519H0259_519H0303_Turnitin.docx

bởi Vinh Hoàng Quốc

Ngày Nộp: 20-thg 8-2024 11:27CH (UTC+0700)
ID Bài Nộp: 2434385311
Tên Tập tin: 519H0259_519H0303_Turnitin.docx (1.15M)
Đếm từ: 10233
Đếm ký tự: 57954

VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

HOANG QUOC VINH – 519H0259
NGUYEN ANH KHOA – 519H0303

HANDLING IMBALANCED DATA IN THE PREDICTION OF CUSTOMER CHURN

INFORMATION TECHNOLOGY PROJECT 2

COMPUTER SCIENCE

HO CHI MINH CITY, 2024

VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

HOANG QUOC VINH – 519H0259
NGUYEN ANH KHOA – 519H0303

HANDLING IMBALANCED DATA IN THE PREDICTION OF CUSTOMER CHURN

INFORMATION TECHNOLOGY PROJECT 2

COMPUTER SCIENCE

Advised by
Prof. Trinh Hung Cuong

HO CHI MINH CITY, YEAR 2024

ACKNOWLEDGMENT

We sincerely thank Prof. Trinh Hung Cuong for his invaluable guidance and support throughout the completion of this project. His expertise and insights into handling imbalanced data in machine learning were instrumental in shaping the direction and success of this work.

We are also thankful for his patience in addressing my numerous questions and for encouraging me to explore various approaches that enriched my understanding of the subject.

Finally, we would like to thank my university's Faculty of Information Technology for providing me with the foundation and resources necessary to undertake this project. The knowledge and skills we acquired during my studies were critical in overcoming the challenges faced during this research.

Ho Chi Minh City, day 20 month 08 year 2024
Author
(Signature and full name)

Vinh
Hoang Quoc Vinh

Khoa
Nguyen Anh Khoa

This thesis was carried out at Ton Duc Thang University.

Advisor: .....
.....

(Title, full name and signature)

This thesis/project report is defended at the Undergraduate Thesis Examination Committee was hold at Ton Duc Thang University on … /…/……

Confirmation of the Chairman of the Undergraduate Thesis Examination Committee and the Dean of the faculty after receiving the modified thesis (if any).

CHAIRMAN	**DEAN OF FACULTY**
……..	……..

DECLARATION OF AUTHORSHIP

I hereby declare that this thesis was carried out by myself under the guidance and supervision of Trinh Hung Cuong; and that the work and the results contained in it are original and have not been submitted anywhere for any previous purposes. The data and figures presented in this thesis are for analysis, comments, and evaluations from various resources by my own work and have been duly acknowledged in the reference part.

In addition, other comments, reviews and data used by other authors, and organizations have been acknowledged, and explicitly cited.

I will take full responsibility for any fraud detected in my thesis. Ton Duc Thang University is unrelated to any copyright infringement caused on my work (if any).

Ho Chi Minh City, day 20 month 08 year 2024
Author
(Signature and full name)

Vinh
Hoang Quoc Vinh

Khoa
Nguyen Anh Khoa

HANDLING IMBALANCED DATA IN THE PREDICTION OF CUSTOMER CHURN

ABSTRACT

Class imbalance problems have been a severe issue that can hinder the performance of standard classification algorithms and have drawn a large amount of attention from researchers across various fields. Therefore, there has been a various number of methods widely used to address these problems including resampling methods, cost-sensitive learning, and ensemble techniques. However, these traditional methods often come with drawbacks, such as the loss of valuable information, the introduction of errors, or an increased risk of overfitting due to alterations in the original data distribution.

To overcome these challenges, we also propose a new method, CSWRF combined with BorderlineSMOTE, this approach aims to enhance the classifier's ability to distinguish between classes. This combination increase the strengths of both methods to improve model performance on imbalanced datasets.

In our study, we compared the performance of our method against various conventional approaches, including Naive Undersampling, Naive Oversampling, SMOTE, ADASYN, BorderlineSMOTE, SVMSMOTE, KMeansSMOTE, SMOTETomek, SMOTEENN, and Borderline Tomek. The evaluation was primarily based on precision, recall, and F1-score, with accuracy as a preliminary measure. The results from experiments Telecom imbalanced datasets demonstrate that our proposed method consistently outperforms traditional techniques, particularly in handling highly imbalanced data.

TABLE OF CONTENT

Chapter 1. INTRODUCTION 1

1.1 Reason For Choosing Topic 1

1.2 Scopes and Objectives.....2

CHAPTER 2. CLASS IMBALANCED DATA PROBLEM.....3

2.1 Literature Review.....3

2.2 Class Imbalance Problem.....4

2.2.2 Problem Definition.....4

2.2.2 Causes of Imbalanced Data 6

2.2.3 Consequences of Imbalanced Data.....6

2.4 Techniques For Class Imbalance Problem.....7

2.4.1 Data-Level Techniques 7

2.4.2 Algorithm-Level Techniques 8

CHAPTER 3. THEORETICAL BASIS.....9

3.1 Over-sampling methods 9

3.1.1 Random Oversampling.....9

3.1.2 Synthetic Minority Over-sampling Technique (SMOTE) and SMOTE Variations.....10

3.1.3 ADASYN (Adaptive Synthetic Sampling) 20

3.2 Under-sampling methods 22

3.2.1 Random Undersampling (Removing examples uniformly) 23

3.2.2 ENN, Tomek Links (Removing noisy observations).....23

3.3 Classification Algorithms.....24

3.3.1 Logistic Regression 24

3.3.2 Random Forest.....25

3.3.3 Support Vector Machine 26

3.4 Common Evaluation Metrics 27

3.4.1 Confusion Matrix 27

3.4.2 ROC – AUC Curve *29*
3.4.3 Precision-Recall curve *29*
CHAPTER 4. METHODOLOGY **30**
4.1 Dataset Analysis 31
4.2 Dataset Preprocessing 32
4.2.1 Data Cleaning *32*
4.2.2 Encoding Categorical Data *33*
4.2.3 Scaling and Normalizing Numeric Data *33*
4.2.4 Eliminating Duplicate Entries *33*
4.2.5 Feature Engineering *33*
4.3 Handling Imbalance Class Methods 35
4.4 Classification Algorithms 35
4.5 A New Method For Handling Imbalanced Data (CSWRF + BorderlineSMOTE) 36
4.5.1 Stratified K-Fold Cross-Validation *37*
4.5.2 CSWRF + BorderlineSMOTE *40*
CHAPTER 5. EXPERIMENTAL RESULTS **44**
REFERENCES **48**

LIST OF FIGURES

Hình 2.1: Scaled Dot-Product Attention .....**Error! Bookmark not defined.**

LIST OF TABLES

Bảng 4.1: Thống kê kiểu thực thể trong tập VLSP 2016 .....**Error! Bookmark not defined.**

ABBREVIATIONS

BERT	Bidirectional Encoder Representations from Transformers
GEC	Grammatical Error Correction
MLM	Masked Language Model
NLP	Natural Language Processing
NSP	Next Sentence Prediction

Chapter 1. INTRODUCTION

1.1 Reason For Choosing Topic

Customer churn is an important term which refers to the rate of customers dropping off a company's products or cancelling subscription within a specific period of time. This has become a crucial priority in businesses' operational performance, especially in subscription business models. The ability of a company to turn new customers into repeat buyers and prevent them from switching to a rival competitor is called customer retention. This process requires more marketing expenses, time and effort than retaining existing customers. Therefore, high-rate level of customer churn can negatively impact revenue, profitability, and overall business growth. Those companies need to apply effective customer churn prediction strategies to minimise churn such as exploiting customer information and developing predictive Machine Learning models to address customer churn problems.

However, customer relationship management (CRM) databases contain thousands or even millions of customers' information. It is a challenge for any company to explore a piece of data that accounts for a minimal ratio of entire databases. This problem is considered as Class Imbalance Problem that occurs when the instances of one class significantly outnumber the instances of other classes. The class has larger number of instances called the majority class, while the smaller one is minority class. The imbalance would lead to poor predictive results of machine learning models because of high accuracy for the majority class (customer not churns) and poor accuracy for the minority class (customer churns) that holds more interest.

Analyzing the collected data has been simpler thanking to the recent development of technology, companies use Machine Learning to create models that discovers what factors lead to Customer Churn. The various common machine learning models in data classification are Support Vector Machine, Random Forest and Logistic Regression. Each model has its own benefits and can be optimized by applying resampling techniques. If adapting the suitable classìication models and resampling techniques for specific datasets, the results of prediction can produce high

accuracy, efficient computation time, strong generalization, and overfitting risk reduction capability. Companies can use the outcomes to support marketing strategies such as improving customers' satisfaction and retaining customers.

1.2 Scopes and Objectives

The aim of customer churn prediction within telecommunications sectors is to explore the effectiveness of various machine learning models, specifically Support Vector Machine (SVM), Random Forest, and Logistic Regression. The research will focus on examining the performance of these models when combined with different resampling techniques, including oversampling methods such as Synthetic Minority Oversampling Technique (SMOTE) and Random Over-Sampling (ROS), as well as Undersampling and Hybrid approaches.

To meet the requirements, the study will collect and prepare a relevant customer data, encompassing factors such as churn indicators, customer characteristics, subscription details, data usage, billing and payment information, and service feedback. Following data preparation, the study will involve developing and training the specified machine learning models. Various resampling techniques will be applied to address class imbalance, and their impact on model performance will be assessed. The effectiveness of each model and technique will be measured through metrics like accuracy, precision, recall, and F1-score. The research will conclude with a comparison of the models and techniques to determine the most effective combination for predicting customer churn, leading to actionable recommendations for telecommunications companies.

