CS7641 A1: A Comparative Analysis of Classifiers for Financial Tasks

Vinit Jogani
https://github.com/vinitjogani/classifier-comparison/

1 Introduction

Many real world problems can be formulated as classification tasks. Supervised Learning (SL) techniques offer effective solutions to such tasks, with automated and consistent decision making, sometimes with even better performance than humans. Among the many industries that have very successfully applied SL to challenging problems, a particularly prominent one is the financial industry. Especially given the greater need for interpretable predictions, there is a resounding preference specifically towards traditional SL algorithms which are often easier to understand. This paper presents a comparative analysis of 5 such learning algorithms on 2 datasets from the financial industry, and presents a reasoning about the selection, tuning and evaluation of machine learning algorithms for problems in this space.

2 Tasks

The first task we will use deals with Credit Score classification [1]. Based on various parameters like age, occupation, annual income, etc, the task is to classify customers into three buckets (Good, Standard and Poor) based on their creditworthiness as seen in historical data. This shall not only lead to more consistent credit score assignments, but also offer quick automated insights for the bank to make better, more efficient lending decisions.

The second task, on the other hand, is to predict, based on a variety of features, the conversion rate of tele-marketing efforts for a financial product (Term Deposits) to customers at a Portuguese bank [2]. The idea is that if, based on the characteristics of a customer like their education, loans, age, marital status, etc, we can predict which customers have a higher likelihood of signing up for term deposits, the bank can focus their marketing efforts only on those customers and thereby save costs.

Summary statistics for the datasets for both the tasks are included in the table below. Note that for categorical features, we perform one- or multi-hot encoding to convert them into numerical features, thereby the number of dimensions is generally higher than the number of features. We also perform scaling using the Power Transformer on numerical features to bring the feature values in a normal-looking distribution so that some models like kNN behave nicely.

	Credit Scores (CS)	**Term Deposits (TD)**
Records	100,000	41,188
Features	22	19
Dimensions	56	61
Classes	3	2
Class Distribution	30:50:20	90:10

Table 1: Dataset Statistics

These datasets are similar in some sense in that they each deal with similar features (age, income, defaults, etc.), have a mix of categorical and numerical variables, and have comparable feature dimensionalities. However, there are interesting differences between them. Obviously, there is the fact that one is multi-class whereas the other is binary. The datasets also have very different sizes, and therefore also the ratio of the features to samples is very different. The class distribution, and imbalance in particular, makes the datasets special when it comes to training as well as evaluation. There is also a sharp difference in the feature space that we discuss later through experiments.

3 Experimental Setup

In the next section, we evaluate five model families (kNN, decision trees, boosting, SVMs and neural networks), on the two tasks discussed earlier. To standardize readings across model types and parameters, we use a systematic protocol as explained in this section.

Firstly, we use an 80:20 split for training and testing. The testing data is not used during tuning or seen by the model in any way to ensure integrity of the final readings. During tuning, we use 5-fold cross validation on the training data to pick the best hyperparameters. That is, for each model and hyperparameter setting, we repeat the experiment 5 times with 20% of the data as a holdout set. The best parameters are ones with best average performance on holdout sets. This has a few advantages. First of all, by repeating each experiment 5 times, we ensure that the model is actually consistently performing well and there is less noise due to random initializations. Second, since there is significant class imbalance in both datasets, this ensures that decent coverage over the sparse classes is achieved during the various repetitions.

Perhaps the most critical aspect of the set up is the evaluation metric. Accuracy is a standard metric for classification but may be severely limited in our case due to the class imbalance. For example, it is trivial to get 90% accuracy on the Term Deposits dataset by always predicting False (the more frequent class). Instead, metrics like Precision / Recall, and the F1 score which combines those two, offer much more valuable information on such imbalanced datasets. However, these metrics are ultimately sensitive to the threshold that we select as a cutoff for labeling. This is not trivial and in practice, we would likely use a calibration step to make sure that the threshold aligns with business goals. Further, like in the Term Deposits task, it would be more useful to find which model achieves the best Precision at any given value of Recall (on average) as the total available capacity for the marketing efforts may depend on the budget which could frequently change. Due to all these reasons, a metric like Area Under Precision Recall Curve (AUPRC) would be the best fit as it captures a snapshot of the overall relationship between Precision and Recall at different thresholds. Other metrics like AUROC are also generally considered misleading on imbalanced datasets [3]. Nevertheless, we compute and show accuracies for all experiments below for comparison.

For all code in this paper, we use Python with Scikit-learn, Matplotlib, Pandas and NumPy. In all tuning experiments, all unchanged parameters are kept as the best found in preceding experiments or library defaults for untuned parameters.

4 Results

3.1 k-Nearest Neighbors

The kNN algorithm is an instance-based learning algorithm that finds "similar" (with respect to some distance metric) examples in the training set to predict labels on the testing set. This has special relevance in finance as this offers a simple mechanism for modeling "lookalikes".

In the first experiment, we evaluate the effect of K, i.e. the number of nearest neighbors to consider, and the weighting mechanism, i.e. whether neighbor labels are combined with a simple or weighted average. Quite interestingly, we see vastly different results for the two datasets.

