POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Sequence classification for credit-card fraud detection

Project report
Course: Numerical Analysis for Machine Learning

Author: **Francesco Buccoliero**

Student ID: [redacted]
Prof. Edie Miglio
Academic Year: 2023-24

Contents

Contents **i**

Introduction **1**

1 Technical Background **3**
1.1 The technical challenges . . . 3
1.2 Class imbalance . . . 4
1.3 Random Forests . . . 4
1.4 Long-Short Term Memory Networks . . . 5

2 Analysis of the paper **9**
2.1 The dataset . . . 9
2.1.1 Feature engineering . . . 10
2.2 Classfiers . . . 10
2.3 Metrics . . . 10
2.4 Results . . . 11

3 The project **13**
3.1 The dataset problem . . . 13
3.1.1 Generating dataset . . . 13
3.1.2 Data preprocessing . . . 15
3.2 Classifiers . . . 16
3.3 Results . . . 16

Bibliography **19**

Introduction

The theme of credit card frauds has been an important issue since their invention, but the rapid growth of e-commerce saw the rise of electronic (ECOM) payments over Face-to-Face (F2F), with an increasing struggle from financial institutions to challenge it.

Already in 2018 when the paper that we use as a reference for this project, **"Sequence classification for credit-card fraud detection"** [2], was published, the problem was felt as important by the academia. Nonetheless, the data used by the authors is dated back to 2015, when the size of e-commerce traffic and potentially the impact of credit card frauds was much smaller.

Since that year, the volume of online shopping has more than **quadrupled** [3].

The paper in its farsightedness already foresaw the growth of a **cashless banking economy** and the need to introduce **automatic tools** based on fraud detection rules to quickly process huge amounts of transactions. In this context the authors suggested incorporating in the model the concept of sequentiality of transactions, making use of a sequential learner such as LSTM and demonstrating its superiority over non-sequential learners (e.g. Random Forests).

The first part of this project report will focus on giving the reader some background context, then will move to analyze the intuitions, the methods and the outcomes of the paper. Finally the results of the project will be presented along with some final considerations.

1 | Technical Background

This chapter will include some sparse knowledge that will be required for the reader to grasp the following content. It might be skipped by readers with a Machine Learning background and an already advanced understanding of the differences between sequential and non-sequential learners.

1.1. The technical challenges

Credit card fraud detection serves as an important testbed for Machine Learning because it embodies numerous intrinsic properties. Firstly, the data distribution in fraud detection evolves over time, leading to what is known as **concept drift**. Concept drift occurs when the patterns observed in reality diverge from those incorporated into the model. This issue is partially mitigated through the implementation of an **ensemble of classifiers**, which are iteratively re-trained according to updating schedules.

Secondly, fraudulent transactions constitute only a small fraction of all transactions, resulting in **imbalanced classes**. This imbalance is addressed through techniques such as minority class oversampling, majority class undersampling, or a combination of both, known as **Synthetic Minority Over-sampling Technique (SMOTE)**.

Also, Fraud detection is inherently a **sequential classification** task, implying a temporal dependence between classes. Transactions from the same user, close in time, are likely to share the same fraudolent outcome.

Due to the nature of the problem, it would also need **real-time detection** (rather than periodically batch-processing transactions) and **online learning** capabilities.

Another complexity in fraud detection is the dynamic nature of misclassification costs: the cost of false positives (legit transactions marked as frauds) or **false negatives** (frauds marked as legit transactions) can increase or decrease based on the needs of the financial institutions. This issue is partially addressed by integrating an updated cost matrix into the learning algorithm, with costs proportional to the available credit limit.

Addressing all these problems would be out of the scope of this project as it would have been out of the scope of the paper. As the paper authors did, we have decided to focus on two main problems: class imbalance and sequentiality.

1.2. Class imbalance

As suggested in the previous chapter the problem of **class imbalance** (only 0.5% of transactions in the original dataset are fraudulent, Face-to-Face transactions reduce this percentage to 0.05%) can be solved making use of **SMOTE**.

In short, SMOTE operates by generating **synthetic examples in the feature space**, rather than by oversampling with replacement. The basic idea is to create new minority class instances by **interpolating** between existing ones.

