Applied Machine Learning and Big Data

62533

Final Report

Tobias Schønau - s224327

Deadline: 30/05-2024

1 Problem Definition

At the end of 2022, personal loans in the US reached $356 billion. [1] This market is highly dependent on classifying borrowers as safe or unsafe. Doing this task with a high accuracy is important, since denying safe borrowers will lose the lender potential customers, and therefore money. The same can be said about approving unsafe borrowers, where the lender now has to take on unintended risk in regards to whether the borrower will pay.

Historically this task has been performed by humans, which is highly expensive and can sometimes fall trap to biases or subjective opinions. Therefore it is very interesting to explore whether this task can be automated or assisted by the use of Machine Learning.

This report plans to explore the use of these Machine Learning methods to classify borrowers as safe and estimating the cost of the errors that the models will have.

1.1 Business oriented goals

To test the functionality of the resulting models some business related goals have to be defined to show the models are commercially viability. The dataset that we are going to use is a dataset of humans lending to humans, with a middleman called Lendingclub to help them find eachother, from 2007-2010. A significant portion of these loans are not paid back in full, in fact only about 80% of the loans were paid out in full. We set a goal that the chosen model should be able to classify loans that will pay out with a higher successrate than humans at 80%.

We also wish for the use of the model to be profitable. To estimate this we are going to use some numbers. We estimate that the standard loan is for an amount of about $3000, with an interest rate of about 12% annually. We also assume that the loan is to be paid back over a year. We define that a classification of "not paid back" means that $0 was paid back, and that a classification of "paid back" means the total amount + interest was paid back. With these assumptions we wish for the model to make a profit.

We also wish for the model to not be too safe. If we are in the business of giving out loans, we wish to give out loans to a significant portion of users. If we only give loans to 1% of users, then we can probably make a profit, but it will take forever. Therefore we wish for our model to give out loans to at least 10% of applicants. If our users know that we will give out loans to 10% of applicants, it will lead to an overwhelming amount of bad applications, but since our dataset is made in advance this is not a problem.

1.2 Technical goals

To solve the loan problem we wish to try a bunch of different models. Therefore a technical goal for the project is to attempt to use at least 7 different models to solve the problem, and compare their performance. This will help us identify the most effective model for our needs.

We will use multiple ways to encode the string variables, to use those that best fit the models. This will ensure that the models can effectively process the data.

We will compare a selected amount of models and pick one to test our business related goals.

We want to explore some explainability. We hope to be able to understand for the model, which of the parameters are most important in desciding whether to classify borrowers as safe or unsafe.

2 Data Extraction and Data Processing

2.1 Dataset

To train these machine learning models a relevant dataset is needed. Here a publically available dataset from Lendingclub is used. This dataset consists of 13 features of borrowers, borrowing money on Lendingclub between 2007 and 2010. These features include the purpose of the loan, amount and interest along with some more personal details about the borrower. Finally the dataset includes whether the loan still has a balance, which is what we are trying to classify.

The dataset was found on Kaggle and is publicly available.[2]

2.2 Column Description

The dataset consists of 14 columns, each depicting an attribute of the borrower. We will describe each column shortly. These desciptions are taken in part from the dataset description from Kaggle

not.fully.paid

Value: 0 or 1
This is the target column. This column depicts whether the loan still has a balance. This also makes for a great target for classification since the variable is binary. This variable is a bit backwards, you might expect a positive classification to mean that a borrower is safe, but in this case a positive classification means that a borrower is unsafe, since we expect them to not pay back in full.

credit.policy

Value: 0 or 1
This column is based on a lending criteria set by Lendingclub. If the value is 1, then Lending club has decided that you meet their criteria as a loaner. We do not know how this is set, but it is probably done by a human, with some tool assistance.

purpose

Value: String
The purpose of the loan (takes values "credit_card", "debt_consolidation", "educational", "major_purchase", "small_business", or "all_other"). This value needs to be encoded to be used in all models used in this report.