CHAPTER 2. CLASS IMBALANCED DATA PROBLEM

2.1 Literature Review

Kaur et al provided a comprehensive review of the challenges associated with imbalanced data in machine learning. Their paper thoroughly examines various solution approaches, including preprocessing techniques, cost-sensitive learning, algorithm-specific methods, and hybrid approaches. The review is organized by domain areas and applications, with a detailed comparison of machine learning algorithms based on metrics from selected studies.

Felix and Lee assessed existing literature on preprocessing techniques for general machine learning applications. Their review emphasizes evaluating the quality of the papers, focusing on data-related issues and preprocessing methods, and providing guidance for future research directions.

Spelmen and Porkodi analyzed solutions for imbalanced data found in the literature, addressing both data-level and algorithmic approaches, including hybrid models. Their study presents and discusses each proposed solution and its outcomes, organized by the method used.

Susan and Kumar conducted a survey on preprocessing techniques for machine learning applications. Their paper provides an in-depth look at sampling methods and details how each reviewed study implemented its proposed solutions. Additionally, the survey summarizes experimental procedures, specifics, and reported results.

Shakeel et al. reviewed literature on preprocessing techniques for both binary and multiclass classification in machine learning. Their review offers a brief overview of classification algorithms, preprocessing techniques, and ensemble methods.Chawla et al. utilized SMOTE to address imbalanced datasets characterized by significant sparsity and class imbalance. They employed Naïve Bayes and decision tree algorithms, demonstrating the effectiveness of SMOTE in handling sparse data.

Tafta et al introduced a method that combined SMOTE with particle swarm optimization (PSO) and the radial basis function (RBF) classifier. They used PSO to

optimize the structure and parameters of RBF kernels, showing that SMOTE PSO delivered competitive performance.

Fernandez-Navarro et al developed two oversampling techniques: a static SMOTE radial basis function method and a dynamic SMOTE radial basis function procedure, the latter being integrated into a memetic algorithm that optimizes RBF neural networks. Their experiments revealed that the dynamic oversampling method achieved the highest accuracy and sensitivity compared to other neural network approaches.

Mazurowski et al examined the impact of combining undersampling and oversampling with two neural network training methods: backpropagation (BP) and particle swarm optimization (PSO). They found that PSO was more responsive to class imbalance, small sample sizes, and a high number of features.

Cohen et al proposed a resampling strategy that incorporates both oversampling and undersampling with synthetic data. They also introduced class-dependent regularization parameters for tuning support vector machines (SVMs), resulting in an asymmetrical soft margin that provided a larger margin for the smaller class.

2.2 Class Imbalance Problem

2.2.2 Problem Definition

An imbalanced binary classification problem has a training set: $\chi_i \in R^d$, $\gamma_i \in \{-,+\}, i = 1,\ldots,(N_+ + N_-)$ where N_+ is the minority class and N_- is the majority class with $N_+ < N_-$. Traditional classification techniques are likely to overfit the majority class due to the large representation of the majority samples in the loss function. Although traditional regularization techniques are designed to balance bias and variance, they cannot successfully handling one-sided issues such as the overfitting of one class leads to the negative impact of the other.

The challenge of handling imbalanced datasets in binary classification is pervasive across various domains, including medical diagnosis, fraud detection, and customer churn prediction. In such scenarios, the minority class often represents the

critical condition or event, making its accurate prediction vital. The skewed class distribution, however, leads to biased classifiers that favor the majority class.

For example: there are only a minimal percentage of potential fraudulent transactions represented in millions of legitimate transactions daily, this imbalanced data makes it challenging for training models to identify fraud ones.

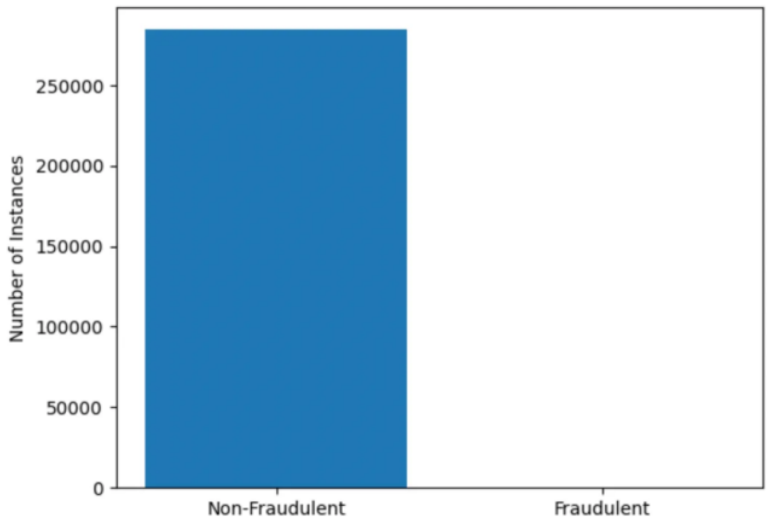


Figure 2.1: Class Imbalance Distribution

Real World Scenarios:

- **Fraud detection** using Machine Learning to detect which are fraudulent transactions out of million transactions. Fraud activities widely happens in finance, healthcare, and e-commerce industries and have impacted negatively on world's economy.
- **Network intrusion detection** requires the examine of large data from network to identify accessing activities to computer systems that are unauthorized.
- **Detection for Cancer** uses machine learning to diagnose cancer relying on the minority class samples, this is crucial for cancerous patients to get earlier treatment.

2.2.2 Causes of Imbalanced Data

- **Inherent in the problem:** Some events hardly occur in life such as fraudulent transactions or rare diseases so the its data is inherently imbalanced.
- **High cost of data collection:** The cost demand for collecting data can be high in some cases. For example, COVID-19 patients data collection requires a large volume of money to obtain because of medical tests and protective equipment.
- **Noisy labels:** This may happen when data labels are in misleading or incorrect category in data collection and contribute to the class imbalance.
- **Labeling errors:** Errors in labeling can skew the dataset bacause if some positive instances are incorrectly labeled as negative. Moreover, human-annotated class might be biases and overlook rare cases, especially inherent class.
- **Sampling bias:** In some cases, collecting data (survey) in a particular region or among a specific group of people can introduce bias in the dataset because the rusults may not represent the entire community.

2.2.3 Consequences of Imbalanced Data

- **Fail to use accuracy metric**: In the context of imbalanced dataset, conventional metrics like accuarcy can be misleading because the model still achieve 99% accuary if the dominated class accounts for 99% of a whole dataset. It leads to high accuracy but poor performance. To evaluate a better model performance, metrics like Precision, Recall and F1 Score is more appropriate to use.
- **Different misclassification costs for different classes**: Sometimes misclassify positive instances can be more expensive or harmful than negative examples. For instance, it takes more serious consequences to misclassify a cancerous patient with the healthy label (false negative) compared to misclassify a healthy patient as cancerous (false positive). This is the cost of misclassification, it makes some cases more complicated due to imbalanced datasets.

- **Computational Constraints:** It is a big challenge for handling a large number of data in sectors such as finance, healthcare or retail. This requires a significant computational power cost and number of consuming time. Techniques such as downsampling or undersampling the majority class can play a critical role in such scenarios. Moreover, it may increase the size of datasets and computational costs if more samples from minority class is acquired. With the large amount of data need to be processed can cause memory limitations.
- **The poor representation of the minority class examples due to lack of diversity in those examples :** Sometimes, having enough large amount of the minority class may still cause a big problem due to the lack of variation in the samples of minority class. Those samples are too similar to each other or small in distribution of the minority class, the model might not learn or distinguish them correctly that decrease the performance level of the model. For example, the model would struggle to detect rare objects in similar images.

2.4 Techniques For Class Imbalance Problem

With the development of computer technology, there are many effective solutions that have been widely applied in handling imbalanced data. These solotions are categorized as two different levels of techniques: Data-Level Techniques, Algorithm-Level Techniques. And each technique comprises methods that create a diversity of approaches to handle different sort of datasets.

2.4.1 Data-Level Techniques

It works, in a pre-processing stage, directly on the data space, and tries to re-balance the class distributions. Techniques can do both methods like decreasing the number of data points in the larger class (undersampling) or increasing the number of data points in the smaller class (oversampling). The primary goal is to create a more balanced dataset that can improve the performance of machine learning models.

- Oversampling: Increasing the number of instances in the minority class (SMOTE, SMOTEN, Borderline SMOTE, SVMSMOTE ADASYN, Random Oversampling,…).
- Undersampling: Decreasing a certain amount of instances in the majority class (Random Under-sampling, TomekLinks, Edited Nearest Neighbour,…).
- Combination of Over and Undersampling methods (SMOTEENN, SMOTETomek).

2.4.2 Algorithm-Level Techniques

This approach aims to modify existing machine learning models by adjusting the parameters to decrease the level of bias towards the majority groups. This requires good understanding of the modified learning algorithm and indentifying precisely of why it fails to failure to learn from the data with skewed distributions.

- Cost-Sensitive Learning is a method that assingn various level of misclassification costs to distinct classes which can lessen the impact of imbalanced classes. By putting misclassification penalties more heavily on the minority class, the model may be trained to detect and classify instances from the minority class more correctly.
- Ensemble Methods: Those techniques such as Bagging, Boosting can improve the ability of capturing minority class patterns by combining various models or assigning higher weights to the minority class. This leads to better model performance on class imbalance.
- One-Class Learning (OCC): This is an adavanced techniques used when the representation of the minority class is poor and difficult to classify. One-Class SVM and Isolation Forest are commonly applied to build a distinguish model that only represents the minority class.

CHAPTER 3. THEORETICAL BASIS

3.1 Over-sampling methods

Sampling is the process of selecting a subset of individuals from a larger population to collect data and develop a conclusion about the features of whole population.

Oversampling is a technique that increases the number of instances in the minority class to tackle imbalanced dataset. Howerver, this technique can be scaled to generate multiple imbalanced classes. The goal is to enhance the learning of the model more effectively in the minority class with sufficient samples. If there are not enough examples of the minority class in the dataset, the model will overlook these examples and give priority to the majority class. This can prevent the learning ability of models for the decision boundary.

