Credit Card Fraud Detection

Data Spaces Exam

Debora Caldarola

Prof. Francesco Vaccarino

Politecnico di Torino

a.y. 2019/2020

Contents

1 Introduction **iii**

2 Credit Card Fraud Detection **1**
2.1 Introduction to Anomaly Detection 1
2.2 Dataset and environment setup 1

3 Dataset Analysis **2**
3.1 Features Distribution 2
3.2 Features Correlation 3

4 Dealing with imbalanced dataset **5**
4.1 Data Preprocessing 5
4.2 Under-Sampling 6
4.3 Over-Sampling 7
4.4 Combining Over-Sampling and Under-Sampling 7
4.5 Synthetic Minority Oversampling Technique (SMOTE) 8
4.6 Features Correlation 10

5 Anomaly Detection **11**
5.1 Metrics 11
5.2 Hyperparameters Optimization 12
5.3 Classification 12
5.3.1 Logistic Regression 12
5.3.2 K-Nearest Neighbors (KNN) 13
5.3.3 Support Vector Machine (SVM) 13
5.3.4 Decision Tree Classifier 14
5.3.5 Random Forest Classifier 15
5.4 Results Analysis 16

1 Introduction

Nowadays, credit cards play a determinant role in many people's life. According to statistics [7], there are 2.8 Billion credit cards in use worldwide of which 1.06 Billions in the United States. Moreover, as reported by *Experian State of Credit Report* [9] and the *Fifth report on credit fraud* published by the European Central Bank [3], 67% of the American people own at least one credit card with an average of four each, while in Europe the number of cards carried per inhabitant ranges from 0.8 to 3.9.
Furthermore, as stated by the *Word Payments Report 2019* [5], non-cash payments are expected to reach the number of more than 1 trillion transactions by 2022: that's a huge quantity!
At the same time, the downside that comes from using credit cards is the presence of fraudulent transactions: in 2018, $24.26 Billion was lost due to payment card fraud worldwide [6] and that loss has been steadily increasing for the last five years [8]. As a consequence, different techniques were born to prevent fraudulent transactions from taking place and especially to detect them: here we will address the *anomaly detection* process applied on the Credit Card Fraud Detection dataset.

2 Credit Card Fraud Detection

2.1 Introduction to Anomaly Detection

Anomaly detection is the process of identifying rare items, observations or events that differ from the majority of the data and therefore raise suspicions. Anomalies are usually referred to as outliers and represent a change in the data pattern. Anomaly detection is applicable to a large variety of domains, such as fraud detection, intrusion detection, fault detection, system monitoring. It is based on two main assumptions:

- Anomalies rarely occur in the data
- Their features can be told apart from the ones belonging to normal instances

The singularity characterizing anomalies is what makes them hard for a human to detect: the main exploitable techniques to address this issue are based on statistics and machine learning algorithms.
Here, we will focus on the **Credit Card Fraud Detection Problem**: starting from past credit card transactions and the knowledge of the ones that turned out to be fraud, we aim to build a model able to identify whether a new transaction is fraudulent or not. Our goal is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications.

2.2 Dataset and environment setup

The analysis and the experiments are performed on the Kaggle CREDIT CARD FRAUD DETECTION dataset (link).
The dataset contains two days-credit cards transactions made by European cardholders in September 2013.
All results are generated using Python programming language and *Colab* environment. Mainly, `sklearn`, `imblearn` and `numpy` libraries are used.
The notebook can be found at https://github.com/debcaldarola/Credit-Card-Fraud-Detection.

3 Dataset Analysis

The CREDIT CARD FRAUD DETECTION dataset contains 284,807 credit card transactions, executed by European cardholders during two days in September 2013.

3.1 Features Distribution

Each transaction is characterized by 31 features:

- `Time`: the seconds elapsed between each transaction and the first one in the dataset (Figure 1).
- `Amount`: the transaction amount (Figure 2).
- `V1`, `V2`, ..., `V28`: the principal components of the original features obtained with Principal Component Analysis (Figure 3). Due to privacy constraints, no other information about the original values or their background is provided.
- `Class`: the binary label dividing the dataset into non-fraudulent (`Class`=0) and fraudulent (`Class`=1) transactions (Figure 4).

As Figure 4 shows, the dataset is highly **imbalanced**: only 492 transactions out of 284,807 are *fraudulent* for a percentage of 0.00172%.

