INFORMATION ASSURANCE COSC - 6375.001
CREDIT CARD FRAUD DETECTION

ABSTRACT

In the modern digital age, credit card fraud poses a serious risk to financial organizations as well as customers or individuals. To improve the efficiency of fraud detection systems, this study offers a machine learning-based method for credit card fraud detection. The proposed methodology involves a systematic process encompassing data acquisition, preprocessing, model selection, and evaluation. Using a comprehensive dataset containing historical credit card transactions, we apply rigorous data preprocessing techniques to ensure data cleanliness and suitability for model training. A wide range of machine learning methods, such as Random Forest, Gradient Boosting, Linear SVC, Neural Networks, and Logistic Regression, are tested to select a model. The performance of the model is evaluated using a variety of evaluation criteria. The results demonstrate the effectiveness of the proposed approach in detecting and mitigating credit card fraud, ultimately contributing towards enhancing financial security and protecting against fraudulent activities in credit card transactions.

INTRODUCTION

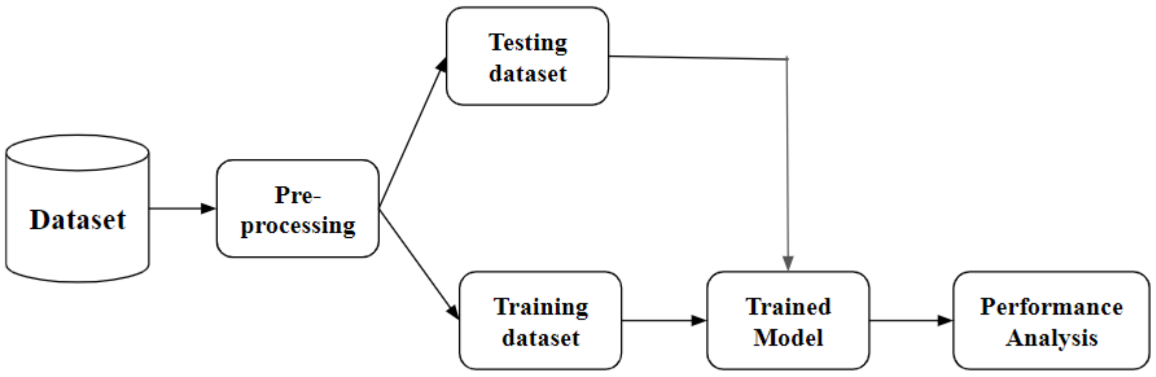
Over the last ten years, the Internet has expanded at an exponential rate. As a result, services like online bill payment systems and e-commerce have grown and are being used more often. Fraudsters have therefore increased their attempts to target credit card transactions. Tokenization and credit card data encryption are traditional methods of credit card transaction security. While these methods are beneficial in most situations, they don't offer total security against credit card fraud.

Detecting and preventing credit card fraud is essential to safeguarding the integrity of financial systems and ensuring the security of individuals' funds. Traditional approaches for detecting fraud frequently depend on laborious manual review procedures and rule-based algorithms, which may fail to recognize complex fraudulent schemes. In contrast, machine learning offers a powerful and automated approach to detecting fraudulent activities by analyzing patterns and anomalies in transaction data.

DATASET

The data set includes cardholder transactions done over two days in September 2013. There are a total of 284,807 transactions, of which 492 (0.172%) are fraudulent transactions. This dataset is quite imbalanced. To maintain client confidentiality, the dataset's characteristics are modified using principal component analysis (PCA). The characteristics V1, V2, V3,..., and V28 are PCA-applied, whereas 'time', 'amount', and 'class' are not. “Time” contains the elapsed seconds between the first and other transactions of each attribute.“Amount” is the amount of the transaction and “ Class” is the binary variable where “1” is known as a fraudulent transaction and “0” is not a fraudulent transaction.

