Machine Learning in Credit Card Fraud Detection: Comprehensive Studies and Real-World Applications

Chong Chen
chen.chon@northeastern.edu

Di Liu
liu.di4@northeastern.edu

***Abstract*—In light of the rising incidence of electronic financial transactions, this paper delves into the application of machine learning (ML) techniques for credit card fraud detection, particularly focusing on the role of reinforcement learning (RL) in this domain. Addressing the challenge posed by imbalanced datasets—a prevalent issue in fraud detection—the study evaluates a suite of ML algorithms, including Logistic Regression, Support Vector Machines, K-Nearest Neighbors, Decision Trees, Random Forest, and Neural Networks, alongside resampling techniques like the Synthetic Minority Over-sampling Technique (SMOTE) and Tomek Links.**

By treating fraud detection as a sequential decision-making problem, we explore the effectiveness of RL agents subjected to distinct reward functions that influence their ability to discern fraudulent from legitimate transactions. An in-depth comparative analysis is conducted, highlighting key performance metrics such as accuracy, precision, recall, and F1-score, as well as introducing a novel savings metric to quantify the financial impact of fraud detection.

Our research illuminates the intricate trade-offs in designing reward functions and their consequential effects on the agent's classification behavior. The findings from this study contribute significantly to our understanding of the potential and challenges associated with the integration of RL into fraud detection frameworks, providing a pathway for developing robust anti-fraud strategies that reinforce the security of credit transactions and bolster consumer confidence in the financial ecosystem.

All source code is available at the following github links:

1) **ML in Credit Card Fraud Detection**
2) **RL in Credit Card Fraud Detection**

I. Introduction

A. Background

A credit card is a payment card, usually issued by a bank, allowing its users to purchase goods or services or withdraw cash on credit. Using the card thus accrues debt that has to be repaid later [1].

According to a publication in the Federal Register, in 2023 there will be nearly 4,000 card issuers in the United States, plus dozens of co-branded merchant partners and four major networks, providing bank cards to more than 190 million consumers.Credit cards offer cardholders the advantage of timing their purchases; credit cards allow consumers to build an ongoing debt balance, but accordingly charge interest in the process.

During this process, problems arise and credit card fraud occurs. Criminals can easily steal large amounts of money from credit card holders in a short period of time, without the cardholders' knowledge. The criminals disguise the fraudulent activities through consumer disguises, making each transaction look as reasonable and legal as possible, creating enormous work pressure for bank staff.

This is when it becomes especially critical for credit card companies to effectively detect credit cards based on various data, using computers and other advanced technologies. This can block the wrongdoing of criminals at the source, and at the same time effectively enhance the competitiveness of their own company's business.

B. Research Significance

As the volume of electronic financial transactions grows and more complex fraud schemes emerge, exploring effective methods for detecting credit card fraud becomes particularly important. This emerging research field faces a dual challenge: on one hand, ensuring the safety of credit transactions, and on the other hand, maintaining their convenience and ease of use.

The importance of credit card fraud detection research lies not only in its aim to mitigate financial losses for consumers and financial institutions due to fraud but also because it helps to broadly reduce the increase in service fees. Moreover, enhancing the efficiency of fraud detection is crucial for maintaining consumer trust in the financial system, effective protective measures encourage continued reliance on credit transactions. Finally, this research area also provides valuable insights for the wider field of cyber security, offering strategies and techniques for dealing with various digital and financial security threats.

Advances in data science, especially breakthroughs in machine learning and pattern recognition, bring unprecedented opportunities to optimize fraud detection mechanisms. These technologies have the potential to significantly improve the accuracy and efficiency of detection by analyzing transaction data to identify anomalies and fraud patterns. In this project, we are committed to leveraging these technological advancements to design scalable and efficient solutions to combat credit card fraud.

C. Expected Goals

This research is driven by the objective to significantly enhance the detection of fraudulent credit card transactions

through the development of a sophisticated machine learning model. The project is structured around several key aims:

1) **Model Accuracy**: To refine the precision of fraud detection models, minimizing the occurrence of false negatives (fraudulent transactions that are not detected) and false positives (legitimate transactions erroneously flagged as fraudulent). Balancing these outcomes is vital for the model's practical application.
2) **Feature Engineering**: To conduct an in-depth analysis and engineering of the dataset's features to identify the most predictive indicators of fraud. This includes assessing the significance of PCA-transformed features and the impact of the 'Time' and 'Amount' variables on fraud prediction.
3) **Algorithm Selection and Optimization**: To evaluate a range of machine learning algorithms, identifying the most effective methodology. Considerations will include the algorithm's performance with imbalanced datasets, computational efficiency, and the interpretability of its findings.
4) **Evaluation Strategy**: To establish a comprehensive evaluation framework that extends beyond conventional accuracy metrics. This will involve utilizing precision, recall, F1-score, and the Area Under the Receiver Operating Characteristic (AUROC) curve to thoroughly assess model performance.
5) **Real-world Application**: To verify the model's scalability and operational viability in real-world scenarios. This encompasses evaluating the model's enduring performance, adaptability to emerging fraud types, and integration potential within existing fraud detection infrastructures.

By pursuing these objectives, the study aims to make a substantial contribution to the field of credit card fraud detection, equipping financial institutions with a more potent tool to thwart fraud and safeguard consumer interests.

II. Biweekly Progress

This project encompasses a comprehensive exploration of machine learning methodologies, from the initial stages of topic determination to the application in real-world scenarios, emphasizing the challenge of imbalanced datasets. In the initial weeks, the focus on *Topic Determination* (Weeks 5-6) laid the groundwork for identifying relevant issues and formulating the project's objectives. This phase is crucial for aligning the project's goals with existing research, ensuring that the chosen topic is both significant and feasible for exploration within the given timeframe.

Subsequent phases, *Visualization with Final Dataset Details* (Weeks 7-8), and *Final Project Modeling* (Weeks 9-10), are dedicated to data preparation and preliminary analyses, crucial steps that dictate the effectiveness of the machine learning models. During these phases, the integration of visual analytics and meticulous dataset examination plays a pivotal role in understanding the data's characteristics and potential biases, a theme recurrent in related literature emphasizing data-centric approaches in machine learning projects.

The project then advances into more technical realms with *Feature Selection and Engineering* (Weeks 11-12), a phase where the art and science of extracting the most informative features are explored. This stage is often highlighted in literature as a critical step for enhancing model performance and interpretability, especially in applications involving complex and high-dimensional data.

Finally, the focus on *Imbalanced Dataset* handling and *Real-World Significance* (Weeks 13-14) reflects an acknowledgment of the challenges faced when deploying machine learning models in practical settings. The imbalance in datasets is a pervasive issue, affecting the model's ability to generalize and perform equitably across different groups. This project aligns with a growing body of research advocating for more robust and fair machine learning practices, underscoring the importance of addressing these challenges to achieve meaningful impact.

Throughout the project, a multidisciplinary approach is employed, integrating insights from various domains to address the multifaceted challenges of applying machine learning techniques. By engaging with a wide range of methodologies and prioritizing the real-world applicability of the developed models, this project contributes to the ongoing dialogue within the scientific community about the ethical and practical implications of machine learning.

III. Dataset and Visualization

A. Dataset

Data from Kaggle, a Google LLC-owned data science competition platform and online community for data scientists and machine learning practitioners.

This Credit Card Fraud Detection Dataset 2023 dataset records credit card transactions at 2023 for European cardholders.

This dataset has only numerical input variables, which are the result of PCA transformations[2]. Due to confidentiality reasons, the dataset does not provide the original features of the data and further background information. Features V1, V2, ... V28 are the principal components obtained by PCA, and only two features, 'ID' and 'Amount', are not transformed by PCA. Feature 'ID' represents the unique identifier for each transaction. The feature 'Amount' is the transaction amount and this feature can be used for sample-based cost-sensitive learning. Feature 'Class' is the response variable, its value of 1 means fraud and 0 means normal.

ID	V1	...	V28	Amount	Class
0	−0.26	...	−0.15	17 982.10	0
1	0.99	...	−0.06	6531.37	0
2	−0.26	...	−0.24	2513.54	0

TABLE I: Example Dataset Presentation

B. Process Data

We get it after looking at all the columns using the df.info command:

No Null Values

No Duplicate Values

No Categorical Data

C. Data Visualization

After we visualized Class, we found that the ratio of his positive to negative samples was at one to one.This means that the dataset is a good training sample, but note that this does not represent the actual situation.The Class visualization result is shown in Fig.1

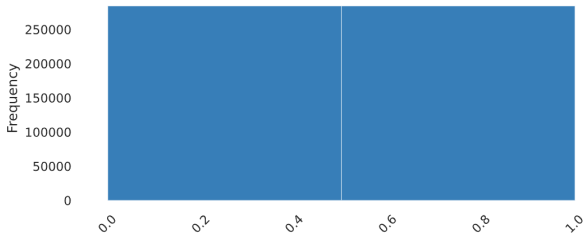

Fig. 1: The ratio of positive to negative samples is one to one

A correlation matrix denotes the correlation coefficients between variables at the same time. A heat map represents these coefficients to visualize the strength of correlation among variables.The heatmap is shown in Fig.2

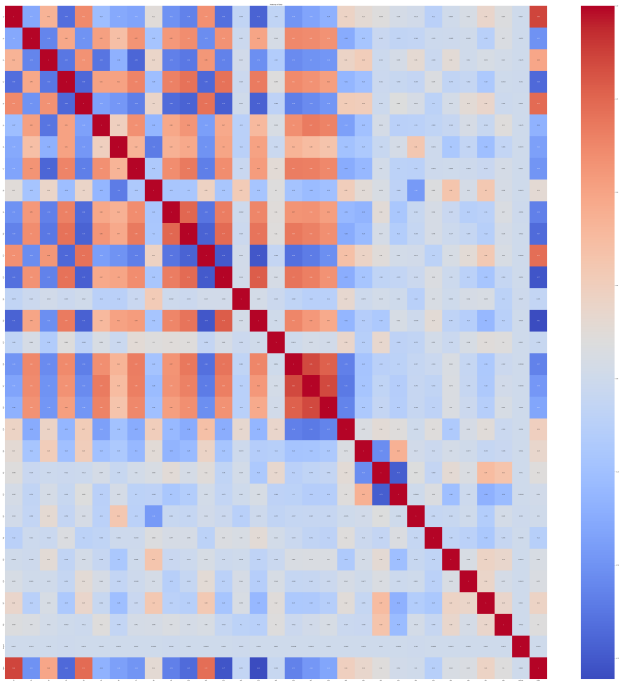

Fig. 2: Strength of relationship for all variables in the data set

IV. Classification Techniques

A. Methodologies

1) Logistic Regression: Logistic regression is a prevalent statistical method used for binary classification tasks[3]. It estimates the probability that a dependent variable belongs to a particular class, which is typically binary—class 1 (positive) or class 0 (negative). The core of logistic regression is the logistic function, often referred to as the sigmoid function, defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

where z is the linear combination of the input features X, weighted by coefficients β:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n \tag{2}$$

The model predicts the probability that X belongs to class 1 by applying the sigmoid function to z. If $\sigma(z) > 0.5$, the model predicts class 1; otherwise, it predicts class 0. The coefficients β are typically learned via maximum likelihood estimation, aimed at maximizing the likelihood of observing the training data given the model.