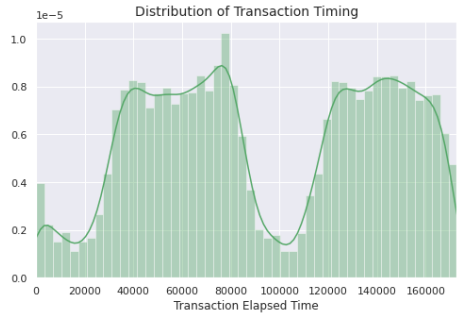


Figure 1: Time distribution

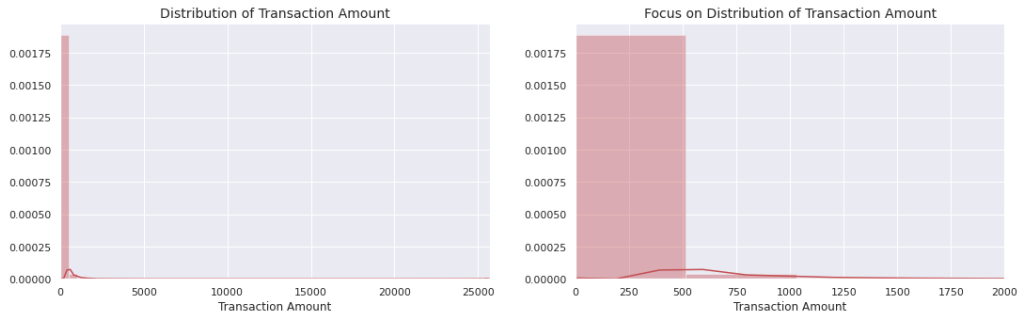


Figure 2: Amount distribution

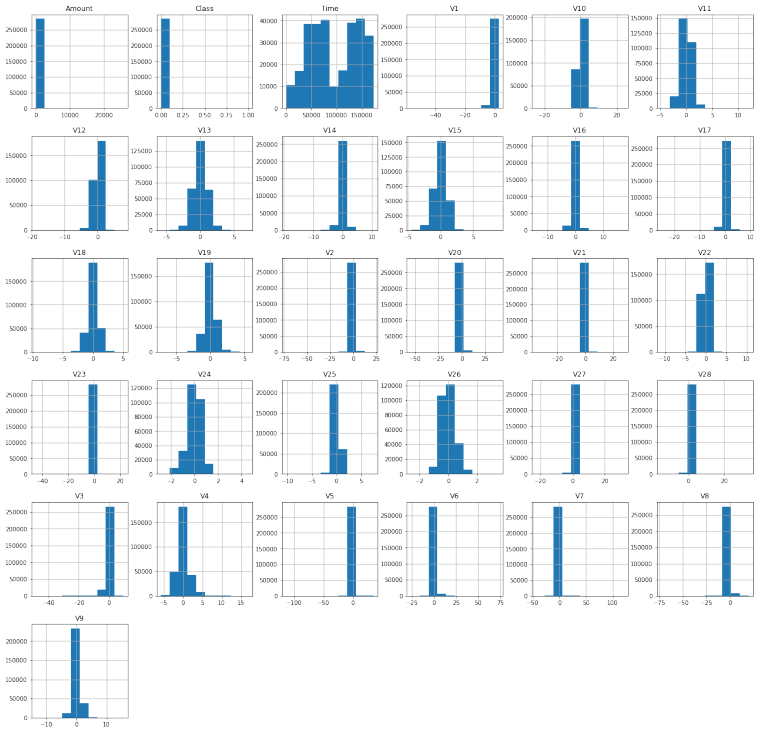


Figure 3: PCA components `V1,V2,...,V28` distribution

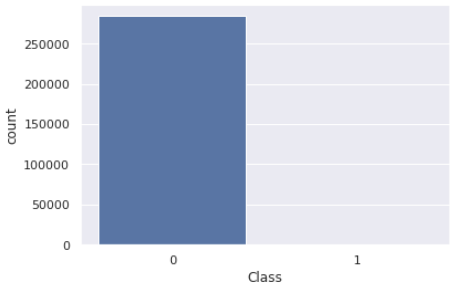


Figure 4: Class distribution

3.2 Features Correlation

In order to better understand our data and the relationships between the different features, we can use correlation matrices. In particular, we want to know if there are features that heavily influence in whether a specific transaction is a fraud. However, if the wrong dataframe is used, that analysis cannot be performed.
Figure 5 shows the correlation matrix built using the whole dataset: we can notice how most of the features do not show any correlation. The reason behind these results is that correlations are more difficult to find within a very unbalanced data frame like ours. Since features correlation has a major impact on models' performances, the imbalanced problem has to be solved.

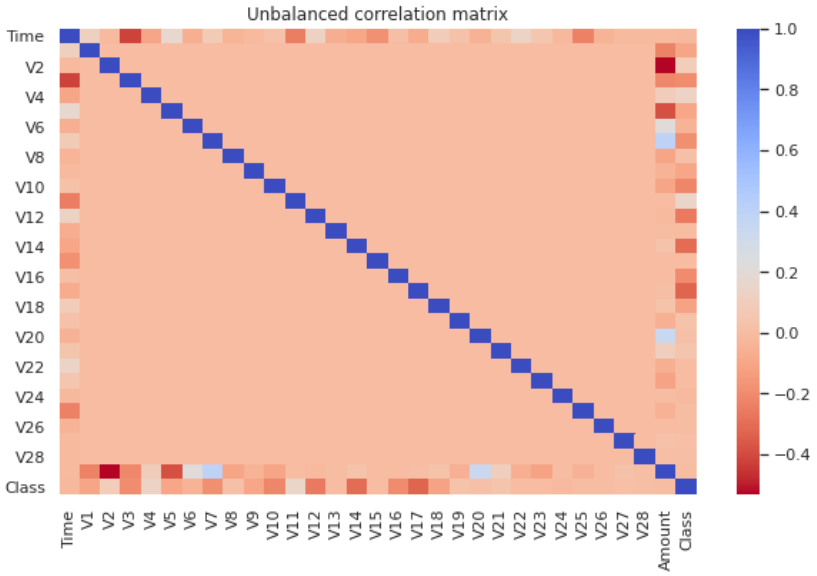


Figure 5: Imbalanced Correlation Matrix

4 Dealing with imbalanced dataset

Imbalanced datasets pose a challenge for predictive modelling as most of the machine learning algorithms used for classification are designed around the assumption of an equal number of examples for each class. That results in models having poor predictive performance, specifically for the minority class. That behaviuor is problematic especially in the anomaly detection field, in which the minority class is more important and should be identified with the least possible margin of error. In our case, we are more interested in the fact that our model is able to correctly classify all fraudulent transactions rather than non-fraudulent ones.
When dealing with imbalanced datasets, two main solutions can be applied:

1. Usage of resampling techniques
2. Change in the evaluation metrics (discussed in Chapter 5).

As for the sampling techniques, they are used to create balanced datasets before fitting a classifier on it. They can be divided into three main classes:

- **undersampling**, consisting in sampling from the majority class in order to keep only a part of those points
- **oversampling**, consisting in replicating some points from the minority class in order to increase its cardinality
- **generating synthetic data**, that is to say, creating new synthetic points from the minority class to increase its cardinality (SMOTE, for example).

