Credit card fraud detection project

Problem Statement

The problem statement chosen for this project is to predict fraudulent credit card transactions with the help of machine learning models.

In this project, you will analyse customer-level data that has been collected and analysed during a research collaboration of Worldline and the Machine Learning Group.

The data set is taken from the Kaggle website and has a total of 2,84,807 transactions; out of these, 492 are fraudulent. Since the data set is highly imbalanced, it needs to be handled before model building.

Business problem overview

For many banks, retaining high profitable customers is the number one business goal. Banking fraud, however, poses a significant threat to this goal for different banks. In terms of substantial financial losses, trust and credibility, this is a concerning issue to both banks and customers alike.

It has been estimated by Nilson Report that by 2020, banking frauds would account for $30 billion worldwide. With the rise in digital payment channels, the number of fraudulent transactions is also increasing in new and different ways.

In the banking industry, credit card fraud detection using machine learning is not only a trend but a necessity for them to put proactive monitoring and fraud prevention mechanisms in place. Machine learning is helping these institutions to reduce time-consuming manual reviews, costly chargebacks and fees as well as denials of legitimate transactions.

The data set includes credit card transactions made by European cardholders over a period of two days in September 2013. Out of a total of 2,84,807 transactions, 492 were fraudulent. This data set is highly unbalanced, with the positive class (frauds) accounting for 0.172% of the total transactions. The data set has also been modified with principal component analysis (PCA) to maintain confidentiality. Apart from ‘time’ and ‘amount’, all the other features (V1, V2, V3, up to V28) are the principal components obtained using PCA. The feature 'time' contains the seconds elapsed between the first transaction in the data set and the subsequent transactions. The feature 'amount' is the transaction amount. The feature 'class' represents class labelling, and it takes the value of 1 in cases of fraud and 0 in others.

EDA:

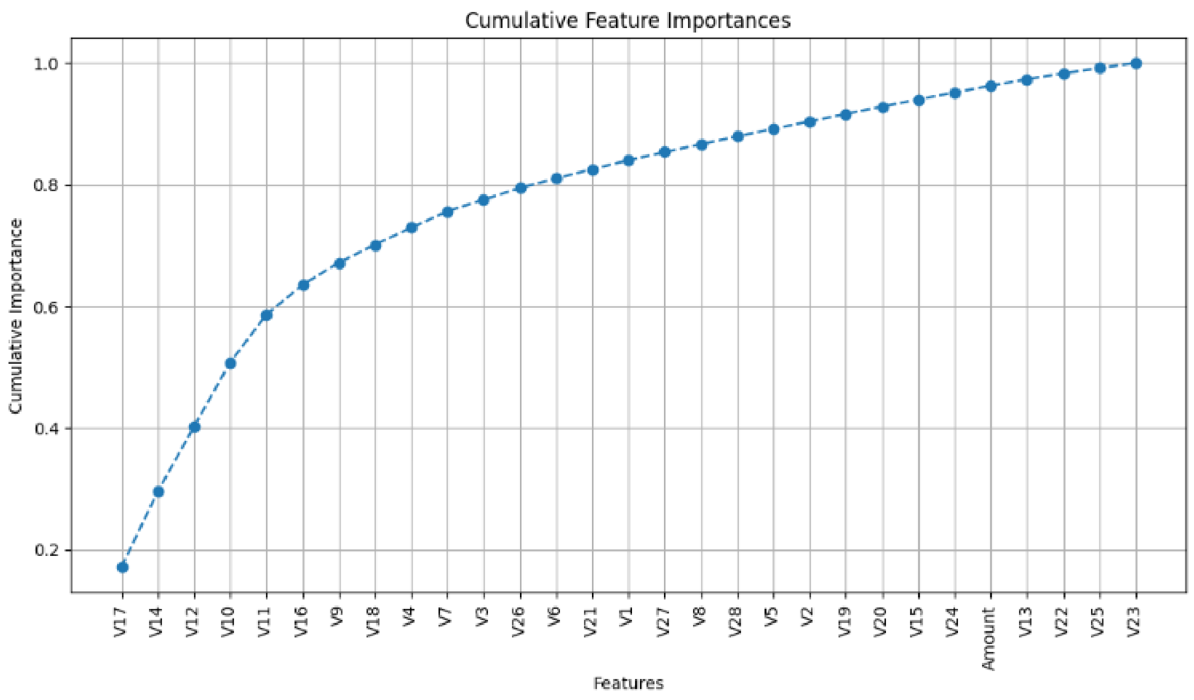
Since the data is highly imbalanced, a few issues can arise if it's not handled properly. When dividing the data into training and testing sets, we must ensure that the model doesn't achieve high accuracy by just predicting the majority class. This isn't just a classification problem; it's also anomaly detection. To address this, use a stratified split based on classes to ensure that both the training and test sets have the same representation of the data. With the training data, employ k-fold cross-validation to get a better idea of which model to choose, and to use all data points for training and validation during model building.

