Credit Card Fraud Detection

Arttu Hannila, Tuomas Laakso, Aleksi Suuronen

October 27, 2025

1 Introduction

In this technical report, we present our group work project for the Introduction to Data Science course at the University of Helsinki. The topic of our project was to predict from a large and unbalanced dataset of credit card transactions if a certain transaction is a fraud or not. This report describes the technical components and methods used in the context of the data science life cycle, and reflects our learning process and choices made along the way. Our technological stack consisted mainly from Python accompanied with its industry-standard machine learning and data science libraries, e.g., scikit-learn, matplotlib, pandas, numpy, seaborn, imblearn, and streamlit. Our streamlit repository can be found at:
https://github.com/AlluSu/credit-card-fraud-detecting

2 Data collection & preprocessing

We decided to go with the topic of detecting potential credit card fraud as we as a group share a common interest for the field of finance and especially in quantitative and computational finance. Hence, it was clear from the beginning that we would do some sort of project related to the field of finance, which would combine our interests with our technical skills, and where we could learn more about data science by applying the various methods and techniques presented during this course.

Credit card fraud is a major issue in the banking industry. For example, working credit card numbers and their customer data are sold continuously in dumps in the dark web, as well as equipment for skimming, i.e., copying the stolen credit card information onto plain credit cards for criminals to use the cards themselves. The topic is not that novel, but this enables us to benchmark and fine-tune our solution in a more trustworthy manner, as there are previous studies to compare with.

The target group of our solution are banks and consumers, namely the customers using the credit cards supplied by the banks and credit institutions. Our aim when designing the solution was to create a system that predicts credit card fraud by analyzing the consumer's typical behavior in terms of transactions and detecting the outliers from this data. Financial institutions, such as banks and lenders, could benefit from the solution to solidify their banking systems to automatically lock or freeze customer's accounts if they detect odd behavior in the customer's transactions.

2.1 Data collection

The data for this project was obtained from Kaggle. The data consists of credit card transactions done in September of 2013 by European cardholders in the timespan of two days. In the data there are over 280 000 rows of transactions, where the fraudulent cases only commit of 0.172% of the whole data, making the data heavily unbalanced. The data consists of 28 different features, which have been anonymized and normalized through Principal Component Analysis (PCA) in addition to the non-normalized time and amount of the transaction. The fraudulent cases are represented by the target variable called "Class", which maps 1 as being a fraudulent transaction, and 0 as a non-fraudulent. The data was stored and read in as a .CSV-file to our app, which worked fine in our project setting. We faced minor issues with git as our used version control system, as the data was too large for it without the git Large File System (git LFS). However, as the data is available through Kaggle, it

can be accesed on one's personal machine and used locally without any extra configuration to git or anywhere else.

2.2 Data preprocessing

As the data was already anonymized through PCA, there was not much left for us to do as a group regarding the preprocessing of the data. We normalized time and amount columns using vectorizers in python. The time we saved by not having to preprocess all columns was used to focus more deeply into the machine learning part of the project, and thus we implemented 6 different models.

3 Exploratory data analysis & visualizations

Initial EDA was performed in the form of looking at the data, its schema, and possible empty values (NaN's) if data imputation would have been needed. As stated earlier, the data was very clean right off the bat, so no big surprises found during the EDA phase. Again, as stated earlier, to have this clean data right at the start might not be the optimal situation in terms of the learning goals of the course. However, we considered it more as a time saver to focus more on the modeling phase during the project.

3.1 Visualizations

Streamlit was used as the underlying technology for creating interactive visualizations for the project in its different phases, such as one of the deliverables that was presented during our spotlight presentation. Streamlit is an open-source app framework used to share data apps more rapidly, and can turn python scripts into interactive web-apps easily without prior expertise in web-development. We also used matplotlib to plot confusion matrices, and different variations of ROC-AUC-curves. Streamlit was very helpful and time saving solution for us, which we can recommend and probably will also rely upon in the future with possible data science projects. Streamlit was specifically chosen because it is familiar to the entire team mainly through previous work experience.

4 Learning

The task of determining whether a certain credit card transaction is fraud or not is a type of classification problem. As a single transaction can either be a fraud or not, we have, more specifically, a binary classification problem on our hands. For this problem we used a variety of ML techniques, such as random forest, logistic regression, and XGBoost. As we are dealing with heavily unbalanced data, we are using Synthetic Minority Oversampling Technique (SMOTE) and bootstrapping to give us more reliable predictions, as the unbalanced data may lead to biased results.

4.1 Evaluated metrics

We used the same variety of metrics for each learning approach to evaluate the probability of possible credit card fraud, and compared the results with and without using SMOTE. The used metrics were:

- ROC-AUC (Receiver-Operating Characteristic and Area-Under ROC-Curve)
- AUC-PR (Area Under the Precision-Recall Curve)
- F2-score

F2-score is an important metric in our business case. F2-score is like F1-score, but puts more emphasis on recall than precision. This means that false negatives will be more costly than false positives. This is essential in our case, as when detecting credit card fraud, we want to be more safe than sorry. This means, flag more cases as fraud, which really are not fraud, rather than flag cases a not fraud which are fraud. This can be problematic especially in a trust-based business like banking.

