Andrea Loizidou

Evaluating the best techniques to deal with an Imbalanced dataset for a credit card fraud detection dataset

Due to immense transaction volumes and continually changing fraudulent activities, credit card fraud detection has become a critical challenge in the digital economy. In this project, we discuss various strategies that may be implemented to deal with the massive imbalance in a credit card fraud dataset obtained from Kaggle, where fraudulent transactions compose merely 0.17% of the dataset. This project implements and compares various techniques, such as random oversampling, random undersampling, SMOTE, and the combination of SMOTE with Tomek Links, and leverages class weight adjustments and hyperparameter tuning to arrive at the best methods for improving model performance. The baseline model is a Random Forest Classifier, while the main metrics guiding the evaluation include Recall, Precision, and F1 Score. Results show a trade-off between catching fraudulent transactions at high Recall versus minimizing false positives at high Precision, indicating how model selection needs to be aligned with business priorities. The results also underscore how balancing and improving the quality of the training data pays dividends in dealing with class imbalance for fraud detection tasks.

1. Introduction:

In today's technologically advanced era, digital transactions and the use of credit cards have grown immensely, and so has credit card fraud. Since fraudsters constantly change how they operate, timely detection of fraudulent transactions has become a significant challenge for financial institutions. So far, Machine learning (ML) models offer promising solutions. However, there is a considerable challenge: if machine learning models are not trained precisely and correctly, they can fail to identify fraudulent transactions. In a fraudulent dataset, most transactions are legitimate, while only a few are fraudulent. This makes it hard to identify fraudulent cases using ML unless we cautiously handle the imbalanced dataset.

In imbalanced datasets, most machine learning models focus on the majority class, hence producing poor results for the minority class. Therefore, we need methods to help models perform well in both classes to solve this. This paper discusses various techniques for handling imbalanced data and tests their performance in detecting credit card fraud.

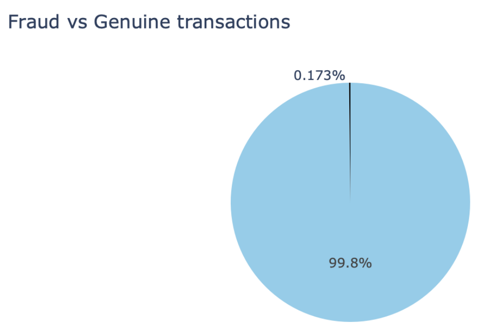
1.1 Idea development

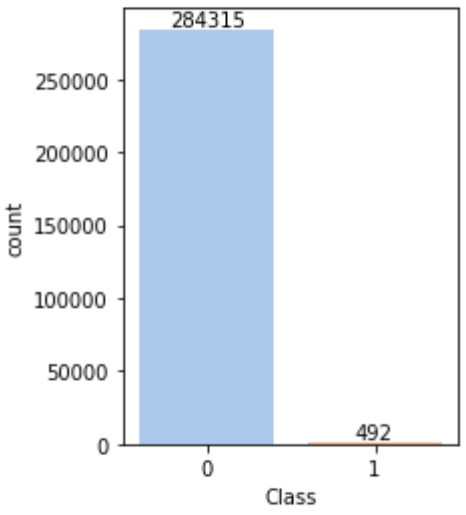
The original idea was to create synthetic datasets for the project using the Synthetic Data Vault (SDV). The objective was to assess how well three distinct models performed on a real dataset compared to an artificial one. Additionally, the main goal was to manually artificial outliers to assess the durability of these models. The prospect of employing SDV was intriguing since it provided a chance to study the creation of synthetic data and investigate its usefulness.

However, as soon as the dataset was evaluated, it became clear that it was incredibly unbalanced. This disparity caused the project's focus to change. Class imbalance emerged as the main obstacle, and considerable effort was made to comprehend and implement the most effective strategies to deal with this problem. Reading about various approaches and contrasting their efficacy motivated the decision to focus the project on assessing strategies to handle imbalanced datasets effectively.

2. Dataset and Preprocessing

The dataset used for this project was the "credit card fraud" dataset taken from Kaggle. The dataset contains transactions made by European cardholders within a two-day period in September 2013. It consists of 284,807 [99.8%] transactions, out of which 492 [0.173%] are fraudulent.