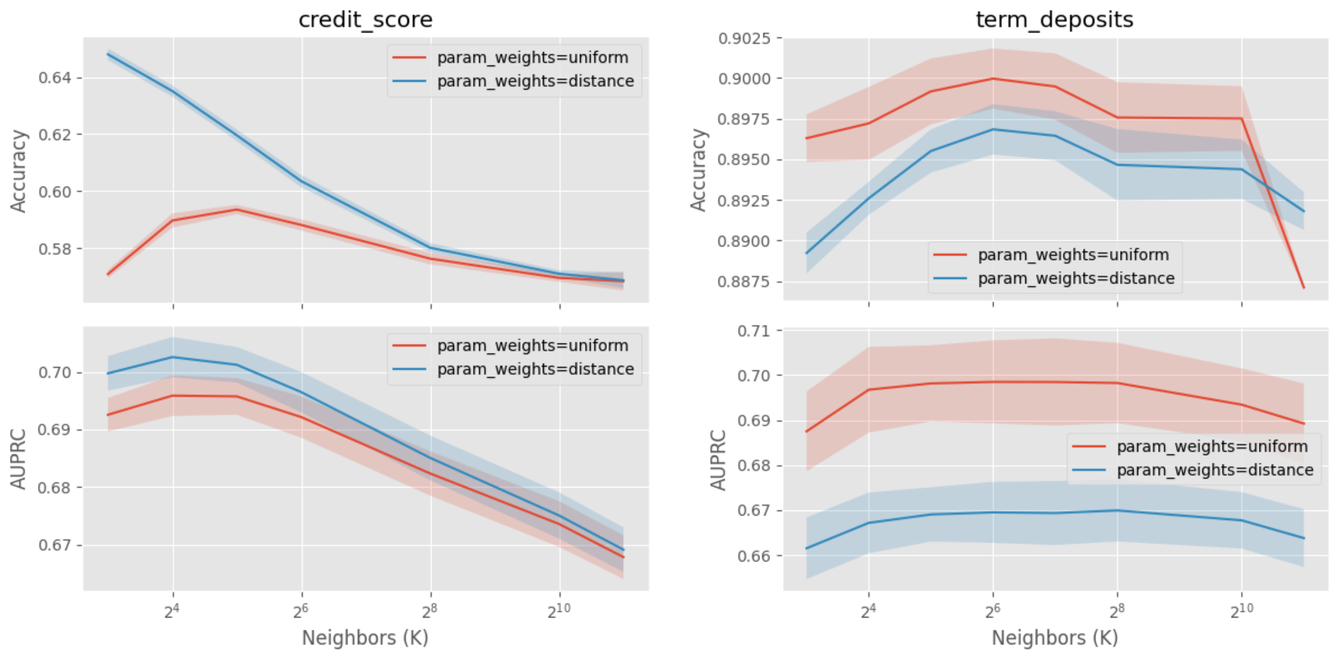

Figure 1: Varying k and weighting function in kNNs

Notice that the performance on the CS dataset peaks at K=16 and then degrades rapidly whereas for the TD dataset, the performance is best at K=64 but very similar in the K=16 to K=256 range. Intuitively, this might be an indicator of the smoothness of the class boundaries across the two datasets. That is, it seems that in the TD dataset, there are much larger hyperspheres having homogeneous labels. On the other hand, in the CS dataset, only the most close-by samples should be considered "similar" as there are sharper boundaries between the classes. This might also explain why different weighting functions work better in the two datasets. In the TD dataset, more close-by points share the same label (likely close to 256) so weighting uniformly would actually help reduce overfitting to a few very close-by points. However, since the class regions in the CS dataset are a lot more local instead of global, the closeness/distance becomes much more important which is why the distance-weighting likely works better.

This theme of locality of class regions comes up again in other models as well. It may seem that this is simply because the CS dataset has more classes so the labels are less homogenous across the feature space. However, empirically, the same pattern emerges even for one-vs-rest classification of the “Bad” Credit Score bucket. This indicates that it is something more inherent about the nature of the feature space itself.

In the second experiment, we evaluate the effect of three different distance metrics on the model performance. The TD dataset is a lot more invariant to the distance metric, likely because it has a lot of categorical features and the distance between binary encodings is roughly the same across the functions. On the other hand, the CS dataset seems to have a more pronounced preference for the L1 distance metric. There could be many reasons for this effect, but one possibility is that L1 distance is more robust to large differences across a single feature whereas L2 norm and cosine both would amplify it much more due to the multiplicative factors. In effect, while L2 penalizes the worst case error, L1 prioritizes close matches on as many features as possible even though some may be way off. For this problem, this seems to work better which indicates that there may be multiple “simpler” (in that they rely on fewer features) rules that influence credit scoring.

	Cosine	**L2**	**L1**
TD	0.6639	0.6699	**0.6703**
CS	0.7029	0.7026	**0.7487**

Table 2: Distance metrics in kNN

3.2 Decision Trees

Decision trees use sequential decision nodes to reduce some impurity criteria such that at the end, all samples in the same leaf are likely to share the same label. In general, the decision tree algorithm runs until there are no more features to split upon or the fit is perfect, whichever comes first. However, this usually leads to overfitting and so we prefer adding some pruning.

There are several hyperparameters that let us control the extent of pre-pruning including the maximum depth, minimum samples required for splitting and more, but for simplicity we choose to tune on the minimum samples required at a leaf node. This implicitly enforces a lower bound on the splitting condition but allows for a more lop-sided tree unlike the maximum depth method. Since we have class imbalance, we might need deeper, lop-sided trees.

Within the same experiment, we also try two different criteria used for optimization: Gini Impurity and Entropy. The results are shown in Figure 2. By and large, the results align with observations from the kNN model. Notice how the best model for the CS dataset uses less pruning, i.e. only a minimum of 8 samples on the leaf. This emphasizes how the nature of the feature space is more local and the subregions with homogenous labels are smaller. At the same time, without pruning, the model does overfit leading to lower performance on the validation set. The models on the TD dataset are once again more robust and perform well even at high levels of pruning which reiterates that there are larger regions of the feature space where labels are homogenous. When it comes to the criterion, there are only minor differences between the two but note that using Gini Index during training is approximately 30% faster which, along with its slightly better AUPRC, makes it the preferred choice.