The algorithm works as follows: for each minority class sample $\mathbf{x}_i$, SMOTE selects one or more of its k-nearest neighbors. A synthetic instance $\mathbf{x}_{\text{new}}$ is then generated by choosing a random neighbor $\mathbf{x}_j$ and creating a point along the line segment joining $\mathbf{x}_i$ and $\mathbf{x}_j$. This is mathematically represented as:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_i + \delta \times (\mathbf{x}_j - \mathbf{x}_i)$$

where δ is a random number between 0 and 1. This process can be repeated for the desired number of synthetic samples. By introducing these synthetic instances, SMOTE helps to balance the class distribution and improve the performance of machine learning models on imbalanced datasets. The interpolated samples ensure that the **decision boundary** is better defined, thereby reducing bias towards the majority class.

In the project, SMOTE is applied making use of the library `imblearn.over_sampling` that provides easy access to this useful technique.

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(sampling_strategy='auto')
X_rf, y = smote.fit_resample(X_rf, y)
```

1.3. Random Forests

Random forests (RF) will be used as non-sequential learners for comparison purposes. The paper authors suggested other possible comparison methods: as the problem is funda-

mentally a **binary classification**, any method from **Logistic Regression** to **Support Vector Machines** could have been used instead. We also chose RF as the paper authors due to the low implementation overhead.

Random forests are an **ensemble learning** method used for classification and regression tasks, which operate by constructing multiple **decision trees** during training and outputting the mode of the classes (classification) or mean prediction (regression) of the individual trees. The key steps in the algorithm are as follows:

1. **Bootstrap Sampling**: For each tree in the forest, a bootstrap sample (a random sample with replacement) of the training data is drawn.
2. **Tree Construction**: A decision tree is built using this bootstrap sample. At each node, a random subset of features is chosen, and the best split among these features is selected based on a predefined criterion (e.g., Gini impurity or entropy for classification).
3. **Tree Aggregation**: Once all trees are built, the forest combines their predictions. For classification, the output is the mode of the classes predicted by the individual trees. For regression, it is the average of the predictions.

The algorithm can be summarized as:

$$\hat{y} = \text{mode}\left(\{T_i(\mathbf{x})\}\right) \quad \text{for classification}$$

$$\hat{y} = \frac{1}{N}\sum_{i=1}^{N} T_i(\mathbf{x}) \quad \text{for regression}$$

where $T_i(\mathbf{x})$ denotes the prediction of the i-th tree for input $\mathbf{x}$. Random forests improve predictive performance by **reducing overfitting** and increasing generalization through the aggregation of diverse trees.

In the project, the standard `sklearn.ensemble.RandomForestClassifier` implementation of Random Forests has been used.

1.4. Long-Short Term Memory Networks

As we have seen we need to incorporate the sequentiality of transactions as two identical transactions can appear as totally legitimate or totally fraudulent according to the context they're in. The authors of the paper identified two methodologies to support the decisions

of the detector: the first is **feature engineering**, but we'll address it when talking about the dataset, the second is to use a **Recurrent Neural Network**.