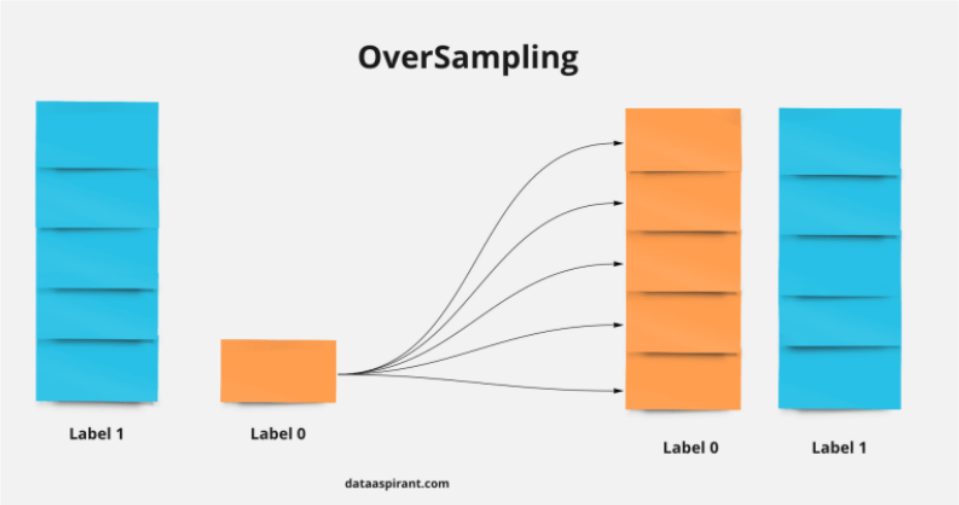


Figure 4.1: Oversampling

3.1.1 Random Oversampling

Random oversampling is a simple technique used in machine learning to address imbalanced datasets, where one class (the majority class) significantly outnumbers the other class (the minority class). It works by selecting the minor class data to be randomly duplicated, however, the result does not necessarily represent the

equivalent size between the minority class and the majority class. If the instances of minority class is oversampled in a large number, the model may not be able to recognize the features of data from different classes. Random Oversampling is straightforward to implement with simple parameter adjustments so it has some limitations for complex dataset. The number of added samples is determined based on a simple formula:

$$N = N_{majority} - N_{minority}$$

Where:

N: The number of samples is added to the minority class.

$N_{majority}$: The number of samples in the majority class.

$N_{minority}$: The number of samples in the minority class.

3.1.2 Synthetic Minority Over-sampling Technique (SMOTE) and SMOTE Variations

3.1.2.1 SMOTE

SMOTE is an oversampling method designed to generate synthetic samples for the minority class. By doing so, it helps mitigate the overfitting issue often associated with random oversampling. The algorithm works by focusing on the feature space, where it creates new instances through interpolation between closely positioned positive instances.

SMOTE process:

1. Set Total Number of Oversampling Observations (N): Determine the number of synthetic samples to generate, N. Typically, N is chosen such that the minority class is balanced with the majority class, but it can be adjusted based on the specific needs.
2. Select a Positive Class Instance: Select randomly an instance from the minority class (X).
3. Use the k-nearest neighbors algorithm to find the K neighbors of the selected instance. By default, K = 5

4. Choose Neighbors for Interpolation: From these K neighbors, randomly choose N neighbors to create new synthetic samples.
5. Interpolate New Synthetic Instances:
   - For each new synthetic instance, select a neighbor $x_{neighbor}$ from the 5 nearest neighbors.
   - Compute the difference vector: Diff = $X_{neighbor} - X$.
   - Choose a random value gap to create new synthetic instance (R): R = X + gap x Diff.

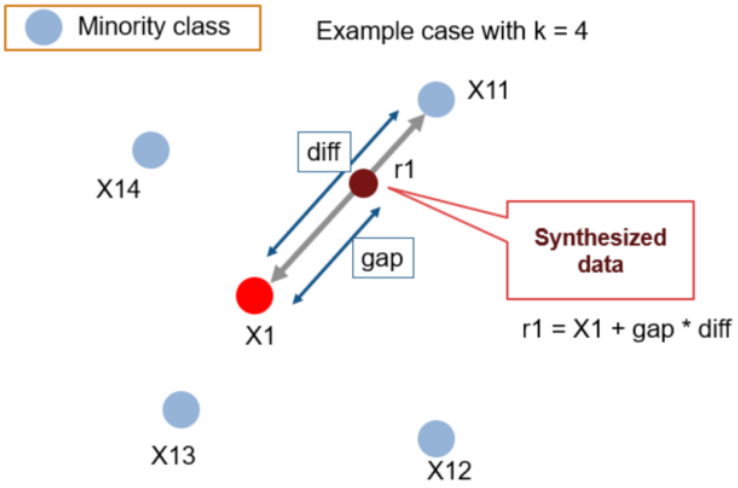


Figure 4. : SMOTE generate a New Synthetic Instance

SMOTE Pseudo Code:

```
Algorithm SMOTE(T, N, k)
Input:
  - Number of minority class samples T
  - Amount of SMOTE as a percentage N
  - Number of nearest neighbors k

Output:
  - (N/100) * T synthetic minority class samples
```

```
(* If N is less than 100%, only a random subset of minority class samples will be
SMOTEd. *)
if N < 100
   then
      Randomize the T minority class samples
      T = (N/100) * T
      N = 100
endif

N = (int)(N/100) (* Convert percentage to an integer value *)
numattrs = Number of attributes
Sample[][]: array for original minority class samples
newindex: counter for the number of synthetic samples generated, initialized to 0
Synthetic[][]: array for synthetic samples

(* Compute k-nearest neighbors for each minority class sample *)
for i ← 1 to T
   Compute k-nearest neighbors for sample i and save indices in nnarray
   Populate(N, i, nnarray)
endfor

Function Populate(N, i, nnarray) (* Generates synthetic samples for sample i *)
while N ≠ 0
   Choose a random neighbor index nn from the k-nearest neighbors
   for attr ← 1 to numattrs
      Compute: dif = Sample[nnarray[nn]][attr] - Sample[i][attr]
      Compute: gap = random value between 0 and 1
      Synthetic[newindex][attr] = Sample[i][attr] + gap * dif
   endfor
   newindex++
   N = N - 1
endwhile
return (* End of Populate function *)
End of Algorithm
```

Drawbacks of SMOTE:

- Noise Introduction: Synthetic instances created by SMOTE might introduce noise or outliers, especially in noisy or overlapping regions of the feature space.
- Overlapping Clusters: SMOTE might not perform well when minority class instances are tightly clustered.
- Computational Cost: Can be computationally expensive for large datasets, especially when dealing with high-dimensional data.
- Imbalanced Distribution: While SMOTE increases the number of minority class instances, it doesn't guarantee a balanced distribution in all feature subspaces.

3.1.2.2 SMOTE-ENN

SMOTENN is a combination of two methods SMOTE and Edited Nearest Neighbors (ENN) that improve limitation of SMOTE. ENN works as a data filtering technique which removes noisy and unimportant samples from the dataset. SMOTENN is able to generate more the synthetic samples from the minority class and reduce the noise level of the samples. As a result, a better generalization performance of the model is produced.

3.1.2.3 BorderlineSMOTE

In improve prediction accuracy, most classification algorithms concentrate on identifying the boundaries of each class during training to achieve better prediction. Data points near the borderline are more likely to be misclassified than the other ones which are far from these boundaries. Therefore this is crucial for improving model' classification.

Since data points far away from the boundaries aren't as crucial for classification, Hui Han, Wen-Yuan Wang, and Bing-Huan Mao (2005) proposed a new technique called 'borderlineSMOTE' . This method focuses on increasing the number of "borderline examples" from the smaller group of data (minority class) to

improve model performance. This approach differs from existing methods that either increase all data points from the minority class or randomly select a portion to increase. All instances of the minority class In Borderline-SMOTE are categorized into three groups: *NOISE*, *DANGER*, and *SAFE*.

- *NOISE*: These are rare instances and likely incorrect, found in regions that are largely dominated by the majority class instances.
- *DANGER*: These instances are located closely to the class boundaries and often overlap with majority instances.
- *SAFE*: These instances are easily recognizable and mainly represent in the minority class.

Suppose that the whole training set is T, the minority class is P and the majority class is N, and $P = \{p_1, p_2, \dots, p_{pnum}\}$, $N = \{n_1, n_2, \dots, n_{nnum}\}$.

BorderlineSMOTE process:

1. Identify Borderline Instances: For every p_i in the minority class P, calculate its m-nearest neighbors from the whole training set T. The number of majority examples among the m-nearest neighbors is denoted by m' $(0 \le m' \le m)$.
2. Evaluate Minority Instances:
   - If $m' = m$, all m-nearest neighbors of p_i all majority examples, p_i is considered as noisy data points and not used in the following steps.
   - If $m/2 \le m' < m$, p_i is considered as misclassified data points and added to DANGER.
   - If $0 \le m' < m/2$, p_i is safe.
3. Select Borderline Instances: The examples in *DANGER* are the borderline data of the minority class P , and $DANGER \subseteq$ P. It's defined as $DANGER = \{p_1, p_2, \dots, p_{pnum}\}, 0 \le d_{num} \le p_{num}$.
   - Generate Synthetic Samples: s x d_{num} synthetic positive samples are created from the *DANGER* data set, and an integer

$s \in (1,..,k)$. With each p_i choose randomly s from its k-nearest neighbors in P.

- First calculate $Diff_j$ (where j = 1, 2, … s).
- Then multiply $Diff_j$ by a random number $r_j \in (1,..,k)$ (where j ranges from 1 to s)
- Finally generating new synthetic minority examples s between $p'_i + r_j$ x $Diff_j$.