PROPOSED MODEL



The machine learning-based credit card fraud detection methodology that has been suggested takes a methodical approach to utilizing the embedded powers of machine learning algorithms to efficiently identify and reduce fraudulent activity in credit card transactions. The procedure begins with acquiring and understanding a comprehensive dataset containing historical credit card transactions, comprising essential features such as transaction amount, time, and the binary target variable indicating fraudulent or non-fraudulent transactions. Subsequently, rigid data preprocessing techniques are applied to ensure data cleanliness and suitability for model training. This includes handling missing values through imputation or deletion strategies, detecting and treating outliers, scaling numerical features, and encoding categorical features. Following data preprocessing, a diverse set of machine learning algorithms, including Logistic Regression, Random Forest, Gradient Boosting, Linear SVC, and Neural Networks, are considered for model selection. The dataset is split into training, validation, and testing sets to facilitate model training, hyperparameter tuning, and evaluation. Model performance is measured using a range of evaluation metrics, including accuracy, precision, recall, and F1-score, to ensure reliability and generalization capability.

IMPLEMENTATION

1. Data Preprocessing
   This plays a pivotal role in preparing the credit card transaction dataset for effective machine learning model training. The dataset undergoes several preprocessing steps to ensure data quality and suitability for analysis. Firstly, to reduce any potential biases caused by incomplete data, missing values are handled using imputation or deletion techniques. Outliers, which could distort analysis and model performance, are detected and treated using statistical methods or domain knowledge. Numerical features such as transaction amounts are scaled to a consistent range to prevent features with larger magnitudes from dominating the model training process. The RobustScaler is used to scale the 'Amount' column of the DataFrame (df). It standardizes the feature by removing the median and scaling per the interquartile range. Min-max normalization is used to normalize the DataFrame's 'Time' column. By doing this, it is ensured that the values of 'Time' are scaled to fall between 0 and 1.

```
from sklearn.preprocessing import RobustScaler
new_df = df.copy()
new_df['Amount'] = RobustScaler().fit_transform(new_df['Amount'].to_numpy().reshape(-1, 1))
time = new_df['Time']
new_df['Time'] = (time - time.min()) / (time.max() - time.min())
new_df
```

2. Balancing Dataset
   Addressing class imbalance is an essential step in credit card fraud detection, as fraudulent transactions often constitute a minority class compared to legitimate ones. In this phase, we handle class imbalance by balancing the dataset to ensure that the machine learning models are trained on a representative sample of both fraudulent and non-fraudulent transactions. To balance the distribution of instances between the minority class (fraudulent transactions) and the majority class (non-fraudulent transactions), we employ a resampling technique by randomly selecting a subset of instances from the majority class. By creating a balanced dataset with equal representation of both classes, we prevent the model from being biased towards the majority class and improve its ability to accurately predict fraudulent activities. This balanced dataset is then used for subsequent model training, validation, and testing, thereby enhancing the reliability and effectiveness of the credit card fraud detection system.

```
not_frauds = new_df.query('Class == 0')
frauds = new_df.query('Class == 1')
not_frauds['Class'].value_counts(), frauds['Class'].value_counts()
```

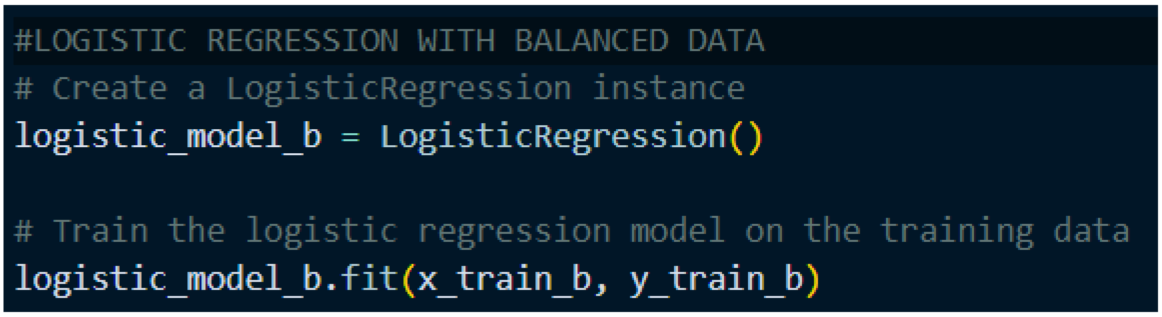
```
(Class
 0    284315
 Name: count, dtype: int64,
 Class
 1    492
 Name: count, dtype: int64)
```

```
balanced_df = pd.concat([frauds, not_frauds.sample(len(frauds), random_state=1)])
balanced_df['Class'].value_counts()
```

```
Class
1    492
0    492
Name: count, dtype: int64
```