Recurrent Neural Networks (RNNs), particularly **Long Short-Term Memory (LSTM)** networks, are employed for sequence classification tasks. These networks are good at capturing the sequential structure of a customer's transaction history by modelling the transition dynamics between transactions. An RNN functions similarly to a **multilayer perceptron** but includes additional connections among layers corresponding to different **time steps**.

The state vectors in an RNN are determined by three parameters: **recurrent weights, input weights, and biases**. Typically, the activation function used is the hyperbolic tangent (tanh), the cost function is a cross-entropy function, and the output distribution is modeled using logistic regression. Classical gradient optimization methods can be enhanced using **Backpropagation Through Time (BPTT)**, which involves "unrolling" the network over time and representing it as a deep multi-layer network with as many layers as there are time steps.

RNNs face two major issues:

1. **Exploding Gradients**: This occurs because the gradient of the loss function with respect to the parameters depends on a Jacobian matrix that includes all component-wise interactions between state vectors. Solutions include L1 or L2 regularization and gradient clipping, which sets a threshold for gradients.
2. **Vanishing Gradients**: This is the opposite of the exploding gradients problem. It was theoretically solved in 1997 and practically applied in 2015 with the introduction of memory cells, the fundamental components of LSTMs.

These memory cells in LSTMs help mitigate the vanishing gradient problem by maintaining **long-term dependencies** in the data. They achieve this through a carefully designed gating mechanism, which regulates the flow of information. The **cell state**, modified by input, forget, and **output gates**, allows relevant information to persist over long sequences, effectively **preserving gradient flow** during backpropagation. This design ensures that gradients do not diminish as they are propagated back through time, allowing the network to learn and remember long-term dependencies.

In the project, the `keras` implementation of LSTM layers has been used, twice, in a sequential model, as long as **Dropout** layers to reduce overfitting and finally output **Dense** layers.

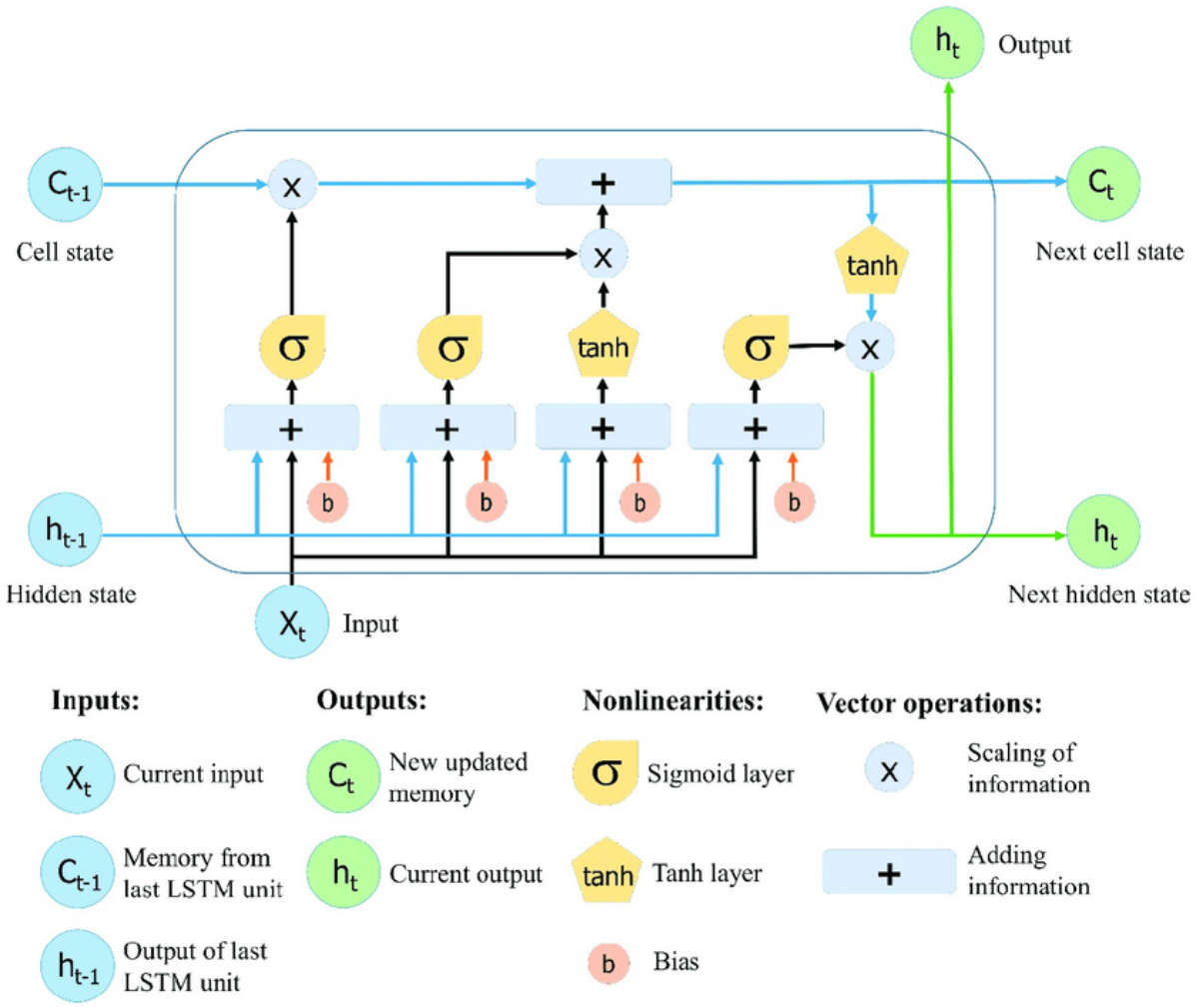

Figure 1.1: Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting - Scientific Figure on ResearchGate. https://www.researchgate.net/figure/The-structure-of-the-Long-Short-Term-Memory-LSTM-neural-network-Reproduced-from-Yan_fig8_334268507

2 | Analysis of the paper

As we have seen, the goal of the paper[2] is to demonstrate the superiority of sequential learners over non-sequential ones. Analyzing the dataset and the methods used we can find some interesting information that we can use in the last part when trying to work with the same problem.

2.1. The dataset

The dataset utilized by the paper's authors is not publicly available and spans a **three-month observation period** in 2015. It was manually constructed by experts based on established rules. Transactions were **grouped by cardholder** and **sorted chronologically**, transforming the transaction-based dataset into a **sequence-based** dataset. The sequences were further split by card entry mode, creating two distinct sequences.

Regarding the **class imbalance**, approximately 0.5% of transactions are fraudulent (and only 0.05% in face-to-face transactions) An account was considered compromised if it contained at least one fraudulent transaction. With a 0.9 probability, an account was randomly picked from genuine ones, and with a 0.1 probability, from compromised ones, repeating this process one million times.