Logistic regression is favored for its simplicity and the interpretable nature of its coefficients, which indicate the strength and direction of the association between each feature and the log odds of the dependent variable being in class 1. Despite its simplicity, logistic regression can be extended with regularization methods to handle more complex relationships and prevent overfitting, making it a versatile tool for many binary classification problems.

2) Support Vector Machine (SVM): Support Vector Machines (SVMs) are a set of supervised learning methods used for classification, regression, and outliers detection[4]. The main idea behind SVM is to find a hyperplane in an n-dimensional space (where n is the number of features) that distinctly classifies the data points.

We are given a training dataset of n points of the form

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n), \ldots$$

where the y_i are either 1 or -1, each indicating the class to which the point $\mathbf{x}_i$ belongs. Each $\mathbf{x}_i$ is a p-dimensional real vector. We want to find the "maximum-margin hyperplane" that divides the group of points $\mathbf{x}_i$ for which $y_i = 1$ from the group of points for which $y_i = -1$, which is defined so that the distance between the hyperplane and the nearest point $\mathbf{x}_i$ from either group is maximized.

Any hyperplane can be written as the set of points $\mathbf{x}$ satisfying

$$\mathbf{w}^{\mathsf{T}}\mathbf{x} - b = 0, \tag{3}$$

where $\mathbf{w}$ is the (not necessarily normalized) normal vector to the hyperplane. The parameter $\frac{b}{\|\mathbf{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\mathbf{w}$.

a) Hard-margin: If the training data is linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the "margin", and the maximum-margin hyperplane is the hyperplane that lies halfway between them. With a normalized or standardized dataset, these hyperplanes can be described by the equations

$$\mathbf{w}^\mathsf{T}\mathbf{x} - b = 1 \tag{4}$$

which means anything on or above this boundary is of one class, with label 1, and

$$\mathbf{w}^\mathsf{T}\mathbf{x} - b = -1 \tag{5}$$

that is anything on or below this boundary is of the other class, with label -1.

Geometrically, the distance between these two hyperplanes is $\frac{2}{\|\mathbf{w}\|}$, so to maximize the distance between the planes we want to minimize $\|\mathbf{w}\|$. The distance is computed using the distance from a point to a plane equation. We also have to prevent data points from falling into the margin, we add the following constraint: for each i either

$$\mathbf{w}^\mathsf{T}\mathbf{x}_i - b \geq 1, \quad \text{if } y_i = 1, \tag{6}$$

or

$$\mathbf{w}^\mathsf{T}\mathbf{x}_i - b \leq -1, \quad \text{if } y_i = -1. \tag{7}$$

These constraints state that each data point must lie on the correct side of the margin.

This can be rewritten as

$$y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i - b) \geq 1, \quad \text{for all } 1 \leq i \leq n. \tag{8}$$

We can put this together to get the optimization problem:

$$\begin{aligned} &\underset{\mathbf{w},\, b}{\text{minimize}} \quad \|\mathbf{w}\|_2^2 \\ &\text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \quad \forall i \in \{1, \ldots, n\} \end{aligned} \tag{9}$$

The $\mathbf{w}$ and b that solve this problem determine our classifier, $\mathbf{x} \mapsto \operatorname{sgn}(\mathbf{w}^\mathsf{T}\mathbf{x} - b)$ where $\operatorname{sgn}(\cdot)$ is the sign function.

An important consequence of this geometric description is that the max-margin hyperplane is completely determined by those $\mathbf{x}_i$ that lie nearest to it (shown in Fig. 3). These $\mathbf{x}_i$ are called support vectors.

b) Soft-margin: To extend SVM to cases in which the data are not linearly separable, the "hinge loss" function is helpful

$$\max(0, 1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i - b)).$$

Note that y_i is the i-th target (i.e., in this case, 1 or -1), and $\mathbf{w}^\mathsf{T}\mathbf{x}_i - b$ is the i-th output.

This function is zero if the constraint in equation (1) is satisfied, in other words, if $\mathbf{x}_i$ lies on the correct side of the margin. For data on the wrong side of the margin, the function's value is proportional to the distance from the margin.

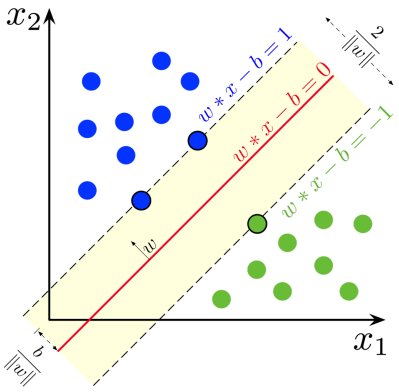

Fig. 3: Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

The goal of the optimization then is to minimize:

$$\|\mathbf{w}\|^2 + C\left[\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i - b))\right], \tag{10}$$

where the parameter $C > 0$ determines the trade-off between increasing the margin size and ensuring that the $\mathbf{x}_i$ lie on the correct side of the margin. By deconstructing the hinge loss, this optimization problem can be massaged into the following:

$$\begin{aligned} &\min_{\mathbf{w},b,\zeta} \quad \|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\zeta_i, \\ &\text{subject to} \\ &y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \zeta_i, \quad \forall i \in \{1, \ldots, n\}, \\ &\zeta_i \geq 0, \quad \forall i \in \{1, \ldots, n\}. \end{aligned} \tag{11}$$

Thus, for large values of C, it will behave similar to the hard-margin SVM, if the input data are linearly classifiable, but will still learn if a classification rule is viable or not.

3) K-nearest Neighbour: The K-Nearest-Neighbours (kNN) is a simple but effective method for classification [5].

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point. KNN's classification is demonstrated in Fig.4

In the K-Nearest Neighbors (KNN) algorithm, the optimal value of K, which represents the number of nearest neighbors to be considered during prediction, is crucial. The algorithm uses the Euclidean distance as the metric to measure the similarity between the target and the training data points. For a new data point $\mathbf{x}$, and each data point $\mathbf{X}_i$ in the training dataset

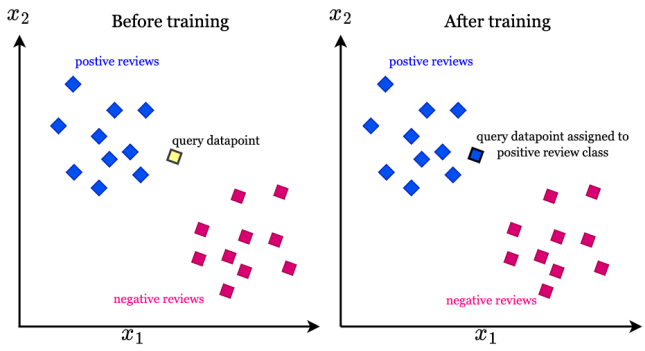

Fig. 4: The K-Nearest Neighbors algorithm assigns a new data point to a specific category based on its similarity to other data points in that category

$\mathbf{X}$ with n data points and d-dimensional feature vectors, the distance is calculated as follows:

$$\text{distance}(\mathbf{x}, \mathbf{X}_i) = \sqrt{\sum_{j=1}^{d} (x_j - X_{ij})^2} \tag{12}$$

where X_{ij} denotes the j-th dimension of the i-th data point in $\mathbf{X}$.

The algorithm selects the K data points from $\mathbf{X}$ that have the shortest distances to $\mathbf{x}$. In classification tasks, the class label y assigned to $\mathbf{x}$ is the one most frequently occurring among the K nearest neighbors. In regression tasks, the predicted value for $\mathbf{x}$ is either the average or the weighted average of the values y of the K nearest neighbors, thereby estimating the output based on local approximations.

4) Decision Tree: A decision tree [6] is a decision support hierarchical model that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility.

In Decision Trees, a versatile supervised learning algorithm, a tree-like model of decisions is built, where each internal node represents a test on a feature, each branch represents an outcome of the test, and each leaf node represents a class label or continuous value (for classification and regression tasks, respectively). The tree is constructed via a process known as recursive partitioning, starting with the root node that includes the entire dataset.

The key to building a decision tree lies in selecting the attribute that best splits the dataset into subsets composed of the most homogeneous class labels (or the most similar values in the case of regression). This is measured using criteria like Entropy and Gini Impurity, defined as follows:

Entropy for a dataset is defined as:

$$H(S) = -\sum_{x \in X} p(x) \log_2 p(x) \tag{13}$$

where $p(x)$ is the proportion of the number of elements in class x to the number of elements in set S.

Similarly, Gini Impurity is calculated as:

$$G(S) = 1 - \sum_{x \in X} p(x)^2 \tag{14}$$

The process of selecting the best attribute and splitting the dataset is repeated recursively on each derived subset in a manner known as recursive partitioning. The recursion is completed when the nodes are sufficiently homogeneous or when further splits are no longer beneficial.

5) Random Forest: In Random Forests[7], an ensemble learning technique, multiple decision trees are constructed during the training phase. The final output for classification tasks is determined by majority voting, whereas for regression, it is the average of the outputs from all trees. This methodology enhances the predictive accuracy and controls over-fitting.

Random Forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. Each tree in the forest is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. Additionally, when splitting each node during the construction of a tree, the best split is chosen from a random subset of features. The mathematical representation for the decision criterion at each split in a tree is based on maximizing the information gain:

$$\text{Information Gain} = H(S) - \sum_{t \in \{left, right\}} \frac{|S_t|}{|S|} H(S_t) \tag{15}$$

where $H(S)$ is the entropy of the set S, and S_t are the subsets created from splitting set S.

a) Feature Importance: Random Forest also provides an excellent feature of estimating the importance of features on an impurity-based criterion. The importance of a feature is computed as the decrease in node impurity weighted by the probability of reaching that node. The node probability is equivalent to the number of samples that reach the node, divided by the total number of samples. The higher the value, the more important the feature.

6) Neural Network: A neural network[8] is an interconnected group of nodes, inspired by a simplification of neurons in a brain. Neural networks are used for predictive modeling, adaptive control, and other applications where they can be trained via a dataset. This method can learn from experience, and can derive conclusions from a complex and seemingly unrelated set of information.

In our work, we mainly focus on convolutional neural network(CNN)[9]. CNN was initially used in the field of image processing and is the mainstream model in the field of image processing. With the development of deep learning, CNN has also been used in the processing of various large-scale data in recent years. It uses the convolution layer in the network to process the overall Features are extracted from the data, and then the dimensionality of the data is reduced through operations such as pooling, so it is suitable for training a large amount of data and has a mechanism to avoid model overfitting. The basic structure of the CNN model is shown in Fig. 5.

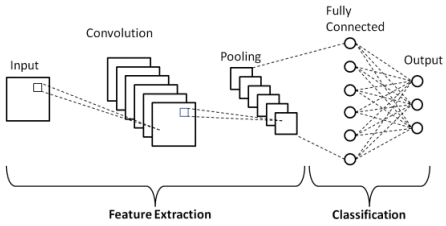

Fig. 5: A CNN structure typically consists of an input layer, convolutional layers, activation functions (ReLU), pooling layers, fully connected layers, and an output layer. The convolutional layers detect features, pooling reduces dimensionality, and fully connected layers used for classification.