2.1 Data Features:

The dataset includes both numerical and transformed features:

- **PCA Transformed Features**: Features V1 through V28 were generated using Principal Component Analysis (PCA) for dimensionality reduction. The original feature names and meanings are not disclosed due to confidentiality concerns.
- **Time**: Represents the seconds elapsed between each transaction and the first transaction in the dataset.
- **Amount**: Indicates the transaction amount, which can be useful for cost-sensitive learning approaches.
- **Class**: The target variable, where 1 represents fraud and 0 represents non-fraud.

2.2 Characteristics:

- **Transaction Amount**: The mean transaction amount is approximately USD 88, with relatively small values overall.
- **Class Distribution**: The dataset is highly imbalanced, with 99.83% of transactions classified as non-fraud and only 0.17% as fraud.
- **Null Values**: The dataset has no missing values, simplifying preprocessing steps.
- **Duplicate Values:** 1001 duplicate values were found and removed.

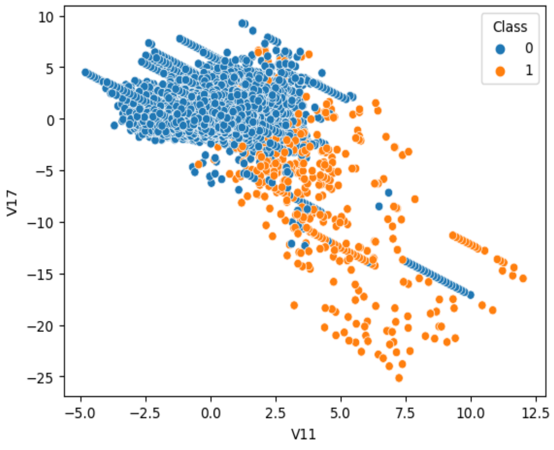
2.3 Transformations and Assumptions

- **PCA Transformation**: All other columns were transformed using PCA except for the 'Time' and 'Amount' features. This ensures reduced dimensionality while preserving essential patterns.
- **Scaling**: Features used in PCA were scaled prior to transformation. It is assumed that this scaling was applied correctly by the dataset creators applied this scaling correctly.