BorderlineSMOTE Pseudo Code:

```
Algorithm: Borderline-SMOTE

Input:
   Training set T
   Minority class P
   Integer k (number of nearest neighbors)
   Integer s (number of synthetic samples to generate)

Output:
   Synthetic samples to augment minority class P

Step 1: Identify Borderline Instances
   For each p_i in minority class P:
      Find m-nearest neighbors in T
      Count majority class neighbors as m'

Step 2: Evaluate Minority Instances
   For each p_i in P:
      If m' = m:
         Continue  // p_i is noise
      Else if m/2 ≤ m' < m:
         Add p_i to DANGER
      Else:
         Continue  // p_i is safe

Step 3: Select Borderline Instances
   For each p_i' in DANGER:
```

```
    Find k-nearest neighbors in P

Step 4: Generate Synthetic Samples
  For each p_i' in DANGER:
    For each neighbor p_j (j = 1 to s):
      diff_j = p_j - p_i'
      r_j = random(0, 1)
      synthetic_j = p_i' + r_j * diff_j
    Add synthetic samples to P

Return augmented minority class P
```

3.1.2.4 KMeansSMOTE

K-means SMOTE is a technique for handling imbalanced datasets that integrates K-means clustering with the SMOTE oversampling method. It involves three main steps:

- Clustering: The dataset is partitioned into *k* clusters using the K-means algorithm. K-means assigns each data point to the nearest cluster centroid and updates the centroid positions iteratively until convergence. For large datasets, more efficient variants like mini-batch K-means can be used.
- Filtering: This step identifies clusters for oversampling based on the proportion of minority class instances within them. Clusters with at least 50% minority samples are chosen. The goal is to oversample in clusters dominated by the minority class, which is less prone to noise. Sampling weights are calculated based on cluster density, where less dense clusters (with fewer minority samples) receive more generated samples.
- Oversampling: SMOTE is applied within each selected cluster to create synthetic minority instances. The quantity of instances to generate is determined by the sampling weight allocated to each cluster. SMOTE

produces new instances by interpolating between existing minority instances.

This method differs from others by clustering The entire dataset is considered irrespective of class labels, which can help avoid oversampling in potentially unsafe areas. It also distributes generated samples based on cluster density rather than just cluster size.

KMeansSMOTE Pseudo Code:

```
Input: X (matrix of observations)
      y (target vector)
      n (number of samples to be generated)
      k (number of clusters to be found by k-means)
      irt (imbalance ratio threshold)
      knn (number of nearest neighbors considered by SMOTE)
      de (exponent used for computation of density; defaults to the number of features
in X)
begin
   // Step 1: Cluster and filter based on minority presence
   clusters ← kmeans(X, k)
   filteredClusters ← empty set

   for each c in clusters do
      imbalanceRatio ← (majorityCount(c) + 1) / (minorityCount(c) + 1)
      if imbalanceRatio < irt:
         filteredClusters ← filteredClusters ∪ {c}

   // Step 2: Compute sampling weights for each filtered cluster
   sparsitySum ← 0
   for each f in filteredClusters do
      avgDist ← mean(euclideanDistances(f))
      densityFactor(f) ← minorityCount(f) / (avgDist ^ de)
      sparsityFactor(f) ← 1 / densityFactor(f)
      sparsitySum += sparsityFactor(f)

   for each f in filteredClusters do
      samplingWeight(f) ← sparsityFactor(f) / sparsitySum
```

```
    // Step 3: SMOTE oversampling
    generatedSamples ← empty set
    for each f in filteredClusters do
        numSamples ← floor(n × samplingWeight(f))
        generatedSamples ← generatedSamples ∪ SMOTE(f, numSamples, knn)

    return generatedSamples
end
```

3.1.2.5 SVMSMOTE

SVM-SMOTE is proposed to use support vector machine (SVM) algorithm that focus on the decision boundary to identify the misclassified instances instead of *K-nearest neighbor algorithm*. The main aim of this method is using extrapolation to expand the area of the minority class where the number of majority class is fewer. Thus, this algorithm can increase more chance of representation of Minority group close to the optimal border. There are two conditions to generate new instances by SVMSMOTE with following steps:

1. Identify support vectors from the minority class $\mathbf{SV^+}$ and $\mathbf{sv^+} \subseteq \mathbf{SV^+}$.
2. Find m nearest neighbors for each supoort vectors $\mathbf{sv^+}$ from the training set, m' is denoted as the number of instances from majority class among m.
3. If $0 \leq m' < \frac{m}{2}$ (Fewer majority class are the nearest neighbors of sv^+), meaning that the majority class does not influence the minority area so that for each $\mathbf{sv^+}$ using this equation to expand the minority region by extrapolation.

   $$Synthetic_i = \mathrm{sv}^+ + p \times (\mathrm{sv}^+ - nn[i][j]),$$

   where $nn[i][j]$ is the j-th positive nearest neighbor of sv^+ and p is a random number $\in [0, 1]$.
4. If $\frac{m}{2} \leq m' < m$ (More majority class are the nearest neighbors of sv^+), meaning that the algorithm need to consolidate the minority region instead of expanding it.

$$Synthetic_i = \mathrm{sv}^+ + p \times (nn[i][j] - \mathrm{sv}^+)$$

SVM-SMOTE pseudo code:

```
Algorithm: SVM-SMOTE

Input:
   SV+ (set of minority class support vectors)
   k (nearest neighbors)
   m (nearest neighbors from the training set)
   Training data set Dtr

Output:
   Augmented minority class with synthetic instances

Step 1: For each support vector sv_i+ in SV+:
   1.1 Calculate its m nearest neighbors from the training set.
   1.2 Count the number of majority instances among these m nearest neighbors and
denote this as m0.

Step 2: Conditions for Generating Synthetic Instances
   2.1 If 0 ≤ m0 < m/2 (fewer than half of the m nearest neighbors are majority
instances):
      // Extrapolation - Expand minority class region
      For j = 1 to k (for each of the k minority nearest neighbors):
         2.1.1 Generate synthetic instance using:
            synthetic_i = sv_i+ + ρ * (sv_i+ - nn[i][j])
            where nn[i][j] is the j-th minority nearest neighbor of sv_i+
                ρ is a random number in the range [0, 1]
         2.1.2 Add synthetic_i to the augmented minority class

   2.2 If m/2 ≤ m0 < m (most neighbors are majority class):
      // Interpolation - Consolidate boundary area of minority class
      For j = 1 to k (in the order of nearest neighbors):
         2.2.1 Generate synthetic instance using:
            synthetic_i = sv_i+ + ρ * (nn[i][j] - sv_i+)
```

where nn[i][j] is the j-th minority nearest neighbor of sv_i+
ρ is a random number in the range [0, 1]
2.2.2 Add synthetic_i to the augmented minority class

Step 3: Repeat for each support vector sv_i+ in SV+ to generate synthetic instances

Return the augmented minority class with new synthetic instances

3.1.3 ADASYN (Adaptive Synthetic Sampling)

The ADASYN method aims to address the issue of imbalanced datasets by generating more synthetic data for the minority class in a way that focuses on the instances that are harder to classify correctly. This helps to reduce the bias that naturally occurs when training a model on an imbalanced dataset, where the model may be more likely to misclassify minority class instances. Additionally, ADASYN can adjust the decision boundary (the line that separates the classes) to give more attention to the difficult-to-learn minority class samples. Suppose training dataset D_{tr} with m samples $\{x_i, y_i\}$, i = 1, ..., m, *where* x_i is an instance in the n dimensional feature space X and $y_i \in Y = \{1, -1\}$ is the class identify label associated with x_i. Define m_s and m_l as the number of minority class examples and the number of majority class examples, respectively. Therefore, $m_s \leq m_l$ and $m_s + m_l = m$.

ADASYN process:

1. Calculate the degree of class imbalance by computing ratio $d = \frac{m_s}{m_l}$ (1), where $d \in (0, 1]$.
2. Compare d with a threshold d_{th} (threshold for the maximum tolerated degree of class imbalance ratio) to determine which imbalance ratio is needed to be proceeded in next steps.
    - If $d \geq d_{th}$: the dataset is balanced enough and no need further proceeding steps.
    - If $d < d_{th}$: continues to generate synthetic samples.

3. Calculate the total number of synthetic samples G to generate the balance dataset. $G = (m_l - m_s) \times \beta$, where $\beta \in (0, 1]$ is parameter used to specify the desired balance level after generation of the synthetic data. For example, $\beta = 1$ means a fully a balanced data set is created.
4. Calculate the Density Ratio:
    - Using Euclidean distance to find *K nearest neighbors* in *n* dimension and count the number of *K neighbors* Δ_i that belong to the majority class .
    - Calculate the ratio r_i : $r_i = \frac{\Delta i}{K}$, where $i = 1, \dots, m_s$.
5. Normalize the ratio r_i : $\hat{r_i} = \frac{r_i}{\sum_{i=1}^{m_s} r_i}$.
6. Determine the number of synthetic samples for each example x_i :

$$g_i = \hat{r_i} \times G.$$

7. Generate synthetic samples for each example x_i with *Loop from 1 to g:*
    - Randomly select a minority sample x_{zi} from one of the *K nearest neighbor* of x_i.
    - Sample $s_i = x_i + (x_{zi} - x_i) \times \lambda$ is generated, where $\lambda \in [0, 1]$.

