

# Credit Card Fraud Detection

Jiban-Ul Azam Chowdhury Shafin

*Electronic Engineering*

*Hochschule Hamm-Lippstadt*

Lippstadt, Germany

jiban-ul-azam-chowdhury.shafin@stud.hshl.de

**Abstract**—Credit card fraud detection is a challenging task due to extreme class imbalance, where fraudulent transactions are rare. This paper applies the Isolation Forest algorithm to detect such anomalies without supervised labels. Using Scikit-learn’s IsolationForest and StandardScaler, the model was trained on a real-world dataset of European credit card transactions. Anomaly scores based on path lengths were computed, and model performance was evaluated using precision, recall, F1 score, and accuracy. Results show that Isolation Forest achieves high recall and provides an efficient unsupervised approach for large-scale fraud detection.

## I. INTRODUCTION

Credit card fraud detection has become increasingly critical as the volume of digital transactions continues to grow rapidly. Modern fraud schemes are highly adaptive, making traditional rule-based systems insufficient to detect sophisticated attacks. Financial institutions are under constant pressure to implement advanced analytics capable of detecting fraudulent behavior with minimal disruption to legitimate customers. [1]

One of the main challenges is the highly imbalanced nature of fraud data—fraudulent transactions represent a tiny fraction of overall activity, making accurate detection a “needle-in-a-haystack” problem. In addition, fraudsters continuously evolve their tactics, requiring detection models that are flexible and robust to unseen patterns. [1]

Anomaly detection approaches, such as Isolation Forest, offer a promising solution by identifying rare and unusual behavior without relying on labeled data. These models isolate anomalies by exploiting their statistical differences from normal data, making them well-suited for dynamic environments where fraud patterns shift over time. [1]

The objective of this paper is to explore the application of Isolation Forest for unsupervised credit card fraud detection. By employing the well-known Credit Card Fraud Detection dataset, preprocess the data using Scikit-Learn’s StandardScaler, and train an Isolation Forest model to detect suspicious transactions. The performance is then evaluated using standard classification metrics to assess the model’s practical utility. [1]

## II. NECESSARY MATHEMATICAL BASICS

**Path Length in Binary Tree.** The path length  $h(x)$  of a data point  $x$  is defined as the number of edges traversed from the root node to the external node where  $x$  ends up during traversal in a binary tree. In the context of Isolation Forest,

data points that are easier to isolate (anomalies) tend to have shorter path lengths, while normal points have longer paths. [2]

**Harmonic Number.** The harmonic number  $H(i)$  is mathematically defined as:

$$H(i) = \sum_{k=1}^i \frac{1}{k}$$

It represents the expected path length of unsuccessful searches in a Binary Search Tree and is used in computing the normalization term  $c(n)$  in the anomaly score formula. [2]

**Logarithmic Component.** The term  $c(n)$ , used to normalize the average path length  $E(h(x))$ , is defined as:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n}$$

where  $n$  is the sub-sampling size, and  $H(n-1)$  is the harmonic number. The logarithmic growth of the harmonic number relates to the logarithmic behavior of Binary Search Tree height, reflecting the theoretical behavior of isolation processes in high-dimensional data [2].

## III. WHAT IS CREDIT CARD FRAUD DETECTION?

Credit card fraud detection is the process of identifying unauthorized or fraudulent credit card transactions among a large number of legitimate transactions. Credit card fraud typically involves the misuse of another person’s credit information or card without their consent. Fraud can take various forms, including application fraud, where false personal information is used to obtain a new card, and behavioral fraud, where the legitimate card details are misused without physically stealing the card. These fraud types are forms of identity theft. [3]

During credit card transactions, many details are logged — up to one hundred or more variables per transaction — such as transaction amount, time, location, merchant ID, and other characteristics. Collecting such transactional data over time results in a dataset that can be used to build fraud detection models. [4]

Fraud detection can be approached as a binary classification problem. [5]

**Dataset:** The dataset can be expressed as  $X = \{x_1, x_2, \dots, x_n\}$ , where each  $x_i$  is a feature vector describing

one transaction. Each transaction also has a corresponding label  $y_i \in \{0, 1\}$ , where  $y_i = 1$  indicates fraud and  $y_i = 0$  indicates a legitimate transaction. [5]

Fraud detection can also be modeled as an anomaly detection problem. In this approach, models are trained to learn what constitutes normal behavior, and transactions that significantly deviate from this behavior are flagged as suspicious. Outlier detection techniques are used to identify such deviations at both the individual transaction level and the overall dataset level. [6]