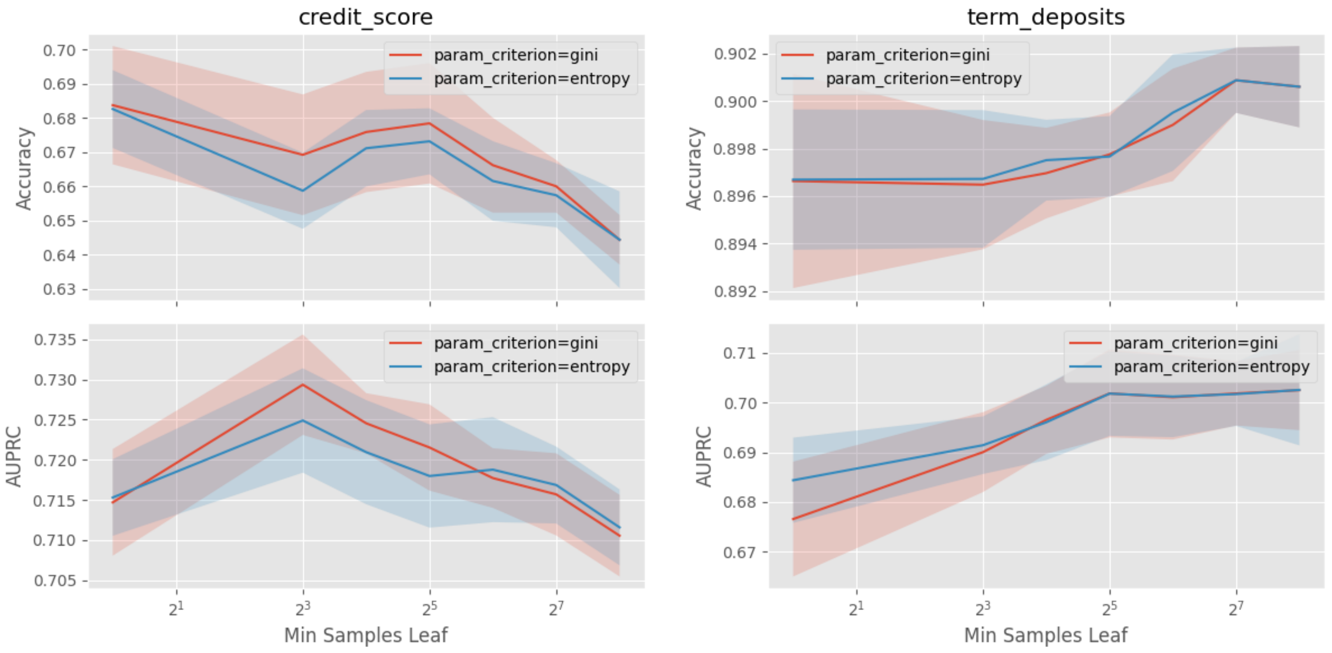

Figure 2: Varying pruning and criterion in Decision Trees

An interesting observation is that when there is no pruning, i.e. the leftmost points on the curves, the performance is not significantly far from the peak performance even though the model fits the training set perfectly. Intuitively, this may indicate that these tasks are like detecting anomalies – a few "lookalikes" may be enough evidence for classification, although it helps to have more. Also note how for the CS dataset, accuracy at 8 samples per leaf is actually lower than at 1 sample per leaf, even though AUPRC is higher. As mentioned earlier, this shows how accuracy may be misleading for imbalanced datasets as sometimes a model that better addresses business needs may actually have a worse accuracy.

3.3 Boosting

Boosting involves using multiple sequentially trained weak learners such that the weighted sum of the individual learners' predictions provides a good estimate of the label. This overcomes a significant limitation of decision trees that may be especially relevant in these datasets: when you make a decision at a node, you are locked into it for the rest of the path down to the leaf. For example, if you select a split at the root node, all subsequent splits are conditioned on that root split. If the underlying data generating process involves many sparse rules (i.e. using only a subset of features), Decision Trees cannot learn a simple representation for them.

Note that instead of AdaBoost, we found that Gradient Boosted Trees work better on both datasets during the initial exploration. While the idea of iteratively training weak learners is the same, Gradient Boosting uses deeper trees (as opposed to stumps) by default. The more important distinction, however, is that instead of re-weighting samples like in AdaBoost, Gradient Boosting will learn to iteratively fit the residual between latest predictions and the ground truth, and apply a constant weight (aka learning rate) to all trees. The reason this strategy works better may be that the datasets are intrinsically composed of slightly deeper rules so each tree can learn a bigger portion of it in a single run, and also perhaps log loss likely works better than reweighting for penalizing confidently wrong predictions in this case.

To tune this, we run an experiment trying different levels of pruning and boosting iterations.