ADASYN pseudo code:

```
Algorithm: ADASYN

Input:
   Training set Dtr with m samples {xi, yi}, i = 1, ..., m
   Integer K (number of nearest neighbors)
   Integer β (desired balance level, 0 ≤ β ≤ 1)
   Float dth (threshold for maximum tolerated degree of class imbalance)

Output:
   Augmented minority class with synthetic samples

Step 1: Calculate Degree of Class Imbalance
   ms = number of minority class examples
```

```
   ml = number of majority class examples
   d = ms / ml

Step 2: Check Imbalance Ratio
   If d >= dth:
      Return

Step 3: Calculate Total Number of Synthetic Samples
   G = (ml - ms) * β

Step 4: Calculate Density Ratio for Each Minority Instance
   For each minority class instance xi:
      Compute k-nearest neighbors of xi
      Δi = number of majority class examples among k-nearest neighbors
      ri = Δi / K
      Normalize ri to get density distribution:
         Σri = sum of ri for all minority class instances
         r̂i = ri / Σri

Step 5: Determine Number of Synthetic Examples per Instance
   For each minority class instance xi:
      gi = r̂i * G

Step 6: Generate Synthetic Samples
   For each minority class instance xi:
      For j = 1 to gi:
         Randomly select a neighbor xzi from k-nearest neighbors of xi
         Generate synthetic sample si:
            si = xi + (xzi - xi) * λ
            where λ is a random number between 0 and 1
         Add si to the augmented minority class

Return the augmented minority class with synthetic samples
```

3.2 Under-sampling methods

Undersampling is a group of techniques that removes samples from the majority class. This helps to balance the dataset with a skewed class distribution and increase the learning ability of the models. It is different from Oversampling that adds more samples from the minority class in reduce the level of class imbalance . Generally, combining both undersampling and oversampling methods will produce more optimized results.

3.2.1 Random Undersampling (Removing examples uniformly)

The most straightforward undersampling method involves arbitrarily selecting instances from the majority class and excluding them from the training dataset. This approach is known as random undersampling. Although simple and effective, this technique has the drawback of removing examples without considering their potential significance in defining the decision boundary between classes. This implies that valuable information could be inadvertently eliminated.

3.2.2 ENN, Tomek Links (Removing noisy observations)

3.2.2.1 Edited Nearest Neighbor

ENN is a common technique used in data processing to address the class imbalance problem by undersampling the majority class. It eliminates the samples that are close the decision boundary to put less effort on distinguishing different classes, it leads to the improvement of dataset quality. Here its process:

1. Choosing the number of k Neighbors ($k = 3$ is a common choice but can be modified depending on the dataset).
2. For each instance x_i, determine its *k-nearest* neighbors by using Euclidean distance.
3. For each instance x_i, determine the class labels of its *k-nearest* neighbors.
4. If the class label of x_i does not match the majority class of its k-nearest neighbors → x_i is considered as a misclassified instance in its nearest neighbors → x_i is removed from the dataset.

3.2.2.2 Tomek Links

1. Identify the nearest neighbors for each instance in the dataset to understand proximity and potential class overlap, use distance metrics, commonly Euclidean distance, to determine the closest data points for each instance.

2. Find pairs of instances where one belongs to the minority class and the other belongs to the majority class, and they are each other's nearest neighbors.

- For a pair (i, j) to qualify as a Tomek Link:
- Instance i (from the minority class) and Instance j (from the majority class) are nearest neighbors to each other.

3. Clean the dataset by removing instances from the majority class that are identified as part of Tomek Links by method:

- Filter out the instances from the majority class that are involved in Tomek Links to reduce noise and improve class separation.

4. After removing Tomek Links, the dataset may still be imbalanced. Further resampling techniques can be used to achieve a better balance.

5. Apply resampling techniques like SMOTE or ADASYN to generate synthetic samples for the minority class and achieve a more balanced dataset.

3.3 Classification Algorithms

3.3.1 Logistic Regression

Logistic regression extends linear regression to tackle binary classification problems by modeling the specific event or class probability with a logistic function. Considering $X \in R^{n \times p}$ as an n by p matrix including n observations and p features, where x_i^T is a p-dimensional row vector containing p features of the i^{th} observation, and β is a p-dimensional column vector containing features' coefficients. $Y \in \{0, 1\}$ represents the binary outcome and $Y_i \sim Bernoulli(\pi_i)$. The linear relationship between the features and the log-odds of the event when $Y_i = 1$ can be written in the following form:

$$l = log[\frac{P(Y_i=1)}{1-P(Y_i=1)}] = log[\frac{\pi_i}{1-\pi_i}] = x_i^T \beta.$$

The likelihood function of $\pi = (\pi_1, \ldots, \pi_n)^T$ is used to estimate model coefficients β. It is the joint probability mass function of n Bernoulli variables.

Regularized logistic regression incorporates a penalty term (or regularizer) into the loss function to mitigate excessive fluctuations in the model. This constraint prevents coefficients from attaining extreme values, resulting in a simpler and less overfitted model.

Advantages:

- Simplicity and Interpretability: Logistic Regression is straightforward and provides coefficients for each feature, which helps in understanding how each feature affects the probability of churn.
- Probabilistic Output: It provides probabilities for the predicted class, which can be useful for decision-making and assessing the confidence of predictions.
- Fast and Scalable: It is computationally efficient and scales well with large datasets, making it a practical choice for many applications.

3.3.2 Random Forest

Random forest is an ensemble learning method that builds a large collection of decision trees using the bagging (bootstrap aggregation) technique. This method generates a set of decorrelated decision trees. In a random forest, each tree makes a class prediction, and the class that receives the most votes is selected as the final prediction. The random forest algorithm benefits from bagging because it reduces correlation among the trees by introducing random splits on subsets of features. As a result, random forests typically perform significantly better than single decision tree classifiers. The algorithm for a random forest is as follows:

1. Set the total number of trees as B. For b = 1 to B:

   (a) Draw a bootstrap sample Z ∗ of size N from the training data.

(b) Build a random forest tree TbT_bTb on the bootstrapped sample by repeating the following steps for each terminal node until the minimum node size n_{min} is reached:

i. Select m features at random from p features (Typically m = √p).

ii. Pick the best features/split-point among m.

iii. Split the node into two daughter nodes.

2. Output the ensemble of trees Tb. For a classification task, the final prediction is determined by majority voting across the B trees.

Advantages:

- Handles Non-Linearity and Interactions: Random Forests can capture complex interactions between features without needing explicit specification of these interactions.
- Robust to Overfitting: By averaging multiple decision trees, Random Forests reduce overfitting and improve generalization.
- Feature Importance: It provides insights into the importance of each feature, which can be useful for feature selection and understanding the drivers of churn.

3.3.3 Support Vector Machine

The goal of the Support Vector Machine (SVM) algorithm is to identify a hyperplane within a high-dimensional space that effectively separates two classes [21]. A hyperplane serves as a decision boundary that maximizes the margin, which represents the distance between the hyperplane and the closest data point. Various hyperplane learning techniques exist, each utilizing different types of kernels. For instance, the dot product functions as the distance metric in linear kernel SVMs. More complex kernels, like the polynomial and radial kernels, enable the separation of classes with curved or even more complex shapes. Given a training set n of $\{xi, yi\}_{i=1}^{n}$, where $xi \in R^p$, p is the number of features, $yi \in \{-1, +1\}$ is the i th output, the SVM classifier is defined as:

$$f(x) = sign\ [\sum_{i=1}^{n} \alpha_i\, y_i K(x, x_i) + b].$$

where α_i are positive real constants, and b is a real constant. $K(x, x_i)$ is the choice of a kernel.

Advantages:

- Effective in High-Dimensional Spaces: SVM performs well in cases with many features, making it useful when data has numerous attributes.
- Robust to Overfitting: By using a kernel trick, SVM can efficiently handle non-linear relationships in the data, which helps in capturing complex patterns that might be indicative of churn.
- Margin-Based Classification: SVM aims to find the optimal hyperplane that maximizes the margin between classes. This can be particularly useful if classes are not well-separated and need a robust decision boundary.

3.4 Common Evaluation Metrics

3.4.1 Confusion Matrix

The performance of a classification model is generally assessed using a confusion matrix, which summarizes the number of correct and incorrect predictions for each actual class, as shown in Table 5.2. In this study, 'positive' refers to the majority class or class 1, while 'negative' refers to the minority class or class 0. TP (True Positives) and TN (True Negatives) represent the number of correctly classified positive and negative instances, respectively. FN (False Negatives) and FP (False Positives) represent the number of positive and negative instances that were misclassified. The model's accuracy can be derived from the values in the confusion matrix.

Table 5.2 Confusion Matrix

	Predicted Positive	Predicted Negative
Actually Positive	True Positive (TP)	False Negative (FN)

Actually Negative	False Positive (FP)	True Negative (TN)

However, when assessing model performance on imbalanced data, accuracy often favors a classifier that excels at predicting the majority class while performing poorly on the minority class. In cases of extreme imbalance, the model can achieve high accuracy even if it correctly predicts all majority instances and misclassifies every minority instance. Therefore, accuracy alone does not provide a reliable measure of the model's ability to predict the minority class, necessitating the use of more appropriate evaluation metrics.

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

3.4.1.1 Sensitivity, specificity, precision, and F-measure

Sensitivity, or recall, the same as the TP rate, is the ratio of correctly predicted positive observations to the observations in the actual positive class. It answers the question: of all customers who are truly positive, or "no churn," how many were correctly predicted or labeled as positive. Specificity, also known as the true negative (TN) rate, measures the proportion of customers who are actually churning and are correctly identified as churn.

$$Sensitivity = Recall = TP\ rate = \frac{TP}{TP + FN}$$

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

Precision measures the proportion of instances that are classified as positive and are truly positive. It answers the question: of all customers labeled as positive, or "no churn," how many are actually "no churn."

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

F-measure is the weighted harmonic mean of precision and recall, and it takes both false positive and false negative cases into account when evaluating the model performance. F-measure is high when both recall and precision are high. It can be

adjusted by a coefficient of β, where β represents the relative importance between precision and recall. For instance, F_1 (β = 1) puts the balanced importance on the precision and recall.

$$F_\beta = (1 + \beta^2)\frac{Recall \times Precision}{\beta^2 \times Precision + Recall}$$

3.4.2 ROC – AUC Curve

The ROC curve (Receiver Operating Characteristic) visually depicts the trade-off between true positive rate (TPR) and false positive rate (FPR). The AUC (Area Under the Curve) quantifies the model's ability to differentiate between classes. A higher AUC indicates superior model performance in correctly classifying both positive and negative instances. Ideally, an excellent model achieves an AUC close to 1, signifying optimal class separability. Conversely, a poor model exhibits an AUC near 0, demonstrating minimal class discrimination.