It is important to remark that resampling techniques should only be applied on the training set, so as not to distort testing outcomes.
In this Chapter, all the techniques and their results will be introduced. Our goal is to find the sampling technique and the classification method that perform best.

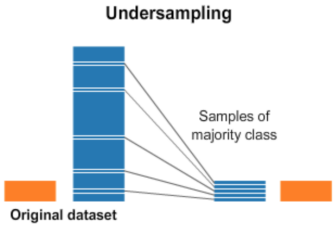
4.1 Data Preprocessing

Considering that the values referring to the features `Time` and `Amount` are much higher than the others, data is **normalized**, in order to reach a Gaussian distribution for all the parameters. In particular, all data - except the label - is scaled according to the following formula:

$$X' = \frac{X - mean(X)}{max(X) - min(X)}$$

Then the scaled data is split into 80% train and 20% test sets, both having the same percentage of positive and negative labels, respectively equal to 0.00172 and 0.99827. Figure 8 shows the training set plotted in 2D before applying any resampling technique is applied: there are 392 fraudulent transactions and 227,451 non-fraudulent ones.

4.2 Under-Sampling



Under-sampling aims to reduce the number of majority samples to balance the class distribution. Since it removes observations from the original data set, it might discard useful information. At the same time, training and test time might decrease as a consequence of the presence of fewer samples.
Here we introduce undersampling performed in two different ways:

1. Manual undersampling: 394 non-fraudulent transactions are randomly picked up by the shuffled data

2. Random undersampling with Python `imblearn` module: ad hoc class to perform random under-sampling. It undersamples the majority class by randomly picking samples without replacement.

In both cases, the resulting class distribution is equal: the dataset is made up of 784 samples, equally distributed between the two labels (Figure 6).
The resampled datasets are plot in Figures 9 and 10.

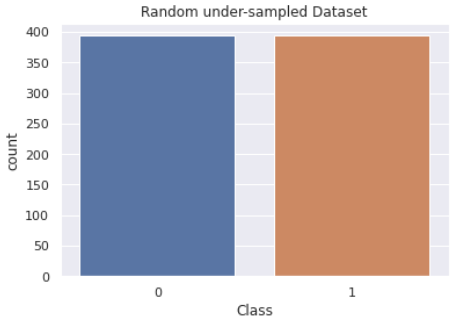
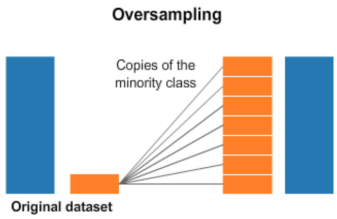


Figure 6: Class distribution after undersampling

4.3 Over-Sampling



Over-sampling increases the number of minority class samples in the training set. The main resulting advantage is that no information from the original training set is lost, as all the observations from the minority and majority classes are kept. On the other hand, it is prone to overfitting.
As in the Under-Sampling case, Over-Sampling is performed both manually (samples having label equal to `1` are randomly repeated in order to obtain a final quantity equal to the number of transactions having label `0`) and with the RandomOverSampler exposed by Python `imblearn` module. The resulting training sets are shown in Figures 11 and 12: the main detectable difference is the presence of more scattered data in the first set. The equal class distribution is represented in Figure 7: there are 227,057 newly inserted samples, for a total of 454,902 transactions, 227,451 for each class.

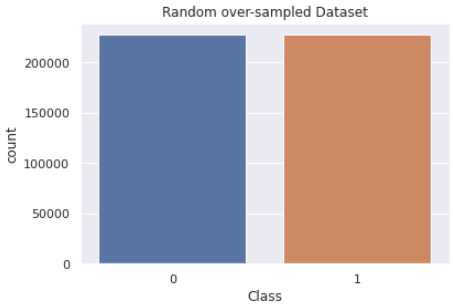
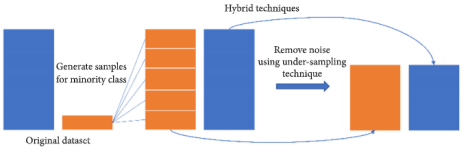


Figure 7: Class distribution after oversampling

4.4 Combining Over-Sampling and Under-Sampling



Over-Sampling and Under-Sampling may be combined to achieve better results: in particular, the combination of those two techniques allows us to improve the

bias towards data points of the minority class by first applying oversampling; then undersampling reduces noise and, therefore, the probability of overfitting. In our case, we oversampled the original training set using the following ratio for the sampling strategy:

$$N_m = 0.1 \cdot N_M$$

where N_m and N_M are the number of samples of the minority and the majority class respectively. So the equation becomes:

$$N_m = 0.1 \cdot 227451 = 22745$$

After that, undersampling is applied and we obtain 22,745 samples for both positive and negative samples.
The resulting training set is displayed in Figure 13.

