NLP and financial marketing campaign via Neural Nets, AdaBoosting, SMVs, KNNs and Decision Trees

Damian Abasto, damian.f.abasto@gmail.com

***Abstract*—The performance of Decision Trees (with pruning), Neural Networks, AdaBoosting, Support Vector Machines and K-Nearest Neighbors was analyzed over two different binary classification problems, the first one related to the financial industry (predicting if a given customer would buy a financial product based on characteristics of the customer and of the selling campaign), while the second one related to natural language processing (predicting the sentiment of a given tweet on stocks). The two sets are very distinct, with the first one exhibiting a mixture of numerical and categorical data, with a total number of 15 features, while the second one related to text classification, which posses unique challenges in terms of numerical representation of text information and high number of features. We analyzed various learning and validation curves as a function of different hyperparameters, which helped us identify the best algorithms in terms of their bias-variance tradeoffs. Finally, hyperparameter tuning was performed to select optimal parameter values (which we contrasted with the validation curves), and their final performance on the testing set and confusion matrices was analyzed.**

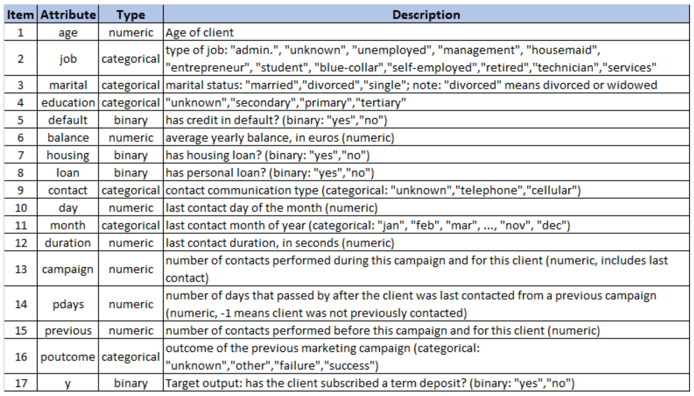
I. Introduction

II. Bank Marketing Dataset

This data set represents a binary classification problem to determine the best client selection strategy for a banking institution to sell long term deposits via phone calls. The objective is try to predict whether or not the customer will subscribe to a term deposit product based on various attributes of the customer. This problem is interesting from the machine-learning standpoint, as it contains a mixture of numerical and categorical features, and is relatively unbalanced, with only 11.7% cases belonging to the positive label (see Section II-A). The classification problem has also practical implications, as an effective algorithm that can predict which customer is worth pursuing can potentially increase the efficiency of the campaign, by reducing the amount of time spent with customers that are likely not to purchase a term loan, and increase sales revenue by better targeting the existing customer base.

A. The data

The data was obtained from the UCI Machine Learning repository [9] (see also [10]), file "bank-full.csv". The data consists of 45,211 instances, with 16 attributes. None of the instances contain missing data. Figure 1 presents a summary of the attributes, their types and meanings (extracted from the file "bank-names.txt", [9]). The target variable is binary, with values "y" (customer brought the financial product) or "no" (customer did not buy the financial product) after the phone call. There are 7 variables of numerical type, and 9 of categorical type. Attributes 1 through 8 come from the existing bank client database, 9 through 12 correspond to the data collected from the last time the customer was contacted.



Item	Attribute	Type	Description
1	age	numeric	Age of client
2	job	categorical	type of job: "admin.", "unknown", "unemployed", "management", "housemaid", "entrepreneur", "student", "blue-collar","self-employed","retired","technician","services"
3	marital	categorical	marital status: "married","divorced","single"; note: "divorced" means divorced or widowed
4	education	categorical	"unknown","secondary","primary","tertiary"
5	default	binary	has credit in default? (binary: "yes","no")
6	balance	numeric	average yearly balance, in euros (numeric)
7	housing	binary	has housing loan? (binary: "yes","no")
8	loan	binary	has personal loan? (binary: "yes","no")
9	contact	categorical	contact communication type (categorical: "unknown","telephone","cellular")
10	day	numeric	last contact day of the month (numeric)
11	month	categorical	last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")
12	duration	numeric	last contact duration, in seconds (numeric)
13	campaign	numeric	number of contacts performed during this campaign and for this client (numeric, includes last contact)
14	pdays	numeric	number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted)
15	previous	numeric	number of contacts performed before this campaign and for this client (numeric)
16	poutcome	categorical	outcome of the previous marketing campaign (categorical: "unknown","other","failure","success")
17	y	binary	Target output: has the client subscribed a term deposit? (binary: "yes","no")

Fig. 1: Description of attributes and target variable.

Figure 2 shows the overall distribution of customers who bought the term loan after the telemarketing campaign. Out of the 45,211 instances, only 11.7% of those represent cases where the customer did buy the term loan. Figure 3 shows that the distribution of the target variable depends on the different values of each of the categorical variables. For example, of the 815 instances that defaulted, only 6.4% accepted a term loan, compared to 11.8% of the population of 44,396 that did not default. Similarly, of the 7,244 instances that had an existing personal loan, 6.7% accepted a term loan, compared to 12.65% for the 37,967 instances that did not have a personal loan.

Similarly, Figure 4 shows a histogram of the customers decision on the term loan across the numerical attributes. A non-uniform distribution of the response variable with respect to the numerical variables is observed.

As stated in UCI Machine Learning repository [9], the variable "duration" represent the duration of the phone call and is known only *after* the interaction with the customer has taken place. In addition, it highly affects the target variable "y" (since if no interaction with the customer has taken place, then `duration`=0 and `y` = "no"). This variable has therefore been removed from the rest of the analysis, since the purpose is to have a realistic predictive model depending on attributes that can only be obtained before the call to the customer is

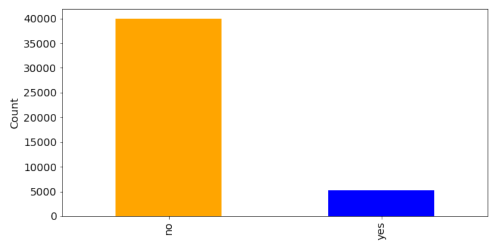


Fig. 2: Distribution of customers who bought the term loan deposit ("yes") versus those who did not ("no"). Only 11.7% of the customers bought the term loan, compared

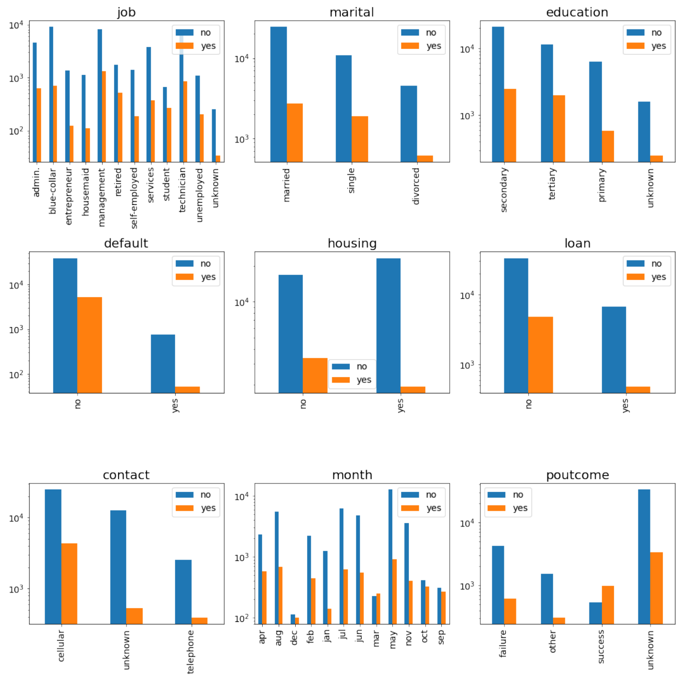


Fig. 3: Distribution of customers who bought the term loan deposit ("yes") versus those who did not ("no"), across each of the categorical attributes: 'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month','poutcome'. We see that the distribution is non-uniform and depends on the values of each of the categorical variables. The counts in the y-axis are in log-scale.

made.

B. Train-Test data split

The totality of the data is split into a train and test data sets. The train data set is used to train the different machine learning algorithms, while the test data set will be used to estimate the final out of sample performance. In the course of hyper-parameter tuning of the different algorithms, the train data set will be spitted further, via cross-validation or other means.