The computation process for the i-th layer of a Convolutional Neural Network (CNN) can be formalized as follows:

$$C_i = f(\text{conv}(C_{i-1}, W_i) + b_i) \tag{16}$$

Here, C_i denotes the feature map of the i-th layer. W_i represents the weights of the convolutional kernel for the i-th layer, and b_i is the bias term for the i-th layer. The function $\text{conv}(\cdot, \cdot)$ specifies the convolution operation between the $(i-1)$-th layer feature map and the i-th layer's convolutional kernel. After adding the bias term b_i, the result is passed through a nonlinear activation function f, yielding the feature map C_i.

The pooling layer follows the convolutional layer C_i, which reduces the dimensionality and therefore the computational complexity while preserving the features that are invariant under small translations, as shown in the following equation:

$$H_i = \text{subsampling}(C_i) \tag{17}$$

Where, H_i represents the feature map after the pooling layer.

After the original data is transformed by the convolution layer and the pooling layer, it is sent to the fully connected layer to realize the classification and recognition of the extracted features. The Softmax function is usually used to receive this N-dimensional data as input, and then convert the value of each dimension into a real number between $(0, 1)$ as the recognition probability, and its formula is:

$$P_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}} \tag{18}$$

In this paper, the CNN Model is defined as follows, starting with its initialization which sets up the layers and their dimensions:

- The first convolutional layer (`conv1`) applies 64 filters of size 3 to the input, assuming the input has a single channel.
- A max-pooling layer (`pool`) with a kernel size of 2 follows, reducing the dimensionality by half.
- The second convolutional layer (`conv2`) further processes the data with 128 filters of size 3.
- The output size after each convolutional and pooling layer is dynamically calculated based on the input size, ensuring compatibility with the subsequent fully connected layer.
- A flattening operation converts the multi-dimensional tensor into a 1D tensor, preparing it for the dense layers.
- The first fully connected layer (`fc1`) takes the flattened output and reduces its dimension to 64.
- A dropout layer (`dropout`) with a rate of 0.5 is applied to reduce overfitting by randomly setting a portion of the inputs to zero during training.
- The second fully connected layer (`fc2`) further compresses the representation to a single output.
- Finally, a sigmoid activation function (`sigmoid`) scales the output to a range between 0 and 1, suitable for binary classification tasks.

The forward pass of the model, mapping inputs to outputs, can be described by the following operations:

1) Apply the first convolutional layer and ReLU activation: $x = \text{ReLU}(\texttt{conv1}(x))$.
2) Apply max-pooling: $x = \texttt{pool}(x)$.
3) Apply the second convolutional layer and ReLU activation: $x = \text{ReLU}(\texttt{conv2}(x))$.
4) Flatten the tensor: $x = \texttt{flatten}(x)$.
5) Apply the first fully connected layer and ReLU activation: $x = \text{ReLU}(\texttt{fc1}(x))$.
6) Apply dropout: $x = \texttt{dropout}(x)$.
7) Apply the second fully connected layer: $x = \texttt{fc2}(x)$.
8) Apply the sigmoid activation function: $x = \text{Sigmoid}(x)$.

And the overall model architecture is shown in Fig. 6

B. Performance Evaluation

This section introduces how to evaluate the performance of the classification model using various metrics and visual representations to ensure a comprehensive understanding of its effectiveness in correctly predicting classes.

1) Confusion Matrix: The confusion matrix[10] is a tabular representation that quantifies the performance of a classification algorithm. It contrasts the predicted classifications against the actual labels, as outlined below:

TABLE II: Classification Confusion Matrix

	Actual Class	
Predicted Class	**Positive** ($y = 1$)	**Negative** ($y = 0$)
Positive ($c = 1$)	True Positive (TP)	False Positive (FP)
Negative ($c = 0$)	False Negative (FN)	True Negative (TN)

In the matrix:

- **TP** – True Positives: Correct positive predictions.
- **TN** – True Negatives: Correct negative predictions.
- **FP** – False Positives: Incorrect positive predictions.
- **FN** – False Negatives: Incorrect negative predictions.

The confusion matrix is instrumental in computing subsequent performance metrics such as accuracy, precision, recall, and the F1-score.

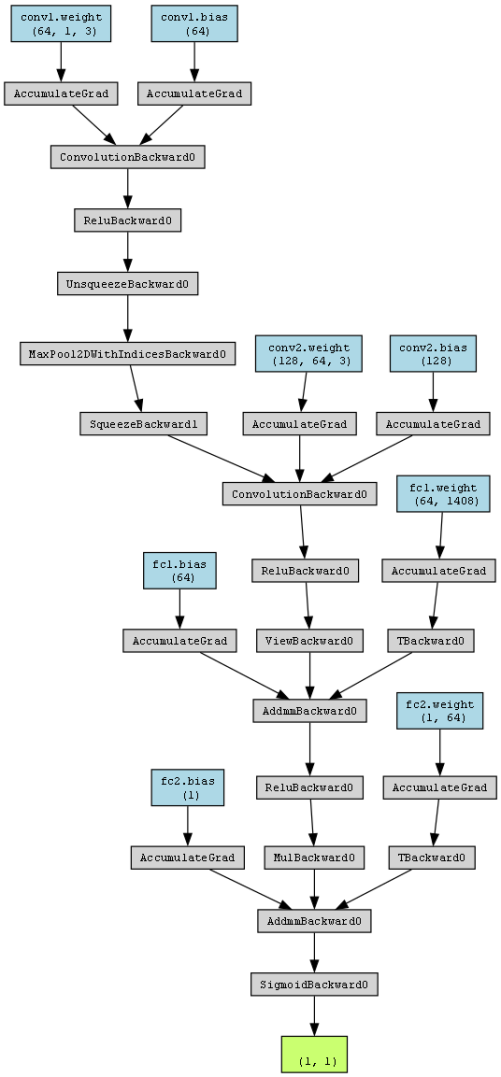

Fig. 6: This diagram illustrates the structure of a Convolutional Neural Network designed for binary classification tasks. It starts with two convolutional layers, each followed by ReLU activation and a max-pooling layer, to extract and downsample features from the input data. The extracted features are then flattened and passed through two fully connected layers, with a dropout layer in between to prevent overfitting. The final output is obtained using a sigmoid activation function, yielding a probability score between 0 and 1.

2) Model Performance Metrics: In the context of evaluating classification models, good metrics[11] that can comprehensively assess the model's ability to distinguish between classes. Below are some basic metrics:

- **Accuracy**
  The Accuracy metric evaluates the overall correctness of the model, representing the ratio of correctly predicted observations (both true positives and true negatives) to the total observations in the dataset.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{19}$$

- **Recall (Sensitivity or True Positive Rate)**
  Recall assesses the model's ability to correctly identify all relevant instances (true positives) among all actual positives in the dataset. This metric is crucial for fraud detection, where failing to detect fraudulent transactions (false negatives) could have significant financial implications.

$$Recall = \frac{TP}{TP + FN} \tag{20}$$

- **Precision (Positive Predictive Value)**
  Precision measures the accuracy of positive predictions made by the model, i.e., the proportion of predicted positives that are truly positive. High precision indicates a low rate of false positives, which is desirable in scenarios where the cost of false alarms is high.

$$Precision = \frac{TP}{TP + FP} \tag{21}$$

- **F1-Score**
  The F1-Score is the harmonic mean of Precision and Recall, providing a single metric to balance the trade-off between the two. It is particularly useful when the costs of false positives and false negatives vary significantly.

$$F1\text{-}Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{22}$$

C. Results

The following table provides a comprehensive overview of the performance metrics for different models on a balanced dataset. Below, we discuss the performance of each model:

- **Logistic Regression (LR)** shows a high overall performance with an accuracy of 96.49%. The precision is excellent at 97.80%, indicating high accuracy in predicting positive classes. The recall and F1-score are also robust at 95.13% and 96.44%, respectively, demonstrating good coverage of positive class samples and a balanced performance.
- **Support Vector Machine (SVM)** displays nearly perfect metrics, with all scores at 99.58%. This suggests that SVM is extremely effective for balanced datasets, achieving almost flawless classification.
- **K-Nearest Neighbors (KNN, N=3)** underperforms relative to other models, with the lowest accuracy of 82.37%. This could indicate sensitivity to noise and sample distribution, given the small neighborhood size used.
- **Decision Tree** achieves around 98% in all metrics, showing very high performance. This indicates that the decision tree model adapts well to the dataset's structure, effectively segmenting the data for accurate predictions.
- **Random Forest** also performs exceptionally well, with metrics approaching or achieving 99.80%. It generally outperforms a single decision tree, indicating better generalization.
- **Convolutional Neural Network (CNN)** has the best performance across all metrics, notably achieving 100% recall. This perfect score in recall indicates that it correctly identified all positive class samples, demonstrating its strong feature extraction capabilities.

Conclusion: For tasks requiring high precision and recall, CNN and SVM offer excellent choices. Random Forest and Decision Tree are also highly reliable for most scenarios, particularly when the dataset features complex structures. KNN might require more careful parameter selection, especially in larger and more diverse datasets. These results suggest that most advanced machine learning models can achieve very high levels of performance on balanced datasets, but the choice of model should be guided by specific application needs and performance requirements.

Model	Accuracy	Precision	Recall	F1-score
LR	0.9649	0.9780	0.9513	0.9644
SVM	0.9958	0.9958	0.9958	0.9958
KNN(N=3)	0.8237	0.8674	0.7636	0.8122
Decision Tree	0.9830	0.9789	0.9872	0.9831
Random Forest	0.9980	0.9983	0.9976	0.9980
CNN	**0.9997**	**0.9994**	**1.0000**	**0.9997**

TABLE III: Performance of Different Models on Balanced Dataset

D. Feature Engineering

In this section, we explore the role of feature engineering[12] in enhancing the predictive accuracy of a logistic regression model used in our study. Feature engineering involves creating new input features or modifying existing ones to improve model performance. We initially employ Recursive Feature Elimination (RFE)[13] alongside Logistic Regression to identify the most significant predictors and refine the feature set. To optimize our model, we implement GridSearchCV for systematic hyperparameter tuning through cross-validation. This section details the methodologies applied—PolynomialFeatures, StandardScaler, and MinMaxScaler—and presents comparative results demonstrating the efficacy of each technique in improving various performance metrics such as accuracy, precision, recall, and F1-score.

1) Basic Model and Parameters: In our study, we employed a logistic regression model to demostrate feature engineering. Initially, we utilized a Recursive Feature Elimination (RFE) method with Logistic Regression to narrow down the most impactful predictors. We tuned the hyperparameters using GridSearchCV.