4.5 Synthetic Minority Oversampling Technique (SMOTE)

The main disadvantage in using standard oversampling is the major probability of overfitting, due to the fact that the model sees the same data points more than once at training time. A possible solution to that problem is to create synthetic samples starting from the existing ones, instead of simply duplicating them: this method can be seen as a form of data augmentation for the minority class and the most widely used approach is the Synthetic Minority Oversampling Technique, or SMOTE.
SMOTE first selects a minority class instance a at random and finds its k=5 nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b [10].
Figure 14 displays the training set sampled with SMOTE. As in the previous oversampling case, the total number of transactions is 454,902.

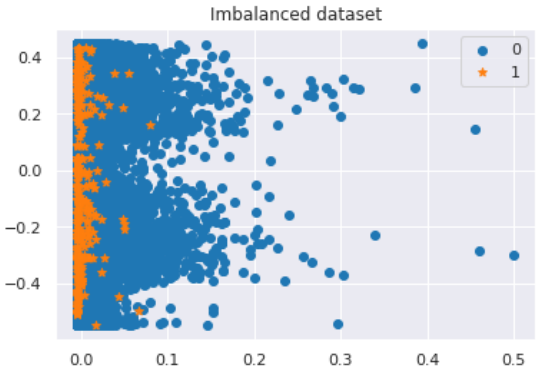


Figure 8: Training set in 2D

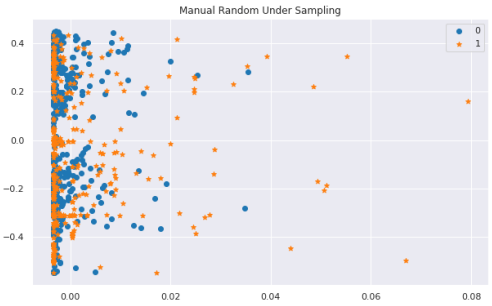


Figure 9: Manual Random Undersampling

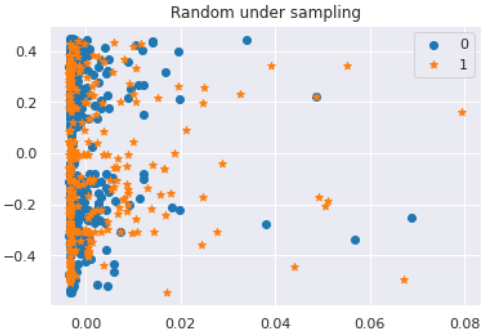


Figure 10: Python Random Undersampling

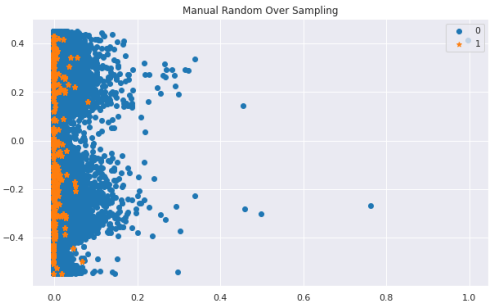


Figure 11: Manual Random Oversampling

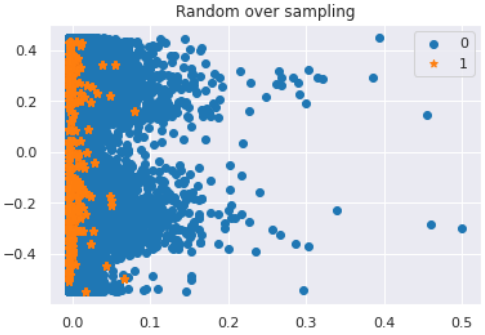


Figure 12: Python Random Oversampling

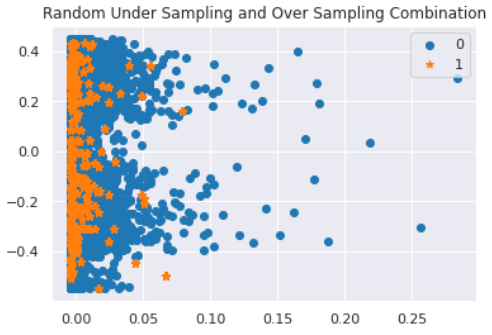


Figure 13: Combining Oversampling and Undersampling

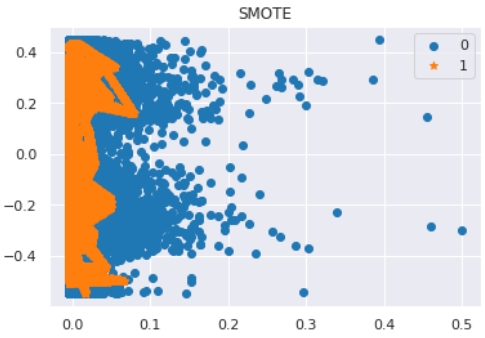


Figure 14: SMOTE

4.6 Features Correlation

Once the dataset is balanced, the correlation matrix can be plotted again (Figure 15). We can notice how the features correlations are now much more highlighted than those showed in Chapter 3.
In particular, those are relations we can point out:

- `V20, ..., V28`, `Amount` and `Time` are little related to the `Class`.
- `V1, ..., V19` are highly correlated.

We can take a look at the box plots in Figure 16: they focus on the distribution of the more correlated features `V17`, `V14`, `V12` and `V10` (negatively correlated) and `V2`, `V4`, `V11` and `V19` (positevely correlated) in fradulent and non fradulent transactions: considering the absolute value of their correlation, how the higher these values are, the more likely the end result will be a fraud transaction.