int.rate

Value: decimal between 0 and 1 (reasonably)
The annualized interest rate of the loan as a fraction (12% interest = 0.12).

installment

Value: dollar amount
The monthly installments owed by the borrower if the loan is funded.

log.annual.inc

Value: log of dollar amount
The natural log of the self-reported annual income of the borrower.

dti

Value: ratio
The debt-to-income ratio of the borrower (amount of debt divided by annual income).

fico

Value: number between 300 and 850
The FICO credit score of the borrower. This is a form of summary of your credit report. A standard value used in USA.

days.with.cr.line

Value: days
The number of days the borrower has had a credit line.

revol.bal

Value: dollar amount
The borrower's revolving balance (amount unpaid at the end of the credit card billing cycle).

revol.util

Value: percentage
The borrower's revolving line utilization rate (the amount of the credit line used relative to total credit available).

inq.last.6mths

Value: number of inquiries
The borrower's number of inquiries by creditors in the last 6 months.

delinq.2yrs

Value: number of late payments
The number of times the borrower had been 30+ days past due on a payment in the past 2 years.

pub.rec

Value: number of records
The borrower's number of derogatory public records (bankruptcy filings, tax liens, or judgments).

2.3 Data values

To see the values of these different columns we made some boxplots.

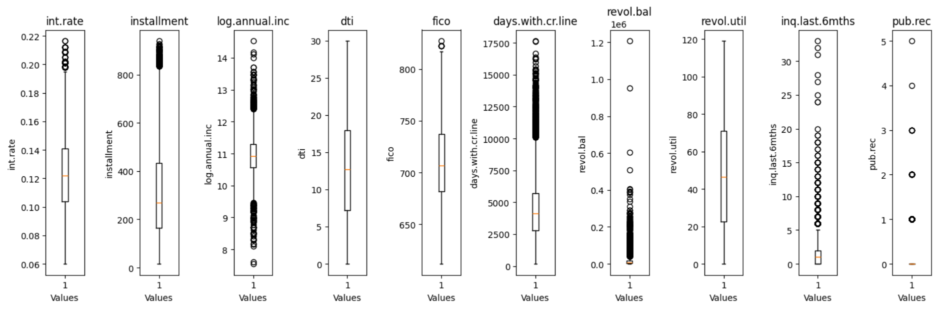

Figure 1: Dataset overview

Here we can see that some of these values are on very different scales, and will need to be standardized to work with some models. We can also see that a lot of these values are very prone to ouliers, such as the log annual income where we see many values outside the 1.5*IQR range. This just means that the standard deviation is also different from different values and will also need to be stanardized.

2.4 Data Processing

Each type of model needs different methods of data processing. Here we will go through all methods used, and explain why they are used.

2.4.1 Standardization of dataset

Some models may need the dataset to be scaled so some features with higher numerical values arent weighted heigher than others. To do standardization the sklearn preprocessing standard scaler is used. This scales numerical values to have a mean of 0 and a variance of 1. This step is only applied to some models, which will be detailed in the modelling chapter.

2.4.2 Encoding of values

The purpose column of data is not a numerical value, and has a string value from one of 6 possible values. The models used in this report cannot use text input, so we have to do some string encoding to be able to use these.

Label encoding: This version of encoding gives each possiblility a number and encodes the string as that number. For example "credit_card" may be mapped to 1 and so forth. This is a very simple way of encoding and it keeps the number of features the same.

One hot encoding: This version of encoding strings gives each possibility for the purpose column its own column, and then sets the value of the column represented to one and all others to 0. This works better with some models that use sums, like neural networks. This is due to the fact that the label encoding implicitly defines taht there is a bigger difference between label 1 and 6 than 1 and 2. This is no the case in our purpose column so one hot encoding may be preferred.

2.4.3 Resampling methods

In order to not test on our training data we need to have some way of resampling our data.