The GridSearchCV method systematically explores a range of specified parameter combinations and implements cross-validation to evaluate each combination. We defined our grid as follows:

- Regularization strengths (`C`): [0.01, 0.1, 1, 10, 100, 1000]
- Penalty types (`penalty`): ['l1', 'l2']
- Solvers: ['lbfgs', 'liblinear', 'newton-cg', 'sag', 'saga']

GridSearchCV evaluated the performance of each combination using a 5-fold cross-validation, which ensures that each subset of the data is used for both training and validation. The selection of the best hyperparameters was based on the highest average cross-validation score. The best parameters identified by GridSearchCV for our logistic regression model were: $\{C : 1000,$ penalty: 'l1', solver: 'liblinear'$\}$.

2) Feature Engineering Methodologies: Feature engineering plays a crucial role in improving the performance of machine learning models by creating new input features or modifying existing ones. We discuss three common feature engineering methods: PolynomialFeatures, StandardScaler, and MinMaxScaler.

a) Polynomial Features: PolynomialFeatures is a method for generating a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree. For a given feature vector $x = (x_1, x_2, \ldots, x_n)$, the transformation to a polynomial feature space up to degree d is represented as:

$$\phi(x) = (1, x_1, x_2, \ldots, x_n, x_1^2, x_1x_2, \ldots, x_n^d) \tag{23}$$

where $\phi(x)$ denotes the feature vector after applying PolynomialFeatures. The inclusion of these polynomial terms allows models (especially linear models) to capture more complex relationships in the data.

b) Standard Scaler: StandardScaler standardizes features by removing the mean and scaling to unit variance. This is a common requirement for many machine learning estimators as they might behave badly if the individual features do not more or less look like standard normally distributed data (Gaussian with zero mean and unit variance).

For a dataset with mean μ and standard deviation σ, the standard score of a sample x is calculated as:

$$z = \frac{(x - \mu)}{\sigma} \tag{24}$$

where x is the original feature value, μ is the mean of the feature values, and σ is the standard deviation of the feature values. This transformation redistributes the features to center around 0 with a standard deviation of 1.

c) Min-Max Scaler: MinMaxScaler scales each feature to a given range, typically 0 to 1, or so that the minimum and maximum of the feature is scaled to a exact range, such as (-1, 1). It works according to the formula:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \times (b - a) + a \tag{25}$$

where x is an original value, x' is the normalized value, and a, b are the new minimum and maximum values of the scaled feature respectively. This scaling compresses all feature values within the specified range thereby ensuring consistent scale across features.

3) Results of Feature Engineering: In this section, we present the results of applying various feature engineering techniques to logistic regression. Feature selection and transformation techniques, such as Recursive Feature Elimination (RFE) and feature scaling, have been used to enhance the model's predictive power. The following table summarizes the performance of each configuration:

Model	Accuracy	Precision	Recall	F1-score
LR	0.9579	0.9805	0.9343	0.9569
LR + RFE	0.9651	0.9794	0.9501	0.9645
LR + Polynomial	0.9449	0.9806	0.9075	0.9426
LR + Standard	0.9647	0.9775	0.9512	0.9642
LR + MinMax	0.9596	0.9844	0.9338	0.9585
LR + RFE + Polynomial	**0.9857**	**0.9865**	**0.9848**	**0.9857**
LR + RFE + Standard	0.9651	0.9794	0.9501	0.9645
LR + RFE + MinMax	0.9591	0.9849	0.9323	0.9579

TABLE IV: Performance metrics of logistic regression models after feature engineering.

4) Analysis: The results derived from the application of various feature engineering techniques highlight the effectiveness of these methods in improving the predictive accuracy of logistic regression models. The application of Recursive Feature Elimination (RFE) coupled with polynomial features notably provided the highest boost in all performance metrics, including accuracy, precision, recall, and F1-score. This outcome emphasizes the advantage of not only selecting the most relevant features but also transforming them to capture non-linear relationships which may be crucial for the model's performance.

The standalone impact of feature scaling, as seen from the Logistic Regression models enhanced with Standard Scaler and MinMax Scaler, suggests that normalization or standardization of feature values plays a significant role in logistic regression, a model sensitive to feature scaling. The improvements in accuracy and other metrics can be attributed to the model's enhanced ability to generalize from a normalized feature space.

Furthermore, the comparison between different combinations of feature engineering techniques suggests a diminishing return when RFE is combined with feature scaling alone, without polynomial features. This could indicate that while RFE effectively reduces the feature space to the most influential predictors, additional transformations such as polynomial features are necessary to extract maximum predictive power from the reduced feature set.

The comparative analysis across different configurations also suggests that while polynomial features alone do not achieve the highest performance (as seen in the LR + Polynomial configuration), their combination with RFE is particularly potent. This synergy likely arises from the ability of polynomial features to model complex interactions which are then refined by RFE to retain only the most impactful of these interactions.

In conclusion, the observed results advocate for a nuanced approach to feature engineering in logistic regression models. While each technique has its merits, their strategic combination, particularly the use of RFE with polynomial feature transformation, appears to be most effective for maximizing model performance. This analysis not only supports the continued use of these techniques but also encourages further exploration into optimal combinations for different types of data sets in future studies.

E. Feature Selection

1) The Curse of Dimensionality and its Impact on K-Nearest Neighbors: As the dimensionality of a dataset increases, the volume of the space in which the data resides grows exponentially. This exponential growth leads to a rapid increase in sparsity; the data points, which were once close in lower-dimensional spaces, now find themselves isolated in higher dimensions.This phenomenon is known as the curse of dimensionality [14].

Consider a one-dimensional space, such as a line, where a small number of points can densely populate the region. Extending this to two dimensions, a square area requires significantly more points to achieve a similar level of density. Progressing further to three dimensions, a cube demands an even greater number of points to fill its volume adequately.

In the context of high-dimensional spaces, this phenomenon poses a substantial challenge to algorithms like K-Nearest Neighbors (KNN). The foundational principle of KNN is to classify data points based on the proximity and similarity of their nearest neighbors. However, in high-dimensional spaces, the concept of 'nearness' becomes distorted. Distances between data points increase and become less meaningful, which often results in a deterioration of the algorithm's ability to discern and classify points accurately based on their closest neighbors. Essentially, what were considered 'nearest neighbors' in lower dimensions may no longer share similarities in higher dimensions, undermining the effectiveness of the KNN algorithm.

2) Principal Components Analysis(PCA): Principal Component Analysis (PCA)[15] is a statistical technique that mitigates the effects of the curse of dimensionality by identifying the most significant directions, or principal axes, in a dataset. These axes, where the variance of the data reaches its maximum, are defined as principal components. Each principal component is a linear combination of the original variables and encapsulates a specific amount of the dataset's total variance.As shown in Fig.7

The procedure begins by determining the first principal component, which captures the greatest variance present in the data. Subsequent principal components are computed orthogonally to the preceding ones and capture progressively smaller portions of the data's variance. This sequential process ensures that each component maximizes the variance it captures, conditional on being orthogonal to the earlier components.

By focusing on the first few principal components, PCA allows for a significant reduction in the dimensionality of the dataset. This reduction is achieved without substantial loss of information, as these components retain the most critical aspects of the original data.

3) PCA Choose Optimal Features: When applying PCA, the first step is to standardize the feature set if the features have different units of measurement. Standardization makes the data unitless and brings different variables to the same scale. Following this, PCA is performed without initially specifying the number of components, which allows for the examination

Algorithm 1 Principal Component Analysis (PCA)

1: **Input:** Data matrix X of dimensions $n \times d$, where n is the number of data points and d is the number of dimensions
2: **Output:** Principal components, variance explained by each component
3: Standardize the columns of X to have mean 0 and variance 1
4: Compute the covariance matrix Σ of X
5: Compute the eigenvalues λ_i and eigenvectors v_i of Σ
6: Sort the eigenvectors by decreasing eigenvalues
7: Select the top k eigenvectors to form the matrix W of dimensions $d \times k$
8: Compute the transformed data $Y = XW$
9: **return** Y, eigenvectors v_i, and eigenvalues λ_i

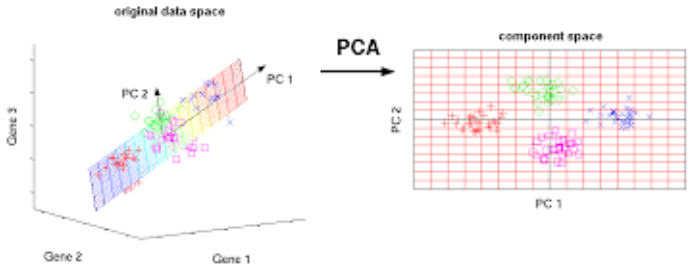

Fig. 7: Three-dimensional gene expression data which are mainly located within a two-dimensional subspace

of the cumulative explained variance ratio as a function of the number of components.

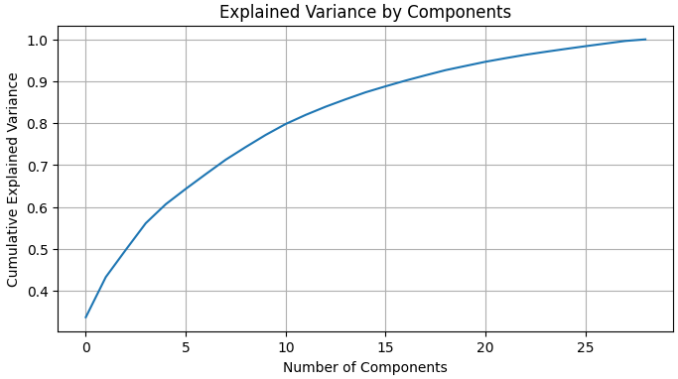

Fig. 8: Cumulative explained variance ratio as a function of the number of principal components.

As illustrated in Figure 8, the cumulative explained variance ratio typically increases with the number of components, eventually plateauing. The goal is to choose a number of components such that the cumulative explained variance is above a predefined threshold, we set at **95**%.

```
Optimal number of components: 22
```

In the provided example, the optimal number of components is found to be 22. By selecting the first 22 principal components, we ensure that at least 95% of the total variance is retained, thereby reducing the dataset to its most informative features without a significant loss of information.

Subsequently, PCA can be applied with this optimal number of components for both training and testing data. This results in a transformed dataset that is lower in dimensionality.

4) Hyperparameter Tuning for KNN: The K-Nearest Neighbors (KNN) algorithm is sensitive to the choice of the hyperparameter k, which represents the number of nearest neighbors to consider when making predictions. Optimal selection of k is crucial to the model's performance.

A series of experiments were conducted where the KNN model was trained with different values of k, ranging from 1 to 20. For each value of k, the model was evaluated on a test set to ascertain its accuracy.

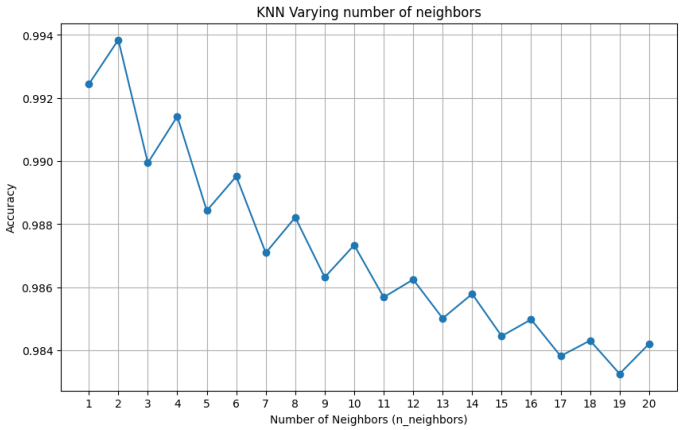

Fig. 9: Accuracy of the KNN model as a function of the number of neighbors ($n\_neighbors$).