#### IV. CHALLENGES: IMBALANCE AND COST OF ERRORS

**Imbalance:** A key characteristic of fraud detection is extreme class imbalance, as the number of fraudulent transactions is typically very small compared to the number of legitimate ones — often less than 0.5% of all transactions. This imbalance makes fraud detection a challenging “needle-in-a-haystack” problem, where it is difficult for analytical models to learn the patterns of the minority class (fraud). [7]

**Cost of errors:** The cost of errors in fraud detection is very important to consider. A false negative (failing to detect a fraud) can result in significant financial loss, while a false positive (wrongly flagging a legitimate transaction as fraud) can harm customer experience. Cost-sensitive learning approaches aim to address this by assigning higher misclassification costs to errors that are more critical (such as false negatives). Mathematically, the total misclassification cost is expressed as:

$$\text{Total cost} = C(-, +) \times FN + C(+, -) \times FP$$

where  $FN$  and  $FP$  represent the number of false negatives and false positives, respectively, and  $C(-, +)$ ,  $C(+, -)$  are the corresponding costs. [8]

#### V. EVALUATION OF CREDIT CARD FRAUD DETECTION MODEL

To evaluate a fraud detection model, various performance measures are commonly used. One fundamental tool is the *confusion matrix*, which organizes the model’s predictions against actual outcomes. The matrix contains **True Positives (TP)**, **True Negatives (TN)**, **False Positives (FP)**, and **False Negatives (FN)**. [9]

From this matrix, several key metrics can be mathematically defined:

- **Accuracy** quantifies the proportion of correct predictions:

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

[9]

- **Error Rate** (or Misclassification Rate) is simply the complement of accuracy:

$$\text{Error Rate} = \frac{FP + FN}{TP + FP + FN + TN}$$

[9]

- **Recall** (also called Sensitivity or Hit Rate) captures the proportion of actual fraud cases correctly detected:

$$\text{Recall} = \frac{TP}{TP + FN}$$

[9]

- **Precision** reflects the proportion of predicted fraud cases that are truly fraudulent:

$$\text{Precision} = \frac{TP}{TP + FP}$$

[9]

- **F1-Measure** provides a balanced harmonic mean of Precision and Recall:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

[9]

- **Specificity** measures the ability of the model to correctly classify legitimate transactions:

$$\text{Specificity} = \frac{TN}{FP + TN}$$

[9]

These measures help understand both the strengths and weaknesses of a fraud detection model. In fraud detection, achieving high Recall is important to catch as many fraudulent transactions as possible, but Precision must also be carefully monitored to avoid excessive false alarms. [9]

#### VI. FORMAL DESCRIPTION OF ISOLATION FOREST

Isolation Forest (iForest) is an anomaly detection algorithm that identifies anomalies by explicitly isolating them, rather than profiling normal data points. Anomalies are “few and different,” making them more susceptible to isolation through recursive partitioning. The algorithm builds a collection of random trees, called *Isolation Trees (iTrees)*. [10]

Each tree is constructed by recursively partitioning the data:

- At each node, an attribute  $q$  is selected at random.
- A split value  $p$  is chosen uniformly between the minimum and maximum values of  $q$ .
- The data is split into two subsets accordingly [11].

The process continues until one of the following stopping conditions is met:

- The maximum tree height is reached.
- Only one data point remains.
- All remaining data points have identical values [11].

The path length  $h(x)$  of a data point  $x$  is defined as the number of edges traversed from the root to an external node. Anomalies tend to have shorter path lengths because they can

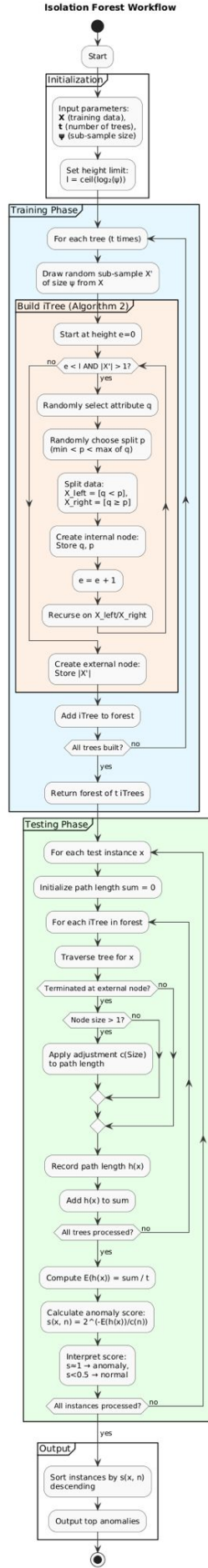


Fig. 1. Isolation Forest Workflow [12]

be isolated with fewer splits, while normal points require more partitions. [11]