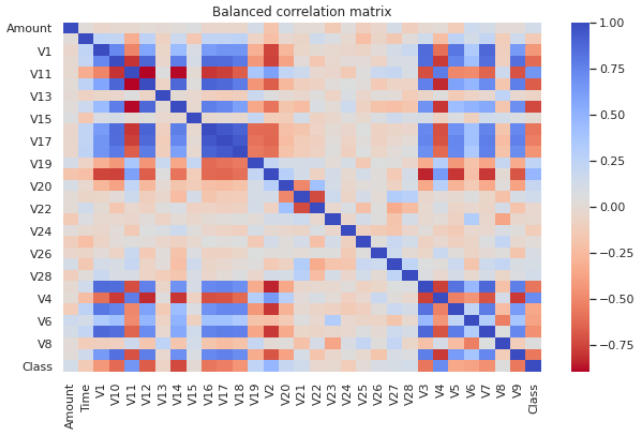


Figure 15: Correlation Matrix of Balanced Dataset

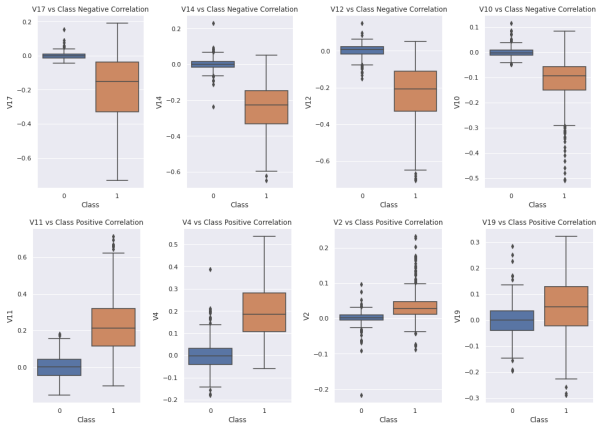


Figure 16: Highly correlated features vs class distribution

5 Anomaly Detection

Given the training sets balanced with the sampling techniques introduced in Chapter 4, now our goal is to perform anomaly detection and to find out the combination of the sampling technique and the classifier performing best on the test set.
First, we discuss which metrics should be used to evaluate the results; then we perform a grid search to optimize the hyperparameters; finally, the following classification techniques are applied and compared:

- Logistic Regression
- K-Nearest Neighbors
- Support Vector Machine
- Decision Tree Classifier
- Random Forest Classifier

Results are showed in Figures 27 and 28.

5.1 Metrics

The metrics with which machine learning models are generally evaluated is accuracy, which is mathematically defined as:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP stands for *True Positive* (fraudulent transactions predicted as fraudulent), TN are the *True Negatives* (non-fraudulent transactions predicted as non-fraudulent), FP are the *False Positives* (non-fraudulent transactions predicted as fraudulent) and FN means *False Negative* (fraudulent transactions predicted as non-fraudulent). So that implies accuracy represents the fraction of predictions got right by the model.
When using imbalanced datasets, accuracy turns out not to be a good evaluation metrics: 99.99% of our dataset consists of non-fraudulent transactions, so even if all data were classified as non-fraudulent and, therefore, all fraudulent transactions were misclassified, we would still have a total accuracy of 99.82%. But that is not what we want to achieve!
The solution is to look at different evaluation metrics. In particular, in this notebook, we make use of the following ones:

- **confusion matrix**, a representation of the results divided into TP, TN, FP, FN
- **recall** = $\frac{TP}{TP+FN}$, so the percentage of fraudulent transactions that were correctly identified by our model: this is what interests us most
- **precision** = $\frac{TP}{TP+FP}$, so percentage of all the transactions predicted to be fraudulent that are actually fraudulent
- **F1-score** = $2 \cdot \frac{precision \cdot recall}{precision + recall}$, a balance between precision and recall.

5.2 Hyperparameters Optimization

In order to find the models parameters performing best with our data, a **Grid Search** is proposed: it performs an exhaustive search over specified parameter values for an estimator and returns the ones achieving the best results at training time according to a certain metrics, which is F1-score in our case. The parameters of the estimator are optimized by a 3-folds cross-validated grid-search over a parameter grid, so the total number of fits performed for each classifier on each set is

$$tot\_fits = \prod_{i=1}^{n\_parameters} n\_candidates_i \cdot 3$$

where $n\_parameters_i$ is the number of parameters to be optimized for the current classifier and $n\_candidates_i$ is equal to the possible candidates for that paramater.
Table 1 shows the total fits evaluated for each classifier and the average execution time over the six training sets: as the number of data increases, search time increases exponentially.

Classifier	Total fits	Average execution time (min)	Minimum time (sec)	Maximum time (sec)
Logistic Regression	42	1.72	0.4	210
KNN	60	1255.44	0.8	25,758
SVC	48	> 1,440	2.8	> 86,400
Decision Tree	270	172.66	2.2	5064
Random Forest	12	> 1,440	16	> 86,400

Table 1: Grid Search statistics

SVC and Random Forest could not be tested on all the possible parameters combinations on all sampled training sets because of a too long an execution time, not supported by *Kaggle*. The number of parameters or their candidates could not be decreased any further, so not-tested parameters were estimated according to results obtained by the same classifier on other sampled sets.

5.3 Classification

According to the optimized parameters obtained in the grid search, each classifier is trained and tested on all the training sets resampled with the previously presented techniques: manual and random undersampling, manual and random oversampling, combination of undersampling and oversampling, SMOTE.