As depicted in Figure 9, the accuracy of the model varies with the number of neighbors, with an observable peak in performance. At $k = 2$, the model achieves the highest recorded accuracy of 0.9938, indicating an optimal balance between bias and variance. The detailed results are summarized in the following table:

Model	Accuracy	Precision	Recall	F1-Score
KNN (k=3)	0.8237	0.8674	0.7636	0.8122
Standard+KNN (k=2)	0.9931	0.9898	0.9990	0.9944
Standard+PCA(22)+KNN (k=2)	**0.9938**	**0.9899**	0.9979	0.9939

TABLE V: Performance metrics for the KNN model with $k = 3$ and after hyperparameter optimization with $k = 2$.

The classification report for the optimal model with $k = 2$ indicates significant improvements in all metrics, as shown in Table V. This suggests that, particularly for datasets that have undergone PCA for dimensionality reduction, careful tuning of the k parameter in KNN can lead to substantial gains in model accuracy and generalization capability.

5) Analysis: The methodology employed in this investigation underscores the importance of hyperparameter tuning in machine learning models. Through iterative training and evaluation, one can systematically identify the hyperparameters

that yield the best predictive performance, as demonstrated by the KNN algorithm's enhanced accuracy with an optimal k.

V. Real-World Application

A. Imbalanced Dataset Generating

To simulate the realistic imbalance typically observed in credit card fraud detection datasets, we modified our initially balanced dataset to create a significantly imbalanced version. The original dataset comprised an equal number of fraudulent and non-fraudulent transactions, each class containing 284,315 samples, thereby making the proportion of both classes equal (50%).

We employed the following procedure to generate the imbalanced datasets:

1) **Data Segregation:** We separated the data into two classes based on the 'Class' label, where class 1 represents fraudulent transactions and class 0 represents legitimate transactions.
2) **Data Splitting:** Each class dataset was then split into training and testing subsets using a 50% split ratio, ensuring both subsets were representative of the overall class distribution.
3) **Subset Rebalancing:** We rebalanced the training and testing datasets to reflect a more realistic fraud scenario, where fraudulent transactions are significantly less common compared to non-fraudulent ones. We set the target proportion of fraudulent samples (positive class) at approximately 1%, mimicking common real-world fraud prevalence.
4) **Sample Calculation:** To achieve this, we calculated the number of positive samples required to maintain a 1% fraud proportion given the total number of negative samples. This calculation was based on the formula:

$$n_{\text{positive}} = \frac{n_{\text{negative}} \times \text{positive proportion}}{1 - \text{positive proportion}}$$

where n_{negative} is the number of non-fraudulent transactions.
5) **Dataset Generation:** Using the calculated number of positive samples, we sampled from the positive class dataset without replacement. We combined these with the negative samples to form the final imbalanced datasets for both training and testing.

The resulting training dataset contained 142,157 non-fraudulent and 1,435 fraudulent transactions, resulting in a proportion of approximately 99% non-fraudulent and 1% fraudulent transactions. The test dataset mirrored this distribution closely. This approach allowed us to create a dataset that better represents the typical imbalance found in operational environments, facilitating more realistic model training and evaluation.

B. Metric of Savings

In this fraud detection classification problem of imbalanced dataset, the performance of each models is assessed using a set of established metrics and a novel metric developed specifically for this context. The evaluation framework includes the accuracy, recall, precision, F1-score, and a custom metric termed savings. While the traditional metrics provide insights into the general effectiveness of the models, Savings quantitatively measures the financial impact of fraud detection, offering a direct assessment of cost efficiency unique to the credit card fraud detection problem.

While common metrics derived from the confusion matrix are widely used, they typically assume that all misclassification errors incur the same cost, which may not be appropriate in problems such as fraud detection. To address this limitation, we introduce the savings metric, which is derived from an example-dependent cost matrix and reflects the variable costs associated with different types of misclassification errors.

The savings metric is calculated as follows:

$$Cost = \sum_{i=1}^{N}((1 - c_i) \cdot y_i \cdot Amt_i) + c_i \cdot C_a \tag{26}$$

$$Cost_l = \sum_{i=1}^{N} y_i \cdot Amt_i \tag{27}$$

$$Savings = 1 - \frac{Cost}{Cost_l} \tag{28}$$

where N is the number of transactions, c_i is the predicted class, y_i is the actual class, Amt_i is the amount of the transaction, and C_a is the administrative cost incurred when investigating a transaction flagged as fraudulent.

This metric specifically accommodates the unique cost considerations inherent in credit card fraud detection by quantifying the economic benefit of the model's ability to correctly identify fraudulent transactions, taking into account the varying costs of false positives and false negatives.

C. Resampling Methodologies

Imbalanced datasets are a common and significant challenge in the field of machine learning, particularly in classification tasks where the unequal distribution of classes can lead to biased models that preferentially predict the majority class. Resampling methodologies are critical in preprocessing steps to balance class distributions and ensure that the predictive modeling is not skewed. These methodologies generally fall into two broad categories: oversampling and undersampling. Oversampling methods increase the size of the minority class by adding new instances, whereas undersampling methods reduce the size of the majority class by removing some instances. Both approaches aim to create a more balanced class distribution, thereby allowing classification algorithms to perform more effectively.

In this section, we delve into several resampling strategies, each with its own merits and drawbacks. We begin with the simplest techniques, random oversampling and undersampling, and then explore more sophisticated methods such as the Synthetic Minority Oversampling Technique (SMOTE) and Tomek Links. We also consider hybrid approaches like SMOTE-Tomek Links, which combine the strengths of these methods to

create a dataset conducive to developing robust and unbiased classification models. The efficacy of these techniques is not only theoretical but also practical, as we will demonstrate through illustrative examples.

1) Random Oversampling and Undersampling: When it comes to methods for dealing with datasets with imbalanced class distributions, random oversampling and undersampling are well known for their simplicity and straightforwardness. However, their simple implementation brings obvious challenges and potential shortcomings to model performance.

Random oversampling is a technique that solves the class imbalance problem by randomly replicating instances of a minority class, aiming to achieve a balanced class distribution. However, its simplicity can lead to overfitting as it increases the likelihood of replicating the same instances repeatedly, possibly biasing the model towards these overrepresented examples.

Random undersampling, on the other hand, involves reducing the size of the majority class by randomly removing instances to balance the class distribution. This approach does reduce the risk of overfitting by simplifying the learning process of the model. Nonetheless, it introduces the risk of underfitting, as random elimination of instances may result in the loss of critical data, thereby impairing the model's ability to generalize from training data to unseen data.

2) SMOTE: The Synthetic Minority Oversampling Technique (SMOTE), developed by Chawla et al. in 2002[16], presents an advanced approach to addressing the challenge of class distribution imbalance. Unlike traditional random oversampling, which merely duplicates random examples from the minority class, SMOTE synthesizes new examples. This is achieved by leveraging the concept of Euclidean distance and the k-nearest neighbors algorithm to create instances that are similar yet distinct from the original minority class data.

The procedure for generating synthetic samples through SMOTE includes several steps:

1) Select a random instance from the minority class.
2) Compute the Euclidean distance to identify the k nearest neighbors of this instance within the minority class.
3) For each neighbor, a synthetic sample is generated by interpolating between the selected instance and its neighbor. This interpolation involves multiplying the difference vector between the two points by a random number between 0 and 1 and adding it to the original instance.
4) This process is repeated until the minority class is sufficiently augmented to achieve the desired class proportion.

Here is the pseudocode Alg. 2:

SMOTE's effectiveness lies in its ability to introduce new information to the dataset by populating the feature space near the minority class with synthetic samples. These samples are conceived to closely mimic the variability within the minority class, thereby enriching the dataset without merely replicating existing data.

Algorithm 2 SMOTE Algorithm

Input: Minority class C_{min}, oversampling rate N, number of nearest neighbors k
Output: Augmented set of minority class samples
procedure SMOTE(C_{min}, N, k)
    Initialize $SyntheticSamples$ to empty
    for $i \leftarrow 1$ to $|C_{min}|$ **do**
        $sample \leftarrow C_{min}[i]$
        $Neighbors \leftarrow$ ComputeKNearestNeighbors($sample$, C_{min}, k)
        for $n \leftarrow 1$ to N **do**
            $nn \leftarrow$ RandomSelection($Neighbors$)
            $diff \leftarrow nn - sample$
            $gap \leftarrow$ RandomNumberBetween(0, 1)
            $syntheticSample \leftarrow sample + gap \cdot diff$
            Add $syntheticSample$ to $SyntheticSamples$
        end for
    end for
    return $SyntheticSamples$
end procedure

However, it is pertinent to acknowledge a limitation of SMOTE: its focus on augmenting the minority class without direct consideration for the distribution of the majority class. This can sometimes result in synthetic instances that blur the distinction between classes, especially in datasets where the class boundaries are not distinctly defined. Such scenarios may complicate the task of classification algorithms in distinguishing between classes.

An illustrative example of how SMOTE operates can be seen in Fig.10, demonstrating the synthetic sample generation process.

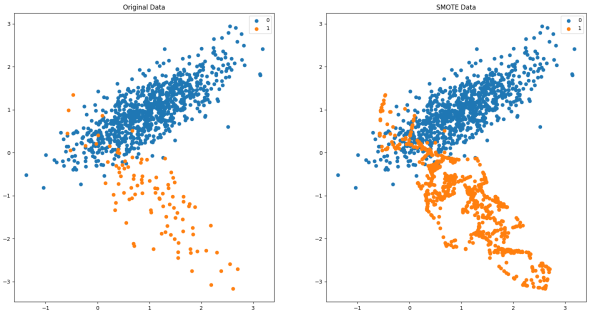

Fig. 10: Comparison of class distribution in a dataset before and after the application of SMOTE: The left plot shows the original data with a class imbalance, while the right plot illustrates the effect of SMOTE in creating synthetic samples to achieve a more balanced class distribution.

3) Tomek Links: Tomek Links is an undersampling technique developed by Tomek in 1976[17], which is a modification of the Condensed Nearest Neighbor (CNN) method. Unlike CNN, which randomly selects samples for removal

based on their proximity to the k nearest neighbors from the majority class, Tomek Links employs a more deliberate rule-based approach for identifying and removing specific samples that contribute to class overlap. This method focuses on pairs of observations, denoted as a and b, that satisfy the following conditions:

1) The observation a's nearest neighbor is b, and vice versa, the observation b's nearest neighbor is a.
2) Observations a and b belong to different classes; specifically, one belongs to the minority class, and the other belongs to the majority class.