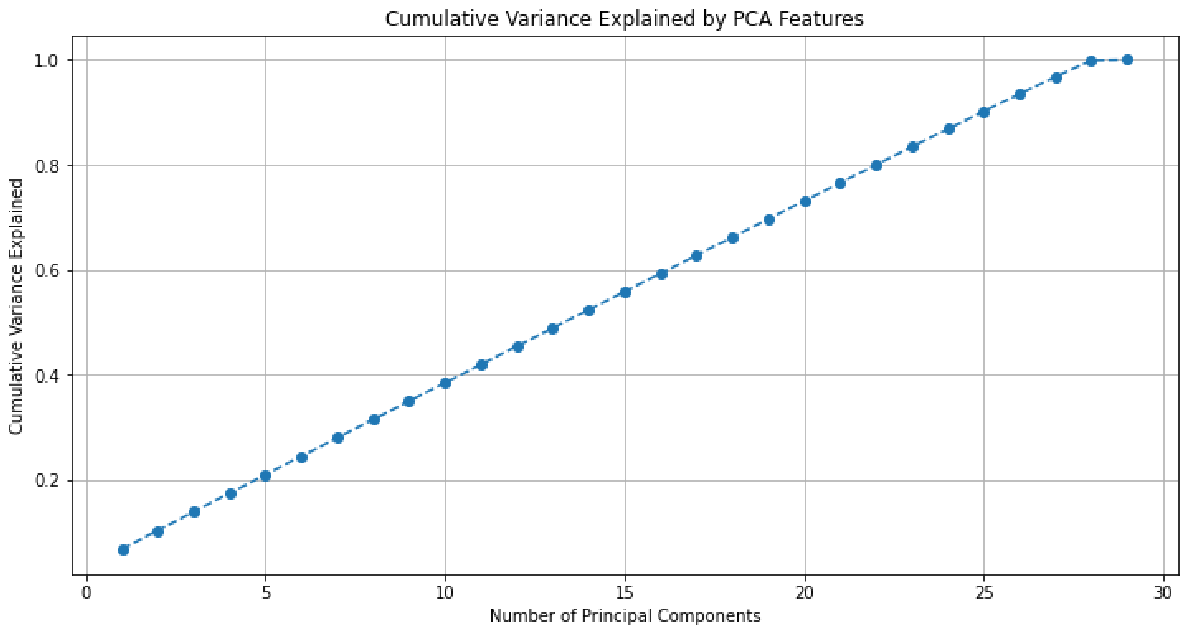
1. **Choosing the value for k in k-fold cross-validation**: Several factors need consideration.

2. Higher k values increase computational cost and reduce the number of fraudulent cases the model can train and validate on effectively.
3. With k=5, each fold has about 56,961 transactions (1/5th of the dataset), providing a good mix of training and validation samples, even for the fraudulent cases. Each fold will contain approximately 98 positive cases (56,961 / 284,807 * 492) and a proportionate number of non-fraudulent cases, ensuring each fold represents the dataset well.
4. 5-fold cross-validation reduces the computational burden compared to using a larger k (like 10), which can be particularly beneficial if working with limited computational resources.

There are 29 features in the dataset. We can use feature selection in a random forest to calculate the importance of each feature and determine if some features can be dropped. After plotting the cumulative feature importance, we can see that the first 12 features account for 80% of the cumulative importance. However, since this is an anomaly detection problem, removing features based solely on this plot might not be the most effective approach. Here's why:

- **Imbalanced Data**: In imbalanced datasets, models tend to focus on the majority class and neglect the minority class. Feature importance scores can be biased towards the majority class, leading to the removal of features that might be crucial for identifying the minority class.
- **Model Performance**: The primary goal is to improve model performance on the minority class. Simply removing features based on cumulative importance might not guarantee better performance, especially if important features for the minority class are removed.





Let's say we want to capture 95% of the variance. From the plot, we can see that this is achieved with around 20 principal components. But the dataset is highly imbalanced so the nuances in identifying minority class might be missed, so we choose to not remove any features.

Algorithms and Considerations:

K-Nearest Neighbors (KNN):

Pros: Simple and intuitive.

Cons: May struggle with imbalance since it relies on majority voting within its neighbors. Computationally expensive with large datasets.

Recommendation: Not ideal for your scenario, but can be used with techniques like SMOTE.