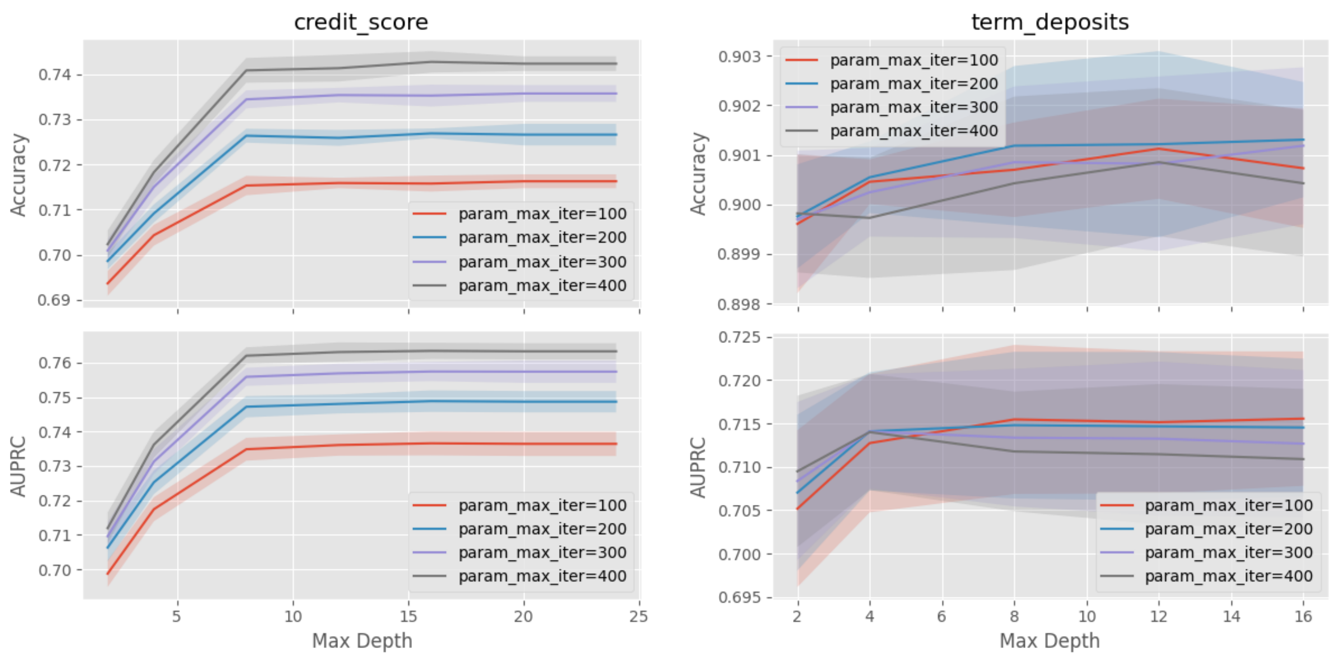

Figure 3: Experimenting with max depth and max iterations in Gradient Boosted Trees

When talking about kNN, we noticed how Manhattan distance performs better on the CS dataset. We reasoned it may be because there are simpler rules at play such that matching subsets of features exactly provides better predictions than focusing on all parameters. We see that reinforced in this experiment as the model seems to perform best when using many rules (400) that are each individually simple (max depth of 8). On the other hand, in the case of the TD dataset, the optimal number of rules is actually much smaller because there are fewer, more global rules applicable. In fact, with more trees and less pruning, there is actually evidence of overfitting indicating that the learner may not be weak enough.

3.4 Neural Networks

With the advent of deep learning, neural networks have become popular as these universal linear algebra machines that can fit any function. As it stands, there are many hyperparameters that we can train for these models but we focus on five: (1) depth, (2) breadth, (3) batch size, (4) regularization and (5) activation.

In the first experiment, we try different sizes of 1-, 2- and 3-layer networks. Firstly, there is more variance in these readings compared to other model types owing to the natural properties of neural networks. That said, we do see that deeper neural networks work best for the CS dataset whereas TD dataset tends to work better at 2 layers. The greater variability on the TD dataset also seems to suggest that there are probably many local minima that the network may get stuck in.