Mathematically, Tomek Links can be described using the concept of Euclidean distance, $d(x_i, x_j)$, between two samples x_i and x_j, where x_i is a sample from the minority class and x_j is a sample from the majority class. A pair (x_i, x_j) is considered a Tomek Link if there is no sample x_k that satisfies either of the following conditions:

1) $d(x_i, x_k) < d(x_i, x_j)$, or
2) $d(x_j, x_k) < d(x_i, x_j)$.

By identifying and removing the majority class samples that are nearest to the minority class samples, the Tomek Links method effectively reduces the overlap between classes. This targeted removal of ambiguous samples near the decision boundary aims to enhance the classifier's ability to distinguish between classes, thereby improving the overall classification performance on imbalanced datasets.

Here is the pseudocode Alg. 3:

Algorithm 3 Tomek Links Algorithm

Input: Dataset D with majority class C_{maj} and minority class C_{min}
Output: Reduced dataset D' without Tomek Links
procedure TOMEKLINKS(D, C_{maj}, C_{min})
    Initialize $TomekLinks$ to empty
    for all sample x_i in D **do**
        $x_j \leftarrow$ NearestNeighbor(x_i, $D \setminus \{x_i\}$)
        $x_k \leftarrow$ NearestNeighbor(x_j, $D \setminus \{x_j\}$)
        if $x_k = x_i$ **and** $x_i \in C_{min} \neq x_j \in C_{maj}$ **then**
            Add pair (x_i, x_j) to $TomekLinks$
        end if
    end for
    for all pairs (x_i, x_j) in $TomekLinks$ **do**
        if x_j in C_{maj} **then**
            Remove x_j from D
        end if
    end for
    $D' \leftarrow D$
    return D'
end procedure

An illustrative example of how Tomek Links undersampling operates can be seen in Fig.11, which demonstrates the identification and removal of Tomek Links.

4) Combine: SMOTE-Tomek Links is a hybrid method that was first introduced by Batista et al. in 2004[18]. This

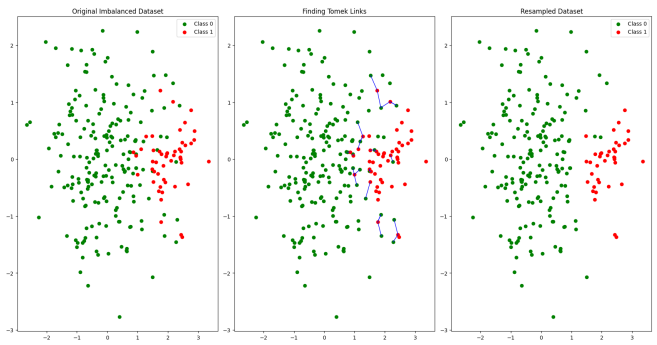

Fig. 11: Visual illustration of Tomek Links undersampling: The left plot shows the original imbalanced dataset, the middle plot identifies the Tomek Links (pairs of samples from different classes that are each other's nearest neighbors), and the right plot shows the dataset after removing the Tomek Links, resulting in a slightly more balanced class distribution.

method leverages the synthetic data generation capability of SMOTE for the minority class and the cleaning effects of Tomek Links, which removes samples from the majority class that are involved in Tomek Links. The procedure for SMOTE-Tomek Links is as follows:

1. **SMOTE:**
    a) Choose random data from the minority class.
    b) Calculate the distance between the random data and its k nearest neighbors.
    c) Multiply the difference by a random number between 0 and 1 and add the result to the minority class as a synthetic sample.
    d) Repeat steps (b) and (c) until the desired proportion of the minority class is achieved.
2. **Tomek Links:**
    a) Choose random data from the majority class.
    b) If the random data's nearest neighbor belongs to the minority class and they form a Tomek Link, remove the Tomek Link.

Here is the pseudocode Alg. 4:

An exemplification of the SMOTE-Tomek Links resampling method can be observed in Fig.12. This figure displays the process of augmenting the minority class with synthetic data followed by the refinement of the class boundaries by removing Tomek Links.

D. Results of Resampling

This section evaluates the impact of different resampling techniques on machine learning models trained on imbalanced datasets. Data imbalance can significantly skew a model's performance, favoring the majority class. Through methods like oversampling the minority class and undersampling the majority class, we aim to determine which resampling strategies enhance model accuracy and robustness. We analyze several models—Logistic Regression, K Nearest Neighbors, Random Forest, XGBoost, Support Vector Machines, and Convolutional Neural Networks—each configured differently to

Algorithm 4 SMOTE-Tomek Links Algorithm

Input: Dataset D with majority class C_{maj} and minority class C_{min}, oversampling rate N, number of nearest neighbors k
Output: Enhanced and cleaned dataset D'
procedure SMOTETOMEKLINKS(D, C_{maj}, C_{min}, N, k)
    /* Step 1: Apply SMOTE */
    Initialize $SyntheticSamples$ to empty
    for $i \leftarrow 1$ to $|C_{min}|$ **do**
        $sample \leftarrow C_{min}[i]$
        $Neighbors \leftarrow$ ComputeKNearestNeighbors($sample$, C_{min}, k)
        for $n \leftarrow 1$ to N **do**
            $nn \leftarrow$ RandomSelection($Neighbors$)
            $diff \leftarrow nn - sample$
            $gap \leftarrow$ RandomNumberBetween(0, 1)
            $syntheticSample \leftarrow sample + gap \cdot diff$
            Add $syntheticSample$ to $SyntheticSamples$
        end for
    end for
    $C_{min} \leftarrow C_{min} \cup SyntheticSamples$
    $D \leftarrow C_{maj} \cup C_{min}$
    /* Step 2: Remove Tomek Links */
    Initialize $TomekLinks$ to empty
    for all sample x_i in D **do**
        $x_j \leftarrow$ NearestNeighbor(x_i, $D \setminus \{x_i\}$)
        $x_k \leftarrow$ NearestNeighbor(x_j, $D \setminus \{x_j\}$)
        if $x_k = x_i$ **and** class(x_i) $\neq$ class(x_j) **then**
            Add pair (x_i, x_j) to $TomekLinks$
        end if
    end for
    for all pairs (x_i, x_j) in $TomekLinks$ **do**
        if class(x_j) = C_{maj} **then**
            Remove x_j from D
        end if
    end for
    $D' \leftarrow D$
    return D'
end procedure

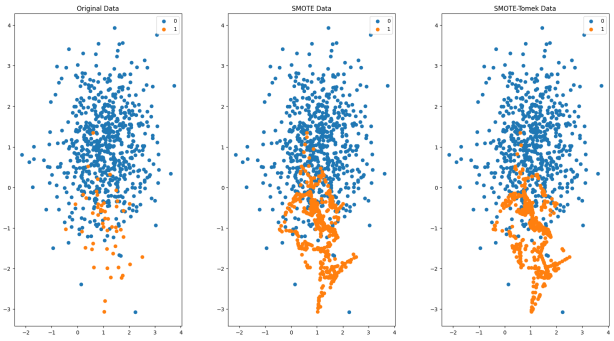

Fig. 12: Graphical representation of the SMOTE-Tomek Links resampling approach: The leftmost plot illustrates the initial dataset exhibiting a significant class imbalance. The center plot depicts the application of SMOTE, enhancing the minority class representation. The rightmost plot demonstrates the subsequent application of Tomek Links, refining the dataset by eliminating the nearest opposite-class pairs, thus potentially improving the classifier's performance.

assess their responses to resampling. This comparative analysis helps identify the most effective techniques for managing class imbalance in predictive modeling.

1) Models Introduction: In our exploration of the effectiveness of various resampling techniques on imbalanced datasets, we employ a diverse set of machine learning models, each selected for its unique strengths and suitability for handling data imbalance. The models include:

- **Logistic Regression:** As a probabilistic linear model, logistic regression provides a straightforward and interpretable methodology, making it an ideal baseline model. It performs well in binary classification tasks, and with appropriate regularization and class weighting, it can be effectively applied to imbalanced datasets.
- **K Nearest Neighbors (KNN):** An instance-based learning methodology, KNN classifies a new sample based on the majority vote of its k nearest neighbors. Its non-parametric nature allows it to adapt effectively to real-world datasets, making it a versatile choice for our analysis. KNN's flexibility and the potential for fine-tuning through distance weighting and the selection of an optimal k make it an excellent candidate for handling imbalanced datasets.
- **Random Forest (Ensemble Trees):** This ensemble learning model improves prediction accuracy by combining the outputs of multiple decision trees. Random Forest is particularly adept at managing imbalanced data, with its performance benefitting from the tuning of parameters like class weights. Its ability to capture non-linear relationships without the need for feature scaling is a significant advantage for our study.
- **XGBoost:** XGBoost is a gradient boosting model known for its speed and performance, XGBoost has built-in mechanisms for dealing with imbalanced datasets, notably through the *scale_pos_weight* parameter. Its robustness across a broad spectrum of classification tasks and capability to uncover complex patterns in data make it a key model for our investigation.
- **Support Vector Machine (SVM):** SVMs operate effectively in high-dimensional spaces and offer versatility through various kernel options. For imbalanced datasets, SVM's sensitivity to the minority class can be enhanced with the right kernel choice and parameter adjustments, such as class weights, making it particularly useful for our analysis.
- **Neural Networks (specifically, CNN):** Offering the flexibility to model complex, non-linear relationships, neural networks are scalable and adaptable, making them suited for large datasets and capable of addressing class

imbalance through modifications in architecture, loss functions, and sampling techniques. The application of focal loss, in particular, allows the model to concentrate on more challenging classifications, often associated with the minority class.

Each model was chosen for its potential to provide insight into the effectiveness of resampling methodologies in addressing the challenges posed by imbalanced datasets. The diversity of these models also allows for a comprehensive evaluation across different algorithmic approaches, enhancing the robustness of our findings.

E. Results and Analysis

In this section, we thoroughly investigate the outcomes of applying various machine learning models to imbalanced datasets, focusing on how different configurations and resampling techniques impact their performance. This section starts with a detailed presentation of the parameters for each model, ensuring that readers can understand the basis of our comparative analysis. Following this, we delve into the results obtained from employing different resampling strategies such as SMOTE, Tomek Links, and their combination, to highlight their effects on the predictive accuracy and other performance metrics of the models. This comprehensive examination aims to provide insights into the optimal strategies for handling imbalanced datasets in predictive modeling.

1) Models Parameters: In this subsection, we present the configuration parameters for each model utilized in our study. These parameters were meticulously chosen to optimize each model's performance on imbalanced datasets. Table 1 provides a detailed overview of the parameter settings for the Logistic Regression, Random Forest, SVM, XGBoost, CNN, and KNN models, which are essential for understanding the differences in model behavior and performance.