3. Model Training
   a. Logistic Regression
      In addressing classification challenges, especially those involving two categories, logistic regression emerges as a prevalent and classical approach. The principle of favoring simplicity over complexity guides the selection of this algorithm. Logistic regression stands out as a well-established statistical

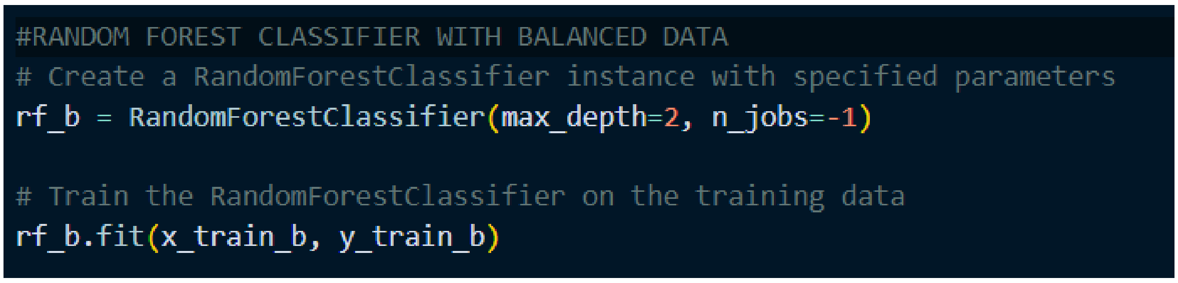
method for forecasting binomial or polynomial outcomes. It is particularly effective when dealing with a categorical field containing multiple potential values, leading to enhanced performance of the classification algorithm.



```
#LOGISTIC REGRESSION WITH BALANCED DATA
# Create a LogisticRegression instance
logistic_model_b = LogisticRegression()

# Train the logistic regression model on the training data
logistic_model_b.fit(x_train_b, y_train_b)
```

b. Random Forest Classifier
The Random Forest Classifier stands as a robust ensemble learning method frequently applied in machine learning for classification purposes. Its operation involves the creation of numerous decision trees during the training phase, with the final output being the mode of the classes from the individual trees in the forest, particularly for classification tasks. Upon instantiation, the RandomForestClassifier incorporates specific hyperparameters. In this instance, the parameter max_depth=2 imposes a limit on the maximum depth of each tree within the forest, a measure aimed at mitigating overfitting. Additionally, the n_jobs=-1 parameter specifies that the algorithm should utilize all available CPU cores for concurrent processing.

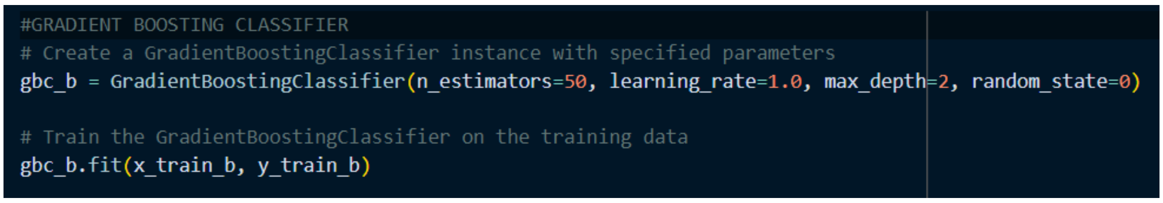


```
#RANDOM FOREST CLASSIFIER WITH BALANCED DATA
# Create a RandomForestClassifier instance with specified parameters
rf_b = RandomForestClassifier(max_depth=2, n_jobs=-1)

# Train the RandomForestClassifier on the training data
rf_b.fit(x_train_b, y_train_b)
```