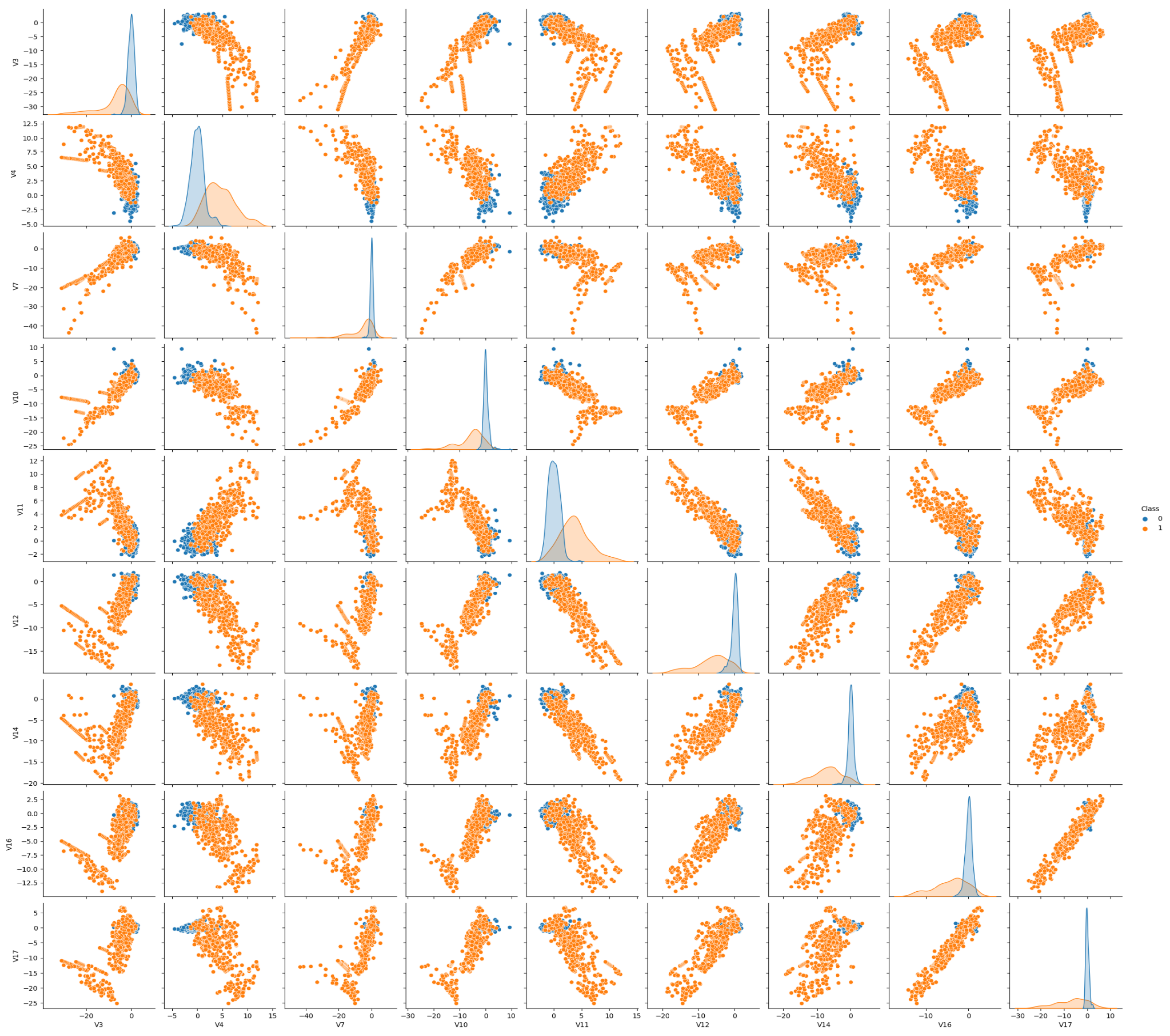
3. Outliers

Outliers are data points that stand out from the rest of the data because they differ significantly from the variable's average. Since fraudulent transactions frequently fall into this category, it is essential to comprehend and manage outliers in fraud detection. These points are difficult to identify and necessary for creating reliable models because they do not fit into conventional classifications.

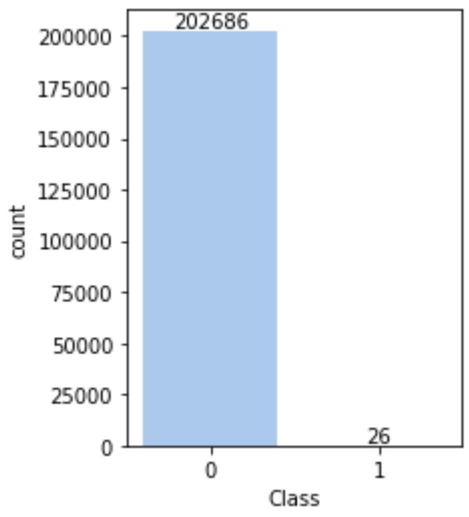
We started by creating a scatterplot of the transactions for one of the variables (V11) in order to look at outliers in the dataset. An intriguing pattern emerged from the visualization: a compact cluster of blue dots, which stood for typical transactions, with very few outliers. On the other hand, Orange dots, which signify fraudulent transactions, were dispersed but did not form a clear cluster. The inability to cluster fraudulent transactions brought to light how challenging it is to identify outliers in these situations. After making this finding, we looked into this further to determine if the pattern persisted across additional factors.



We proceeded to create scatterplots for all variables in the dataset. Regular transactions consistently showed up as discrete clusters across all variables, but fraudulent transactions were dispersed and lacked an identifiable grouping. This demonstrated that fraudulent operations are naturally varied and defy conventional distributions or patterns. Their dispersed character highlights the difficulty of spotting fraudulent transactions as anomalies.



We applied the Interquartile Range (IQR) method to address the outliers in the dataset. The IQR method is very suitable for skewed or non-bell-shaped data as it makes no distributional assumptions. The IQR is calculated as the difference between the third quartile (Q3) and the first quartile (Q1), providing a measure of statistical dispersion. Any data points falling outside the range of [Q1−1.5]×IQR and [Q3+1.5]×IQR are classified as outliers. This approach helps to identify and manage outliers without relying on subjective judgment or predefined thresholds. However, after using this technique, we quickly realized that most outliers removed were from the fraudulent transaction class. Thus, eliminating outliers using the IQR would have resulted in only 26 fraud cases left in the dataset, which is not viable for building effective ML models. Therefore not deleting the outliers was a better solution.