For **dataset alignment**, transactions not followed by at least nine others were excluded to ensure comparability with Random Forest (RF) results and to meet LSTM requirements. Given an average of four fraudulent transactions per compromised account, a **sequence length** of five transactions (SHORT) was chosen, with a second scenario doubling this length (LONG).

Since this kind of data has to comply with international standards it is very homogeneous. Business-specific identifiers have been for generalization.

Three **feature sets** were defined: BASE, TDELTA (adding the time delta feature), and AGG (including aggregation features). Gaussian normalization was applied to variables where applicable. Categorical variables were encoded using **classical label encoding** and time was similarly encoded by splitting each temporal resolution into categorical

variables.

2.1.1. Feature engineering

Manual feature engineering was employed by the paper's authors. They focused on the **time between consecutive transactions** rather than the exact time of purchase. New features were created by **aggregating values** of certain variables over time. Aggregation was based on matching parameters and using a **24-hour timeframe**. Datasets were separated for face-to-face (F2F) and e-commerce (ECOM) transactions to highlight behavioural patterns.

2.2. Classfiers

For the two classifiers the paper used:

- **LSTM**: Sequential model with two recurrent layers, followed by a single hidden layer. A logistic regression classifier is placed atop the final layer. Dropout is applied to the nodes within the LSTM.
- **Random forest**: For comparative purposes, the SciKit-Learn implementation of Random Forest (RF) is utilized.

A grid search has been also performed to tune the following hyper-parameters

- **LSTM**: learning rate, dropout rate, recurrent nodes per layer, hidden nodes
- **Random Forest**: minimum sample leaf, splitting criterion, maximum number of features, number of trees

2.3. Metrics

The following metrics have been considered:

- **AUCPR**: The area under the precision-recall curve is employed instead of the ROC metric due to class imbalance, which reduces the significance of ROC by its insensitivity to false positives.
- **AUCPR@0.2**: For business purposes, a low recall/high precision is preferred. An acceptable recall value (0.2) is set, and the area under the curve up to this value serves as a business-driven metric.
- **Jaccard Index**: Given two sets of true positives, A and B, the Jaccard Index

$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$. The sets are the true positives selected at 0.2 recall for RF and LSTM.

For each combination of feature set, dataset, and sequence length (irrelevant for RF), the model is tested over 24 test days individually, with results averaged.

2.4. Results

The model over ECOM class demonstrates higher overall detection accuracy compared to the F2F class, likely due to a higher fraud ratio in ECOM transactions. The length of the input sequence has minimal effect on accuracy. However, **incorporating previous transactions using LSTM significantly improves results** for F2F.

Feature aggregations enhance fraud detection, particularly for ECOM. Interestingly, feature aggregation benefits scenarios where sequence modelling does not, indicating that **these context representations are complementary**. Adding aggregate features considerably improves random forest results, making them nearly comparable to LSTM. In ECOM, performance is consistently comparable across various subsets.