- When AUC is 0.7, it means there is a 70% chance that the model will be able to distinguish between positive class and negative class.
- When AUC is approximately 0.5, the model has no discrimination capacity to distinguish between positive class and negative class.
- When AUC is approximately 0, the model is predicting a negative class as a positive class and vice versa.

3.4.3 Precision-Recall curve

The Precision-Recall Curve (PR curve) is a graphical representation used to evaluate the performance of a classification model, particularly in scenarios where the classes are imbalanced. The PR curve is a plot of Precision (y-axis) versus Recall (x-axis) for different threshold values used to classify instances as positive or negative.

PR Curve Characteristics:

- High Precision with Low Recall: Indicates that the model is very confident in its positive predictions, but it might miss many positive instances.
- High Recall with Low Precision: Indicates that the model captures most of the positive instances but at the cost of many false positives.
- The Area Under the PR Curve (AUPRC): A key metric that summarizes the performance of the model across different thresholds. A higher AUPRC indicates better overall performance, especially in cases of imbalanced data.

CHAPTER 4. METHODOLOGY

Diagram step to Handle Imbalance Dataset in this study

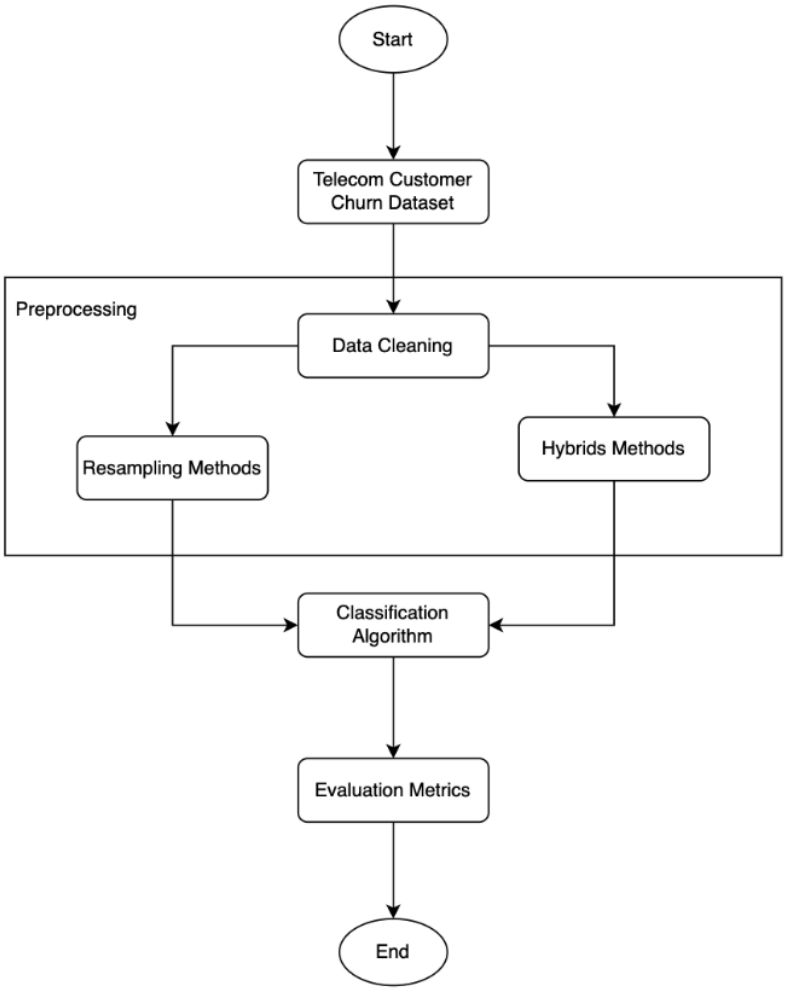


Figure 5.1: Diagram System

4.1 Dataset Analysis

The Tele Customer churn dataset contains 7043 customers information in California in Q3 that use Phone and Internet services provided by a fictional Telecom Company. This dataset presents 1869 churning customers out of 7043 customers. The dataset is highly unbalanced, the positive class (churn) account for 26.54 of all transactions. A brief description of the dataset characteristics is outlined as follows:

Dataset Name	# Total Examples	# Minority examples	# Majority examples	#Attributes
Tele Customer Churn	7043	1869	5174	21

Table 5.1: Dataset characteristics

Modifications:

- Imbalance Adjustment: The dataset is naturally imbalanced with a minority class representing customers who have churned and a majority class representing customers who have not. The positive class (Churn) accounts for approximately 26.54% of the dataset.
- Feature Engineering: For improved model performance, features related to customer behavior and demographic information are used. Features such as 'MonthlyCharges' and 'TotalCharges' are considered for scaling and normalization to handle skewed data distributions.

4.2 Dataset Preprocessing

4.2.1 Data Cleaning

Data cleaning is an essential preprocessing step in machine learning, especially for customer churn prediction. It changes raw data to high-quality data that improve models' accuracy and reliability. Below are some key tasks and techniques used in data cleaning.

4.2.1.1 Managing Missing Values

- Detection: Identify any missing values in the dataset.
- Imputation: Replace missing data using methods like mean, median, mode, or more advanced techniques such as KNN imputation or multiple imputation by chained equations (MICE).
- Removal: If the missing data is insignificant, consider removing the corresponding rows or columns.

4.2.1.2 Handling Outliers

- Detection: Identify outliers using statistical measures like Z-scores or interquartile range (IQR) and visualization tools like box plots.
- Resolution: Choose whether to remove, transform, or use robust models that are less sensitive to outliers.

4.2.2 Encoding Categorical Data

- Label Encoding: Convert categorical variables into numerical ones.
- One-Hot Encoding: Create binary columns for each category in the categorical variable.

4.2.3 Scaling and Normalizing Numeric Data

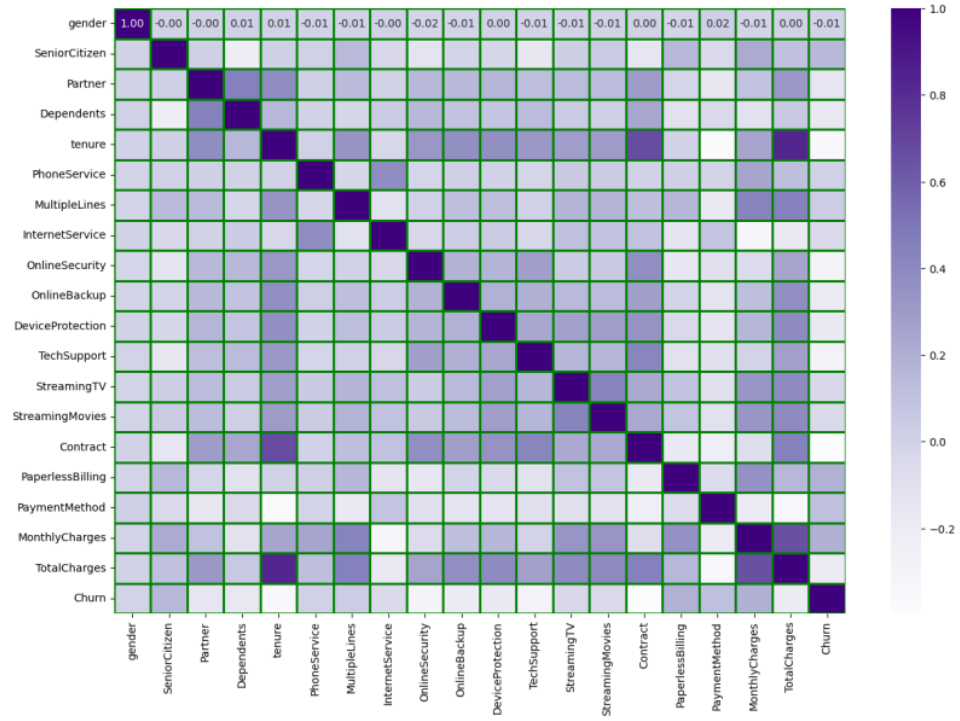
- Standardization: Adjust features to have a mean of 0 and a standard deviation of 1.
- Normalization: Scale features to a specific range, usually [0, 1].

4.2.4 Eliminating Duplicate Entries

- Detection: Search for duplicate records in the dataset.
- Removal: Drop duplicates to maintain the uniqueness of each entry.

4.2.5 Feature Engineering

- Creation: Generate new features that might enhance the model's predictive power, such as interaction terms or aggregate statistics.
- Selection: Use techniques like feature importance or correlation analysis to select the most relevant features.



Correlation measures the strength and direction of a linear relationship between two variables. It ranges from -1 to 1:

- 1: Perfect positive correlation
- -1: Perfect negative correlation
- 0: No correlation

gender: -0.008612

- Interpretation: Very weak negative correlation with the target variable Churn. Gender has almost no linear relationship with churn.

SeniorCitizen: 0.150889

- Interpretation: Weak positive correlation with Churn. Senior citizens are slightly more likely to churn.

InternetService: -0.047291

- Interpretation: Very weak negative correlation with Churn. Internet service has a negligible impact on churn.

OnlineSecurity: -0.289309

- Interpretation: Moderate negative correlation with Churn. Lack of online security is associated with a higher likelihood of churning.

OnlineBackup: -0.195525

- Interpretation: Weak negative correlation with Churn. Lack of online backup is slightly associated with a higher likelihood of churning.

DeviceProtection: -0.178134

- Interpretation: Weak negative correlation with Churn. Lack of device protection is slightly associated with a higher likelihood of churning.