An important assumption of the machine learning algorithms is that the data is independent and identically distributed

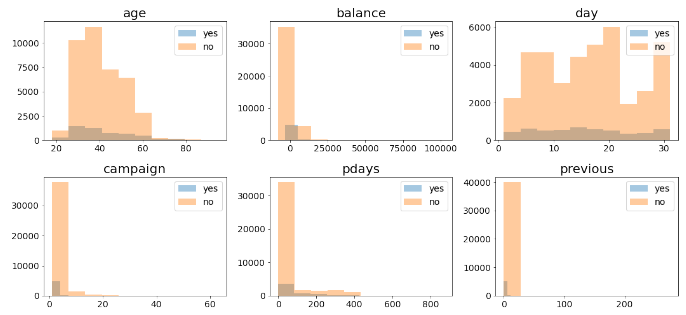


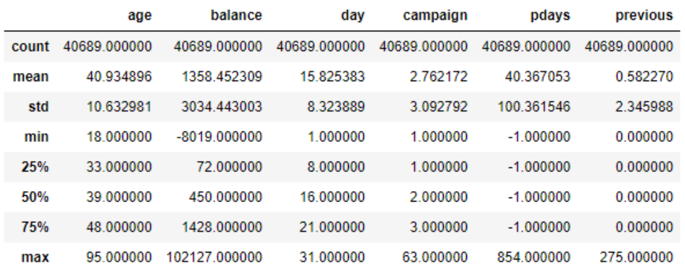
Fig. 4: Histogram of customers who bought the term loan deposit ("yes") versus those who did not ("no"), across each of the numerical attributes: 'age', 'balance', 'day', 'campaign', 'pdays', 'previous'. We see that the distribution is non-uniform and depends on the distribution of the numerical variables.

("i.i.d"). Therefore, an effort should be made for the test sample to be as close as possible to the train sample. The data exploration above revealed that the distribution of term loan acceptance depended to various degrees on the underlying values of different attributes. Not only the overall distribution of term loans is non-uniform, with 11.7% of the total instances accepting a term loan, but that percentage drops to 6.4% for the subset of instances where there was a default, or 6.7% for the subset with an existing personal loan. The personal loan therefore has a strong influence over the distribution of term loans acceptance. A *Stratified* sampling [5], based on the loan categorical variable, was performed to do the train-test split, to ensure that the train and test distributions are as close as possible, in particular for the loan variable, that has a strong influence on the target variable.

C. Data representation

Categorical variables were encoded into a numerical representation for the purposes of training and testing the machine learning algorithms. We use one-hot encoding for them, whereby each of the categorical variables is mapped to an n-dimensional vector, where n is the number of distinct values that the categorical variable can have, and where all of the entries in the vectors are zero, except for one entry equal to 1, indicating the value of the categorical variable. For example, for the case of `contact`, a value of "cellular" would be mapped to a three dimensional vector (since there are three possible values for `contact`), of the form $(1, 0, 0)$. In addition, the is a wide variety of scales and ranges across numerical attributes. We have normalized the numerical data via scikit-learn `StandardScaler` [2] to account for this and allow machine learning algorithms to perform better [5]. The standard scaler substracts the mean from all the variables, and divides them by their standard deviation.

We have only used the *train* set to obtain the mean and standard deviations to do this normalization (or any other type of data transformation), and only then apply it to the test set, in order to avoid test data leaking into the training set and affecting the results of the machine learning algorithms and subsequent analyses.



	age	balance	day	campaign	pdays	previous
count	40689.000000	40689.000000	40689.000000	40689.000000	40689.000000	40689.000000
mean	40.934896	1358.452309	15.825383	2.762172	40.367053	0.582270
std	10.632981	3034.443003	8.323889	3.092792	100.361546	2.345988
min	18.000000	-8019.000000	1.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	1.000000	-1.000000	0.000000
50%	39.000000	450.000000	16.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	63.000000	854.000000	275.000000

Fig. 5: Summary of some statistics for the numerical variables. Notice the wide variety of scales and ranges across numerical attributes. We have normalized the numerical data via scikit-learn `StandardScaler` [2] to account for this.

D. *Performance measure*

Accuracy (as measured by the tota number of instances with loans, divided by the total number of instances), is typically a poor measure of performance, especialy in situations where the data is *skewed*, so that a subset of classes are more frequent than others, [5]). In our case, 11.7% of the instances do contain customers that accepted a term loan. Therefore, a simple classifier that predicts "no" always will have an accuracy of $100\% - 11.7\% = 88.3\%$. The area under the curve "AUC" for the receiver-operating characteristic curve ("ROC curve") was chosen as the performance metric for the different binary classifiers, as it captures the trade-off between the recall (or true positive rate, which measures the fraction of positive labels correctly detected by the classifier), versus the false positive rate (the ratio of negative instances that are incorrectly classifier by the model). Increasing the recall increases the false positive rate, and viceversa.

E. *Cross validation*

In addition, cross-validation with $K = 10$ folds will be used through the hyperparameter tuning of the different models to assess the out of sample performance of the models. This effectively produces validation sets that are 10% hod-out of the data, where the model is evaluated, after being fitted to the 90% of the data. By repeating this 10 times over non-overlapping segments of the data and averaging the `auc` metric, we can get a better estimate of the true out of sample performance of the model, for a given choice of hyperparameters.

F. *Parameter tuning*

The parameters of the different learning algorithms will be optimized via cross-validation over random sets of the parameter space, using `scikit-learn` `RandomizedSearchCV`. In this way we can have better control on the number of total iterations, reduce the computational time, while still covering the space of possible values for a subset of parameters chosen.

G. *Algorithms*

We used the library `scikit-learn` [2] to analyze the performance of Decision Trees, Neural Networks, Boosting, Support Vector Machines and K-Nearest Neighbors on this data. We implemented our own Pruning algorithm for Decision trees, sub-classing sklearn `DecisionTreeClassifier`.

1) Decision Trees: Decision trees were fitted to the present data set using `scikit-learn` `DecisionTreeClassifier` [2]. The `DecisionTreeClassifier` was sub-classed and its `fit` modified so it could accommodate pruning, as way to overcome over-fitting. Pruning was implemented by over-riding the `fit` method from the `DecisionTreeClassifier` in a subclass and by removing leaf nodes from the decision tree, after it was fitted to the training data. The number of nodes to be pruned can be controlled by an external parameter called `pruneLevel` in our code, which effectively controls the number of leaf nodes removed, starting from the bottom up. The pruning was done from the bottom up since the over-fitting typically happens in the leaf nodes, as lower nodes can sometimes produce unnecessary splits to fit the noise in the data. Figure 6 shows our implementation of pruning for the iris data set. As can be seen, the higher the level of pruning, the simpler the resulting trees are, indicating that the pruning algorithm was implemented correctly.

Figure 7(a) shows the learning curve for the Decision Tree, the validation curve as a function of the maximum depth of the tree (second plot) and as a function of the amount of pruning (third plot), and finally the fitting times versus the training sample size. Across the plots in Figure 7, the solid curves represents the mean of the given metric (training score, validation score or fit times), while the shade represents $\pm\sigma$ one standard deviation from it. The perfect training AUC score of one across all sample sizes, with very small cross-validation scores on the out of sample, validation sets, highlights large *over-fitting* of the Decision Trees. Since the parameters of the `DecisionTreeClassifier` were not specified in the constructor when building the learning curve, there was no limit to the depth of the tree, which could then split until all the leaves were pure, fitting potentially to the noise in the data, and causing the overfit. The validation curve as a function of the maximum depth of the tree also points to overfitting, as the depth of the tree increases, the training curve quickly reaches to one, with the subsequent drop on the validation sets. Notice how at the smallest `max_depth`=10 the model performs better in terms of validation score, since it avoids fitting to closely to the data, and thus, limits the level of overfitting. The validation curve as a function of prune level also exhibits over-fit, as the Decision tree achives perfect AUC in the training data, with much lower performance on the validation set. Notice however the "inverted U" shape in the the validation curve as a function of prune level, with a maximum around 500, at which point we achieve the best generalization with the smallest variance (given by the gap between the train and corss-validation score curves). Figure 7(a), last plot from left, indicates that the Decision Tree fitting times tends to grow linearly with the sample size.

2) Neural Networks: Neural networks were fitted to the data set using `scikit-learn` `MLPClassifier` [2], which implements a multi-layer perceptron (MLP) that outputs probabilities for the binary classes using a softmax function at the end of the network. Figure 7(b) shows the learning curve for a MLP with one hidden layer of 10 neurons, two validation curves, one as function of the number of hidden layers (each layer with 10 neurons each), and another as a function of neurons in a single hidden layer (width), and finally, the performance plot (Validation set AUC as a function of fitting time) for the MLP with one single layer with 10 neurons. The learning curve shows a decreasing auc score on the training size, while the validation score increases. This indicates that increasing the sample data does provide better generalization of the model, with the gap between them narrowing. The gap between the curves signals the variance of the model. Given that the curves do not show a sign of reaching a plateau, this means that the neural network would benefit from a larger data set than currently available.