Support Vector Machine (SVM):

Pros: Effective in high-dimensional spaces and robust with imbalanced data, especially when combined with class weights.

Cons: Can be slow with large datasets. Kernel SVMs can be computationally intensive.

Recommendation: Use with caution, opt for Linear SVM or leverage class weights.

Decision Tree:

Pros: Simple and interpretable.

Cons: Prone to overfitting and might need class weight balancing.

Recommendation: Can be a base model, but not the strongest candidate alone.

Random Forest:

Pros: Robust to overfitting, handles imbalance well by averaging.

Cons: Requires more resources but scales relatively well.

Recommendation: Good choice, especially with class_weight='balanced'.

XGBoost:

Pros: Handles imbalance using scale_pos_weight, highly effective with large datasets, and provides feature importance.

Cons: Computationally intensive but highly optimized.

Recommendation: Excellent choice for your data. Tune scale_pos_weight and other hyperparameters.

Logistic Regression:

Pros: Simple, interpretable, works well with large and sparse datasets.

Cons: May not perform well without balancing class weights.

Recommendation: Use class_weight='balanced' or resampling techniques.

Upon the above considerations we selected **XGBoost, random forrest, logistic regression.**

EVALUATION METRIC:

Evaluation metric plays an important role here as the dataset is highly imbalanced.

Why Threshold-Dependent Metrics Aren't Ideal for Imbalanced Datasets:

1. **Accuracy**:

Issue: Accuracy measures the proportion of correct predictions (both true positives and true negatives) out of all predictions. In an imbalanced dataset, where the majority class vastly outweighs the minority class, a model can achieve high accuracy by simply predicting the majority class most of the time.

Example: If 99.8% of transactions are non-fraudulent, a model that always predicts "non-fraudulent" will have an accuracy of 99.8%, but it won't correctly identify any fraudulent transactions.

2. **Precision**:

Issue: Precision measures the proportion of true positive predictions out of all positive predictions (true positives + false positives). While useful in cases where the cost of false positives is high, precision alone doesn't account for false negatives, which can be critical in fraud detection.

Example: A model might have high precision if it predicts only a few fraudulent transactions and gets most of them right. However, it might miss many actual fraud cases, making it unreliable for detecting fraud.

3. **Recall** (Sensitivity):

Issue: Recall measures the proportion of true positive predictions out of all actual positives (true positives + false negatives). High recall means detecting most fraudulent transactions, but it doesn’t reflect how many false positives (incorrectly predicted frauds) there are.

Example: A model with high recall might flag a large number of transactions as fraudulent, capturing most frauds but also creating many false alarms.

4. **F1 Score**:

Issue: The F1 score is the harmonic mean of precision and recall, providing a balance between the two. However, in highly imbalanced datasets, it might still not fully capture the model's performance, especially regarding false positives and true negatives.

Example: The F1 score balances precision and recall, but given the vastly different class sizes, it might not fully reflect the model's effectiveness in distinguishing between fraudulent and non-fraudulent transactions.

Best Evaluation Metric for Imbalanced Datasets:

AUC-ROC (Area Under the Receiver Operating Characteristic Curve):

- **Explanation**: AUC-ROC measures the model's ability to distinguish between positive (fraudulent) and negative (non-fraudulent) classes across all classification thresholds. It calculates the true positive rate (recall) against the false positive rate, providing a single value that summarizes performance.
- **Why It's Better**: A high AUC indicates that the model has a good measure of separability between classes. It is not dependent on a specific threshold, making it more robust for imbalanced datasets.

Now, let us build models using stratified k-fold cross-validation with different hyperparameters using GridSearchCV, with AUC-ROC as the evaluation metric. Once we find the best model with the highest AUC-ROC score among all combinations, we will use it to predict the test set.

Model name	Best hyper parameters	Best auc-roc score
RandomForest	{'class_weight': 'balanced', 'max_depth': 10, 'n_estimators': 200}	0.9766836379895016
XGBoost	{'max_depth': 7, 'n_estimators': 50, 'scale_pos_weight': 599.4708994708994}	0.9771974431334293
LogisticRegression	{'C': 0.1, 'class_weight': 'balanced'}	0.9754748865765472

The best model we've identified is XGBoost, with an AUC-ROC score of 0.977. When this model was used to predict the test set, it achieved an AUC-ROC score of 0.9746312073210387, which is comparable and quite promising.

To further enhance the model's performance, we'll balance the dataset using oversampling techniques such as SMOTE and ADASYN, along with random oversampling. Additionally, we'll fine-tune the hyperparameters based on the best individual model's parameters and make adjustments accordingly.