TechSupport: -0.282492

- Interpretation: Moderate negative correlation with Churn. Lack of tech support is associated with a higher likelihood of churning.

By thoroughly addressing these steps, the quality of data improves, leading to better performance in customer churn prediction models.

4.3 Handling Imbalance Class Methods

Mainly methods are used to experiment is from Resampling and Hybrid Techniques.

4.4 Classification Algorithms

When tackling a churn classification problem, selecting the right classification algorithms is crucial for building an effective model. Each algorithm—Support Vector Classifier (SVC), Logistic Regression, and Random Forest—has its unique strengths and characteristics that make it suitable for different aspects of churn prediction. Here's why each of these algorithms is considered a good choice:

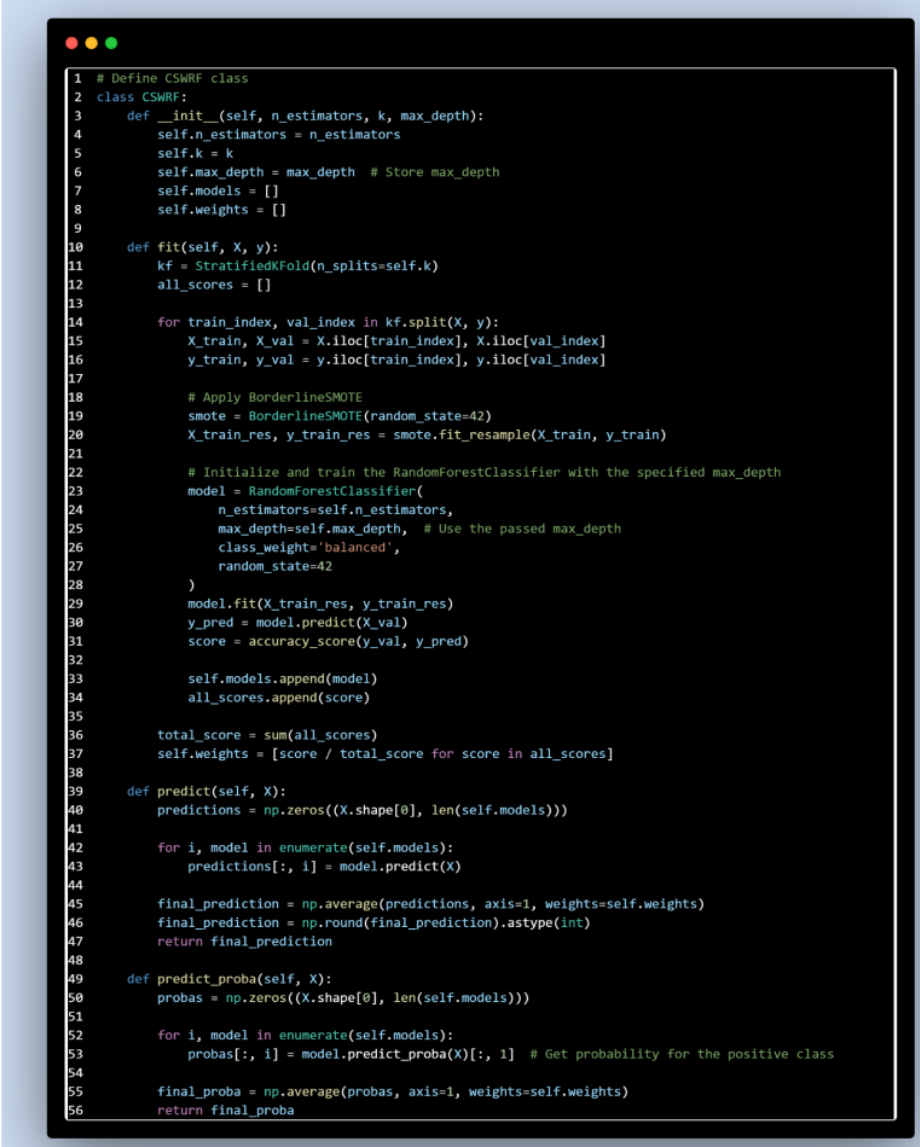
- Complex Decision Boundaries: If churn data has complex, non-linear relationships between features and the target variable, SVC's ability to use different kernels (like RBF or polynomial) can help capture these complexities.

- Baseline Model: It is often used as a baseline model due to its simplicity. It's a good starting point to understand the relationship between features and churn and to evaluate feature importance.
- Interpretation of Effects: Helps in understanding which factors (e.g., tenure, monthly charges) are statistically significant in predicting churn.
- Handling Large Feature Sets: Suitable for datasets with many features and complex relationships.
- High Accuracy and Robustness: Effective in achieving high classification accuracy and robustness in the presence of noisy data.

SVC is ideal when dealing with complex, high-dimensional data and when expect non-linear relationships in churn data. While Logistic Regression provides a good balance between interpretability and performance, making it useful for understanding the effects of individual features on churn. Finally, Random Forest excels in handling large datasets with many features and capturing complex interactions, making it a powerful and flexible option for churn prediction.

By using a combination of these algorithms, leverage their individual strengths to build a more robust and accurate churn prediction model. Each algorithm can be tested and compared to determine which provides the best performance for specific dataset. SVC is ideal when dealing with complex, high-dimensional data and when expect non-linear relationships in churn data.

4.5 A New Method For Handling Imbalanced Data (CSWRF + BorderlineSMOTE)



```python
# Define CSWRF class
class CSWRF:
    def __init__(self, n_estimators, k, max_depth):
        self.n_estimators = n_estimators
        self.k = k
        self.max_depth = max_depth  # Store max_depth
        self.models = []
        self.weights = []

    def fit(self, X, y):
        kf = StratifiedKFold(n_splits=self.k)
        all_scores = []

        for train_index, val_index in kf.split(X, y):
            X_train, X_val = X.iloc[train_index], X.iloc[val_index]
            y_train, y_val = y.iloc[train_index], y.iloc[val_index]

            # Apply BorderlineSMOTE
            smote = BorderlineSMOTE(random_state=42)
            X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

            # Initialize and train the RandomForestClassifier with the specified max_depth
            model = RandomForestClassifier(
                n_estimators=self.n_estimators,
                max_depth=self.max_depth,  # Use the passed max_depth
                class_weight='balanced',
                random_state=42
            )
            model.fit(X_train_res, y_train_res)
            y_pred = model.predict(X_val)
            score = accuracy_score(y_val, y_pred)

            self.models.append(model)
            all_scores.append(score)

        total_score = sum(all_scores)
        self.weights = [score / total_score for score in all_scores]

    def predict(self, X):
        predictions = np.zeros((X.shape[0], len(self.models)))

        for i, model in enumerate(self.models):
            predictions[:, i] = model.predict(X)

        final_prediction = np.average(predictions, axis=1, weights=self.weights)
        final_prediction = np.round(final_prediction).astype(int)
        return final_prediction

    def predict_proba(self, X):
        probas = np.zeros((X.shape[0], len(self.models)))

        for i, model in enumerate(self.models):
            probas[:, i] = model.predict_proba(X)[:, 1]  # Get probability for the positive class

        final_proba = np.average(probas, axis=1, weights=self.weights)
        return final_proba
```

Figure 5.2: Set Up CSWRF Class

4.5.1 Stratified K-Fold Cross-Validation

4.5.1.1 Purpose of Stratified K-Fold Cross-Validation:

- Evaluation Consistency: Ensures that each fold in the cross-validation process has a similar class distribution to the entire dataset, leading to more consistent evaluation of the Random Forest model.
- Handling Imbalance: For imbalanced datasets (e.g., in customer churn prediction, where churn cases may be much fewer than non-churn cases), stratified sampling helps to ensure that each fold contains a representative number of churn cases, preventing bias in model evaluation.
- Kaur et al provided a comprehensive review of the challenges associated with imbalanced data in machine learning. Their paper thoroughly examines various solution approaches, including preprocessing techniques, cost-sensitive learning, algorithm-specific methods, and hybrid approaches. The review is organized by domain areas and applications, with a detailed comparison of machine learning algorithms based on metrics from selected studies.
- Felix and Lee assessed existing literature on preprocessing techniques for general machine learning applications. Their review emphasizes evaluating the quality of the papers, focusing on data-related issues and preprocessing methods, and providing guidance for future research directions.
- Spelmen and Porkodi analyzed solutions for imbalanced data found in the literature, addressing both data-level and algorithmic approaches, including hybrid models. Their study presents and discusses each proposed solution and its outcomes, organized by the method used.
- Susan and Kumar conducted a survey on preprocessing techniques for machine learning applications. Their paper provides an in-depth look at sampling methods and details how each reviewed study implemented its proposed solutions. Additionally, the survey summarizes experimental procedures, specifics, and reported results.

- Shakeel et al. reviewed literature on preprocessing techniques for both binary and multiclass classification in machine learning. Their review offers a brief overview of classification algorithms, preprocessing techniques, and ensemble methods.Chawla et al. utilized SMOTE to address imbalanced datasets characterized by significant sparsity and class imbalance. They employed Naïve Bayes and decision tree algorithms, demonstrating the effectiveness of SMOTE in handling sparse data.
- Tafta et al introduced a method that combined SMOTE with particle swarm optimization (PSO) and the radial basis function (RBF) classifier. They used PSO to optimize the structure and parameters of RBF kernels, showing that SMOTE PSO delivered competitive performance.
- Fernandez-Navarro et al developed two oversampling techniques: a static SMOTE radial basis function method and a dynamic SMOTE radial basis function procedure, the latter being integrated into a memetic algorithm that optimizes RBF neural networks. Their experiments revealed that the dynamic oversampling method achieved the highest accuracy and sensitivity compared to other neural network approaches.
- Mazurowski et al examined the impact of combining undersampling and oversampling with two neural network training methods: backpropagation (BP) and particle swarm optimization (PSO). They found that PSO was more responsive to class imbalance, small sample sizes, and a high number of features.
- Cohen et al proposed a resampling strategy that incorporates both oversampling and undersampling with synthetic data. They also introduced class-dependent regularization parameters for tuning

support vector machines (SVMs), resulting in an asymmetrical soft margin that provided a larger margin for the smaller class.