Validation set: The simplest way is just to split the data into a section of training and test data. In our case we use 20% test and 80% training data. This is very quick but does not allow us to train on all of our data.

K-Fold validation: To try to use all of the data for training we implement K-Fold validation for some models. This means splitting the data into some k parts, each to be used as the validation set once. Then for each of the k parts the rest of the data will be used to train. This process is then repeated until all elements of the training set have been used as validation. This way no data is used as training and validation at the same time, but all data is used as training data at some point. This process also means training on the same data multiple times, which can increase compute costs.

2.4.4 Upsampling

The dataset is not balanced between Class 1 and 0. About 80% of loans are paid back. This means that some models may just predict that all loans are paid back and get an 80% accuracy and get stuck in that local minima. To combat this we can upsample the data from class 0. It is important that we only do this on the training data, since this will impact the accuracy metric on the test data if this is also upsampled. To upsample, we simply sample with replacement until the dataset is balanced.

2.4.5 Dropping missing values

Dropping missing values is the first step taken before any other processing. This process ensures that the data set is clean in the way of not having any missing data points. Missing datapoints affect different models differently, but in general just avoiding using datapoints with missing values is a good approach

3 Modeling

3.1 Models used

We wish to test multiple models to see which performs the best on our data. We chose the following models:

- Logistic Regression
- Neural Network(31602 parameters)
- Support Vector Machine
- Random Forest
- Ada Boost
- Gradient Boosting
- Bagging
- Voting classifier(All other ensembles)

All of the models are implemented using scikit learn[3]. I will give a quick overview of the implementation of each model:

3.1.1 Parameters for models

Logistic Regression:
The logistic regression model is very simple in terms of arguments needed to pass to the model. We just upped the amount of iterations the model can use and lowered the tolerance a bit. Also added verbose to see output in console.

```
model = LogisticRegression(max_iter=1000, tol=1e-3, verbose=True)
```

Neural Network: Here i use the MLPClassifier since this just represents a fully connected neural network. I added four hidden layers with 100 nodes each, which is also how i get my 31602 parameters: 1400+10100+10100+10100+202 = 31602. We then set the activation to relu and told it to use stochastic gradient descent. Again we upped the amount of iterations.

```
model = MLPClassifier(hidden_layer_sizes=(100, 100, 100, 100),
activation='relu', solver='sgd', verbose=True, max_iter=1000)
```

Support Vector Machine: Support vector machine does not have that many arguments to set, the kernel is set to rbf, by default, so we just upped the iterations and lowered the tolerance.

```
svc = SVC(max_iter=1000, tol=1e-3, verbose=True)
```

Ensemble models: All ensemble methods were run using 100 estimators, the only other parameter set was the min sample split for the random forest, since it was performing a lot better on its training data than the test data, meaning an overfit was occurring. To fix this we only allow the model to split down to five samples in each "section".

```
RandomForestClassifier(n_estimators=100, min_samples_split=5)
AdaBoostClassifier(n_estimators=100)
GradientBoostingClassifier(n_estimators=100)
BaggingClassifier(n_estimators=100)
VotingClassifier(estimators=[
('rf', random_forest_classifier.model),
('ada', ada_boost_classifier.model),
('gb', gradient_boosting_classifier.model),
('bc', bagging_classifier.model)])
```

Now that we have seen how each model is configured, we want to show which data preprocessing steps are used for each model:

3.1.2 Logistic Regression

Steps taken:

- Dropping missing values
- One hot encoding
- Standardization of dataset
- Validation set approach
- Upsampling training data

3.1.3 Support Vector Machine

Steps taken:

- Dropping missing values
- One hot encoding
- Standardization of dataset
- Validation set approach

3.1.4 Neural network

Steps taken:

- Dropping missing values
- One hot encoding
- Standardization of dataset
- Validation set approach
- Upsampling training data

3.1.5 Ensemble Models

Steps taken:

- Dropping missing values
- Label encoding
- K-Fold validation

The implementation of all models can be seen in the attatched source code

3.2 Baseline model

The baseline model that we have chosen is the random forest model that we implemented in class. This model is the same as the model released on Learn as MyRandomForest. [4]

```
import numpy as np
from scipy.stats import mode
from sklearn.tree import DecisionTreeClassifier
class MyRandomForestClassifier:
    def __init__(self, n_estimators=10, max_depth=None):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.trees = []
    def fit(self, X, y):
        n_samples, n_features = X.shape
        for _ in range(self.n_estimators):
            row_indices = np.random.choice(n_samples, n_samples, replace=True)
            n_selected_features = int(np.sqrt(n_features))
            feature_indices = np.random.choice(
                n_features, n_selected_features, replace=False)
            X_sample, y_sample = X[row_indices][:, feature_indices], y[row_indices]
            tree = DecisionTreeClassifier(max_depth=self.max_depth)
            tree.fit(X_sample, y_sample)
            self.trees.append((tree, feature_indices))
    def predict(self, X):
        predictions = np.array(
            [tree.predict(X[:, features])
            for tree, features in self.trees]
        ).T
        return mode(predictions, axis=1).mode.ravel()
```

We had some problems running this model on our data, and did not have the time to figure out exactly what the problem was, but it would likely have performed worse than the sklearn random forest model, since the baseline model just uses sklearns decicion tress, and some very simple logic.

4 Performance Evaluation

We tested all of these models and compared them using their accuracy. This is just the simplest metric to compare models, and is just defined as the amount of correct guesses over the amount of guesses in total:

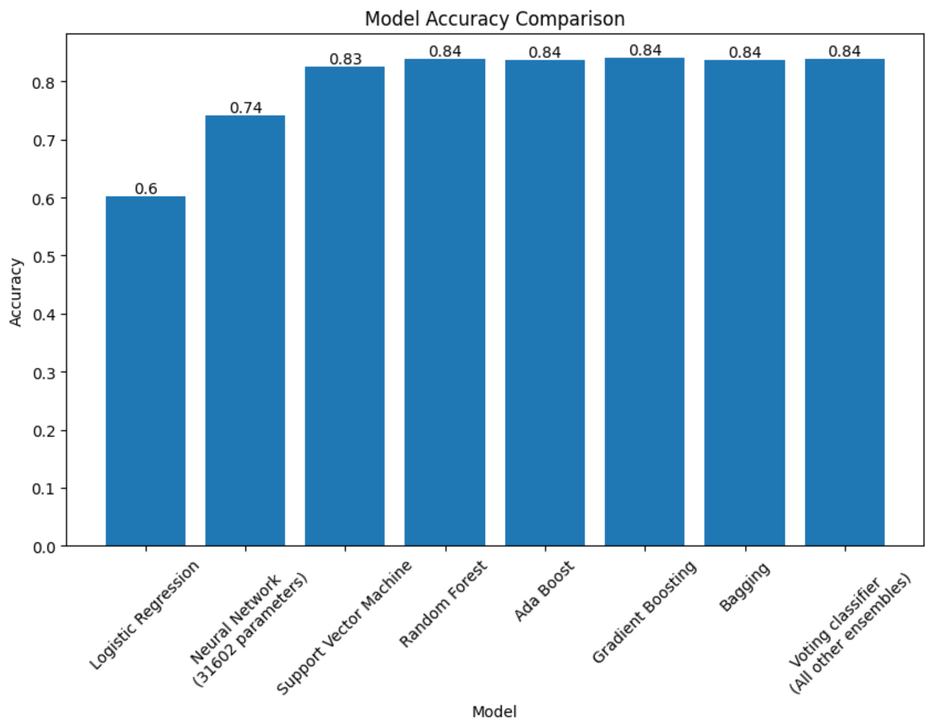

Figure 2: Model Comparisons

As we can see in this figure the Logistic regression performs the worst and then the neural network. After this the support vector machine and all of the ensemble methods are very close. We chose to work further with some of these ensemble methods and compare more than just the accuracy. The reason we chose to work further on the ensemble methods is mostly out of interest, but these are also performing slightly better than the support vector machine model.