The validation curve for the neural network as a function of the number of layers is analyzed next (second plot from the left). Each layer consisted of 10 neurons each, and the training and validation auc scores were monitored as a function of the number of layers. Initially the training auc increases, with decreasing validation auc, signaling overfitting. After 3 layers, both the training and validation sets show a drop in auc, with a higher variability of the auc scores in the cross validation sets (notice the widening of the standard deviation). This signals that the model severely underfits the data (since the training score decreases), or that it has failed to converge. Given that at 3 layers the neuron has $3 \times 10 = 30$ neurons, the large number of parameters would lead to overfit, with increasing training scores as a function of layers. Given that the opposite behavior is observed, we believe this signals an issue with the depth of the network, either indicating a vanishing gradient problem (common to deep networks, see [6]), or that the back-propagation algorithm did not converge[1]. The validation curve as a function of the number of neurons in a single layer (7(b), third plot from the left) shows a clear sign of over-fitting, with the training score increasing as the number of neurons in the single layer increases, and the subsequent drop in the validation score, signaling poor generalization. Notice that the first point in the plot corresponds to 10 neurons, the same configuration as the learning curve plot. This indicates that 10 neurons (or less) is best for generalization. Next, the validation curve as a function of the learning rate γ is shown in 7(b), fourth plot from the left. The learning rate controls the step-size of the neural network weights updates during each step of gradient descent during backpropagation. Small values of the learning rate can cause the backpropagation to converge too slowly to the optimal minimum of the loss function and get stuck in a sub-optimal solution (specially if the number of iterations is fixed). On the contrary, the learning rate is too large, the updates to the weights will be too large and tend to "over-shoot", exhibiting an herratic zig-zag around the optimal minimum of the loss function, preventing proper convergence of the algorithm. In both situations the in-sample fit and the generation to out of sample of the network will suffer, due to improper convergence of the algorithm. We see this clearly in 7(b), fourth plot from the left, with both training and cross-validaton score curves exhibiting an "inverted U-Shape", with an optimal point around $\gamma = 10^{-3}$.

[1]While running the MLP, we received a `ConvergenceWarning` message, stating that the Maximum iterations (200) were reached and the optimization hasn't converged yet.

Finally, the performance plot (7(b), first plot on the right) shows the fit time as a function of sample size for a MLP with one layer and 10 neurons. The fitting time seems to scale sub-linear as a function of sample size.

3) Boosting: AdaBoost (for Adaptive Boosting) was chosen as the boosting algorithm, using `scikit-learn` `AdaBoostClassifier` [2], based on the AdaBoost multi-class classification algorithm introduced in [7].

Figure 7(c), first curve from the left, shows the learning curve of the AdaBoost classifier with a fixed number of 10 weak learners (tree stumps), as a function of sample size. The cross validation seems relatively unchanged as a function of the sample size, with a wide dispersion of training and cross validation scores, potentially due to the low number of weak learners. The two curves are statistically equal within one standard deviation, with a relative low score of 66.5% (compare this to a random classifier that always achieves an auc score of 50%). This could signal that the model with only ten weak learners is not fitting the training data well (high bias), but at the same time exhibits low variance, since the training and validation curve are statistically indistinguishable from each other.

The validation curve for Boosting exhibits a unique behavior, in that the `auc` score for the cross-validation sets seems to keep increasing until reaching a maximum around around 100 weak learners, after which it decreases slightly, while the training curve keeps increasing. This is because as the number of weak learners increases, the confidence that boosting places on its predictions increases, effectively widening the margin between negative and positive answers, which minimizes overfit [8]. High bias, or under-fit, may be occurring below 100 weak learners, as both the training and cross-validation scores climb together and overlap with one another. This could happen if the algorithm is not picking up any useful signal from the data, thus performing similarly between training and validation data. Therefore, keeping everything else constant, 100 weak learners seems the optimal value for this hyperparameter.

The performance plot of the `AdaBoostClassifier` with 10 weak learners is shown in the last plot in Figure 7(c). AdaBoost fit time seems to scale linearly as a function of the sample size data.

4) Support Vector Machines: We employed the Support Vector Classifiers (SVC) from the `scikit-learn` library,

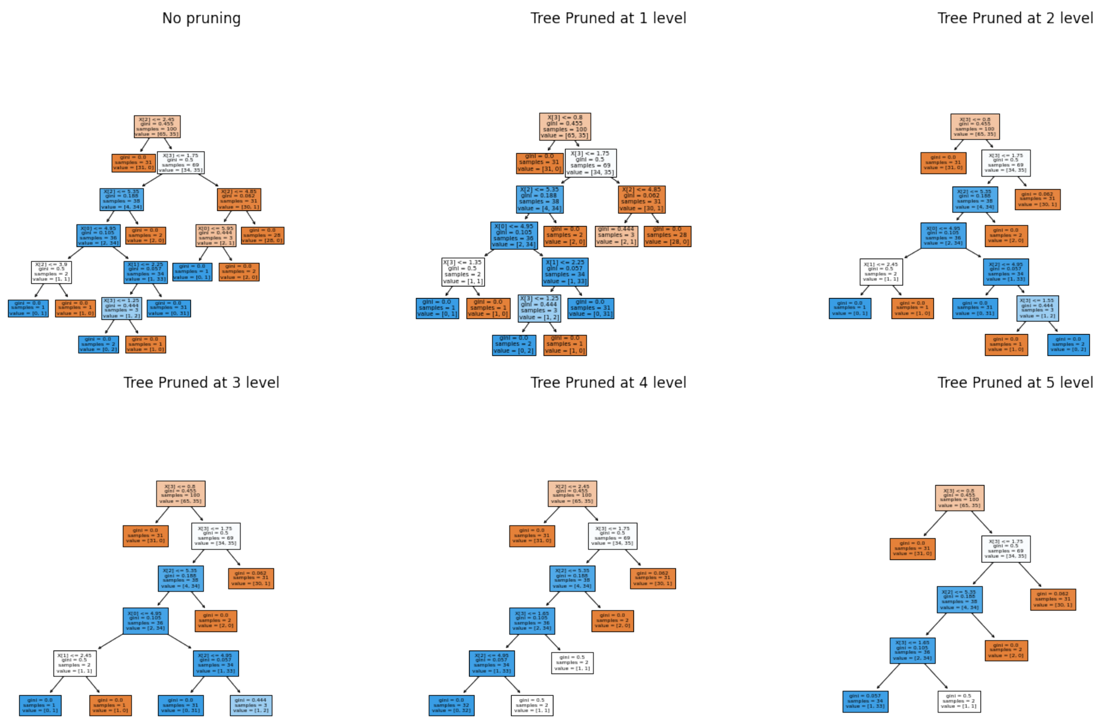


Fig. 6: Example of our implementation of pruning on the iris dataset [2]. Pruning reduces overfitting by removing leaf nodes from the tree. In our implemetation of pruning, the parameter `pruneLevel` measures the number of sets of leaf nodes removed.

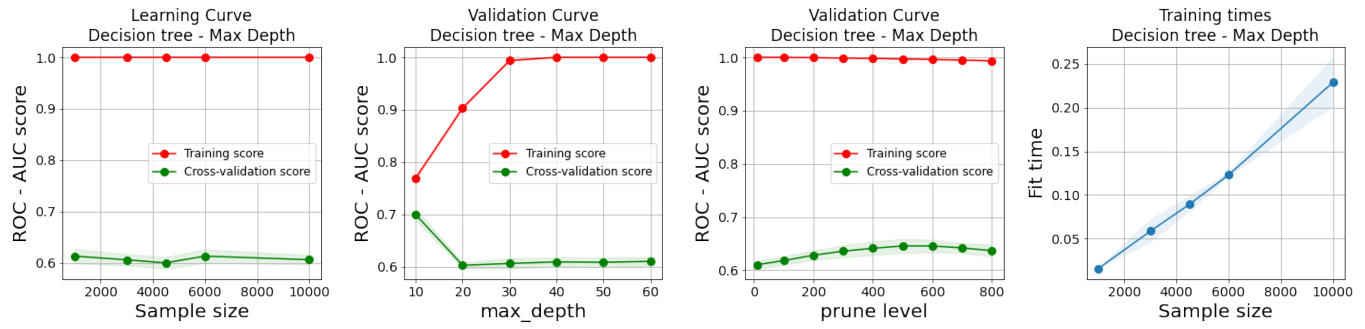
using the module `SVC` [2].