Using random oversampler:

Fitted the random oversampler on train set and after resampling there is a 50-50 distribution between the classes.

```
y_train_resampled.value_counts()
```

```
1    226600
0    226600
```

Now lets train the models and we have also added code to track the training time for each model.

Model name	Best hyper parameters	Best auc-roc score	Training Time
RandomForest	{'class_weight': 'balanced', 'max_depth': 20, 'n_estimators': 200}	1.0	3894.786039352417
XGBoost	{'max_depth': 7, 'n_estimators': 50}	nan	1350.3861269950867
LogisticRegression	{'C': 0.5}	0.9862525896069844	26.62443232536316

Inference:

1.RandomForest:

Best Hyperparameters: {'class_weight': 'balanced', 'max_depth': 20, 'n_estimators': 200}

Best AUC-ROC Score: 1

Training Time: 3894.786039 seconds

Inference: The RandomForest model appears to achieve a perfect AUC-ROC score of 1, indicating flawless distinction between fraudulent and non-fraudulent transactions. However, this result is unusual and might suggest overfitting or an issue with the validation process. The training time is the highest among the models, reflecting the complexity and robustness of the model.

2. **XGBoost**:

Best Hyperparameters: {'max_depth': 7, 'n_estimators': 50}

Best AUC-ROC Score: NaN

Training Time: 1350.386127 seconds

Inference: The NaN value for AUC-ROC score suggests there may have been an issue during the evaluation process, such as a lack of sufficient data points to calculate this metric reliably or a possible bug. Despite this, the model has a moderate training time.

3. **LogisticRegression**:

Best Hyperparameters: {'C': 0.5}

Best AUC-ROC Score: 0.98625259

Training Time: 26.62443233 seconds

Inference: Logistic Regression shows a very high AUC-ROC score of 0.98625259, indicating excellent performance in distinguishing between classes, though slightly lower than the RandomForest. Its training time is significantly shorter, making it a more efficient option for quick iterations and deployments.

The best model here is a random forrest with an auc-roc score of 1.0 (the highest possible). We got an auc-roc score of 0.979324383332559 when predicted test set on this model. This suggests that the model might have over fit and also the training time is the highest which might not be the best when considered all things.

Using SMOTE technique:

SMOTE is used to address the class imbalance problem by generating synthetic samples of the minority class. It creates synthetic samples by interpolating between existing minority class samples. This helps in making the decision boundaries for the minority class more generalizable.

Since the training time for logistic regression is shorter compared to other models we've tested and the evaluation score is also favorable, we'll proceed with building more models using this algorithm with diverse hyperparameters to further improve the score.

Model scores:

Logistic Regression Best AUC-ROC Score: 0.988628927791311
Logistic Regression Best Hyperparameters: {'C': 1.0, 'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}
Logistic Regression Training Time: 1774.9560732841492 seconds

Test AUC-ROC Score: 0.9802339387745622

Using ADASYN technique:

ADASYN is another oversampling technique aimed at addressing class imbalances, but it focuses more on adaptive learning. ADASYN generates synthetic samples similar to SMOTE but with a twist: it generates more synthetic samples for minority class instances that are harder to classify (closer to the decision boundary). This adaptive approach helps in focusing on the difficult-to-classify regions of the minority class, leading to a more balanced and potentially more robust classifier.

Model scores:

Logistic Regression Best AUC-ROC Score: 0.9590921710161291

Logistic Regression Best Hyperparameters: {'C': 1.0, 'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}
Logistic Regression Training Time: 2056.857583761215 seconds

Test AUC-ROC Score: 0.983358201328564

The **overview** of all the best models and their test auc-roc scores.

Oversampling technique	Model name	Best hyper parameters	Best Train auc-roc score	Best Test auc-roc score	Training Time
None	XGBoost	{'max_depth': 7, 'n_estimators': 50, 'scale_pos_weight': 599.4708994708994}	0.9771974431334293	0.9746312073210387	
Random oversampling	RandomForest	{'class_weight': 'balanced', 'max_depth': 20, 'n_estimators': 200}	1.0	0.979324383332559	3894.786039352417
SMOTE	LogisticRegression	{'C': 1.0, 'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}	0.988628927791311	0.9802339387745622	1774.9560732841492
Adasyn	LogisticRegression	{'C': 1.0, 'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}	0.9590921710161291	0.98335820132856 4	2056.857583761215

Conclusion:

The choice of the best model can vary based on the available computational resources and the optimal score obtained. If you have significant computational power, you might consider experimenting with ensemble models such as Random Forest or XGBoost. Additionally, the dataset's size and variety are crucial; more examples and diverse scenarios of fraudulent cases can enhance the model's performance over time.