I chose to make plots of MSE scores, Precision and recall.

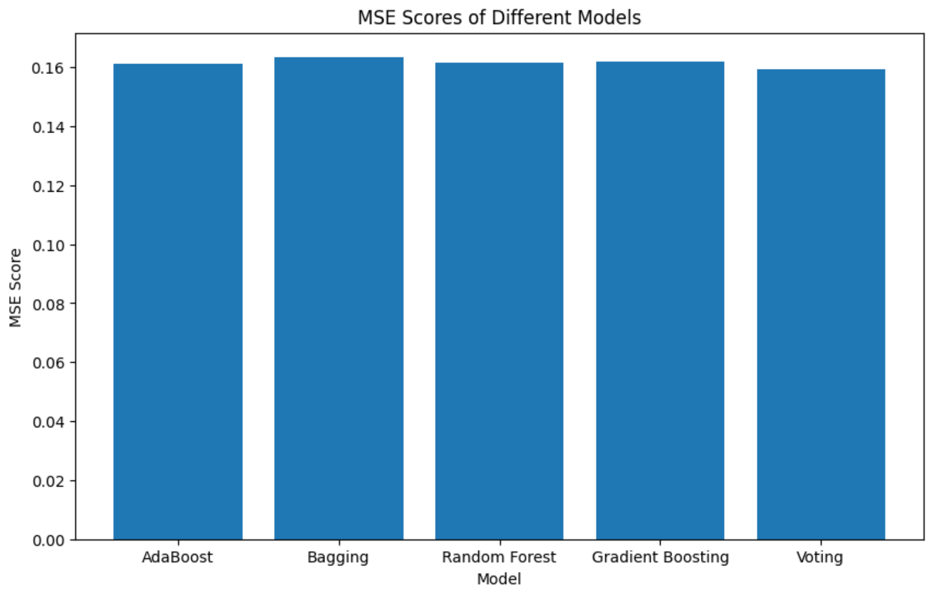

Figure 3: MSE Scores

As we can see from these graphs, these models are very simmilar. There are barely any differences to point out. Essentially all models make about the same amount of total errors.

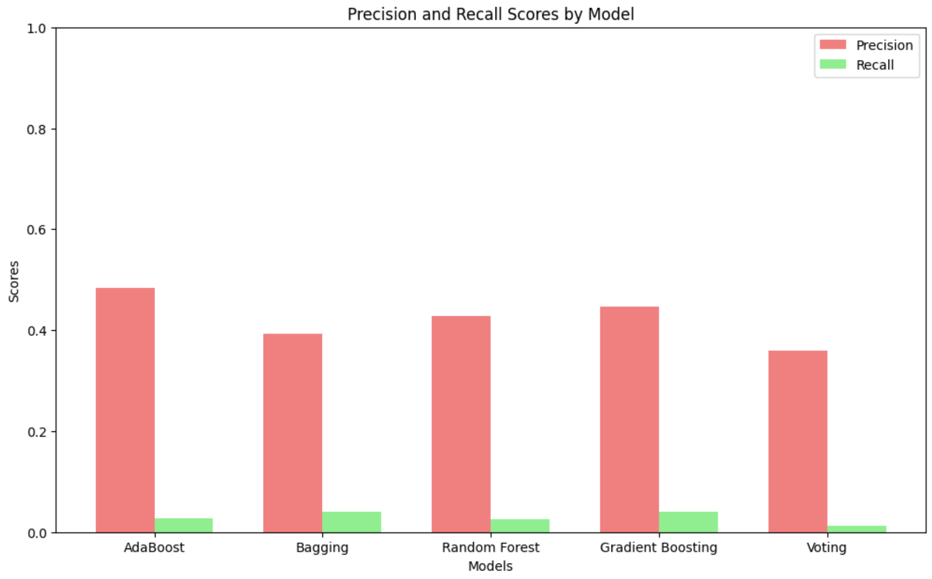

Figure 4: Precision and recall

We can see that our ensemble methods have a pretty low precision and a very low of a recall. If we remember that a positive classification means classifying as unsafe, then this means that when our model classifies a borrower as unsafe, they have about a 50% chance of being unsafe. The lower recall means that we are not quite as good at identifying all unsafe borrowers.

