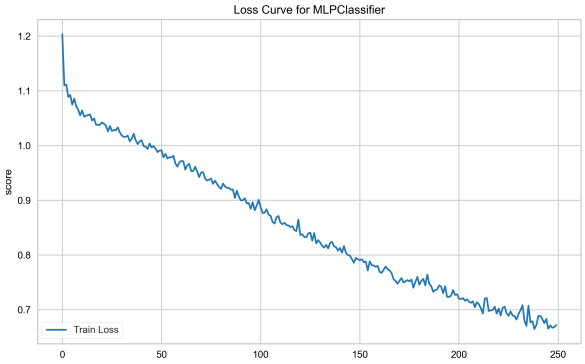
Despite making several attempts to narrow down the search space for each parameter, we ultimately found it necessary to conduct a comprehensive grid search to identify the best possible sets of hyperparameters.

Resulting learning curves depicted in Figures 10c and 11c for the Neural Network-based classifiers in both problems exhibit a common feature: the cross-validation score slowly increases with the number of samples. However, the training score learning curves display more distinct behavior, making it challenging to draw conclusive insights from these graphs.

Due to this complexity, we opted to utilize loss curves (refer to Figure 12). It is evident that for the credit card fraud dataset, the loss curve converges. This, coupled with the demonstrated performance (R@PR(0.9) scores of 0.8396 and 0.7627 on the training and testing sets respectively, as shown in Table **??**), suggests that the model fitting was successful and did not exhibit signs of overfitting. On the contrary, for the wine quality dataset, the loss curve does not converge; it continues to decrease. This, along with a very high training score and a relatively low testing score (as indicated in Table 2), indicates that the Neural Network overfitted the training dataset. It might have been caused by the fact that during grid search we chose a model with too many neurons in the hidden layer. As we noted earlier too complex architectures showed signs of overfitting.



(a) Credit Card Fraud Dataset



(b) Wine Quality Dataset

Figure 12—Loss Curves for Neural Network.

6 CLOSING REMARKS

In this project, we set out to examine the performance of five different machine learning algorithms across two distinct datasets. Our primary aim was to gain a practical understanding of how these algorithms operate in various scenarios, taking into account factors like the nature of the data and adjustable settings.

The machine learning methods under investigation encompassed Decision Tree, Boosting, KNN, SVM, and Neural Network.

The knowledge acquired from lectures, which elucidated the inner workings of these machine learning algorithms and underscored the importance of tweaking settings, played a pivotal role in guiding our experimentation and analysis.