The algorithm computes an *anomaly score* for each point:

$$s(x, n) = 2 - \frac{E(h(x))}{c(n)}$$

where:

- $E(h(x))$  is the average path length of  $x$  across the ensemble of iTrees.
- $c(n)$  is the average path length of unsuccessful searches in a Binary Search Tree of size  $n$ , given by:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n}$$

where  $H(i)$  is the  $i$ -th harmonic number. [11]

Interpretation of the score:

- $s(x, n) \rightarrow 1 \Rightarrow$  highly likely to be anomaly.
- $s(x, n) \rightarrow 0 \Rightarrow$  likely to be normal.
- $s(x, n) \approx 0.5 \Rightarrow$  no clear decision. [11]

## VII. RUNTIME / OVERHEAD OF ISOLATION FOREST

The **training stage** has time complexity:

$$O(t \cdot \psi \cdot \log \psi)$$

where:

- $t$  = number of trees
- $\psi$  = sub-sampling size (default typically  $\psi = 256$ ) . [12]

The **evaluation stage** has time complexity:

$$O(n \cdot t \cdot \log \psi)$$

where:

- $n$  = number of test points . [12]

Isolation Forest achieves **linear time complexity with a low constant**, making it suitable for large-scale, high-dimensional data. It also requires **low memory**, since only partial trees are built . [12]

The process begins with initialization, where the training dataset  $X$ , the number of trees  $t$ , and the sub-sampling size  $\psi$  are provided as input. The height limit for each Isolation Tree (iTree) is set to  $l = \lceil \log_2 \psi \rceil$  . [13]

During the training phase, the algorithm builds an ensemble of  $t$  Isolation Trees. For each tree, a random sub-sample of  $\psi$  instances is selected from  $X$ . Each iTree is then constructed recursively by partitioning the data: at each node, an attribute  $q$  and a split value  $p$  are chosen randomly. The data is split into left and right subsets, and the process continues until a stopping condition is met — either the maximum height is reached, only one instance remains, or all instances have identical values . [13]

Once the Isolation Forest is trained, the testing phase begins. For each test instance  $x$ , the algorithm traverses each iTree to compute the path length  $h(x)$ . If an external node with size greater than one is reached, an adjustment  $c(\text{Size})$  is

applied to the path length. The expected path length  $E(h(x))$  is calculated as the average over all trees. The anomaly score  $s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}$  is then computed for each instance . [13]

Finally, the instances are sorted by their anomaly scores in descending order. The top anomalies, corresponding to the instances with the shortest expected path lengths (and thus the highest scores), are output . [13]

## VIII. IMPLEMENTATION

**Tools:** Scikit-Learn is a widely used Python library that provides efficient implementations of many machine learning algorithms and preprocessing tools. It uses a clean, uniform API design that allows easy integration of data transformation, model training, and prediction into one pipeline [14]. Its `fit()`, `transform()`, `fit_transform()`, and `predict()` methods standardize interaction across different estimators, including preprocessing tools and learning algorithms . [14]

For preprocessing, Scikit-Learn offers `StandardScaler`, a transformer that standardizes input features by removing the mean and scaling to unit variance. This ensures that numerical attributes are centered around zero and have a variance of one, which is important for algorithms sensitive to feature scale. Unlike min-max scaling, standardization is more robust to outliers . [15]

To implement a model for credit card fraud detection using Scikit-Learn, we begin by scaling the data with `StandardScaler`, then apply `IsolationForest`, an unsupervised anomaly detection algorithm. The training data is passed through a pipeline that first standardizes the features and then fits the Isolation Forest model. This structure helps isolate rare and anomalous transactions based on how easily they are separated in randomly constructed trees . [16]

The dataset used in this study was the Credit Card Fraud Detection dataset provided by ULB and publicly available on Kaggle. It contains 284,807 transactions made by European cardholders in September 2013, with 492 fraudulent transactions (approximately 0.17%) indicating a strong class imbalance. The dataset consists of 30 features, including 28 anonymized principal components (V1 to V28), the transaction amount, and time. 492 out of 284,807 transactions labeled as fraud. The highly imbalanced nature of the data presents a realistic challenge for anomaly detection algorithms such as Isolation Forest . [17]

**Step 1: Mounting Google Drive** In the first step, the Google Drive was mounted into the Google Colab environment using the `drive.mount()` function from the `google.colab` library. This allowed Colab to access files stored in Google Drive. The credit card fraud detection dataset is loaded into the Colab notebook for further processing.