4. Train-test splitting

This study used stratified splitting to divide the dataset into training and testing sets. In both the training and testing datasets, stratified splitting guarantees that the distribution of the target variable (Class), which includes fraudulent and non-fraudulent transactions, stays constant. This guarantees that the minority class (fraudulent transactions) is fairly represented in both sets, which is especially crucial for unbalanced datasets. The splitting process was carried out before using any oversampling or undersampling techniques.

This sequence is critical because oversampling techniques, such as SMOTE, artificially create new data points for the minority class. If these synthetic points are introduced before splitting the data, the same synthetic samples could appear in both the training and testing sets, leading to data leakage. Data leakage artificially inflates the performance metrics of machine learning models, as the model is essentially being tested on data it has already seen. Similarly, undersampling reduces the majority class, which could skew the original class distribution if performed prior to splitting. By splitting the dataset first and then applying these techniques only to the training set, we preserve the integrity of the evaluation process and ensure that the test set remains a realistic representation of unseen data. This practice is essential for obtaining reliable and unbiased model performance metrics.

5. Techniques:

Cross Validation

The first thing done was to use the cross-validation resampling procedure, which is used to evaluate ML models. The main idea is that the model is to test the ability of the ML model to predict new data. If we use simple cross-validation in cases where datasets are unbalanced, we might produce subsamples with varying class distributions. This means that some unbalanced samples may produce exceptionally high scores, leading to scores that have exceptionally high cross-validation scores, which are undesirable. Therefore, we must create stratified subsamples that preserve the overall class frequencies. This stratification prevents the creation of folds with disproportionately few or no instances of the minority class, which could skew the results and provide an inaccurate picture of the model's performance. Since, in this project, the plan was to undersample/oversample the data, we had to first cross-validate to ensure that we avoided the "data leakage' problem. Data leakage is a common mistake where information from the validation set influences the training process, leading to inflated performance metrics. This stratification prevents the creation of folds with disproportionately few or no instances of the minority class, which could skew the results and provide an inaccurate picture of the model's performance.

This was implemented using the StratifiedKFold class from sklearn.model_selection, which guaranteed the same splits for runs. As an example model, a Random Forest Classifier was used, and the stratified folds were passed for cross-validation. The model's performance was evaluated with metrics such as Recall to reflect its capability to detect fraudulent transactions. This was a comprehensive approach to getting reliable and unbiased insights from our baseline model into the challenges posed by our dataset.

5.1 Why Random Forest Classifier?

The Random Forest Classifier was chosen as one of the models for this project because of its versatility, robustness, and effectiveness in handling imbalanced datasets. Random Forest is an ensemble learning method that builds multiple decision trees during training and then combines their outputs to improve predictive performance and reduce overfitting. It is particularly well-suited for binary classification tasks like fraud detection because it naturally handles complex, nonlinear relationships between features and the target variable. Moreover, another advantage of using random forests is the fact that it works well with datasets that have a mix of scales and upscaled features, as it does not require extensive preprocessing. This was beneficial in our dataset, where features like 'Time' and 'Amount' were not PCA-transformed. Moreover, Random Forest includes an inherent mechanism to address imbalanced datasets by adjusting class weights during training, ensuring the minority class is appropriately prioritized.

5.2 The most critical Metrics for Imbalance datasets:

In the case of imbalanced datasets, the standard classification accuracy is often misleading. For instance, if 99% of the data is of one class, then predicting that class every time would give 99% accuracy without actually solving the problem for the minority class effectively. Instead, other metrics are more appropriate, such as Precision, Recall, F1 Score, AUC-ROC, and

Average Precision. Precision tells how many predicted positives are correct, while Recall measures how many actual positives were correctly identified. The F1 Score balances Precision and Recall, making it useful when the positive class is critical. From a broader point of view, AUC-ROC is very useful because it measures the ability to distinguish between classes.

5.3 Recall VS Precision

For this project, Recall (**The ability of a model to find all the relevant cases within a data set**.) is a more important evaluation metric than Precision. That is because, in the case of fraudulent transactions, we want to minimize or avoid false negatives as much as possible. Fraud transactions cost a lot, and we want to take appropriate measures to prevent them as much as possible. For instance, a false negative means that a fraud transaction is not flagged as fraud. Instead, it is assessed to be genuine, which is detrimental.
Precision is more important than Recall when you would like to have less False Positives in trade off to have more False Negatives. Meaning, getting a False Positive is very costly, and a False Negative is less.