c. Gradient Boosting Classifier
The Gradient Boosting Classifier represents another form of ensemble learning technique that constructs decision trees sequentially, with each subsequent tree aimed at rectifying the errors of its predecessor. This approach often yields heightened predictive accuracy by emphasizing challenging-to-classify instances. The n_estimators parameter determines the number of boosting stages, corresponding to the count of weak learners or decision trees incorporated into the ensemble. Meanwhile, the learning_rate parameter regulates the influence of each weak learner within the ensemble; a higher learning rate signifies a greater contribution from each tree toward the final prediction. Additionally, the max_depth parameter is set to 1, dictating the maximum depth of the individual decision trees within the ensemble. By restricting it to 1, the complexity of the trees is constrained, thereby guarding

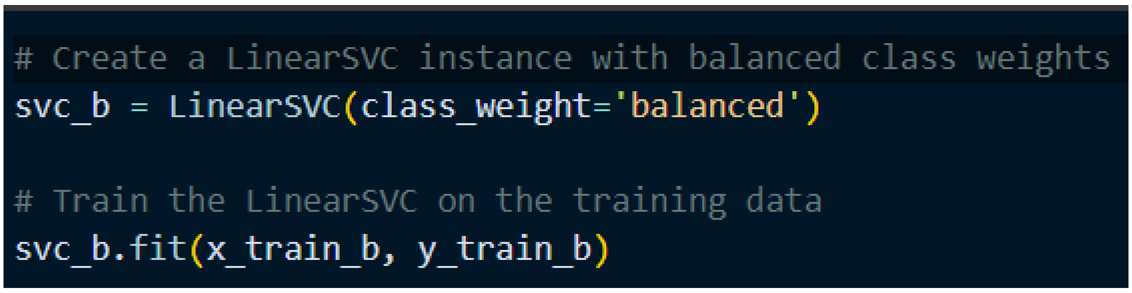
against overfitting and enhancing the model's capacity for generalization.



```
#GRADIENT BOOSTING CLASSIFIER
# Create a GradientBoostingClassifier instance with specified parameters
gbc_b = GradientBoostingClassifier(n_estimators=50, learning_rate=1.0, max_depth=2, random_state=0)

# Train the GradientBoostingClassifier on the training data
gbc_b.fit(x_train_b, y_train_b)
```

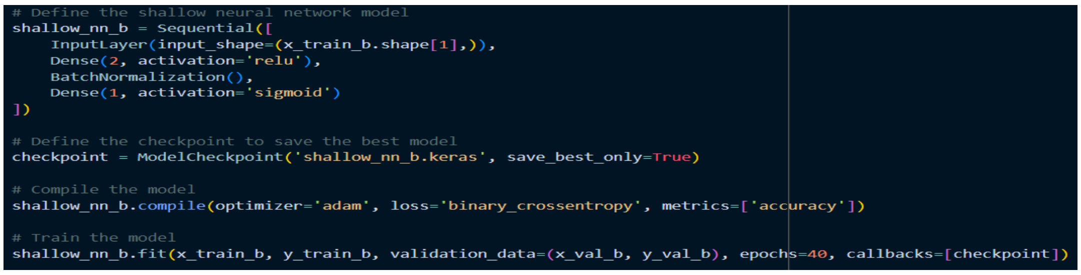
d. Linear SVC
LinearSVC is a Support Vector Machine (SVM) classifier that uses a linear kernel. This classifier is initialized with parameter class_weight='balanced'. This parameter tells the classifier to automatically adjust the class weights inversely proportional to class frequencies in the input data



```
# Create a LinearSVC instance with balanced class weights
svc_b = LinearSVC(class_weight='balanced')

# Train the LinearSVC on the training data
svc_b.fit(x_train_b, y_train_b)
```