5.3.1 Logistic Regression

The logistic model is used to model the probability of a certain class or event existing. Using the Sigmoid function, each object being detected in the image

would be assigned a probability between 0 and 1, with a sum of one. A threshold is defined in order to map that probability to a binary outcome.
The tuned parameter is $C = \frac{1}{\lambda}$, the inverse of regularization strength. λ aims to find a tradeoff between model simplicity (high λ value), which might lead to underfitting, and too high a complexity (low λ value), which might result in overfitting the data: as a consequence, lowering C implies strengthening the lambda regulator.
For all the resampled training sets, C is set equal to 1000.
The resulting F1-score is low because of a poor precision: many non-fraudulent transactions are wrongly classified as fraudulent. On the other hand, the average recall is high: 89.5% of the fraudulent transactions were correctly detected.
The best F1-score is obtained by the training set sampled with the combination of oversampling and undersampling and is equal to 0.13. The randomly undersampled data achieves the best recall score with a result of 91%. Their confusion matrices are shown in Figures 25 and 26.

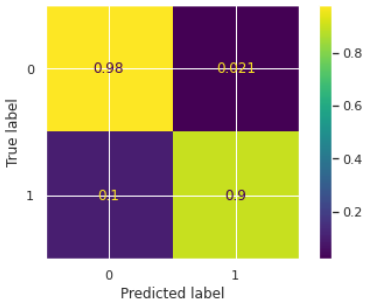


Figure 17: Confusion Matrix: Logistic Regression best F1-score (C=1000)

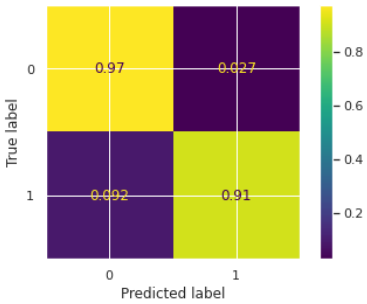


Figure 18: Confusion Matrix: Logistic Regression best recall (C=1000)

5.3.2 K-Nearest Neighbors (KNN)

The KNN algorithm assumes that similar data points are close to each other: it captures the idea of similarity by calculating the distance among the samples. The object is classified according to the most frequent class of its k neighbors. The grid search was used to find the best value for k, the weight function used in prediction and the distance metrics.
The best F1-score is much higher than the one obtained with Logistic Regression: 82% in the training set manually oversampled having *k=1*. The best recall score is 85% on the manually undersampled set, having *K=6*.
In general, KNN performs better on precision with respect to Logistic Regression, while we loose some percentage points when it comes to recall results.

5.3.3 Support Vector Machine (SVM)

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. Their main advantage is the effectiveness in high dimensional spaces thanks to the application of the *kernel*

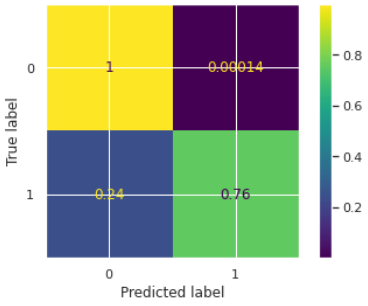


Figure 19: Confusion Matrix: KNN best F1-score (K=1)

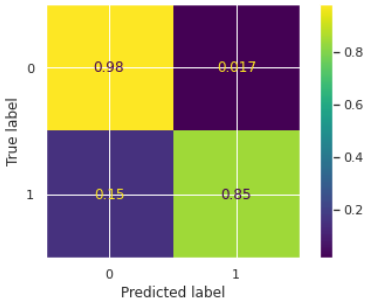


Figure 20: Confusion Matrix: KNN best recall (K=6)

trick. For performing binary classification, SVC (Support Vector Classifier) is used.
The *Radial Basis Function* (RBF) is chosen as kernel function: according to [2], it is the one performing best on fraud detection. It is defined as follows:

$$K(x, x') = \exp(-\gamma \|x - x'\|^2)$$

where γ defines how far the influence of a single training example reaches and can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.
Another important parameter is C, which builds a tradeoff between the correct classification of the training samples and the decision function's margin.
Both C and γ are chosen accordingly to the grid search results.
Best results achieved: F1-score = 0.21 on the randomly undersampled set; recall = 0.90 on both manually and randomly oversampled and oversampled-then-undersampled sets.

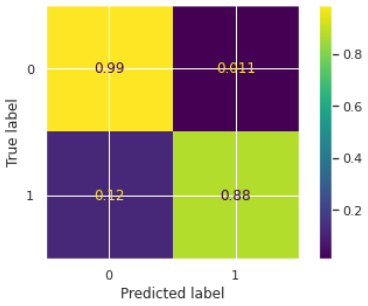


Figure 21: Confusion Matrix: SVC best F1-score (C=10, γ=1)

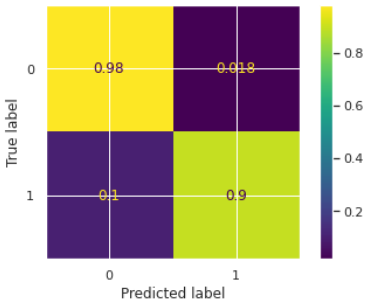


Figure 22: Confusion Matrix: SVC best recall (C=10, γ=1)

5.3.4 Decision Tree Classifier

When using Decision Trees, the goal is to create a model predicting the value of a target variable by learning simple decision rules inferred from the data features. Their main advantage is interpretability, but they might easily lead to

overfitting because of a poor generalization.
The tuned parameters are: *criterion*, which states how objects are classified as belonging to a particular class; the maximum depth of the tree; the minimum number of samples required to split a node and per each leaf.
Best F1-score: 0.65 on randomly oversampled set. When having more data at their disposal, DTs are able to generalise better.
Best recall: 0.89 on randomly undersampled set.