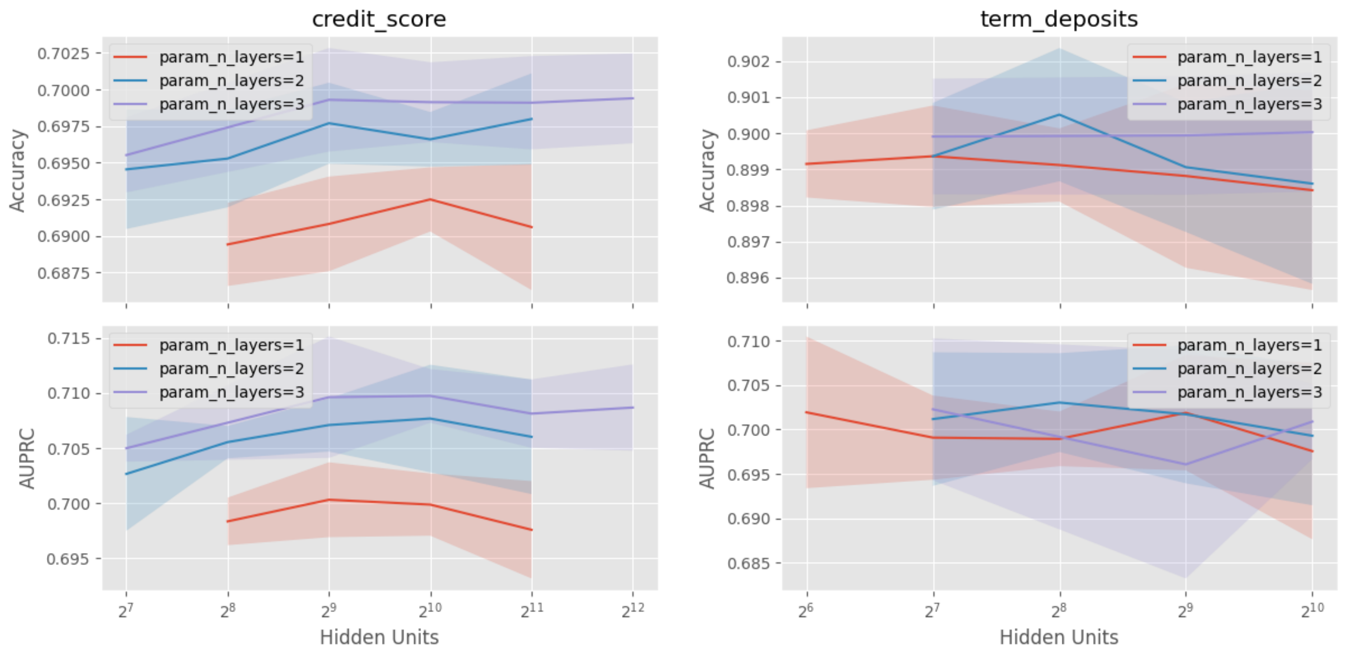

Figure 4: Finding optimal neural network size

In the second experiment, we try different activation functions with the optimal breadth and depth selection from the previous experiment. The readings are fairly close but one counterintuitive piece is that the CS dataset, which uses a deeper network, prefers tanh over relu even though the former is known to have a vanishing gradients problem. We reason that this issue was not as significant at 3 layers but may be more important for still deeper networks, and perhaps tanh helps pack more information into fewer units because it is less sparse.

	Tanh	**Relu**
TD	0.7018	**0.7033**
CS	**0.7118**	0.7061

Table 3: Comparing neural network activation functions

Finally, when looking at the learning curves (summarized in the next section) we found that there was some evidence of overfitting as more data actually led to worse test performance. We tried to address this by tuning the batch size and regularization parameters which are known to have an effect on overfitting. The peak readings, however, turn out to be fairly close and don't seem to drastically curb the overfitting. There are likely even more parameters that may address this, but we leave it out of scope for this paper due to limited computational capacity.

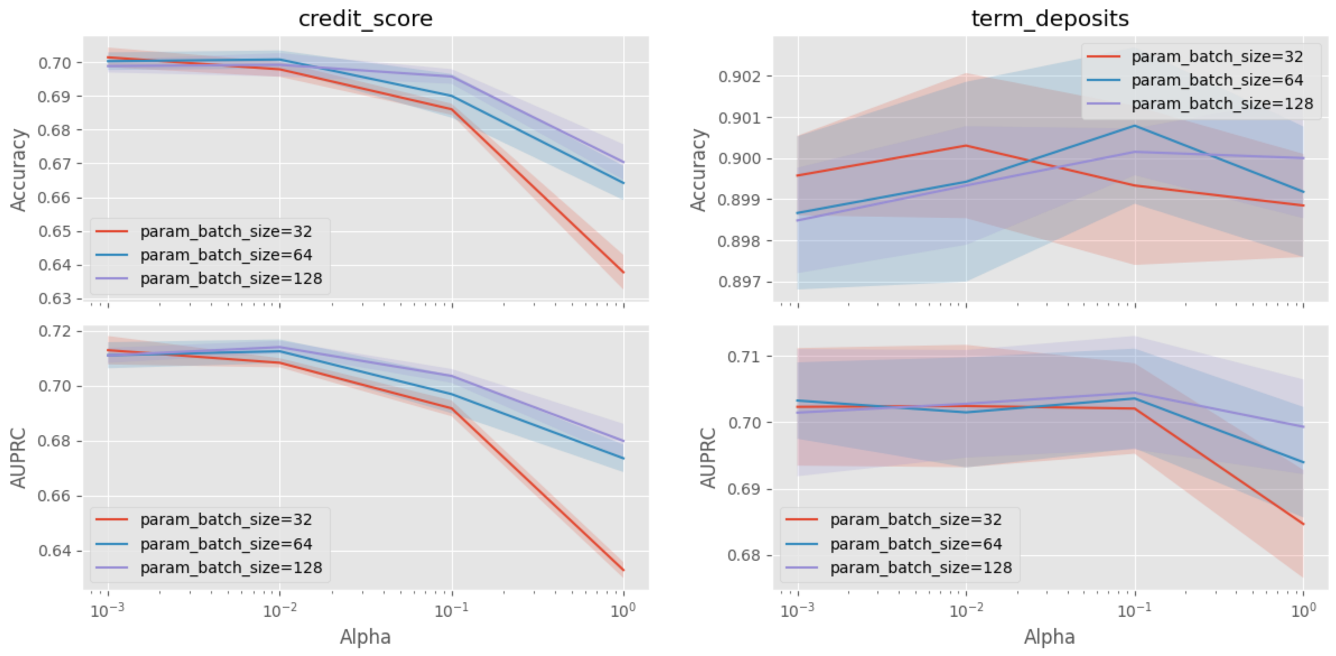

Figure 5: Tuning neural network regularization and batch size

3.5 Support Vector Machines

SVMs try to maximize the margin between the decision boundary and classes. The linear kernel would only perform well for datasets that are approximately linearly separable but with e.g. the RBF kernel, we can allow for spherical decision boundaries. However, most commonly used kernels would still divide the space into only two contiguous regions in some kernel-projected space (with multiple one-vs-rest classifiers for multi-class problems).