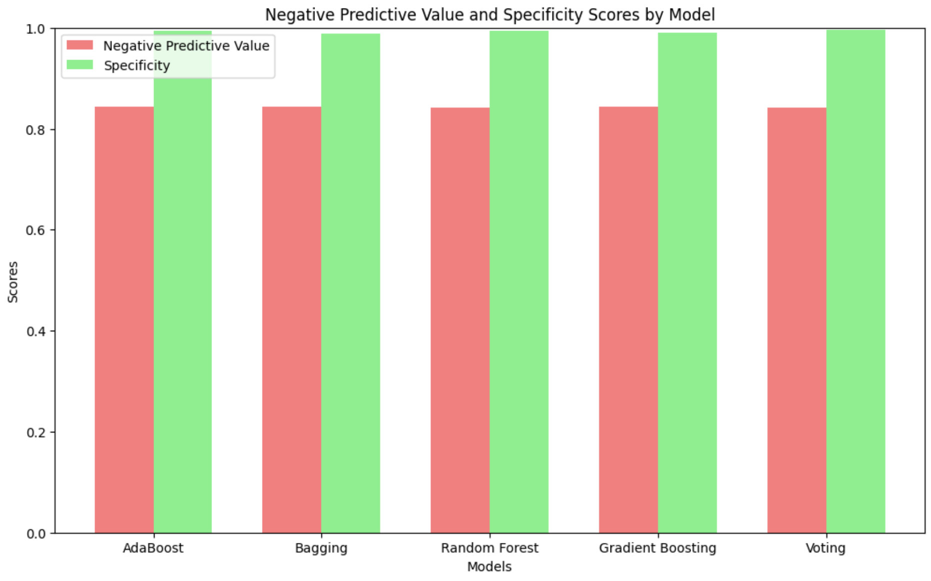

Figure 5: Negative Predictive Value and Specificity Scores by Model

If we however look at the opposite values, meaning how likely the people we classify as safe are to be safe, we see that this number is about 80-85%, and means that the people we give loans are about this likely to pay back in full. The specificity means that we are also very likely to find alm,ost all of the good borrowers. In a business it might make sense to try and balance these values so that the precision is a bit higher and the specificity falls a bit. This would make for a safer lending strategy.

On top these plots we also made a SHAP plot of our dataset to see which of our parameters have the biggest importances of wheter a case is positive or negative:

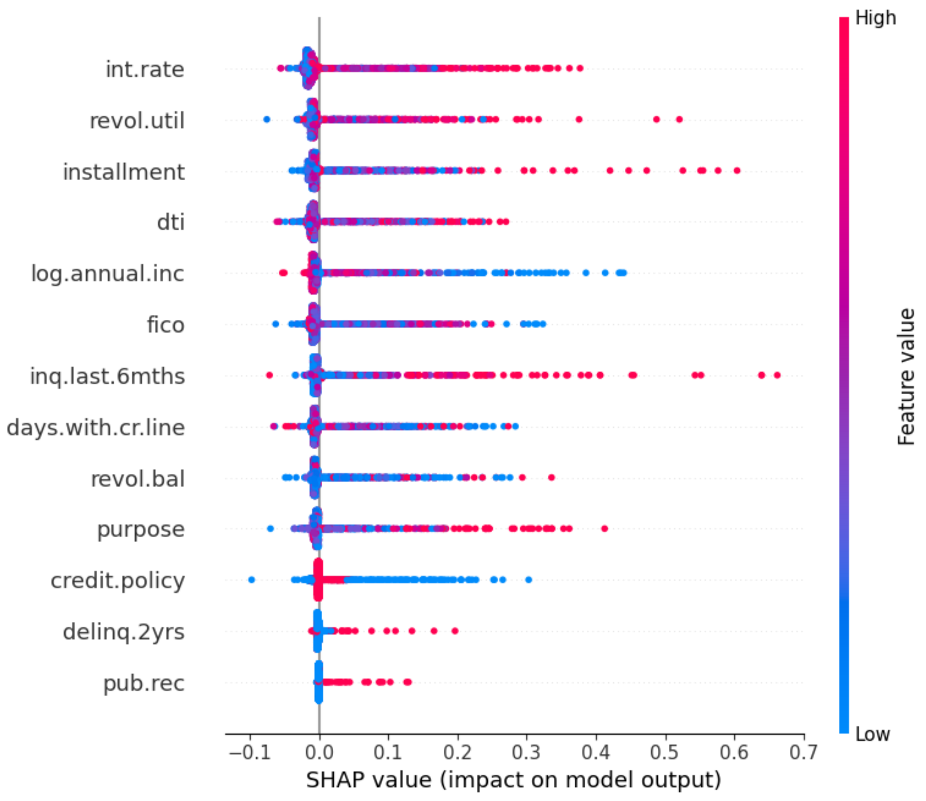

Figure 6: SHAP plot

On this plot we see the SHAP value. When this value is larger that means that a loan is less likely to pay back. This is because of our Class variable being not.fully.paid. As we can see from this plot, there are really no individiual variables that can have a huge positive impact on whether a loan is good or not, (big negative SHAP impact). This makes intuitive sense, since you need almost all of these variables to be good to be a good loaner. On the opposite side however, there are some variables that are very clearly bad to have in a loan, (big positive SHAP impact). These make a lot of intuitive sense aswell, like loans with higher interest rate and installments are less likely to be paid. All in all this plot is very close to what we would expect and gives a good insight into why a loan may be good or bad.

5 Results Explanation

5.1 Business goals

We had a goal of getting a higher than 80% accuracy. This was achieved with multiple models, with the highest being 84%. Although we made a better classification than humans, this is not that big of a difference. This shows that humans are pretty good at classifying loans, and also that loan classification with the given data is not quite as easy as thought. This makes a lot of intuitive sense, since many things can change over the course of the lending period, maybe the loaner gets sick, or looses their job and cant pay because of this. This means that either more data is needed, probably in the form of variables, or loan classification may have to stay as a hard task to classify with very high accuracy.

We also had the goal of making the model profitable. To figure our if the model is profitable we have to do some calculations. Recall that the loan that we are basing these calculations on is for \$3000 with a 12% interest to be paid over one year. This means that we can classify our cases like this:

- True Positive (Loan is not given and correctly predicted not to be paid back): +\$0
- True Negative (Loan is given and correctly predicted to be paid back): +\$360
- False Positive (Loan is given but incorrectly predicted to be paid back): -\$3000
- False Negative (Loan is not given but incorrectly predicted not to be paid back): +\$0

Now we can show the amount of money earned, by looking at a confusion matrix from the random forest classifier. This confusion matrix uses the average results over iterations of the K-Fold validation approach.