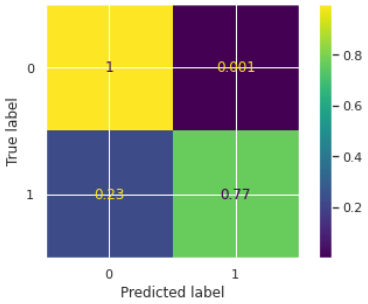


Figure 23: Confusion Matrix: DT best F1-score

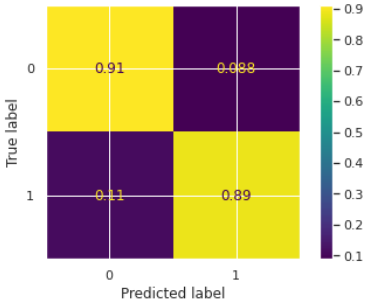


Figure 24: Confusion Matrix: DT best recall

5.3.5 Random Forest Classifier

A Random Forest Classifier can be thought of as an ensemble of different Decision Trees, fit on various subsamples of the dataset, whose decisions are averaged in order to better generalise the model and therefore prevent overfitting.
The main parameter is the number of estimators, which is to say the number of decision trees populating the forest.
When looking at the results, the first thing we can notice is that, in this case, the difference between models trained on undersampled and oversampled sets is evident: when testing the models trained on undersampled sets, the maximum obtained F1-score is 0.12; in the other case, instead, the minimum F1-score is equal to 0.8. So when looking at more data, the model is able to generalise much better.
As for recalls, we reach high results on all sets: model trained on undersampled ones are able to reach better recalls due to less generalization, from which the minority class benefits.

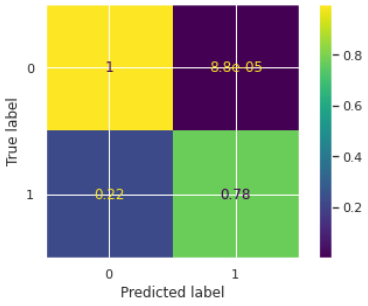


Figure 25: Confusion Matrix: RF best F1-score

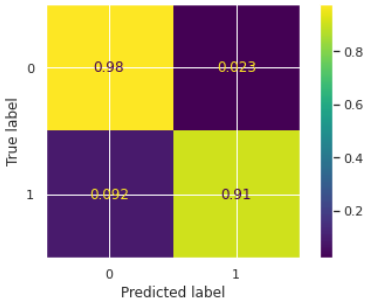


Figure 26: Confusion Matrix: RF best recall

5.4 Results Analysis

In this section, results obtained on different models and sampling techniques are compared. In particular, we can look at Figures 27 and 28 for testing results, evaluated according to the F1-score and recall metrics, respectively. The histograms in Figures 29 and 30 focus on the performance with respect to the F1-score: in the first one, scorings obtained by the same model when trained on differently sampled sets are compared; in the second one, instead, we want to highlight how each sampling technique favors or not a specific classification model. Figures 31 and 32 follow the same principles, but recall is the reference metrics.
Moreover, Table 2 displays a comparison of the different training times of the classifiers, where $TT_{sampling\_technique}$ stands for the training time (in seconds) of the model trained on the set sampled with *i-th* specified technique: KNN is the fastest model with an average of 2.08 seconds, while the slowest model is SVC with trainings lasting 56 minutes on average.

Clf	TT_1	TT_2	TT_3	TT_4	TT_5	TT_6	Avg
Logistic Regression	0.1	55.42	0.11	49.86	4.13	57.20	27.80
KNN	0.01	4.10	0.01	4.05	0.22	4.09	2.08
SVC	0.01	6215.56	0.01	11985.60	17.85	1962.60	3363.61
Decision Tree	0.01	12.01	0.02	13.35	0.96	26.47	8.80
Random Forest	5.22	158.71	5.23	176.16	12.32	7097.41	1242.51

Table 2: Training Time Perfomances
1 = manual undersampling, 2=random undersampling, 3=manual oversampling, 4=random oversampling, 5=oversampling and undersampling, 6=SMOTE

Finally, we can perform one general comparison and find the best combination of the sampling technique and the classifier that reaches the highest

performances on the test set:

- with respect to **F1 score metrics**, the most performing classifier is **Random Forest** and, according to averaged results, the best sampling technique is **manual oversampling**
- with respect to **recall metrics**, the most performing classifier is **Logistic Regression** and, according to averaged results, the best sampling technique is **random under-sampling**.