Prediction accuracy varies significantly depending on the test day, suggesting that some tasks are inherently more challenging. A density analysis confirms high variability in fraud distribution over days. Jaccard Index analysis reveals that **RF and LSTM detect different frauds**, with *inter-model agreement ranging from 25% to 35%* in ECOM. Notably, models trained with aggregate features tend to detect unique frauds, showing the potential of a **combined approach** for future research.

3 | The project

Recreating the outcomes of the given paper was an interesting challenge due to the very peculiar nature of the data we work with.

3.1. The dataset problem

The most famous **public dataset** (https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud) contains transactions (492 frauds out of 284,807 transactions) made by credit cards in September 2013 by European cardholders.

Unfortunately, due to confidentiality issues, features have been anonymized and we cannot apply the same **feature engineering** as the paper authors did. The very same situation can be observed for other less common datasets. The choice of datasets is therefore limited to two, both limited, options:

- Using publicly available datasets, already anonymized, normalized and where it's impossible to perform feature engineering nor implement sequentiality
- Resort to **synthetic datasets**, trying to incorporate the sequentiality found in real-life datasets and where it would be possible to perform feature engineering.

Unfortunately, both options are not satisfactory with regard to the premises of the paper, as they both achieve very **unrealistic results** even with simple implementation of the classifiers.

At last, in order to proceed, the second option was chosen, given the availability of a very detailed handbook[1] for this very task, to try to recreate at best the experiments of the paper authors.

3.1.1. Generating dataset

This study utilized the *"Reproducible Machine Learning for Credit Card Fraud Detection"* handbook [1] to create a synthetic dataset, modified to enhance realism. The dataset features include **geolocation** data for both customers and merchants, **transaction times**,

and **scenario-based** fraudulent signaling (e.g., compromised terminals or cards). These scenarios introduce real-life **concept drift** into the model.

The code generates customer and terminal profiles using **random distributions** for various properties such as geographic coordinates, card issuer, and merchant category codes (MCCs). Transactions are simulated based on customer profiles and considering **distance-based transaction types** (Face-to-Face or E-commerce). The labels of the dataset are defined by two fraud scenarios:

1. Compromised terminal
2. Stolen card

Fraudulent transactions are generated with specific probabilities and temporal patterns to **mimic real-world fraud activities**. The fraud scenarios introduce randomness in the number of compromised cards/terminals, days of compromise, and transaction outcomes, especially highlighting the **higher likelihood of fraud in e-commerce** transactions compared to Face-to-Face transactions.

The complete synthetic dataset can be stored in files if needed, for ease of access and reproducibility.

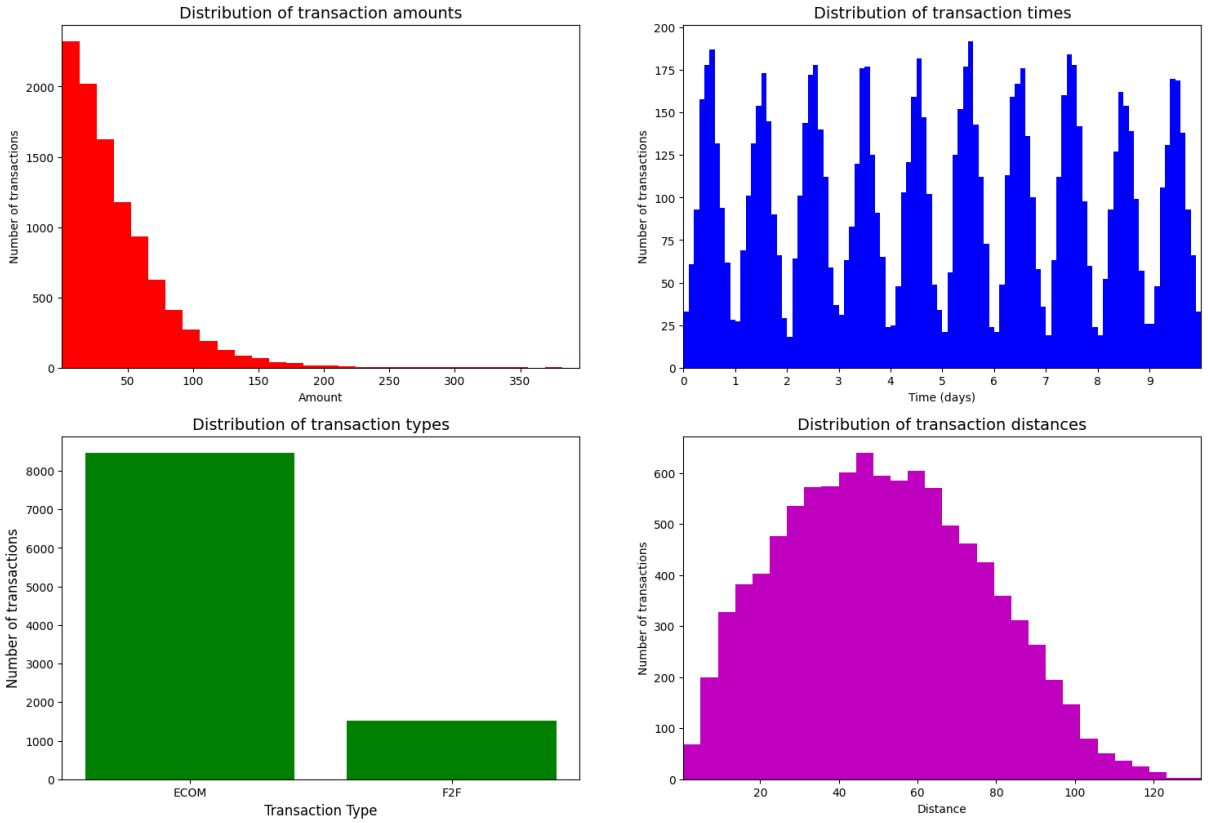

Figure 3.1: Some graphs related to the synthetic dataset, highlighting the design choices taken.

3.1.2. Data preprocessing

An appropriate function preprocesses transaction data by normalizing numerical features and binary encoding categorical features using the `StandardScaler` and `BinaryEncoder` implementations of `sklearn`. When a non-zero time step parameter is provided, the function also adds a specific preprocessing step for LSTM: it generates sequences with the specified time step, retaining transaction IDs and fraud labels for sequential model training.