Figure 7(d), first curve from the left, shows the learning curve for the SVC using a radial-basis-function as a kernel, as a function of sample size. While the training score decreases, the cross validation seems relatively unchanged as a function of the sample size, indicating poor generalization and potentially underfitting (high bias) as the sample size increases. The validation curve as a function of the regularization parameter `C` is shown next. This parameter controls how much misclasification should be avoided. Large values of C produces separating hyperplanes with smaller-margins, while smaller values of C produces larger-margin separating hyperplane, at the cost of potentially misclasifying instances. Everything else held constant, higher values of C decreases bias (better in-sample fit), but increases variance (worse generalization), and viceversa. We clearly see this behavior in the learning curve, as the training auc score increases monotonically with C (higher C $\rightarrow$ lower bias $\rightarrow$ better in sample fit), but increases the variance, as the validation curve remains relative unchanged, with the gap between the two increasing. The second plot shows a SVC with a linear kernel. We see that both the training and validation scores move together in a somewhat erratic fashion. This signals that the model under-fits the data, as the SVC cannot find any useful signal from the data, thus performing similarly between training and validation data. Finally, the fit time as a function of the sample size for the SVC model with rbf kernel is shown last, and it exhibits a quadratic behavior with sample size.

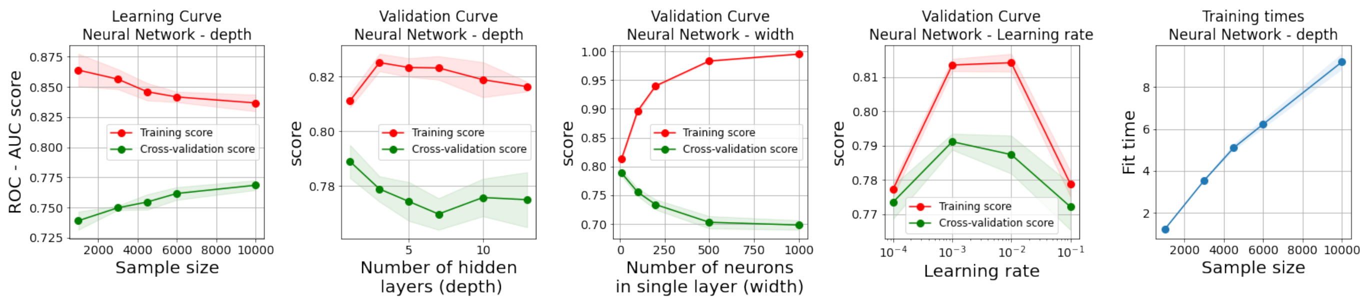
5) K-Nearest Neighbors: KNN-bank We employed the K-Nearest Neighbours Classifier (KNN) from the `scikit-learn` library, using the module `KNeighborsClassifier` [2]. The default class uses the Euclidean distance, with $k = 5$ neighbours.

Figure 7(e), first curve from the left, shows the learning curve for the K-NN algorithm using $k = 5$ neighbors. We can see that both the training curve as well as the cross validation curve increases with training size, with a wide gap between the two, with relatively training scores at or above 90%. This indicates overfitting from the model, given that the in sample scores are high, while the cross-validation scores are relatively, indicator poor generalization.

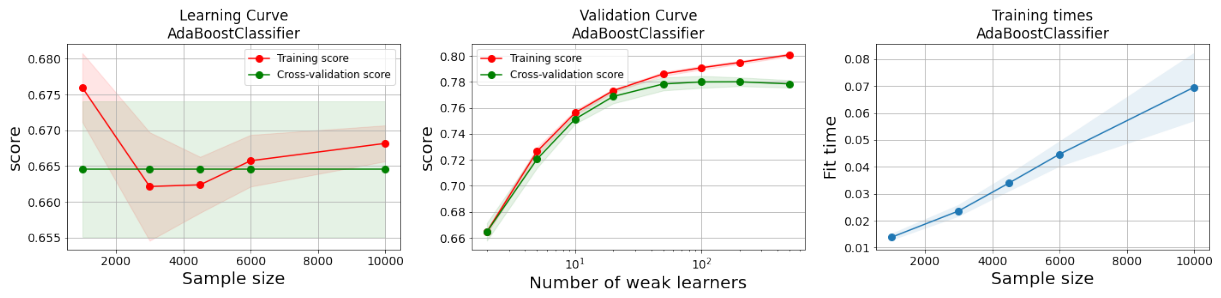
In K-NN, as the number of neigherst neighbours increases, more and more of the data is taken into account to perform classification, reducing the impact of any given point in the decision of the classifier, and lowering fitting too closely to noise. Therefore, as K increases, the bias (or underfit) will tend to increase, but the variance (or overfitting) will decrease. This is exactly what we see in the validation curve for K-NN (seconf plot from left in Figure 7(e)) as a function of the number of neighbors K. This plot show a “typical” behavior of increasing generalization, as the training curve decreases and validation curve increases as the number of neighbors increase, somewhat converging around a score of 80%. This indicates that the model increases its generalization ability across the



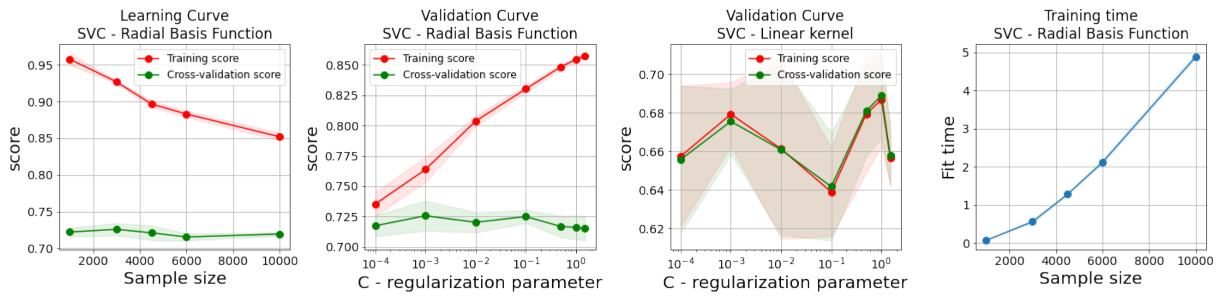
(a) Decision Trees



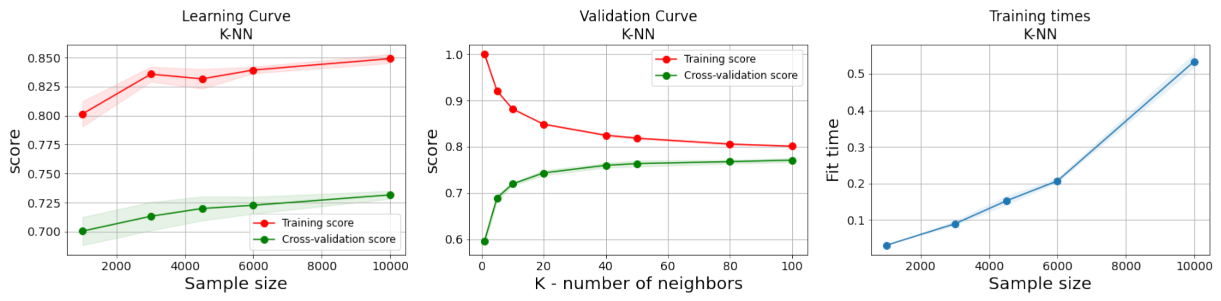
(b) Neural Networks



(c) AdaBoosting



(d) Support Vector Classifier



(e) K-Nearest Neighbors

Fig. 7: Learning curves, validation curves and performance curves for Decision Trees (top row), Neural Networks (second row), AdaBoost with tree stumps (third row), Support vector classifiers (last row, with rbf and linear kernels), and K-Nearest neighbors. See main text for details

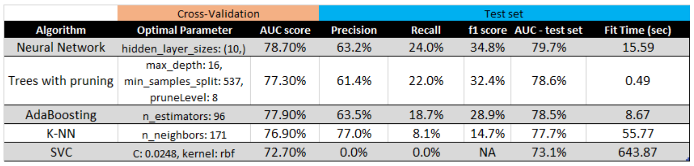
out of sample data, but it bias somewhat increases, as the auc score drops to around 80%.