**Step 2: Loading and Exploring the Dataset** In this step, the credit card fraud detection dataset was loaded into the Colab environment using the `pandas` library. The dataset,

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...
5 rows × 31 columns											
...		V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
...		-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
...		-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
...		0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
...		-0.108300	0.005274	-0.190321	-1.175575	0.847376	-0.221929	0.062723	0.061458	123.50	0
...		-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

Fig. 2. Loading and Exploring the Dataset

provided as a CSV file (`creditcard.csv`), was read into a `DataFrame` using `pd.read_csv()`. The first five rows of the dataset were displayed using the `head()` function to verify the correct loading of the data. The dataset contains various features, including anonymized principal components (V1 to V28), the transaction amount, time, and the `Class` label, which indicates whether a transaction is fraudulent (1) or legitimate (0). [17]

**Step 3: Installing Required Packages** In this step, the `scikit-learn` library was installed and updated using the `pip` package manager. Using the command `!pip install -U scikit-learn` the latest compatible version of Scikit-learn is used in the project. The successful installation of Scikit-Learn, enables the required functionalities for the following steps. [14]

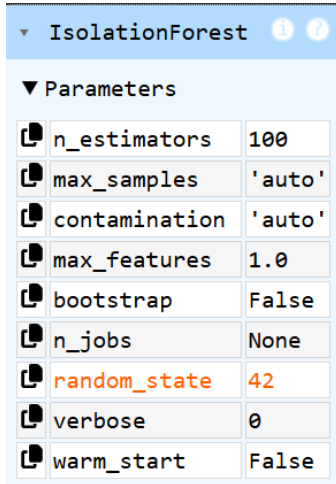
**Step 4: Importing Required Libraries** In this step, the necessary Python libraries were imported to support data processing, model building, and visualization. The `pandas` and `numpy` libraries were used for handling and manipulating data. The `StandardScaler` from Scikit-learn was imported to standardize feature values. Additionally, `matplotlib.pyplot` and `seaborn` were imported for visualization purposes to support later stages of the analysis. [15]

**Step 5: Data Preprocessing with StandardScaler** In this step, the dataset was preprocessed to prepare it for model training. The `Time` and `Class` columns were removed from the feature set, as the `Time` feature was not relevant for the Isolation Forest model, and the `Class` label was reserved for evaluation purposes only. [16]

The remaining features (V1 to V28 and `Amount`) were standardized using the `StandardScaler` from Scikit-learn. This transformation centers each feature around zero and scales it to have unit variance, which is important for the Isolation Forest algorithm, as it is sensitive to feature scales . [15]

After preprocessing, the resulting feature matrix contained 284,807 instances and 29 standardized features, as shown by

the output shape of the scaled data: (284807, 29).



IsolationForest	
Parameters	
n_estimators	100
max_samples	'auto'
contamination	'auto'
max_features	1.0
bootstrap	False
n_jobs	None
random_state	42
verbose	0
warm_start	False

Fig. 3. Training the Isolation Forest Model

**Step 6: Training the Isolation Forest Model** In this step, the Isolation Forest model was instantiated and trained on the standardized feature set. The model was implemented using Scikit-learn's `IsolationForest` class. The following parameters were used:

- `n_estimators=100`: The number of Isolation Trees to be built.
- `max_samples='auto'`: The sub-sample size for each tree was automatically determined using a random sub-sample.
- `contamination='auto'`: The proportion of outliers in the data was automatically estimated.
- `random_state=42`: A fixed random seed was used to ensure reproducibility of results.