F1 scores

	MyUnderSampler	MyOverSampler	RandomUnderSampler	RandomOverSampler	UnderAndOverSampler	SMOTE
LogisticRegression	0.09	0.12	0.1	0.12	0.13	0.12
KNN	0.15	0.82	0.32	0.61	0.62	0.8
Support Vector Classifier	0.14	0.16	0.21	0.16	0.15	0.19
DecisionTreeClassifier	0.14	0.64	0.03	0.65	0.43	0.45
RandomForestClassifier	0.12	0.85	0.12	0.84	0.81	0.8

Figure 27: F1 scores obtained on all the models trained on all sampled training sets

Recall scores

	MyUnderSampler	MyOverSampler	RandomUnderSampler	RandomOverSampler	UnderAndOverSampler	SMOTE
LogisticRegression	0.9	0.89	0.91	0.89	0.9	0.88
KNN	0.85	0.76	0.84	0.8	0.81	0.81
Support Vector Classifier	0.88	0.9	0.88	0.9	0.9	0.88
DecisionTreeClassifier	0.84	0.78	0.89	0.77	0.81	0.81
RandomForestClassifier	0.9	0.78	0.91	0.77	0.81	0.79

Figure 28: Recall scores obtained on all the models trained on all sampled training sets

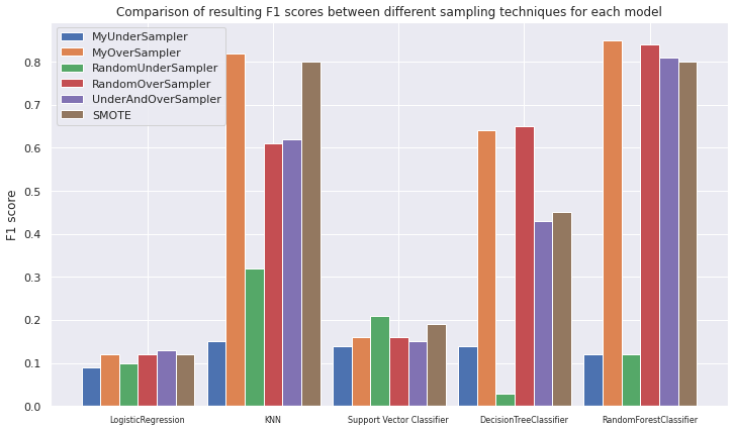


Figure 29: Comparison of resulting F1-scores between different sampling techniques for each model

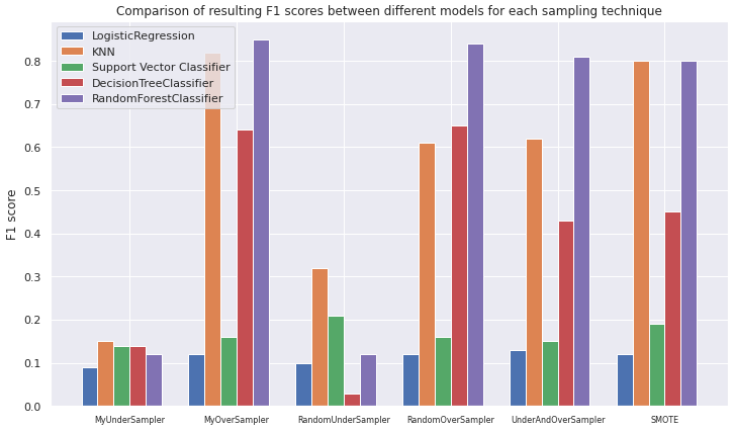


Figure 30: Comparison of resulting F1 scores between different models for each sampling technique

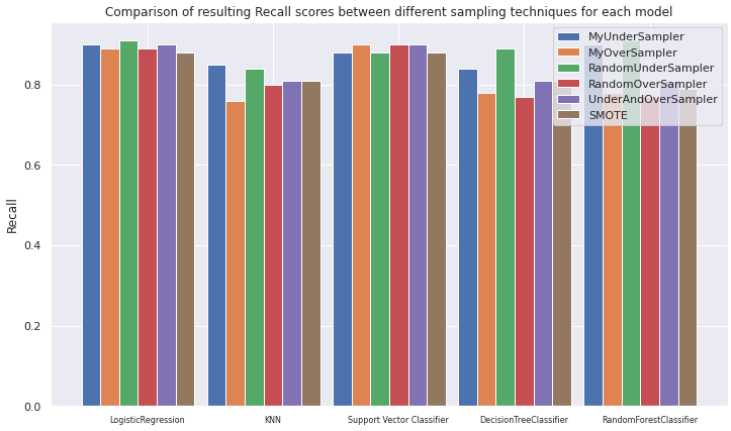


Figure 31: Comparison of resulting Recall scores between different sampling techniques for each model

Figure 32: Comparison of resulting Recall scores between different models for each sampling technique

References

[1] Credit card fraud detection dataset. https://www.kaggle.com/mlg-ulb/creditcardfraud.

[2] A machine learning approach for detection of fraud based on svm. https://www.researchgate.net/publication/236230678_A_Machine_Learning_Approach_for_Detection_of_Fraud_based_on_SVM, 2012.

[3] Fifth report on card fraud. https://www.ecb.europa.eu/pub/cardfraud/html/ecb.cardfraudreport201809.en.html, September 2018.

[4] Credit card fraud detection. https://medium.com/@vitorgabo/credit-card-fraud-detection-34935dda1a4d, 2019.

[5] Word payments report 2019. https://worldpaymentsreport.com/non-cash-payments-volume/, 2019.

[6] Credit card fraud statistics. https://shiftprocessing.com/credit-card-fraud-statistics/, 2020.

[7] Credit card statistics. https://shiftprocessing.com/credit-card/, 2020.

[8] Credit card statistics for 2020. https://blog.spendesk.com/en/credit-card-statistics-2020, 2020.

[9] Experian state of credit report. https://www.experian.com/blogs/ask-experian/consumer-credit-review/, 2020.

[10] Y. M. Haibo He. *Imbalanced Learning: Foundations, Algorithms, and Applications*. 2013.