The fit times as a function of sample size seems to indicate that the algorithm scales roughly quadratic with the size of the data.

6) Comparison across algorithms: Among all the algorithms discussed, AdaBoost, K-NN and neural networks (with a single layer of 10 neurons) seem to have the similar out of sample performance in the cross-validation sets, with an optimal cross-validated auc score of around 78%, at which point the gap between train and validation auc curves is minimum, indicating low variance/overfitting. AdaBoost for example generalizes best on the validation data at around 100 weak learners (see Figure 7(c), second plot), while the neural network with one single layer and 10 neurons performs best in terms of bias/variance (see Figure 7(b), third plot from left). K-NNs, which display an increasing generalization to the validation data as the number of neighbors increases, while achieving a validation score of almost 80% when K=100, very similar to AdaBoost. In addition, the validation curve for AdaBoost exhibits the unique behavior of increasing and then plateauing as a function of weak learners, due to the higher confidence in its predictions, which increases the margin between positive and negative instances. The fact that KNN has performed relative well indicates that the data may be amenable to clustering, as KNN seems to indicate that instances that are close to data points of a given binary class tend to belong to it. Decision Trees (with no limit on depth) or SVC with rbf are among the worst, in that they tend to over-fit the data quite significantly, as can be seen from the difference between the training and cross validation scores, both in the learning curves and validation curves studied.

H. Hyperparameter tuning and final model evaluation

`RandomizedSearchCV` from the `scikit-learn` library was used to tune the hyperparameters that were selected for performing the valildation curves in Figure 7. The results can be seen on Figure 8, where we display the final hyperparameter values for the tuned models, their cross validation AUC score, as well as their statistics on the test set, including precision, recall, f1 score, AUC score and the total time to fit to the train data[2]. The models were sorted by their AUC score on the test set, in descendent order. Several observations can be made from these results. Neural networks with a single layer of 10 nodes showed to be the best model in terms of test AUC. However, in terms of AUC test performance, neural networks, trees with pruning and AdaBoosting all seem to perform similarly well, something we have pointed out in Section II-G6. The AUC scores estimated via cross validation were good estimates of the out of sample test auc scores, as they are similar to each other. The optimal value of the hyperparameters coincided in many cases with the analysis on the learning curves we did in the prior section and using Figure 7. For example, the neural networks best hypermarater is a shallow network of one layer, with 10 neurons, which coincided with our observations made in Section II-G2 in reference to the validation curve in Figure 7(b). The optimal parameters for Decision Trees however do not coincide with the validation plots in Figure 7, though this is expected, as the `RandomizedSearchCV` procedure searched for a "maximum" across all three parameters simultaneusly, while the analysis in Figure 7 was done changing one parameter at a time, while keeping everything else fixed. It is interesting to observe though that the final tree end up being much shallow, with a maximum depth of 16 and a prune level of 8. This coincided with our analysis from Section II-G1, in that deeper trees tend to overfit. We have also included the precision, recall and f1 scores on the test set. Precision is the fraction of `f1` score is the harmonic mean between precision and recall, and tries to strike a balance between the two. Even though we optimized for AUC, it is interesting to see how they compare against AUC. For highly unbalance data sets, it is known that AUC can over-estimate the results, and then a precision-recall curve analysis would be preferable [5]. Even though our dataset is unabalanced (only 11.6% of positive classes overall, see Section), the test AUC score and f1 scores produce the same ranking across models, potentially reassuring that the AUC score was sufficient for choosing the best algorithm. Notice however how misleading AUC was for SVC. We observed that SVC was not performing well during our analysis in Section II-G4, and the test results show it clearly for precision and recall, as both are zero.



	Cross-Validation		Test set				
Algorithm	Optimal Parameter	AUC score	Precision	Recall	f1 score	AUC - test set	Fit Time (sec)
Neural Network	hidden_layer_sizes: (10,)	78.70%	63.2%	24.0%	34.8%	79.7%	15.59
Trees with pruning	max_depth: 16, min_samples_split: 537, pruneLevel: 8	77.30%	61.4%	22.0%	32.4%	78.6%	0.49
AdaBoosting	n_estimators: 96	77.90%	63.5%	18.7%	28.9%	78.5%	8.67
K-NN	n_neighbors: 171	76.90%	77.0%	8.1%	14.7%	77.7%	55.77
SVC	C: 0.0248, kernel: rbf	72.70%	0.0%	0.0%	NA	73.1%	643.87

Fig. 8: Models after hyperprameter tuning using cross validation with `RandomizedSearchCV`, as well as their performance on the test set. We show precision, recall, f1 score, AUC score and in sample fit time. The best model is the neural network, both in terms of having the highest AUC and f1 scores on the test set.

Figure 9 shows the confusion matrices for each of the tuned models on the test set. These figures are in correspondence with the precision, recall numbers from Figure 8. For example, in the case of AdaBoost, the precision is given by the ratio of correctly identified positive labels (62) out of the total number of predicted positive labels $(108 + 62 = 170)$, giving a precision of $62/170 = 63.5\%$ (which matches the results in Figure 8, while the recall is given by the ratio of correctly identified labels out of the total true positive labels $(108/(108 + 470) = 18.7\%$, again matching results from Figure 8). A perfect classifier would have zero along the off-diagonal entries of the confusion matrices. SVC was the poorest model performers as quantified in Figure 8, and the

[2]To produce the test scores, we cloned the model with the best hyperparameters as found by `RandomizedSearchCV` and then we used the *totality* of the train data set to fit the model, and finally, measured performance (precision, recall, f1, AUC) on the test set. This is the only time the test set was used.

confusion matrices confirm this: SVC classifies all the samples as negative (the customer does not buy term loans), which produces a zero false positive rate (ratio of negative instances that are incorrectly classified as positive), but zero recall. It is also evident that neural networks achieve the best trade-off between precision and recall.

Figure 10 shows the learning curves before parameter tuning (lighter color, "x"' markers, same data as in Figure 7, first plots from the left for each algorithm), and after parameter tuning (solid lines, "o"' line marker). For Classification trees, AdaBoosting and K-NN, there has been a clear improvement in the cross-validation curves after parameter tuning, and in addition, the gap between training and cross-validation decreased, signaling a reduction in variance (with the exception of AdaBoosting: the increase in cross-validation (bias) came at the expense of higher variance (gap between the train and cross-validated curve increased). On the contrary, the improvement in terms of in-sample fit (bias) and out-of-sample generalization (variance) was somewhat marginal for neural networks or SVC, as the learning curves before and after parameter tuning are practically overlapping within error. The gap between training and cross-validation curves for Neural Networks and AdaBoosting seem to close as the sample size increases, while the cross-validation increases, potentially indicating that both algorithms would benefit from collecting more data for this problem.

III. Stock sentiment analysis on tweets

The second data set we have analyzed consists of the binary classification task on tweet messages from the "Stock-Market Sentiment Dataset" from the Kaggle website [3], which consists of a collection of 5,792 short text messages related to the stock market, and the task is to classify if the text message conveys a positive sentiment or a negative sentiment. Applying machine learning techniques to text (natural language processing) is unique and presents its own challenges: a) it needs to be mapped to some sort of numerical values, as machine learning algorithms cannot process raw text b) text is sequential data, and there is a dependency between prior words and future works in a sentence c) words can present ambiguity in their meaning. Consider for example the following tweet: "Indian Stocks Rebound Sharply, But Suffer Worst Week Since 2009". The presence of "Rebound Sharply" could be picked up by a machine learning algorithm as being a sign of positive sentiment, however, the subsequent "Suffer Worst Week" clearly indicates the opposite. An algorithm that does not capture the sequence of the words and their interdependence would fail to capture the contrast that the tweet wants to convey, and could end up miss-classifying this example.

A. Data representation

We used `CountVectorizer` from [2] to convert a given sentence into a vector representation, where each element of the vector represents the unique word, and the vector values are the counts of the word in the sentence. For our set of 5,792 text messages, the whole data is mapped into a matrix of 5,792 rows, with as many columns as unique words in the matrix, and the values equal to the word counts in each sentence. In a way, the "features" are the unique set of words (also called vocabulary), and the values of these features is their counts. This is a "Bag of Words" type of approach [1], where the sequential nature of language is completely disregarded, and the features only contain information about the presence or absence of words in a vocabulary. The library `nltk` was used to tokenize and remove stopwords from the data. After processing there where 7,161 unique tokens or features in the dataset, higher than the number of samples available, with the resulting matrix being highly sparse, as only a very small subset of the total number of words in the vocabulary appear in the a given sentence. We used a 10% hold out test set to measure the final performance of the models, and similar to Section II-B, the vectorization `CountVectorizer` was fitted only using the train data set, and then applied to both the train and test data sets. In this way we prevent "data leaking" from the test set. Figure 11 shows the distribution of tweets with positive sentiment (target = 1) versus negative sentiment (target = -1), showing that the data is relatively balanced, with 36.4% of the tweets having negative sentiment.