5.4 Hyperparameter Tuning Using GridSearchCV

Hyperparameter tuning is the process of manually defining the parameters while we build ML models. GridSearchCV is one of the most famous techniques that could be used to systematically explore a set of predefined hyperparameters for the best performance. It basically works by creating a grid of all possible parameter combinations, training the model for each of the combinations, and then evaluating it using cross-validation. The result is the "best" set of hyperparameters from the given grid, which may not guarantee global optimality but at least guarantees the best within the defined search space. Once the best parameters have been identified, they are used to make predictions, thus effectively improving model performance.

The best-performing combination of hyperparameters, as identified by cross-validation, is {'max_depth': 12, 'n_estimators': 100, 'random_state': 13}, yielding a recall score of approximately 77.3% on the training set before oversampling.

5.5 Baseline Model:

Confusion Matrix:

```
[84972     4]
[   33   109]
```

5.5.1 Results:

The confusion matrix shows the model's predictions: it correctly classified 109 instances of the minority class while misclassifying 33 as negatives. Metrics are calculated to summarize performance: **Recall** (77.6%) measures the model's ability to correctly identify positive cases, while **Precision** (96.5%) reflects the proportion of true positives among predicted positives. The **F1 Score** (85.5%) balances Precision and Recall, and **Accuracy** (99.96%) shows overall correctness but is less meaningful given the dataset's imbalance. These results set a baseline for further improvement, such as using oversampling techniques to address class imbalance.

5.6 Random Oversampling

After we have created the baseline model, we continued by evaluating random oversampling, which is a technique used to balance the dataset by increasing the representation of the minority class [fraud] in the training dataset. Afte this technique was applied, we ended up having 50% of the dataset as non-fraudulent and 50% as fraudulent

While this method balances the training data and can help improve model performance, it can result in overfitting because the model may "memorize" duplicated examples rather than generalize patterns. For example, if each minority data point is duplicated multiple times, and cross-validation is used, folds may contain identical examples, making it easier for the model to perform well during training but not on unseen data. It is important that oversampling is done only on the training dataset and never on the test or holdout set to ensure model evaluation is unbiased.

5.6.1 Building a pipeline

A pipeline is a means of chining together multiple steps in a single process. n this case, the pipeline consists of RandomOverSampler thatbalances the dataset by duplicating examples from the minority classand a Random Forest Classifier to train the model. The use of a pipeline allows for oversampling only the training segments when doing cross-validation, thus leaving the test and validation sets intact. This willindeed ensure the evaluation is unbiased, and the model performance is not artificially high. The reason why we used the pipeline is because we can automate this process in a smart and efficient way. Note: imblearn package was used.

The cross-validation (cross_val_score) evaluates the pipeline, splitting the dataset into training and validation segments. For each fold:

1. The minority class in the training segment is oversampled to balance the data.
2. The Random Forest model is trained on the oversampled training data.
3. The model is then tested on the untouched validation segment.

Recall for every fold is calculated as a measure of the model's ability to correctly identify positive cases. The average Recall across all folds is about 76.4%, which gives a good idea about how well the model detects instances of the minority class while doing cross-validation.

For the final improvement on the model, GridSearchCV is used within the pipeline. GridSearchCV systematically tries out different hyperparameters such as the number of trees n_estimators and tree depth max_depth to find the combination that best optimizes the Recallmetric. The best parameters are max_depth=4 and n_estimators=50, the Recall score is about 87.0% which shows an improvement over the initial average.

5.6.2 Evaluating the Model:
The optimized pipeline is then tested on the test set. The predictions are evaluated using a confusion matrix:

Confusion Matrix:

```
 [84593   383]
[   23   119]
```

- **True Negatives (84,593):** Correctly predicted majority class.
- **False Positives (383**): Majority class instances misclassified as minority.
- **False Negatives (23):** Minority class instances misclassified as majority.
- **True Positives (119):** Correctly predicted minority class.
- From this, key metrics are calculated:

- **Recall (83.8%):** A strong performance in identifying minority class instances.
- **Precision: 23.7%** - relatively low because of the too-high number of false positives.
- **F1 Score: 37.1%** - Optimal between Precision and Recall.
- **Accuracy: 99.5%** - Very high correct, yet meaningless for imbalanced datasets.