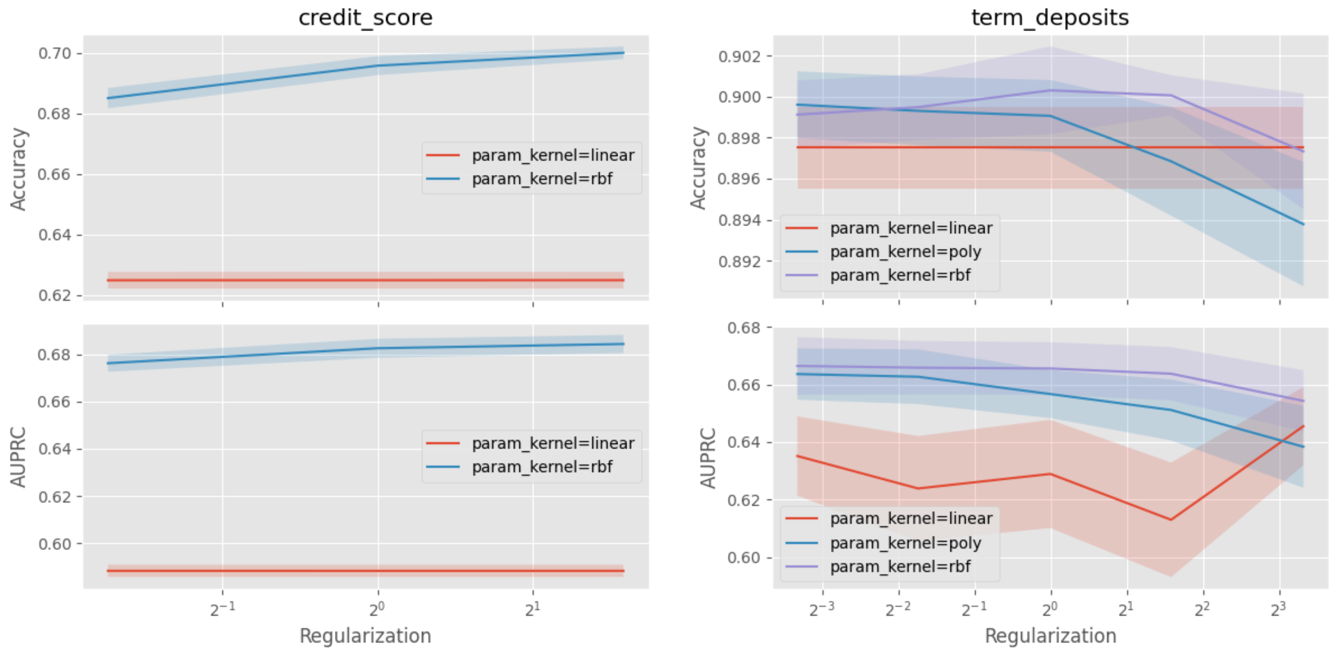

Figure 6: Comparing SVM kernels with different values of C

For the CS dataset, which seems to have many disjoint subregions for various classes, we would not expect it to work very well. However, there is little evidence that even the TD dataset is separable in such a way. To evaluate this, we run an experiment with various kernels and regularization parameters.

Note that training these SVMs was a lot more computationally intensive, especially with the large number of samples, so we had to pick fewer parameter combinations to tractably run the experiment. That said, these models also perform worse than the others on both datasets. Theoretically, there could be a kernel that would make the classes more separable for these datasets but it would be non-trivial to design such a function in practice. To demonstrate this, we create a quick experiment with a simple synthetic dataset as seen below with 2 features (on either axis) and colors to indicate classes/predictions.

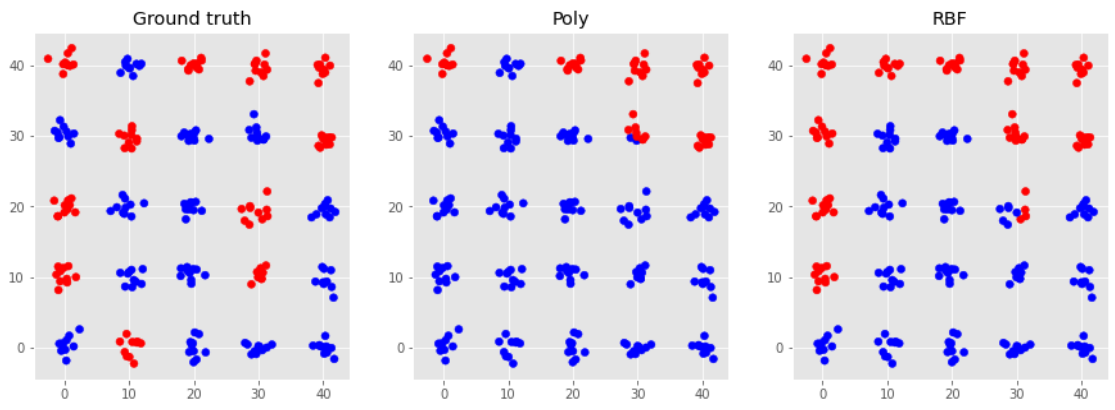

Figure 7: Demonstrating SVM limitations with synthetic dataset

Whereas a model like kNN or Decision Tree can fit this data perfectly, the usual SVM kernels cannot find a good fit. Financial classification tasks seem to leverage some form of rule-based or lookalike-based matching where local information is much more important than global information and this is where SVMs don't work very well.

5 Discussion

After tuning, we pick the best found parameters and evaluate the model types on the test sets. The results are summarized in the table below. They reveal some interesting tradeoffs.