B. Algorithms

We proceed to use the same algorithms as the ones used to analize the bank data in Section II-G, including our own implementation of Decisions Trees with pruning, sub-classing sklearn `DecisionTreeClassifier`.

1) Decision Trees: Decision trees were fitted to the present data set using `scikit-learn DecisionTreeClassifier` [2], with our own implementation of prunning by sub-classing `DecisionTreeClassifier`. For more details see Section III-B1.

The learning curve in Figure 12(a) shows large overfitting from Decision Trees, the training AUC score achieves 1.0 quickly as a function of sample sizes, with very small cross-validation scores on the out of sample, validation sets. Since the parameters of the `DecisionTreeClassifier` were not specified in the constructor when building the learning curve, there was no limit to the depth of the tree, which could then split until all the leaves were pure, fitting potentially to the noise in the data, and causing the overfit.

The validation curves as a function of pruning level shows that this parameter is not effective at reducing the overfitting, as the gap between in-train AUC scores and out of sample remains high throughout the plot. Similar remarks pertain to `max_depth`, since even though the gap between the training score and validation score is minimum at around `max_depth`=10, the validation score seems almost independent of `max_depth`, potentially showing the inability of Decision Trees to generalize well for this data set. Interestingly, for this data set, the fit time as a function of sampling size exhibits a non-linear behavior, compared to the linear behavior we observed in Section II-G1. Given the high number

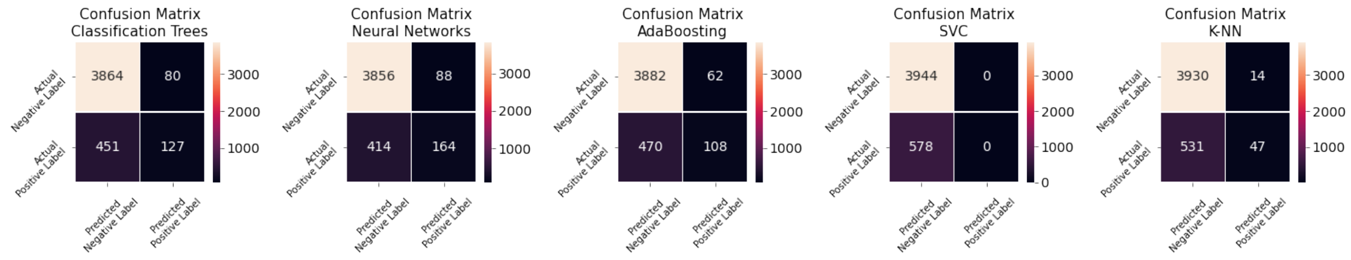


Fig. 9: Confusion matrices of models after hyperprameter tuning.

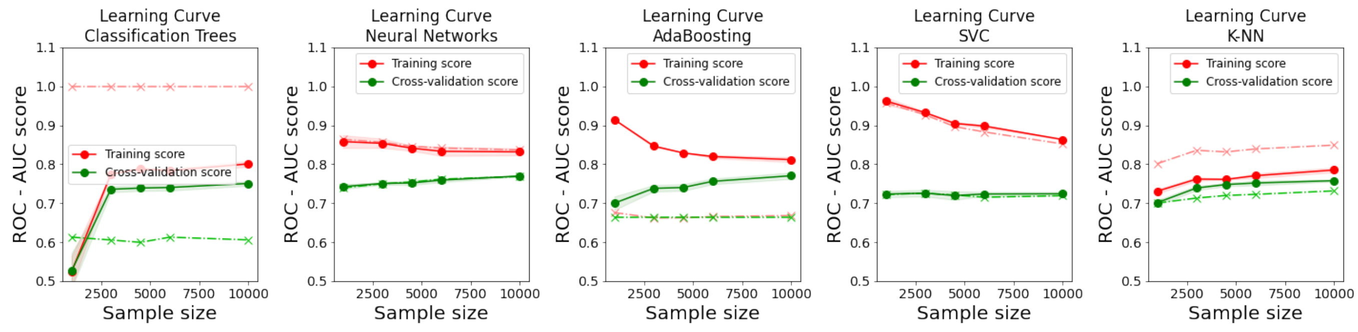


Fig. 10: Learning curves before hyperparameter tuning (lighter color, “x”’ markers, from Figure 7), after after parameter tuning (solid lines, “o”’ line marker). All plots have the same y-scale, to facilitate comparison among them.

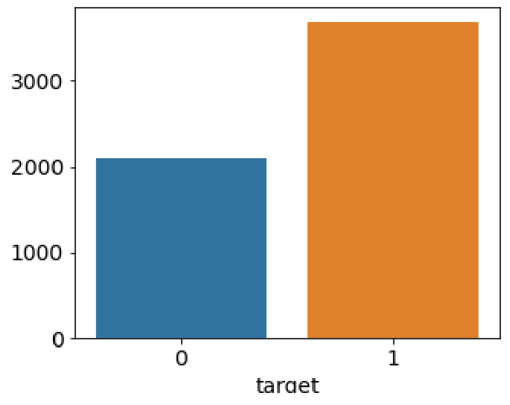


Fig. 11: Distribution of tweets with positive sentiment (target = 1) versus negative sentiment (target = 0)

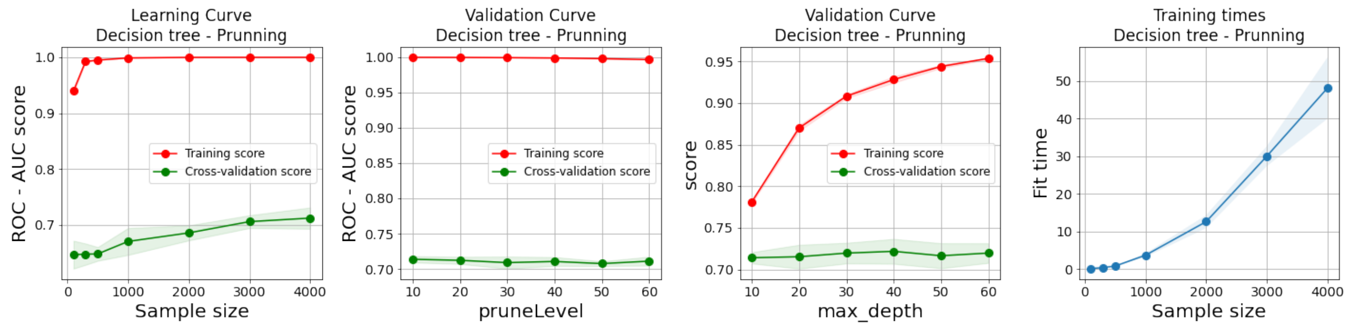
of features and the sparsity of the data, the decision tree potentially is performing an increasing number of branches as new data is fed into it and new features (i.e unqiue words) are provided to the tree discovered.

2) Neural Networks: Neural networks were fitted to the data set using `scikit-learn` `MLPClassifier` [2], which implements a multi-layer perceptron (MLP) that outputs probabilities for the binary classes using a softmax function at the end of the network. Figure 8(b) shows the learning curve for a MLP with one hidden layer of 10 neurons, two validation curves, one as function of the number of hidden layers (each layer with 10 neurons each), and another as a function of neurons in a single hidden layer (width), and finally, the performance plot (Validation set AUC as a function of fitting time) for the MLP with one single layer with 10 neurons.