5.7 Random Undersampling
Random undersampling is a technique for handling class imbalance by reducing the number of instances of the majority class in the training dataset. It involves the random selection and removal of instances from the majority class in order to create a balanced dataset. This can be effective when the minority class has sufficient data to build a reliable model, as it simplifies the dataset and prevents the majority class from dominating model training.

The above example was balanced using RandomUnderSampler from the imblearn library. The effect of this transformation is the reduction of the number of examples from "Genuine" to 331 (majority class) and "Frauds" to 331 (minority class), which both account for 50% of the new dataset. This will significantly reduce the overall size of the dataset to a mere 662 total records. This small dataset size may reduce the model's learning ability, pointing to the fact that while random under-sampling can balance classes, it may not be suitable in all cases, especially for those with a limited dataset size.

5.8 SMOTE (Synthetic Minority Oversampling Technique)¶

SMOTE is a technique aimed at handling class imbalance by over-sampling the minority class with synthetic examples. Unlike random oversampling, which simply replicates existing examples of the minority class, MOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space, and drawing a new sample at a point along that line this technique increases the diversity of the minority class, helping the model generalize better.

Like before, the pipeline was created using SMOTE and a Random Forest Classifier for model training. During cross-validation, smote oversamples the minority class in the training set, making sure the class distribution is balanced in each fold. Cross-validation results show an average of 81.3% recall scores for each fold. Moreover, GridSearchCV was used for further optimization. The best combination of parameters (max_depth=6, n_estimators=50) achieved a Recall score of **87.3%** during cross-validation, showing an improvement from the initial average.

5.8.1 Evaluating the Model:

Then, the optimized pipeline was used on the test dataset, and the results of the prediction were evaluated against a confusion matrix:

```
[84621   355]
[   21   121]
```

- **True Negatives**: 84,621 (correctly identified as the majority class)
- **False Positives:** 355 (majority class misclassified as minority)
- **False Negatives**: 21 (minority class misclassified as a majority)
- **True Positives:** 121 (correctly identified as minority class)

- **Recall 85.2%:** A great ability to identify the minority class examples correctly.
- **Precision (25.4%):** Indicates room for improvement, as many false positives were predicted.
- **F1 Score (39.2%):** Balances Recall and Precision.
- **Accuracy (99.6%):** Reflects overall correctness but is less meaningful for imbalanced datasets.

The combination of SMOTE and GridSearchCV resulted in a model with high Recall, proving its effectiveness in handling class imbalance. However, the relatively low Precision underlines a trade-off: the model predicts more false positives while prioritizing the identification of minority class examples.

5.9 Combining SMOTE and Tomek Links

5.9.1 Tomek Links

Tomek Links is an undersampling technique proposed by Ivan Tomek in 1976. It selects samples between the minority and majority classes by removing the majority class samples that are closer to the minority class samples. A Tomek Link exists between two samples if they are each other's nearest neighbors and belong to different classes. By removing the majority of class samples involved in these links, the method clears the boundary between the two classes and reduces the overlap and ambiguity.

As mentioned above, SMOTE generates synthetic samples based on the minority class. Then, the Tomek Links technique is used to remove noisy and overlapping majority-class samples. This combination was expected to enhance and balance the dataset, improving the model's training.

Just like before, a pipeline is created with SMOTETomek and a Random Forest classifier. The pipeline is evaluated using cross-validation, where the data is split into training and validation. It is also evaluated using GridSearchCV to find the best hyperparameters for the Random Forest Classifier. Due to the computational demand, the process was completed on Jupyter notebooks.

5.9.2 Evaluating the Model

Confusion Matrix:

```
[84967     9]
[   34   108]
```

- **True Negatives (84,967)**: Correctly identified majority class.
- **False Positives (9)**: Majority class instances misclassified as minority.
- **False Negatives (34)**: Minority class instances misclassified as majority.
- **True Positives (108)**: Correctly identified minority class.

- **Recall (76.1%)**: The model effectively identifies most minority class instances.
- **Precision (92.3%)**: Indicates a high proportion of predicted positives are correct, minimizing false positives.
- **F1 Score (83.4%)**: Balances Recall and Precision, showing overall effectiveness.
- **Accuracy (99.9%)**: High correctness but less indicative for imbalanced datasets.