Please note that in the LSTM preprocessing sequences have been considered as fraudulent only if **all the transactions** in the sequence are fraudulent. This approach yields slightly better results rather than a majority vote or considering the sequence fraudulent if at least one transaction is.

The preprocessing function also takes one parameter named `full_dataset`, that allows to yield two different type of datasets, in a similar fashion as the paper authors did:

- The **BASE** dataset, including only essential information, no sequentiality and no time feature engineering.
  - Feature columns include: `TX_DATETIME, TX_AMOUNT, TX_TYPE`.
- The **FULL** dataset, including all the informations available
  - Feature include all previous and: `TX_TIME_SECONDS`, `TX_TIME_DAYS`, `CUST_TX_ID` ,`CUSTOMER_ID`, `TERMINAL_ID`, `CUSTOMER_CARD_ISSUER`, `CUSTOMER_CARD_TYPE`, `TERMINAL_MCC`

where `TX_TYPE` is ECOM/F2F, `TERMINAL_MCC` is the merchant category code, used in the synthetic model to incorporate information about the probability of terminal compromission, `CUSTOMER_CARD_ISSUER` and `CUSTOMER_CARD_TYPE` (e.g. Visa/Debit) also incorporate probability of fraud (stolen card).

3.2. Classifiers

The two classifier implementations follow shortly from the paper ones, with an important step of SMOTE oversampling, followed by classic train/test split and finally model fit.

It is interesting to note how addressing the class imbalance has been the single **most significant contribution** to the final result, with poor performances over all classes of classifiers when the SMOTE step was skipped.

Tuning hyperparameters doesn't leave much room for improvement in results, infact no **grid-search** has been implemented, but rather some **manual tuning** considering the structure of the dataset. The reader might argue that such considerations could have not been made with such precision on a real dataset, and the writer agrees, but since a preliminary study of the dataset could have been made under any circumstance, parameters such as the **LSTM time step** could have also been estimated with a sufficient precision.

3.3. Results

As anticipated, results are not comparable with ones of the papers because under the given circumstances not enough real-life "noise" and randomness could have been incorporated in the synthetic dataset, leading to **very high results** both in accuracy and AUCPR. Yet, despite these **technical limits**, and even if barely noticeable, LSTM did

show some **improvements over RF**, and **feature engineering** is also shown to increase the metrics. The results of the paper are therefore reproduced correctly.

In the following table the results, where the first digit is accuracy and the second is AUCPR.

	RF	**LSTM**
BASE	0.898627 / 0.917177	0.984581 / 0.996151
FULL	0.999474 / 0.999883	0.999883 / 0.999998

Table 3.1: Performance Comparison between RF and LSTM Models

Bibliography

[1] Reproducible machine learning for credit card fraud detection - practical handbook, 2020. URL https://fraud-detection-handbook.github.io/fraud-detection-handbook/Foreword.html. Machine Learning Group (Université Libre de Bruxelles - ULB).

[2] J. J. et al. Sequence classification for credit-card fraud detection. *Expert Systems with Applications*, 2018. URL https://doi.org/10.1016/j.eswa.2018.01.037.

[3] Statista. Retail e-commerce sales worldwide from 2014 to 2027, 2024. URL https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/.