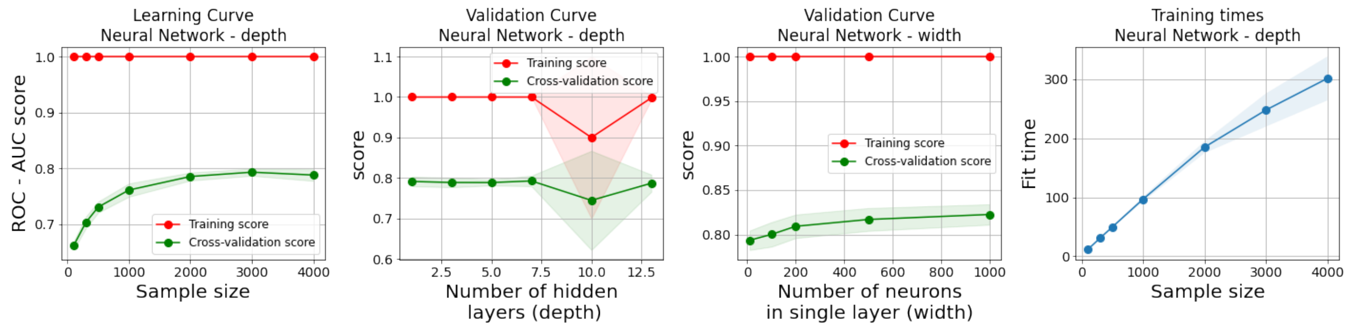
The learning curve and validation curve as a function of number of layers and neurons exhibit a high degree of overfit, as the in-sample auc curves are at 1.0, while the cross-validation curves are only mildly dependent of sample size of number of neurons, or even completely independent of the number of layers. This dataset contains a large number of features, and relatively sparse data, and we hypothesize that the neural network may require more neurons in a given layer, with increasing amount of data to generalize better (mild improvements in the validation curve can be seen as we increase the number of neurons in a single layer). Similar to Section II-G2, 8(b), last plot from the left, shows the fit time as a function of sample size for a MLP with one layer and 10 neurons. The fitting time seems to scale sub-linear as a function of sample size.

3) Boosting: AdaBoost (for Adaptive Boosting) was chosen as the boosting algorithm, using `scikit-learn` `AdaBoostClassifier` [2], based on the AdaBoost multi-class classification algorithm introduced in [7].

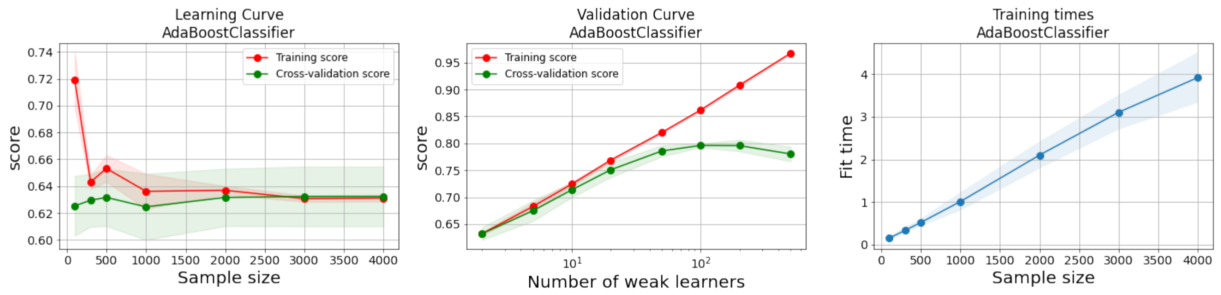
Figure 8(c) shows a similar behavior than we observed with the Bank data in Section II-G3, with the learning curve showing low variance as the training and cross-validation curves are statistically identical within one standard deviation, with no gap. However, there is under-fitting, with a relative low auc score of 63% (compare this to a random classifier



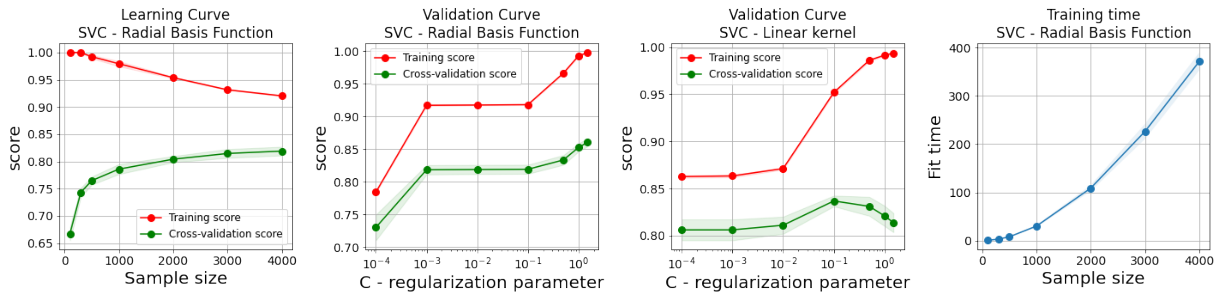
(a) Decision Trees



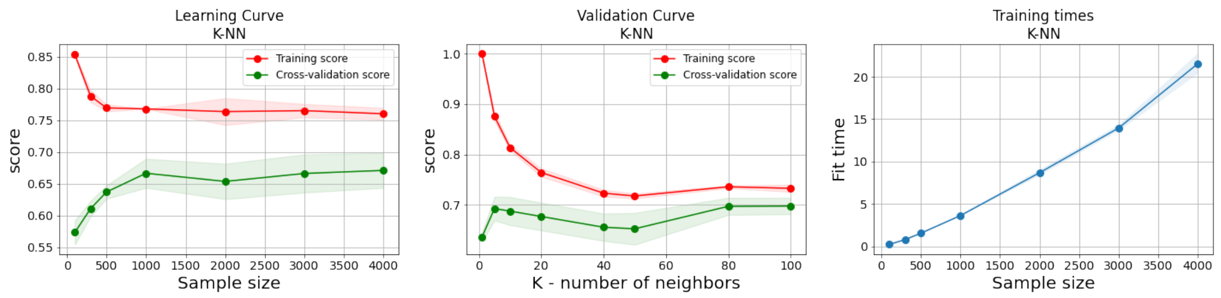
(b) Neural Networks



(c) AdaBoosting



(d) Support Vector Classifier



(e) K-Nearest Neighbors

Fig. 12: Learning curves, validation curves and performance curves for Decision Trees (top row), Neural Networks (second row), AdaBoost with tree stumps (third row), Support vector classifiers (last row, with rbf and linear kernels), and K-Nearest neighbors. See main text for details

that always achieves an auc score of 50%), potentially from using only ten weak learners.

The validation curve for Boosting as function of weak learners does exhibit a more distinctive peak in the validation curve performance, compared to Section II-G3, with a peak in the cross-validation score at around 100 weak learners, with optimal generalization. The performance plot of the `AdaBoostClassifier` with 10 weak learners is shown in the last plot in refFig7(c). AdaBoost fit time seems to scale linearly as a function of the sample size data.

4) Support Vector Machines: We employed the Support Vector Classifiers (SVC) from the `scikit-learn` library, using the module `SVC` [2].

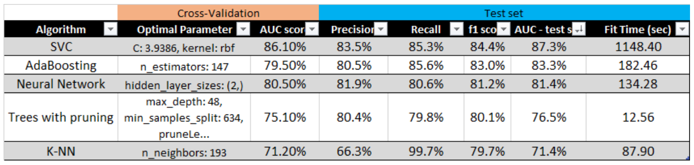
Suppot vector machines do seem to perform better for this dataset compared to the bank data from Section II-G4, with the learning curve in Figure 8(d) showing increasing generalization with larger amount of data for the SVC with radial basis function, as the gap between in-sample train auc score and cross-validation scores decreases. This indicates that the algorithm may benefit from larger amounts of data.

As explained in Section II-G4, the parameter `C` controls how penalization is given to miss-classifications, with higher values of C generally decreasing bias (better in-sample fit), but increases variance (worse generalization), and viceversa. we observe this behavior again here, for both the radial-basis kernel (Figure 8(d), second plot from left) and a linear kernel (Figure 8(d), third plot from left). The `rbf` kernel seems to achieve the best compromise between bias and variance, as its in-sample auc score is 100%, and cross-validation slightly above 85% for the highest value of the parameter C that we considered. Similar to the bank-dataset, we observe that the fit time as a function of the sample size seems to scale quadratically.

5) K-Nearest Neighbors: We employed the K-Nearest Neighbours Classifier (KNN) from the `scikit-learn` library, using the module `KNeighborsClassifier` [2]. The default class uses the Euclidean distance, with $k = 5$ neighbours.

The learning curve for the K-NN algorithm using $k = 5$ neighbors is shown on Figure 8(e), first curve from the left, and shows that the algorithm struggles to improve the in-sample fit as a function of sample size (a signal of under-fit), while the cross-validation curve reaches a plateau at around 1,000 samples, with a constant gap between the two (which shows high variance or poor generalization). This points that the K-NN may not be a good algorithm to tackle this dataset. Similar observations can be made from the validation curve as a function of number of neighbors, in that while the generalization may be seen to improve (the gap between the training and cross-validation curves decreased as a function of K), the in-sample and out of sample auc scores are relatively low at 70%. Similar to the results for the bank dataset, the fit times as a function of sample size seems to indicate that the algorithm scales roughly quadratic with the size of the data.