The SMOTE and Tomek Links in SMOTETomek make a nice ensemble to improve the training data's balance and quality. The model attains good Recall and Precision by the use of oversampling on the minority class and removing the noisy majority class examples. The trade-off between detecting minority instances (Recall) and minimizing false positives (Precision) is well-managed, with a balanced F1 Score of 83.4%. This strategy, even though computationally expensive, gave strong, consistent, and trustworthy performances on imbalanced datasets.

6 Adjusting Class Weights in Models

Most of the machine learning models, including Random Forest Classifiers, have a parameter class_weight. Using this parameter, you can attach a higher weight to the minority class, thus adjusting how the model will learn during training. If no weights are used, all the points in the data have an equal say in the model. On the other hand, if weights are attached, the model scales the error for each point according to the weight. Heavier weights amplify the error signal so that the model minimizes mistakes in the minority class.

In this example, a Random Forest Classifier is trained with class_weight="balanced ."This setting automatically assigns weights to classes inversely proportional to their frequency in the dataset. Cross-validation is used to evaluate the model, yielding Recall scores across different folds. The average Recall score is approximately 74.9%, indicating that the model successfully identifies a majority of the minority class instances during validation.

6.1 Evaluating the Model

The model is tested on the test dataset, and its predictions are analyzed using a confusion matrix:

Confusion Matrix:

```
[84721   255]
[   25   117]
```

- **True Negatives (84,721)**: Correctly identified majority class.
- **False Positives (255)**: Majority class instances misclassified as minority.
- **False Negatives (25)**: Minority class instances misclassified as majority.
- **True Positives (117)**: Correctly identified minority class.

- **Recall (82.4%)**: The model identifies most of the minority class instances.
- **Precision (31.5%)**: Lower Precision indicates a higher number of false positives.
- **F1 Score (45.5%)**: Balances Recall and Precision.
- **Accuracy (99.7%)**: Reflects overall correctness but is less meaningful for imbalanced datasets.

This technique allows class_weight to work well by giving more importance to the minority class while training, resulting in higher Recall. On the other hand, the model did this at the cost of a lower Precision by misclassifying some instances from the majority class as the minority class. Despite these trade-offs, class weights have often provided a simple and effective solution to deal with imbalanced data.

7. Performance Comparisons

Random Forest with	Recall	Precision	F1 Score	Accuracy
SMOTE Oversampling	0.852113	0.254202	0.391586	0.995583
Random Oversampling	0.838028	0.237052	0.369565	0.99523
Class weights	0.823944	0.314516	0.455253	0.99671
No Under/Oversampling	0.767606	0.964602	0.854902	0.999565
SMOTE + Tomek	0.760563	0.923077	0.833977	0.999495

The results demonstrate a trade-off between **Recall** and **Precision** across different models. The **SMOTE Oversampling** model achieved the highest **Recall (85.2%)**, indicating its strength in identifying minority class instances, but at a significant cost to **Precision (25.4%)** and **F1 Score (39.2%)**, making it less suitable for scenarios where false positives are costly. The baseline model, **No Under/Oversampling**, achieved a more balanced performance, with the highest **Precision (96.5%)** and **F1 Score (85.5%)**, but a lower **Recall (76.8%)**, reflecting a conservative approach. The **Class Weights** model provided a moderate **Recall (82.4%)** and improved **F1 Score (45.5%)**, offering a compromise. Interestingly, the advanced **SMOTE + Tomek** model did not excel in **Recall (76.1%),** performing similarly to the baseline. Ultimately, the best model depends on business priorities—whether it is more critical to minimize false negatives (favoring higher Recall) or false positives (favoring higher Precision).

Conclusion:
In this project, we examined some ways to deal with the difficulties of managing unbalanced datasets in the context of credit card fraud detection. We illustrated the trade-offs associated with model performance by using and comparing a range of methods, such as class weight modifications, hyperparameter tuning, SMOTE, random oversampling, random undersampling, and the SMOTE-Tomek combination. The findings demonstrated that although techniques such as SMOTE can greatly increase recall, they frequently do so at the expense of precision, which raises the rate of false positives. However, the SMOTE-Tomek combo and class weight changes produced a more balanced performance, maximizing both precision and recall. Whether reducing false negatives or false positives is more important, the strategy chosen ultimately depends on the main goals of the business.