The model was trained on the entire scaled dataset using the `fit()` method, which constructs the forest of Isolation Trees. This enables the model to learn the structure of the data and to isolate anomalies based on path lengths through the trees. [18]

#### Step 7: Computing Anomaly Scores

In this step, anomaly scores were computed for each instance in the dataset using the trained Isolation Forest model. The `decision_function()` method of Scikit-learn's `IsolationForest` class was used to compute these scores. The scores are based on the average path length of an instance across all Isolation Trees. The anomaly score  $s(x, n)$  is calculated using the following equation:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}$$

where  $E(h(x))$  is the average path length of instance  $x$ , and  $c(n)$  is the average path length of unsuccessful searches in a Binary Search Tree. Higher scores indicate instances that are likely to be normal, while lower scores indicate instances that are likely to be anomalies. [19]

The first ten computed scores were as follows: Anomaly scores example: [ 0.12540493, 0.14272779, 0.05345558, 0.10818359, 0.12636705, 0.14446305, 0.13256987, -0.00329639, 0.1164912, 0.1351488 ].

#### Step 8: Predicting Anomaly Labels

In this step, the trained Isolation Forest model was used to predict anomaly labels for all instances in the dataset. The `predict()` method was applied to the scaled data, which assigns a label to each instance based on its computed anomaly score. The method outputs `-1` for anomalies and `1` for normal instances. To match the binary format used in the original dataset, the predictions were converted as follows: `-1` was mapped to `1` (fraud), and `1` was mapped to `0` (normal) [20].

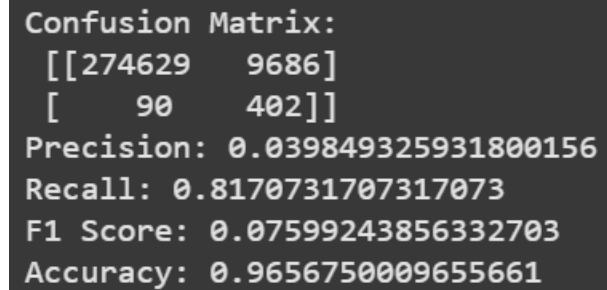
The first ten predicted labels were:

[0 0 0 0 0 0 0 1 0 0]

In this format, `1` indicates a predicted fraudulent transaction, while `0` indicates a predicted legitimate transaction.

**Step 9: Evaluating the Model** The `confusion_matrix()` function was used to compute the confusion matrix, which shows the number of true positives, false positives, true negatives, and false negatives [21].

In this step, the performance of the trained Isolation Forest model was evaluated by comparing its predictions to the true labels in the dataset. Several common evaluation metrics were computed using Scikit-learn's `metrics` module. The following metrics were calculated:



```
Confusion Matrix:
[[274629   9686]
 [    90    402]]
Precision: 0.039849325931800156
Recall: 0.8170731707317073
F1 Score: 0.07599243856332703
Accuracy: 0.9656750009655661
```

Fig. 4. Evaluating the Model

A true positive corresponds to a correctly detected fraudulent transaction, and a true negative corresponds to a correctly detected legitimate transaction. The results show that the model achieved a high recall, meaning that most fraudulent transactions were successfully detected. The precision was lower, which is typical for fraud detection problems with extreme class imbalance. The F1 Score and accuracy provide additional insight into the overall performance of the model [22].

**Conclusion:** Isolation Forest can be applied to credit card fraud detection by modeling the isolation of instances in high-dimensional feature space. This method has proven effective in

identifying anomalous transactions without requiring labeled data for training. The results demonstrated high recall, making the model suitable for detecting most fraudulent activities, although precision remains a challenge due to the extreme class imbalance. Overall, Isolation Forest offers a powerful and efficient approach for unsupervised fraud detection, providing a practical foundation for real-world applications while highlighting the importance of carefully evaluating false positive rates and considering complementary techniques to improve performance.

#### ACKNOWLEDGMENT

I, JIBAN-UL AZAM CHOWDHURY SHAFIN, herewith declare that I have composed the present paper and work by myself and without the use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form have not been submitted to any examination body and have not been published. This paper was not yet, even in part, used in another examination or as a course performance.