Credit Scores Dataset				
Model	**Train AUPRC**	**Test AUPRC**	**Training Time**	**Testing Time**
kNN	1.0000	0.7592	**0.0054s**	80.0837s
Decision Trees	**0.9283**	0.7403	1.4161s	**0.0050s**
Boosting	0.8492	**0.7617**	33.0582s	0.2083s
Neural Networks	0.8843	0.7149	459.9918s	0.6367s
SVM	0.7406	0.6792	636.9347s	67.4639s
Term Deposits Dataset				
Model	**Train AUPRC**	**Test AUPRC**	**Training Time**	**Testing Time**
kNN	0.7252	0.6971	**0.0026s**	12.9797s
Decision Trees	0.7287	0.7077	0.1199s	**0.0016s**

Boosting	0.7491	**0.7210**	1.2508s	0.0063s
Neural Networks	0.7138	0.7078	23.0119s	0.0366s
SVM	**0.7752**	0.6712	48.1310s	4.7801s

Table 4: Final evaluation summary on test datasets

Unsurprisingly, kNNs are the fastest to train because they don't need to do much learning other than just index the data. With appropriate feature scaling, they also work reasonably well on both datasets and can be a simple and interpretable model for many financial tasks. However, one assumption in regular distance metrics for kNNs is that all features are equally important. Due to this, kNNs get close to being the best performer on the datasets but fall a bit short. Perhaps a learned distance function with feature weights can prove to be a strong candidate for these problems.

Decision Trees are the fastest when it comes to inference, but this is likely only due to the heavily pruned trees – in general, the sequential nature of inference on trees would make inference much more expensive on larger trees and harder to parallelize. Balancing under- and over-fitting on decision trees seems to be hard as seen in the learning curves – mostly because as mentioned earlier, all subtrees under a node are conditioned on all parent branches. Thus, especially in the CS dataset where there is evidence of many sparser rules, the tree inevitably becomes much larger or underfits due to heavy pruning.

Boosted trees seem to address this problem very well. Surprisingly, even with many estimators (400 in the CS dataset), they are very fast during training and inference. Plus, they achieve the best test performance on both datasets with relatively very little tuning. In fact, with more iterations, boosting continues to provide better performance on the CS dataset whereas the neural network does not improve much after 500 iterations as seen in the figure below. That said, it was trickier to tune it on the TD dataset with more iterations actually leading to worse performance and pruning doesn't seem to help much either. This may further indicate that the TD dataset focuses on a few global/deeper rules instead of many simpler rules.

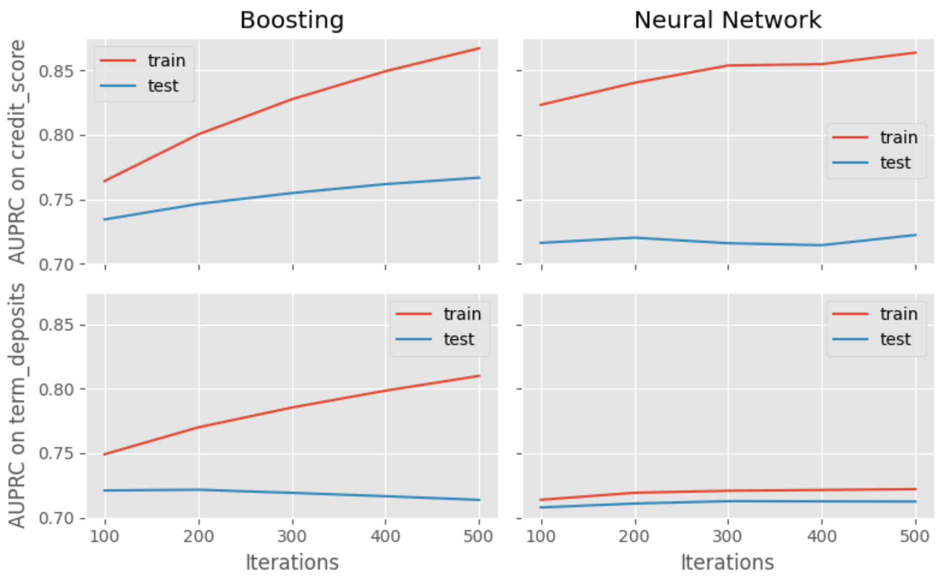

Figure 8: Training and testing errors as a function of iterations

Neural networks are powerful, but have many salient hyperparameters that can each affect the training significantly. They are thus hard to tune and they fall victim to overfitting rather easily. Further, they are more sensitive to random initializations and are also less stable across iterations. We see all of these properties in the learning curves.

SVMs overall are really slow to both train and test, and they scale badly as we increase the regularization parameter (C) or the number of samples. The non-linear kernels seem to be the culprits, as the Scikit-learn implementation stores an NxN matrix of pairwise distances due to the non-linearity. Neural networks take a while to train but inference is reasonably fast and can be sped up further with GPUs.