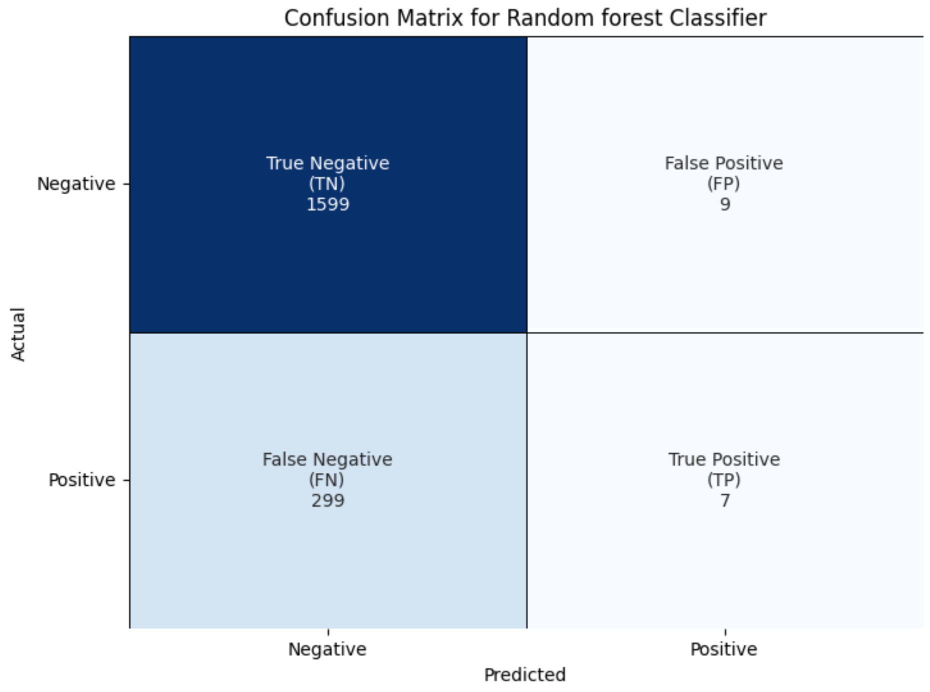

Figure 7: Confusion matrix for random forest

From this chart we can read that over an average iteration of K-Fold we made a total of $\$360*1599 = \$575640$, and lost $\$3000 * 9 = \$27000$, meaning that the model in total made:

$$\$575640 - \$27000 = \$548640$$

This number represents the amount of money that the random forest classifier would make on average on a 5th of the dataset. This number can then be multiplied by 5, to get the total number.

$$\$548640 * 5 = \$2.743.200$$

This number means that if we go by the data that we were given, we would take on 7.995 loans for a total amount of $23.985.000 and make $2.743.200. We do however have to remember that the values for the loans are made up, and cannot be used to predict the future. This does however mean that on our historical data, our random forest model is profitable.

The last goal of giving at least 10% of applicants a loan is easily met, as we can see from the confusion matrix, the True negatives + False positives are the ones that would be given loans and this percentage is way above 10%

5.2 Technical goals

We have tested 8 different models and ended up comparing these on their accuracy score. We then compared some selected models on their precision and recall and based on these picked our random forest model to be the one that best fit out business related goals.

We used both one hot encoding as well as label encoding to encode our string data. These were both used on different models where they fit best.

We explored explainability of our dataset, by making a SHAP plot and seeing which features were regarded as positive and negative for loan classification.

5.3 Final Thoughts

All in all I am pretty happy with the results, and feel like I have learnt a lot about how to practically build a machine learning solution to a classification problem. It feels very good to be able to use the theory to solve a problem i was not able to solve half a year ago. Although the model is not perfect, I am very happy with the results of the model and the process to get there.

6 Appendix

References

[1] Jessica N. Flagg and Simona M. Hannon. “An Overview of Personal Loans in the U.S.” In: *Finance and Economics Discussion Series* 2023–057 (Aug. 2023), pp. 1–13. ISSN: 2767-3898. DOI: 10.17016/feds.2023.057. URL: http://dx.doi.org/10.17016/feds.2023.057.

[2] Sarah Mahmoud. *Loan Data*. 2023. URL: https://www.kaggle.com/datasets/saramah/loan-data.

[3] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[4] Lei You. *Course Manual: Applied Machine Learning*. Accessed: 2024-05-30. 2024. URL: https://learn.inside.dtu.dk/d2l/home/187963.