e. Shallow Neural Networks
Shallow neural networks typically denote neural networks comprising only a limited number of hidden layers, distinguishing them as simpler models in contrast to deep neural networks (DNNs). These networks typically consist of an input layer, one or more hidden layers, and an output layer. They find application in tasks such as binary classification, regression, or straightforward pattern recognition, particularly suited for datasets featuring a moderate number of features and relatively uncomplicated decision boundaries. To construct a shallow neural network, a hidden layer is appended using Dense with 2 units (neurons) and ReLU activation, facilitating the processing of input data while employing the rectified linear activation function. Furthermore, the inclusion of a Batch normalization layer serves to standardize the activations from the preceding layer, thereby enhancing training stability and speed. Finally, the output layer is introduced using Dense with 1 unit and sigmoid activation, thereby yielding a probability score conducive to binary classification (fraudulent or non-fraudulent).



```
# Define the shallow neural network model
shallow_nn_b = Sequential([
    InputLayer(input_shape=(x_train_b.shape[1],)),
    Dense(2, activation='relu'),
    BatchNormalization(),
    Dense(1, activation='sigmoid')
])

# Define the checkpoint to save the best model
checkpoint = ModelCheckpoint('shallow_nn_b.keras', save_best_only=True)

# Compile the model
shallow_nn_b.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
shallow_nn_b.fit(x_train_b, y_train_b, validation_data=(x_val_b, y_val_b), epochs=40, callbacks=[checkpoint])
```

EVALUATION METRICS

The classification report serves as a pivotal tool in evaluating the performance of a model by providing detailed insights into its classification accuracy. It delineates the counts of correctly and incorrectly classified instances across different classes, thereby elucidating the model's predictive efficacy. Accuracy, a fundamental metric, quantifies the proportion of correctly predicted outputs relative to the total number of instances. Precision, on the other hand, illuminates the precision of the model by delineating the number of correctly classified instances out of those predicted as positive, thereby encapsulating the model's exactness. Recall, also known as sensitivity, gauges the model's ability to identify true positives accurately, depicting its efficacy in capturing relevant instances from the positive class.

Metrics / Models	Accuracy	Precision		Recall	
		Not Fraud	Fraud	Not Fraud	Fraud
Logistic Regression	94%	0.96	0.93	0.93	0.96
Random Forest	95%	0.93	0.97	0.97	0.93
Gradient Boosting	93%	0.94	0.92	0.92	0.94
Linear SVC	94%	0.96	0.93	0.93	0.96
Shallow Neural Network	93%	0.96	0.91	0.90	0.96

RESULTS AND CONCLUSION

Based on the metrics obtained from the various models used in the project, it can be concluded that each model performed reasonably well in fraud transactions. The metrics used for evaluation, including accuracy, precision, and recall, provide insights into the performance of each model.

The Logistic Regression model achieved an accuracy of 94%, indicating that it correctly classified 94% of the transactions in the validation dataset. It exhibited high precision and recall values for both the "Not Fraud" and "Fraud" classes, demonstrating its ability to accurately identify both types of transactions.

Similarly, the Random Forest model achieved a slightly higher accuracy of 95% and showed strong precision and recall values for both classes. This ensemble method proved to be effective in capturing the complexities of the data and making accurate predictions.

The Gradient Boosting model, although slightly lower in accuracy compared to the Logistic Regression and Random Forest models, still performed well with an accuracy of 93%. It exhibited balanced precision and recall values for both classes, indicating its robustness in handling imbalanced datasets.

The Linear SVC model also achieved an accuracy of 94% and demonstrated high precision and recall values for both classes. Its performance was comparable to that of the Logistic Regression model, showcasing its effectiveness in classification tasks.

Lastly, the Shallow Neural Network model achieved an accuracy of 93%, which is slightly lower than the other models. While it exhibited high precision for the "Not Fraud" class, its recall for the "Fraud" class was comparatively lower, indicating a potential area for improvement.

In conclusion, all the models performed well in detecting credit card fraud, with each having its strengths and weaknesses. The choice of model may depend on various factors such as computational resources, interpretability, and the specific requirements of the application. Overall, the project demonstrates the efficacy of machine learning techniques in addressing the challenges of fraud detection in financial transactions.