13/06/23, Lippstadt, Jiban-ul Azam Chowdhury Shafin  
Date&Place - Jiban-ul Azam Chowdhury Shafin

#### REFERENCES

- [1] B. Baesens, V. V. Vlasselaer, and W. Verbeke, *Fraud Analytics Using Descriptive, Predictive, and Social Network Techniques*. Wiley, 2015, chapter 1, Pages 5, 24, 29. Provides background, motivation, and challenges in credit card fraud detection, highlighting the need for anomaly detection models like Isolation Forest.
- [2] F. T. Liu, K. M. Ting, and Z.-H. Zhou, *Isolation Forest*. IEEE ICDM, 2008, section 4, Pages 7. Defines path length in binary trees, harmonic numbers, and the logarithmic normalization component used in the Isolation Forest anomaly score.
- [3] B. Baesens, V. V. Vlasselaer, and W. Verbeke, *Fraud Analytics Using Descriptive, Predictive, and Social Network Techniques*. Wiley, 2015, chapter 1, Page 5. Defines credit card fraud and its types, including application and behavioral fraud, as forms of identity theft.
- [4] —, *Fraud Analytics Using Descriptive, Predictive, and Social Network Techniques*. Wiley, 2015, chapter 1, Page 24. Describes the transactional data recorded for credit card fraud detection and its use in analytics.
- [5] —, *Fraud Analytics Using Descriptive, Predictive, and Social Network Techniques*. Wiley, 2015, chapter 4, Page 123. Explains binary classification modeling in fraud detection, including target variable definition.
- [6] —, *Fraud Analytics Using Descriptive, Predictive, and Social Network Techniques*. Wiley, 2015, chapter 1, Pages 24–25. Describes anomaly detection modeling and outlier detection techniques for fraud detection.
- [7] —, *Fraud Analytics Using Descriptive, Predictive, and Social Network Techniques*. Wiley, 2015, chapter 1, Page 29. Explains the extreme class imbalance in credit card fraud detection and its implications.
- [8] —, *Fraud Analytics Using Descriptive, Predictive, and Social Network Techniques*. Wiley, 2015, chapter 4, Pages 199–200. Provides the mathematical formulation of misclassification cost and discusses cost-sensitive learning in fraud detection.
- [9] —, *Fraud Analytics Using Descriptive, Predictive, and Social*. Wiley, 2015, chapter 4, Pages 176–177. Defines the key performance measures (confusion matrix, accuracy, error rate, precision, recall, F1, specificity) in fraud detection evaluation.
- [10] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM)*. IEEE, 2008, section 1. Introduces the Isolation Forest algorithm and its key idea of isolating anomalies.
- [11] —, “Isolation forest,” in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM)*. IEEE, 2008, section 2. Defines Isolation Trees, path length, anomaly score, and mathematical foundation of the algorithm.
- [12] —, “Isolation forest,” in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM)*. IEEE, 2008, sections 4 and 6. Explains steps of the workflow for the Isolation Forest.
- [13] —, “Isolation forest,” in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, section 4, Pages 417–418. Describes the full workflow of the Isolation Forest algorithm, including training phase (Isolation Trees construction) and testing phase (anomaly scoring).
- [14] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O’Reilly Media, 2017, chapter 2, Pages 61–66. Describes Scikit-Learn’s architecture and its uniform API design, including fit/transform methods.
- [15] —, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O’Reilly Media, 2017, chapter 2, Pages 66–67. Explains the use of StandardScaler for standardizing features and the advantages of standardization over min-max scaling.
- [16] —, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O’Reilly Media, 2017, chapter 9, Pages 262–264. Describes how Scikit-Learn Pipelines can be used to combine data preprocessing (such as StandardScaler) with model training, and how anomaly detection can be implemented with IsolationForest.
- [17] A. D. Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi, “Credit card fraud detection dataset,” <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>, 2015, provides a dataset of European credit card transactions with severe class imbalance, commonly used for benchmarking fraud detection algorithms.
- [18] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O’Reilly Media, 2017, chapter 9, Pages 248–253. Explains how to use Scikit-learn’s IsolationForest class, its parameters and the training process.
- [19] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 9, 2008, section 4, Pages 6–7. Explains the computation of anomaly scores using the average path length across Isolation Trees, and presents the formula for  $s(x,n)$ .
- [20] —, “Isolation forest,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 9, 2008, section 4, Pages 6–7. Describes the prediction of anomaly labels using Isolation Trees and the output interpretation of -1 (anomaly) and +1 (normal).
- [21] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O’Reilly Media, 2017, chapter 3, Pages 94–96. Explains how to evaluate classification models using confusion matrix, precision, recall, F1 score, and accuracy with Scikit-learn.
- [22] B. Baesens, V. V. Vlasselaer, and W. Verbeke, *Fraud Analytics Using Descriptive, Predictive, and Social*. Wiley, 2015, chapter 4, Pages 176–177. Defines confusion matrix, precision, recall, F1-score, accuracy, and error rate with formulas and fraud detection example.