Model	**Parameters**
LR	max_iter=1000, class_weight='balanced', penalty='l1', C=1000, solver='liblinear'
RF	n_estimators=200, max_depth=20, min_samples_split=4, min_samples_leaf=2, class_weight='balanced', bootstrap=True, random_state=42
SVM	kernel='rbf', C=1.0, gamma='scale', class_weight='balanced', max_iter=50000, probability=True
XGBoost	max_depth=3, learning_rate=0.1, n_estimators=100 (default settings)
CNN	Conv layers (64 filters, 128 filters), Kernel size=3, Pool size=2, Dropout rate=0.5, Linear layers, Optimizer=Adam, Learning rate=0.001, Epochs=75
KNN	n_neighbors=3, weights='uniform', algorithm='auto', metric='minkowski' (p=2)

TABLE VI: Configuration Parameters of Different Machine Learning Models

2) Results: We evaluated the performance of each model under different conditions: without resampling, using SMOTE, using Tomek Links, and a combination of SMOTE and Tomek Links. These conditions are designed to illustrate how each resampling technique influences the predictive accuracy and other relevant metrics of the models.

a) Baseline Performance: Initially, we assess the models' performance without any resampling method. This serves as a baseline for comparing the efficacy of the different resampling techniques applied. The results are summarized in Table 2, which provides insight into how well each model performs when confronted with imbalanced data without any intervention.

Model	Accuracy	Precision	Recall	F1-score	Savings
LR	0.9786	0.3127	0.9505	0.4706	0.9435
KNN	0.9991	0.9611	0.9463	0.9537	0.9491
RF	0.9988	0.9795	0.8983	0.9371	0.9004
XGBoost	**0.9992**	**0.9824**	0.9324	0.9567	0.9375
SVM	0.9982	0.8605	**0.9756**	0.9144	**0.9733**
CNN	**0.9992**	0.9720	0.9449	**0.9583**	0.9448

TABLE VII: Performance of Different Models on Imbalanced Dataset

b) Performance with SMOTE: Subsequently, we explore the impact of the Synthetic Minority Over-sampling Technique (SMOTE) on model performance. SMOTE is designed to artificially increase the presence of the minority class by creating synthetic examples rather than by oversampling with replacement. The enhanced results, depicted in Table 3, indicate the effectiveness of SMOTE in improving model recall and overall accuracy.

Model	Accuracy	Precision	Recall	F1-score	Savings
LR+SMO	0.9908	0.5230	0.9345	0.6707	0.9341
KNN+SMO	0.9982	0.8611	**0.9763**	0.9151	**0.9769**
RF+SMO	0.9990	0.9600	0.9366	0.9481	0.9375
XGB+SMO	0.9991	0.9485	0.9617	0.9550	0.9601
SVM+SMO	0.9985	0.8922	0.9631	0.9263	0.9622
CNN+SMO	**0.9993**	**0.9523**	0.9742	**0.9631**	0.9710

TABLE VIII: Performance of Different Models on Imbalanced Dataset with SMOTE

c) Performance with Tomek Links: We also examine the effect of using Tomek Links, a technique for undersampling by removing overlapping examples between classes. This approach aims to clarify the boundaries between classes, potentially leading to better model generalization on unseen data. The results are detailed in Table 4, showcasing how each model's performance is tweaked when the class overlap is minimized.

d) Performance with SMOTE and Tomek Links: Lastly, we consider the combination of SMOTE and Tomek Links, an approach that integrates the benefits of both oversampling and undersampling. This hybrid method not only augments the minority class through synthetic samples but also enhances class separation by removing Tomek Links. The comparative results, presented in Table 5, illustrate the synergistic effects of this combined approach on the models' performance metrics.

Model	Accuracy	Precision	Recall	F1-score	Savings
LR+TL	0.9784	0.3105	0.9505	0.4681	0.9434
KNN+TL	0.9991	0.9604	0.9463	0.9533	0.9491
RF+TL	0.9988	0.9809	0.8962	0.9366	0.8988
XGB+TL	0.9991	**0.9822**	0.9254	0.9530	0.9273
SVM+TL	0.9981	0.8578	**0.9756**	0.9129	**0.9733**
CNN+TL	**0.9993**	0.9663	0.9596	**0.9629**	0.9629

TABLE IX: Performance of Different Models on Imbalanced Dataset with Tomek Links

Model	Accuracy	Precision	Recall	F1-score	Savings
LR+SMO-TL	0.9908	0.5224	0.9345	0.6702	0.9340
KNN+SMO-TL	0.9982	0.8616	**0.9763**	0.9154	**0.9769**
RF+SMO-TL	0.9990	0.9607	0.9366	0.9485	0.9388
XGB+SMO-TL	0.9991	0.9528	0.9568	0.9548	0.9589
SVM+SMO-TL	0.9985	0.8922	0.9631	0.9263	0.9622
CNN+SMO-TL	**0.9994**	**0.9719**	0.9638	**0.9678**	0.9601

TABLE X: Performance of Different Models on Imbalanced Dataset with SMOTE-Tomek Links

Each of these scenarios highlights different aspects of the models' capabilities and limitations, providing valuable insights into the optimal strategies for handling imbalanced datasets in various machine learning tasks.

3) Discussion: The exploration of different resampling techniques demonstrates their varied impacts on machine learning model performances, particularly reflected in metrics like accuracy, precision, recall, F1-score, and savings. SMOTE typically enhances recall, critical for detecting the minority class in our imbalanced dataset of credit card transactions. However, it may also introduce synthetic noise, potentially leading to overfitting. Tomek Links improve precision by removing overlapping samples between classes, refining the decision boundaries.

The combination of SMOTE and Tomek Links balances recall and precision, showcasing the effectiveness of integrated resampling approaches in complex datasets.

a) SVM Performance: The SVM model's performance fluctuation with oversampling techniques, including declines in precision, highlights its sensitivity to synthetic sample addition. The inherent properties of SVM can be adversely affected by new, synthetic data points, disrupting its optimization landscape and potentially impairing its generalization capabilities.

Model	Accuracy	Precision	Recall	F1-score	Savings
SVM	0.9982	0.8605	0.9756	0.9144	0.9733
SVM+SMO	0.9985	0.8922	0.9631	0.9263	0.9622
SVM+TL	0.9981	0.8578	0.9756	0.9129	0.9733
SVM+SMO-TL	0.9985	0.8922	0.9631	0.9263	0.9622

TABLE XI: Performance of different resampling methods on SVM

b) KNN's Savings and Precision: Despite KNN's high savings scores, its precision remains relatively low, indicating a high rate of false positives. This suggests that while KNN effectively identifies fraudulent transactions, it also mistakenly classifies legitimate transactions as fraudulent. This trait can lead to increased operational costs in real-world fraud detection systems, where managing false alarms is crucial to maintaining system efficiency and cost-effectiveness.

Model	Accuracy	Precision	Recall	F1-score	Savings
KNN	0.9991	0.9611	0.9463	0.9537	0.9491
KNN+SMO	0.9982	0.8611	**0.9763**	0.9151	0.9769
KNN+TL	0.9991	0.9604	0.9463	0.9533	0.9491
KNN+SMO-TL	0.9982	0.8616	**0.9763**	0.9154	0.9769

TABLE XII: Performance of different resampling methods on KNN

c) CNN's Robust Performance: CNNs exhibit substantial improvements across all evaluated metrics, benefitting from their deep architectures and convolutional layers capable of robust pattern recognition. This makes them particularly adept at handling complex features in imbalanced datasets, illustrating the importance of aligning model architecture with dataset characteristics.

Model	Accuracy	Precision	Recall	F1-score	Savings
CNN	0.9992	0.9720	0.9449	0.9583	0.9448
CNN+SMO	**0.9993**	0.9523	0.9742	**0.9631**	0.9710
CNN+TL	**0.9993**	0.9663	0.9596	0.9629	0.9629
CNN+SMO-TL	**0.9994**	0.9719	0.9638	0.9678	0.9601

TABLE XIII: Performance of different resampling methods on CNN

d) Correlation Between Savings and Recall: The correlation observed between savings and recall suggests that models with higher recall rates lead to greater savings, as more fraudulent transactions are correctly identified. However, the real-world complexities of fraud detection systems introduce various costs associated with false positives, which are not fully captured by the current savings metric. These costs include time, human resources, and customer complaints, among others, suggesting that our simplistic assumption of a constant administrative cost (C_a) may not adequately reflect real-world scenarios.

e) Optimizing the Savings Metric: To better represent comprehensive model performance in operational settings, adjusting the savings metric to reflect the complex costs associated with false positives is recommended. This could involve incorporating variable costs based on factors such as time, human resources, and customer impact into the savings formula, providing a more nuanced evaluation of the economic benefits and drawbacks of fraud detection models.

This discussion underscores the significant effects of resampling techniques on model performance within imbalanced datasets, particularly in fraud detection contexts. Our findings enhance understanding of model selection and resampling strategy effectiveness, contributing to advancements in fraud detection methodologies and offering insights into optimizing economic impacts in practical applications.

VI. Real-World Explorations

A. Reinforcement Learning

Reinforcement Learning (RL) [19] constitutes a significant paradigm of Machine Learning (ML), a discipline wherein

agents learn to make decisions through trial-and-error interactions with a dynamic environment. This computational approach to learning from action enables software agents to determine an optimal behavioral strategy, typically in pursuit of a specific long-term goal.

In RL, an agent learns by iteratively making decisions, receiving feedback in the form of rewards or punishments, and refining its actions accordingly. This learning process is inspired by the cognitive processes observed in sentient beings—adapting actions based on the outcomes of past experiences in order to achieve a desired objective.

Fundamentally, RL involves two primary components that interact within a feedback loop:

1) **Agent:** The agent represents the decision-maker, which perceives the state of the environment and decides upon an action to take. The action selection process is informed by a policy—a mapping from perceived states of the environment to actions—to maximize the expected sum of future rewards.
2) **Environment:** The environment encapsulates the context within which the agent operates. It responds to the agent's actions and provides feedback that guides the learning process. The feedback typically takes the form of rewards, signaling the agent's proximity to the goal.

B. Problem Definition and Modeling Environment

Given Dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where $x_i$represents the feature vector for the i^{th} transaction and y_i indicates the corresponding label of fraudulent activity. Here, a fraud transaction constitutes the positive class, denoted by $y = 1$. The dataset is temporally ordered to maintain the sequential nature of transactions, thereby casting the fraud detection task into a Sequential Decision Making framework. At any given time step t, the agent is presented with a transaction x_t, and must decide whether to authorize (action $a_t = 0$) or decline the transaction (action $a_t = 1$). The environment then dispenses a reward contingent on the current decision's efficacy and reveals the subsequent transaction x_{t+1}. The agent's objective is to discern transaction authenticity in a manner that optimizes utility, aiming to mitigate substantial fiscal losses attributable to fraudulent transactions while also approving legitimate transactions. Concurrently, the agent strives to maintain an optimal trade-off between the fraud rate (fr) and the decline rate (dr), operationalized through the reward function R.