6) Comparison across algorithms: Among all the algorithms discussed, Support vector machines with radial basis function seem to have performed the best in terms of their cross-validation curves for the learning and validation curves. The learning curve for SVC clearly shows that the SVC algorithm would benefit from larger amount of data, as the gap between the training and cross-validation curves decreases in the learning curve, indicating better generalization/lower overfit. Decision Trees (with no limit on depth) tend to over-fit the data quite significantly, while K-NN exhibit high bias (demonstrated by a low `auc` score in the train set) as the number of neighbors increased, were among the worst two performing models.



	Cross-Validation		Test set				
Algorithm	Optimal Parameter	AUC scor	Precision	Recall	f1 sco	AUC - test s	Fit Time (sec)
SVC	C: 3.9386, kernel: rbf	86.10%	83.5%	85.3%	84.4%	87.3%	1148.40
AdaBoosting	n_estimators: 147	79.50%	80.5%	85.6%	83.0%	83.3%	182.46
Neural Network	hidden_layer_sizes: (2,)	80.50%	81.9%	80.6%	81.2%	81.4%	134.28
Trees with pruning	max_depth: 48, min_samples_split: 634, pruneLe...	75.10%	80.4%	79.8%	80.1%	76.5%	12.56
K-NN	n_neighbors: 193	71.20%	66.3%	99.7%	79.7%	71.4%	87.90

Fig. 13: Models after hyperprameter tuning using cross validation with `RandomizedSearchCV`, as well as their performance on the test set. We show precision, recall, f1 score, AUC score and in sample fit time. The best model is the SVC, both in terms of having the highest AUC and f1 scores on the test set.

C. Hyperparameter tuning and final model evaluation

Similarly to the bank dataset, `RandomizedSearchCV` from the `scikit-learn` library was used to tune the hyperparameters that were selected for performing the valildation curves in Figure 12. The results can be seen on Figure 13, where we display the final hyperparameter values for the tuned models, their cross validation AUC score, as well as their statistics on the test set, including precision, recall, f1 score, AUC score and the total time to fit to the train data[3]. The models were sorted by their AUC score on the test set, in descendent order.

As noted before for the bank data set II-H, the test AUC score and f1 scores produce the same ranking across models, potentially reassuring that the AUC score was sufficient for choosing the best algorithm. Interestingly, for this dataset, SVC showed the best performance in terms of auc score and f1 scores in the test set, with an rbf kernel and a relatively high C parameter value. As mentioned in Section III-B4 and Section II-G4, larger values of C tend to produce separating hyperplanes (or hyper-surfaces for the case of `rbf`) with smaller margins, which tend to decrease bias and increase variance.

The in-sample training times are considerable longer compared to the bank data-set from the previous section, potentially impacted by the high number of features of the current data set.

Figure 14 shows the confusion matrices for each of the tuned models on the test set. These figures are in correspon-

[3]To produce the test scores, we cloned the model with the best hyperparameters as found by `RandomizedSearchCV` and then we used the *totality* of the train data set to fit the model, and finally, measured performance (precision, recall, f1, AUC) on the test set. This is the only time the test set was used.

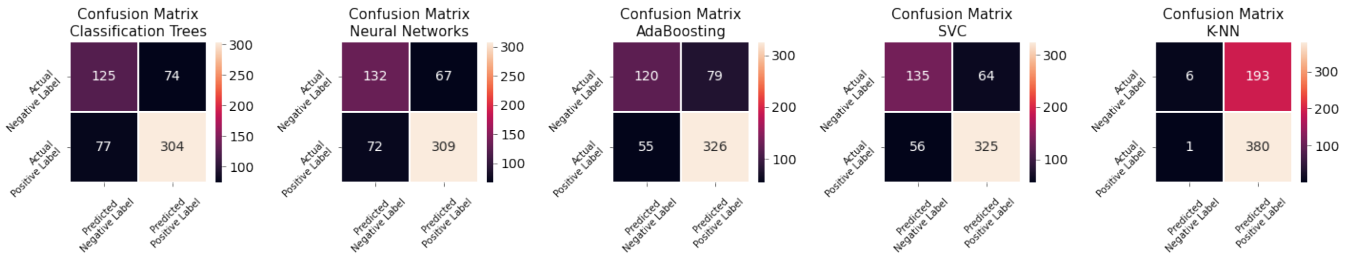


Fig. 14: Confusion matrices for tuned models on the test set.

dence with the precision, recall numbers from Figure 13. For example, in the case of SVC, the precision is given by the ratio of correctly identified positive labels (325) out of the total number of predicted positive labels ($325 + 64 = 389$), giving a precision of $325/389 = 83.5\%$ (which matches the results in Figure 13, while the recall is given by the ratio of correctly identified labels out of the total true positive labels ($325/(325+56) = 85.3\%$, again matching results from Figure 13). A perfect classifier would have zero along the off-diagonal entries of the confusion matrices, and SVC has the best balance in terms of smallest off-diagonal matrix elements across all clasifiers. K-NN, on the other hand, predicts that all instances were positive, giving it a very high recall. However, the confusion matrix clearly indicates the model underperforms, compared to the others.

It is interesting to look at some of the examples that SVC miss-classified. Among the test examples that SVC classified incorrectly as having a positive sentiment is "Take Profits & Tighten stops. SA indicating major correction looms. OCN NSM EGN BID GMC FB NKD", while an example of a correctly classified example is "'AYI 2013 Q1 operational Cash flow negative 14.5 million. decline of 41 million vs. Q1 2012. 152% decline"'. The first example contains a technical term, "tighten stops", which indicates negative sentiment, but a positive term, "profit". We hypothesize the algorithm may have used the presence of the word "profit" to classify the instance as positive sentiment, neglecting to understand the technical term "tighten stops" (in a bag of words approach, each word taken separatedly has a very different meaning that both words put together). On the other hand, the second example contains "negative" and "decline" (twice), which taken in isolation correspond typically to negative sentiment words, which must have helped the algorithm to correctly classify the instance via the bag-of-word approach.

IV. Conclusions

We have analyzed the performance of various machine learning algorithms on the bank dataset and the sentiment classification task on tweets. The datasets were very different, in that the first one contained a mixture of numerical and categorical data (which we converted to numerical via `one-hot encoding`), while the second dataset contained only text, and was converted to a numerical representation via a bag-of-words approach (which mapped sentences to a high-dimensional vector space, representing the count frequency of the words in a given vocabulary), producing a high number of features, with a spare representation in the data. The learning and validation curves allowed us to determine which algorithms performed best in terms of bias-variance tradeoffs, and identify the possible location of the best hyperparameter values. The comparison across learning algorithms revealed clear differences in their behavior, with decision trees among the algorithms with strongest over-fitting. AdaBoosting showed exceptional performance in terms of fitting time, and very good balance between in-sample and out of sample performance. Neural Networks, Decision Trees and AdaBoosting all performed similarly well in terms of cross-validation `auc` scores, while for the second dataset SVC with `rbf` kernels and a relative high value of the C parameter performed best. The confusion matrices on the test set helped us analyze the type of errors made by the algorithms, and confirm the auc, precision and recall values on the test set. For the second text classification case, we noticed that SVC (the bets performing algorithm) seemed to pick on the count frequency of certain words, and failed to recognize the meaning of two word phrases, a known drawback to the bag-of-words approach [5]. Recurrent neural networks are known for capturing better the sequential nature of text and the long-dependencies (with `LSTM` and `GRU`), and to produce better representations for the text via word embeddings [4]. We leave this as a future enhancement.

References

[1] Bag-of-words. https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html.
[2] Scikit-learn. https://scikit-learn.org/stable/.
[3] Y. Chaudhary. Stock-market sentiment dataset.
[4] B. M. Delip Rao. *Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning*. 2019.
[5] A. Geron. *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*. 2019.
[6] Y. B. Ian Goodfellow and A. Courville. *Deep Learning*. 2016.
[7] S. R. Ji Zhu, Hui Zou and T. Hastie. Multi-class adaboost. *Statistics and Its Interface*, 2:349–360, 2009.
[8] P. B. Robert E. Schapire, Yoav Freund and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Machine Learning: Proceedings of the Fourteenth International Conference*, 1997.
[9] P. R. Sergio Moro, Paulo Cortez. Bank marketing data set.
[10] P. R. Sergio Moro, Paulo Cortez. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.