4.2 Learning approaches

Since we had experience with machine learning models from either the corporate world or academic we already had some knowledge on what models work well with classification tasks. We decided to implement Logistic regression, random forest and XGBoost (Extreme Gradient Boosting) models on our data and analyze their performance. Since the data was extremely top heavy with one label we decided to test each model with and without SMOTE.

4.2.1 SMOTE

SMOTE is a statistical method for oversampling samples when having a heavily unbalanced dataset. SMOTE is used to reduce the bias in results, as when doing statistical analysis with unbalanced datasets there is always bias towards the majority class. In our case with the heavily unbalanced dataset, our initial results when using the aforementioned algorithms were close to 99%. With this the case was that the algorithms mostly flagged majority of the cases as non-fraud, as it was the majority class.

4.2.2 Logistic regression

Logistic regression was used with the "liblinear" solver, meaning both L1 (Lasso) and L2 (Ridge) regularizations were in place to prevent overfitting of the model. This solver was also sufficient enough due to our problem being a binary classification task.

4.2.3 Random forest

The random forest algorithm in our setting used 5% splits due to the large and unbalanced dataset, with 400 estimators and maximum depth of 6, and cross-validation with 5 splits. These hyperparameters were obtained after testing and fine-tuning by hand.

4.2.4 XGBoost

XGBoost can be considered as an enhanced version of the random forest algorithm. Our XGBoost used similarly to the random forest a 5% split in the testing dataset, with 400 estimators and maximum depth of 6, and cross-validation with 5 splits.

4.3 Frontend

We decided to use Streamlit as our frontend library because we had all used it in previous projects. The syntax is extremely straightforward, allowing us to avoid writing any HTML or CSS while still producing a visually appealing result. Building an interactive webpage that runs on the end-user's localhost was a satisfying goal, and we're very happy with the outcome. We also implemented a JSON caching system, which was practically mandatory for large machine learning algorithms. Running all the algorithms from scratch takes around 20–30 minutes, depending on the hardware. However, with the caching system, once they've been executed once, the results load instantly—even if the user resets the instance.

Further development could focus on enhancing the frontend to allow more interactivity, such as enabling users to choose parameters. This would be relatively easy to implement since we designed the functions so that the frontend calls them with parameters. However, due to time constraints, we decided not to include these features in the current version.

5 Fairness & data privacy

As stated earlier, the data was already anonymized since we pulled it from Kaggle. Hence, data privacy and fairness do not concern us that much at this stage of the project. However, in a real-world setting more emphasis should be put on these factors. It should be made clear that there is no hidden biases towards, e.g., gender, race, sex, or some other factor that could make the model biased. There have been known cases where machine learning applications based on financial data have been biased, and

discriminative towards certain ethnic groups based on the various features that the model uses, e.g., zip-code known to habit different ethnic groups.

6 Conclusions

6.1 Recap of the results

In the following table, we can see the performance comparison of the used machine learning methods with and without SMOTE. These same results are also available in a more interactive way in the streamlit application of the project. As a key takeaway, we can see that when SMOTE is in use, we get better recall that improves the F2-score at the cost of precison. This is important for our business-case, as we want to flag a more false positives, i.e., frauds, than there really is due to the costliness of flagginga fraud as legitimate transaction.

Identifying a learning task and prefered outcome is crucial when creating machine learning algorithms. We first had to figure out how we can tackle a dataset which is extremely imbalanced which is why we decided to use SMOTE. With SMOTE we found significant improvements in F2-score with the random forest model but XGBoost and LogReg suffered setbacks in F2-score. We found out that XGBoost is supreme in almost all aspects with or without smote which does make sense because it is the only model that backtracks it's progress with each split. In our real-world scenario banks need to protect their customers and minimize friction with their services. Our hypothesis is that customer sentiment will be better if the banks block too many transactions that are not fraud rather than missing those few frauds.

Model	Variant	ROC AUC	PR-AUC	F2-score
LogReg	No SMOTE	0.9401	0.7413	0.5859
LogReg	SMOTE	0.9182	0.5149	0.1834
RandomForest	No SMOTE	0.9524	0.8575	0.3704
RandomForest	SMOTE	0.9741	0.7247	0.7377
XGBoost	No SMOTE	0.9740	0.8934	0.8333
XGBoost	SMOTE	0.9927	0.8687	0.7836

Table 1: Performance comparison of models with and without SMOTE

7 LLM usage during the course project

We used a sprint like process where each individual owned modules in our repository and in order to glue these together with our streamlit front end we used ChatGPT to refactor the code for compatibility reason. ChatGPT was not used in writing or coding from scratch.