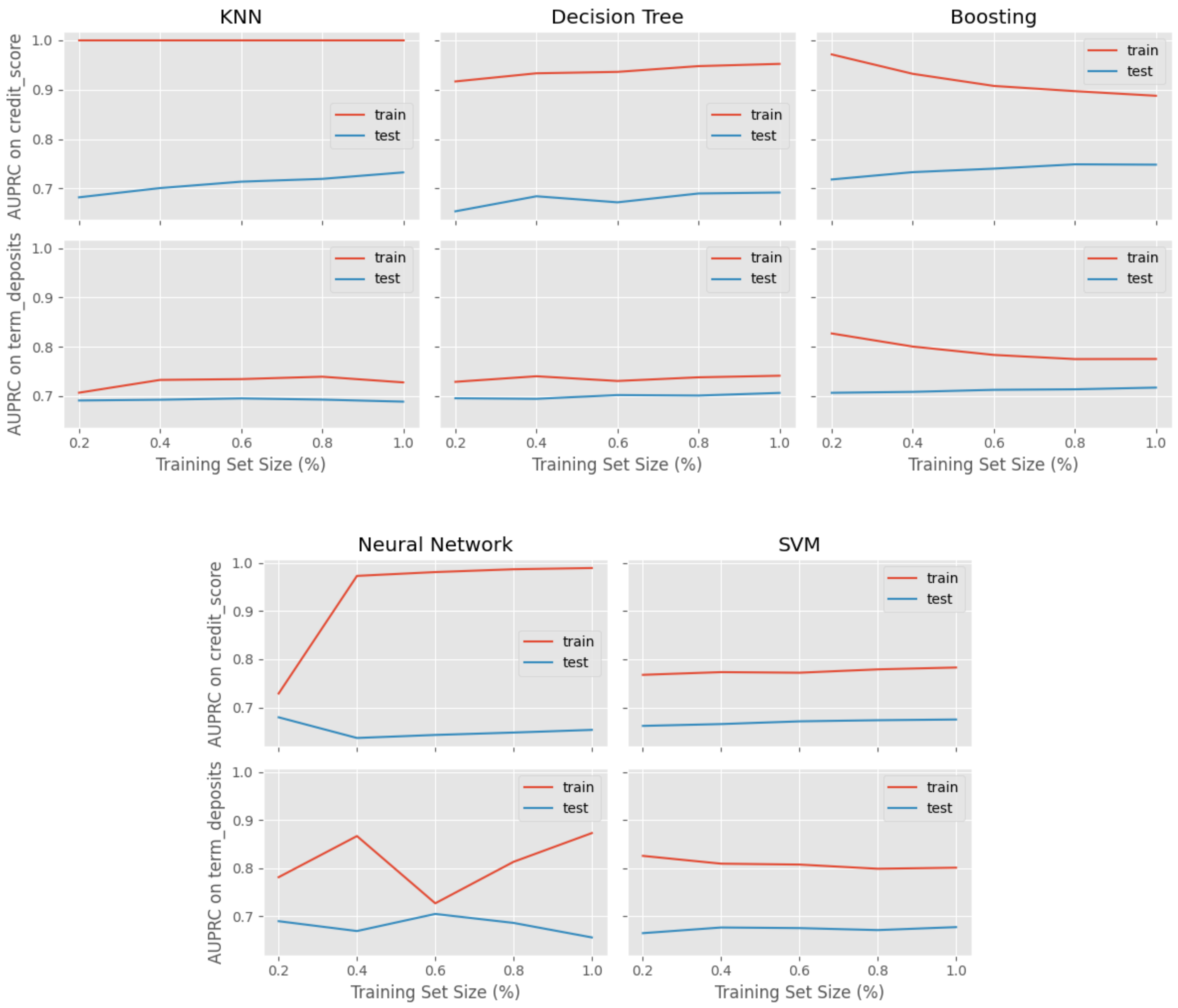

Figure 9: Training and testing errors as a function of training set size

On the CS dataset, the best models are kNN, Decision Trees and Boosting – all of which can fit well to a small number of samples with a certain class (e.g. in anomaly detection). This seems to be characteristic of the dataset, as the customers with a bad credit score are sparse but provide a strong signal. In other words, we showed through a variety of experiments how the dataset has more local patterns, whereas the individual patterns may be simple which is why Boosting seems to work best.

On the TD dataset, all models except SVMs achieve close-by performance. However, this largely seems to be due to inherent noise in the dataset given that all models tend to underfit even with optimal hyperparameter selections. This can be seen in all learning curves (in Figure 9) where the train/test errors are much closer than on the CS dataset. This may be referred to as Bayes error where by nature, the observations have some randomness or alternatively, we don't have access to all variables. This seems likely for tele-marketing as the bank might not have complete information about all factors affecting the customer's behavior. We see that Boosting does lower the bias, reducing the underfitting with a greater ensemble of models, but is still ultimately restricted by the data. Neural networks do work well on this dataset and perhaps with even more tuning, e.g. adding learning rate decay, there may be scope for improving performance even further.

To summarize, we were able to see two different parts of a general spectrum of datasets: the first kind relies on more local information and has sharper class boundaries, whereas the second kind has more global and smoother boundaries. We saw how the position of a dataset on this spectrum can affect the choice of the model significantly. Even for Boosting, which is a swiss-army-knife that works well on both datasets, the hyperparameter choices would greatly vary depending on the nature of the problem. Future work could quantify using more specific metrics where datasets lie on this spectrum. Further, we discussed limitations of the various models in terms of performance as well as training/tuning/testing time and difficulty, which is often a practical concern. Future work can involve crafting special model types e.g. kNN with learnable weights that better leverage the nature of the datasets.

6 References

[1] R. Paris, "Credit Score Classification," Kaggle, 22-Jun-2022. [Online]. Available: https://www.kaggle.com/datasets/parisrohan/credit-score-classification.

[2] P. Bhowmik, "Bank marketing campaign subscriptions," *Kaggle*, 12-Feb-2021. [Online]. Available: https://www.kaggle.com/datasets/pankajbhowmik/bank-marketing-campaign-subscriptions.

[3] D. Rosenberg, "Imbalanced data? stop using ROC-AUC and use AUPRC instead," *Medium*, 07-Jun-2022. [Online]. Available: https://towardsdatascience.com/imbalanced-data-stop-using-roc-auc-and-use-auprc-instead-46af4910a494 .