C. Modeling Environment

A Markov Decision Process (MDP) is employed to formulate the environment for a fraud detection task, which is structured as a finite-horizon, episodic task within the reinforcement learning framework. The environment, encapsulating the dynamics of a fraud detection system, can be described by the following components:

1) **States (S):** The state space S in our MDP is defined by the observations from the dataset. Each state corresponds to a transaction characterized by a vector of features, excluding the 'id' field, as it does not provide discriminative information for fraud detection.
2) **Actions (A):** The action space A comprises two discrete actions: 0: 'not_fraud', 1: 'fraud'. An agent can decide whether to label a transaction as fraudulent or not, corresponding to the binary decision required in fraud detection tasks.
3) **Rewards (R):**The reward function $R(s, a)$ provides immediate feedback to the agent after taking an action a in state s. The rewards are structured to reflect the consequences of correct or incorrect fraud detection. For example, correctly identifying a fraudulent transaction yields a high positive reward, while false predictions incur penalties to varying degrees, depending on their impact.
4) **Transitions (T):** The transition probabilities $T(s'|s, a)$, describe the likelihood of moving from state s to state s' given an action a. In our fraud detection MDP, the transition to the next state is deterministic, as the states follow a sequential order determined by the dataset's indices.

D. Agent

In the context of fraud detection, the objective of a reinforcement learning (RL) agent is to determine an optimal policy π^* that maximizes the sum of the cumulative rewards G_t, where G_t is given by the equation

$$G_t = \sum_{m=0}^{\infty} \gamma^m r_{t+m}. \tag{29}$$

The Deep Q-Network (DQN)[20] agent employs a neural network architecture to approximate the optimal action-value function, as described by

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi], \tag{30}$$

which yields the maximal expected return of rewards discounted by γ at each timestep t, under the policy $\pi = P(a|s)$.

The architecture of the Deep Q-Network (DQN) employed in our study is a neural network designed for approximating the Q-values for each action given a particular state. The network is defined by a sequence of fully connected layers with the following structure:

- The first layer, `fc1`, takes an input of size 30, corresponding to the dimensionality of the state space, and maps it to a hidden layer of size 16.
- The second layer, `fc2`, connects the 16 nodes from the previous layer to 18 nodes.
- Subsequently, `fc3` expands the network from 18 to 20 nodes.
- The fourth layer, `fc4`, further increases the complexity by mapping these 20 nodes to 24 nodes.
- Finally, the output layer `fc5` reduces the dimensionality to 2, corresponding to the estimated Q-values for the two possible actions in the binary decision-making problem of fraud detection.

Each layer is followed by a Rectified Linear Unit (ReLU) activation function, except for the last layer, which uses a sigmoid activation due to the binary nature of the output. Additionally, dropout regularization with a probability of $p = 0.25$ is employed after the second layer to prevent overfitting.

The DQN's loss function at iteration i is represented by

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right], \quad (31)$$

where θ_i are the Q-network parameters at iteration i, and θ_i^- are the target network parameters.

The action space is discrete, with DQN updates performed on mini-batches sampled uniformly from the replay buffer. Experience replay allows the agent to learn from past actions, ensuring a diverse range of experiences is considered during training.

The Q-learning updates take place on mini batches (batch size = 64) drawn uniformly at random from the memory pool.The agent is trained using an ϵ-greedy policy, where ϵ is annealed linearly from 1 to 0.01 using a decay rate δ.

E. Reward Function

In the reinforcement learning framework, the reward function is paramount for guiding the agent towards beneficial actions. Two reward functions are defined to quantify the immediate feedback provided to the agent upon taking an action in response to a state that represents a financial transaction.

The first reward function R^1 is designed with fixed rewards and penalties:

$$R^1(s_t, a_t, y_t) = \begin{cases} 1 & \text{if } a_t = 0 \text{ and } y_t = 0, \\ -10 & \text{if } a_t = 1 \text{ and } y_t = 0, \\ -1000 & \text{if } a_t = 0 \text{ and } y_t = 1, \\ 100 & \text{if } a_t = 1 \text{ and } y_t = 1. \end{cases} \quad (32)$$

- a_t: The action taken by the agent at time step t, which, in the case of a binary classification task, can be either 0 (representing a positive action such as approving a transaction) or 1 (representing a negative action such as declining a transaction).
- y_t: The ground truth label for the outcome at time step t, which in the context of fraud detection is 0 for a non-fraudulent transaction and 1 for a fraudulent transaction.

The second reward function R^2 incorporates a logarithmic scale influenced by the transaction amount, while also integrating an adjustable parameter α, which scales the reward and penalty dynamically:

$$R^2(s_t, a_t, y_t) = \begin{cases} \alpha \log(\text{amount}) & \text{if } a_t = 0 \text{ and } y_t = 0, \\ -\log(\text{amount}) & \text{if } a_t = 1 \text{ and } y_t = 0, \\ -\alpha \log(\text{amount}) & \text{if } a_t = 0 \text{ and } y_t = 1, \\ \log(\text{amount}) & \text{if } a_t = 1 \text{ and } y_t = 1. \end{cases} \quad (33)$$

- α: A scaling factor used in the reward function to adjust the magnitude of the reward or penalty. It serves to amplify the importance of certain outcomes and, thereby, guide the agent's learning process more effectively.

The first approach offers simplicity and clear distinctions between different outcomes. In contrast, the second approach provides a nuanced and transaction amount-dependent reward structure, which may better reflect the varied financial stakes inherent in fraud detection.

In both cases, the overarching goal is to optimize the agent's policy to maximize cumulative rewards, thereby ensuring fraudulent transactions are minimized and legitimate transactions are processed correctly.

F. Result

When examining the impact of two distinct reward functions in a reinforcement learning scenario for fraud detection, attention is particularly drawn to the classification of fraudulent transactions, denoted as Class=1. Performance metrics such as Accuracy, Precision, Recall, and F1-Score are critical for evaluating the efficacy of the learning agent.

Under the first reward function, the agent demonstrates a remarkable Recall for fraudulent transactions, indicating a propensity towards identifying potential fraud. However, this comes at the expense of Precision, suggesting many legitimate transactions are also flagged erroneously as fraud, leading to numerous false positives.

Conversely, the second reward function shows an increase in Precision, implying that when the agent predicts a transaction to be fraudulent, it is more likely to be correct. Nonetheless, the Recall metric experiences a decline, signaling that the agent now misses a larger proportion of fraudulent transactions.

Here is a succinct representation of the comparison:

Reward Function	Accuracy	Precision	Recall	F1-Score
First Reward Function	0.1120	0.0103	**0.9018**	0.0203
Second Reward Function	**0.8914**	0.0124	0.1228	0.0225

TABLE XIV: Performance metrics for Class=1 under two reward functions

From the above comparison, it is evident that the design of the reward function significantly influences the agent's behavior. The first reward function might be overly punitive towards misclassifications, leading to a cautious strategy. This conservative policy, despite achieving a high Recall, results in a lower Precision, which is undesirable in a real-world scenario due to the operational implications of false positives.

The second reward function, informed by the transaction amounts, prompts a more discerning approach, potentially decreasing the cost associated with the investigation of false alarms, as indicated by improved Precision. However, the lower Recall suggests that the agent has become too conservative, overlooking a considerable number of fraudulent transactions.

The challenge lies in calibrating the reward function to achieve a balanced outcome, where both Precision and Recall

are optimized. This may also indicate that reinforcement learning may not be suitable for such a categorization problem.

VII. Summary

Our work systematically addresses the pervasive issue of credit card fraud by deploying a series of methodical steps, each contributing to the development of an effective fraud detection system using machine learning. Initially, we defined the scope and impact of credit card fraud, setting the stage for a targeted analytical approach. Subsequent data preparation involved meticulous cleaning and preparation, ensuring high-quality inputs for model training and evaluation.

Visualization techniques were then employed to explore data distributions and correlations, providing insights that guided the feature selection and engineering phase. This phase was critical in enhancing the model's predictive power by identifying and engineering key features that significantly influence fraud detection outcomes.

The core of our study involved the development and iterative refinement of multiple machine learning models. Through rigorous model tuning, we optimized parameters to achieve optimal performance, particularly focusing on handling imbalanced data with techniques such as SMOTE and Tomek Links to ensure robustness and fairness.

Our models were then tested in real-world settings to evaluate their practical viability and ethical implications, ensuring that they not only perform well statistically but also adhere to ethical standards in deployment. Lastly, we explored advanced techniques like reinforcement learning, paving the way for future enhancements that could offer even more sophisticated solutions to credit card fraud detection.

Overall, our research demonstrates a comprehensive approach to using machine learning in combating credit card fraud, highlighting significant improvements in detection capabilities and offering a roadmap for future innovations in this critical field.

VIII. Contributions

A. Chong Chen

- Implemented models of Logistic Regression, SVM, and Neural Networks, including introduction and performance testing.
- Applied feature engineering techniques of Polynomial Features, Standard Scaler, and Min-Max Scaler to enhance the Logistic Regression model's performance.
- Developed a novel Savings metric for evaluating model cost efficiency and created imbalanced datasets to simulate real-world conditions.
- Integrated resampling techniques like SMOTE and Tomek Links to imbalanced datasets, applied across models, and evaluated and discuss their impacts through detailed analysis.

B. Di Liu

- Visualize data, perform preliminary data analysis, and decide on data processing methods.
- Implement KNN, decision tree, and random forest models, including an introduction and performance tests.
- Used PCA to process data for dimensionality reduction and adjusted hyperparameters to optimize the model and get stronger model prediction ability
- Implemented data prediction experiments using DQN Reinforcement Learning Neural Networks, designed two reward formulas, analyzed and evaluated them in detail and explored their significance.

References

[1] Arthur O'sullivan, Steven M Sheffrin, and Kathy Swan. Economics: Principles in action. 2003.

[2] Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (pca). *Computers Geosciences*, 19(3):303–342, 1993. ISSN 0098-3004. doi: https://doi.org/10.1016/0098-3004(93)90090-R. URL https://www.sciencedirect.com/science/article/pii/009830049390090R.

[3] David G Kleinbaum, K Dietz, M Gail, Mitchel Klein, and Mitchell Klein. *Logistic regression*. Springer, 2002.

[4] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4): 18–28, 1998.

[5] Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. Knn model-based approach in classification. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings*, pages 986–996. Springer, 2003.

[6] Yan-Yan Song and LU Ying. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130, 2015.

[7] Steven J Rigatti. Random forest. *Journal of Insurance Medicine*, 47(1):31–39, 2017.

[8] Herve Abdi. A neural network primer. *Journal of Biological Systems*, 2(03):247–281, 1994.

[9] Keiron O'shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[10] Jingsai Liang. Confusion matrix: Machine learning. *POGIL Activity Clearinghouse*, 3(4), 2022.

[11] Bradley J Erickson and Felipe Kitamura. Magician's corner: 9. performance metrics for machine learning models, 2021.

[12] C Reid Turner, Alfonso Fuggetta, Luigi Lavazza, and Alexander L Wolf. A conceptual basis for feature engineering. *Journal of Systems and Software*, 49(1): 3–15, 1999.

[13] Xue-wen Chen and Jong Cheol Jeong. Enhanced recursive feature elimination. In *Sixth international conference on machine learning and applications (ICMLA 2007)*, pages 429–435. IEEE, 2007.

[14] Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, 14(5):503–519, 2017.

[15] Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (pca). *Computers & Geosciences*, 19(3):303–342, 1993.

[16] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[17] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *Icml*, volume 97, page 179. Citeseer, 1997.

[18] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1):20–29, 2004.

[19] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[20] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.