4.5.1.2 Stratified K-Fold Cross-Validation Process

1. Dataset Splitting: The entire dataset is divided into k folds using stratified sampling, ensuring that each fold maintains the same class distribution as the overall dataset.
2. Training and Validation:
   - For each iteration, the Random Forest model is trained on k-1 folds (the training set) and validated on the remaining fold (the validation set).
   - This process is repeated k times, with each fold serving as the validation set once.
3. Model Evaluation:
   - After each fold, performance metrics (e.g., accuracy, F1-score, ROC-AUC) are computed and recorded.
   - The final model performance is assessed by averaging these metrics across all k folds.

4.5.2 CSWRF + BorderlineSMOTE

1. Class Initialization (__init__ method).

Parameters:

- n_estimators: The number of trees in the Random Forest.
- k: The number of folds for cross-validation.
- max_depth: The maximum depth of each tree in the forest.

Initialization:

- self.models: An empty list to store the Random Forest models trained on each fold.
- self.weights: An empty list to store the weight of each model, determined by its validation accuracy.

2. Training the Model (fit method):

Stratified K-Fold Cross-Validation:

- kf = StratifiedKFold(n_splits=self.k): The data is split into k folds, ensuring each fold has a similar class distribution (important for imbalanced datasets like customer churn).

Loop Over Each Fold:

- The data is split into training and validation sets (X_train, X_val, y_train, y_val).

Handling Imbalanced Data with BorderlineSMOTE:

- smote = BorderlineSMOTE(random_state=42): This technique generates synthetic samples for the minority class (e.g., customers who churn), focusing on the decision boundary where misclassification is more likely.
- X_train_res, y_train_res = smote.fit_resample(X_train, y_train): The training set is resampled to balance the class distribution.

Training a Random Forest Model:

- model = RandomForestClassifier(...): A Random Forest model is trained using the resampled data.
- model.fit(X_train_res, y_train_res): The model learns patterns from the balanced training data.

Validation:

- y_pred = model.predict(X_val): The model predicts on the validation set.
- score = accuracy_score(y_val, y_pred): The accuracy of the model on the validation set is calculated.

Storing Models and Weights:

- self.models.append(model): The trained model is stored.

- self.weights.append(score / total_score): The model's weight is stored, proportional to its validation accuracy.

3. Making Predictions (predict method):

   Aggregating Predictions:

   - predictions[:, i] = model.predict(X): Each model in the ensemble predicts the class labels for the input data.
   - final_prediction = np.average(predictions, axis=1, weights=self.weights): The final prediction is a weighted average of predictions from all models.
   - final_prediction = np.round(final_prediction).astype(int): The final prediction is rounded to obtain the class label.

4. Predicting Probabilities (predict_proba method):

   Aggregating Probabilities:

   - probas[:, i] = model.predict_proba(X)[:, 1]: Each model predicts the probability of the positive class (e.g., the customer will churn).
   - final_proba = np.average(probas, axis=1, weights=self.weights): The final probability is a weighted average of probabilities from all models.

Why This Approach is Effective for Imbalanced Data:

- Stratified K-Fold Cross-Validation: Ensures each fold has a similar distribution of classes, making the training process more robust, especially for imbalanced datasets.
- BorderlineSMOTE: Focuses on generating synthetic data near the decision boundary, which is where models often struggle to differentiate between classes. This helps improve the model's ability to correctly classify the minority class (e.g., predicting customers who are likely to churn).

- Ensemble Learning with Weighted Voting: Multiple models are trained on different subsets of the data, and their predictions are aggregated. By weighting models based on their validation accuracy, the final prediction is more reliable and less prone to overfitting.
- Class Weighting in Random Forest: The class_weight='balanced' parameter ensures that the model gives appropriate attention to both classes during training, further addressing the imbalance.

The CSWRF class combines several powerful technique Stratified K-Fold Cross-Validation, BorderlineSMOTE, and Random Forests with weighted voting to effectively handle imbalanced data. This makes it well-suited for tasks like customer churn prediction, where accurately identifying the minority class (e.g., customers likely to churn) is crucial.

CHAPTER 5. EXPERIMENTAL RESULTS

Classification Algorithms	SVC	Logistic Regression	Random Forest
Precision	0.67	0.69	0.66
Recall	0.40	0.43	0.44
F1-score	0.50	0.53	0.53
Accuracy	0.79	0.80	0.79

Table 6.1: Perfomance for three classification algorithms without balancing methods.

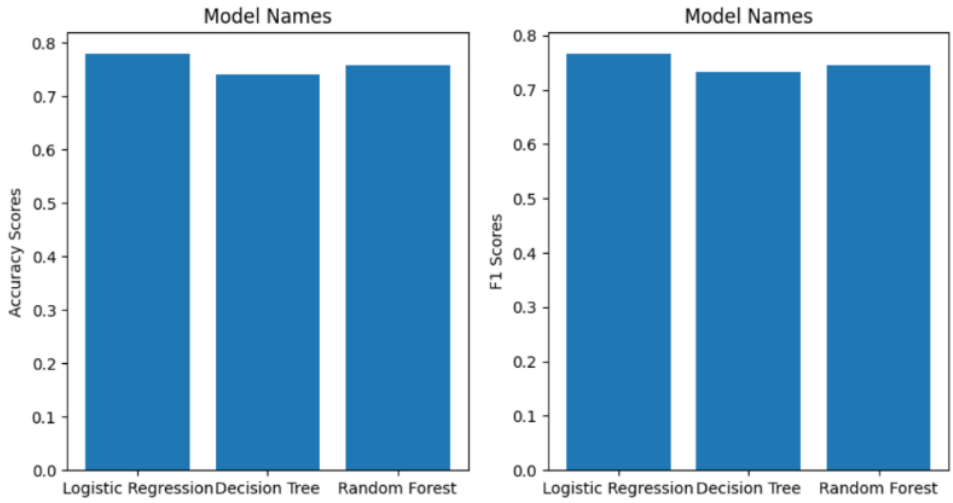


Figure 6.1: Accuracy and F1 Scores for three classification algorithms

Apply randonUndersampler with Random Forest and compare its performance.

Random Forest	With randonUndersampler method	Without randonUndersampler method
Precision	0.51	0.66
Recall	0.76	0.44
F1-score	0.61	0.53
Accuracy	0.74	0.79

Table 6.2: Perfomance with randomUndersampler method

Apply SMOTE with Random Forest and compare its performance.

Random Forest	With SMOTE method	Without SMOTE method
Precision	0.51	0.66
Recall	0.76	0.44
F1-score	0.61	0.53
Accuracy	0.74	0.79

Table 6.3: Perfomance with SMOTE method

Apply Borderline-SMOTE with Random Forest and compare its performance.

Random Forest	With Borderline-SMOTE method	Without Borderline-SMOTE method
Precision	0.55	0.66
Recall	0.68	0.44
F1-score	0.61	0.53
Accuracy	0.77	0.79

Table 6.4: Perfomance with Borderline-SMOTE method

Apply SVM-SMOTE with Random Forest and compare its performance.

Random Forest	With SVM-SMOTE method	Without SVM-SMOTE method
Precision	0.57	0.66
Recall	0.64	0.44
F1-score	0.60	0.53
Accuracy	0.77	0.79

Table 6.5: Perfomance with SVM-SMOTE method

Apply ADASYN with Random Forest and compare its performance.

Random Forest	With ADASYN method	Without ADASYN method
Precision	0.54	0.66
Recall	0.67	0.44
F1-score	0.60	0.53
Accuracy	0.77	0.79

Table 6.6: Perfomance with ADASYN method

Apply CSWRF + BorderlineSMOTE

	CSWRF + BorderlineSMOTE	Borderline-SMOTE + Random Forest	Random Forest
Precision	0.53	0.55	0.66
Recall	0.78	0.68	0.44
F1-score	0.63	0.61	0.53
Accuracy	0.76	0.77	0.79

Table 6.7: Perfomance with CSWRF + BorderlineSMOTEmethod

REFERENCES

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-term Memory. *Neural Computation*, *9*, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). *Attention Is All You Need* (arXiv:1706.03762). arXiv. https://doi.org/10.48550/arXiv.1706.03762

519H0259_519H0303_Turnitin.docx

BÁO CÁO ĐỘC SÁNG

16%	16%	11%	8%
CHỈ SỐ TƯƠNG ĐỒNG	NGUỒN INTERNET	ẤN PHẨM XUẤT BẢN	BÀI CỦA HỌC SINH

NGUỒN CHÍNH

1	escholarship.org Nguồn Internet	6%
2	Submitted to Ton Duc Thang University Bài của Học sinh	2%
3	docplayer.net Nguồn Internet	1%
4	Submitted to Manchester Metropolitan University Bài của Học sinh	1%
5	doczz.net Nguồn Internet	1%
6	medium.com Nguồn Internet	1%
7	hdl.handle.net Nguồn Internet	1%
8	Submitted to Imperial College of Science, Technology and Medicine Bài của Học sinh	1%

9	Vitor WernerdeVargas, Jorge Arthur SchneiderAranda, Ricardo dos SantosCosta, Paulo Ricardo da SilvaPereira et al. "Imbalanced data preprocessing techniques for machine learning: a systematic mapping study", Knowledge and Information Systems, 2022 Xuất bản	1 %
10	Submitted to Loughborough University Bài của Học sinh	1 %
11	ousar.lib.okayama-u.ac.jp Nguồn Internet	1 %
12	Submitted to University of Wollongong Bài của Học sinh	1 %

Loại trừ Trích dẫn	Mở	Loại trừ trùng khớp	< 1%
Loại trừ mục lục tham khảo	Mở		