

Sentimental Analysis

*Project report submitted in partial fulfillment of the requirement for the degree of
Bachelor of Technology*

in

Computer Science and Engineering/Information Technology

By

Tripti Gupta (191346)

Pankaj Kumar (191506)

UNDER THE SUPERVISION

OF

Dr. Rakesh Kanji

Assistant Professor (SG)



Department of Computer Science &
Engineering and Information Technology

Jaypee University of Information Technology,

Waknaghat , Solan - 173234 , Himachal Pradesh

Candidate's Declaration

I hereby declare that the work presented in this report entitled “ **Sentimental Analysis**” in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from July 2022 to May 2023 under the supervision of Dr. **Rakesh Kanji** Assistant Professor.

I also authenticate that I have carried out the above mentioned project work under the proficiency stream Data Science.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Tripti Gupta 191346

Pankaj Kumar 191506

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr Rakesh Kanji

Assistant Professor

Computer Science Engineering

Dated:

ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for His divine blessing makes it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Supervisor Dr. Rakesh Kanji Assistant Professor, Department of CSE Jaypee University of Information Technology, Wakhnaghat. Deep Knowledge & keen interest of my supervisor in the field of “Information Security” to carry out this project. This project was made possible by his never-ending patience, academic leadership, constant encouragement, constant and energetic supervision, constructive criticism, insightful counsel, reading numerous subpar versions and fixing them at all levels.

I would like to express my heartiest gratitude to Dr. Rakesh Kanji , Department of CSE, for his kind help to finish my project

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Project Group No-96

Tripti Gupta 191346

Pankaj Kumar 191506

CONTENTS

| Titles | Page no |
|---|----------------|
| 1. Introduction | 1-4 |
| 1.1 Introduction | 1 |
| 1.2 Problem Statement | 2 |
| 1.3 Objective | 3 |
| 1.4 Methodology | 3 |
| 1.5 Organization | 4 |
| | |
| 2.Literature Survey | 5-8 |
| | |
| 3.System Development | 10-31 |
| 3.1 Natural Language Processing | 10 |
| 3.2 Extractive Summarization | 11 |
| 3.2.1 Architecture of Smote Alogrithm | 11 |
| 3.2.2 Transformer and pipelining | 12 |
| | |
| 3.3 Smote | 16 |
| 3.3.1 Working procedure of oversampling technique | 17 |
| 3.3.1 Methodology | 18 |
| 3.3.3 Steps involve while implemmenting | 19 |
| 3.3.4 Tf-Idf | 21 |
| 3.3.5 Sequence-to-sequence Modeling | 22 |
| 3.3.6 Encoder-decoder architecture of transformer | 23 |
| 3.4 Technologies used | 25 |

| | |
|---|-----------|
| 3.4.1 Keras | 25 |
| 3.4.2 Tensorflow | 25 |
| 3.4.3 Scikit-learn | 25 |
| 3.4.4 Pandas | 26 |
| 3.4.5 Imblearn | 27 |
| 3.4.6 Sklearn | 27 |
| 3.4.7 Numpy | 28 |
| | |
| 3.5 Mathematical Model Development | 29 |
| 3.5.1 Term Frequency | 30 |
| 3.5.2 Keyword Frequency | 30 |
| 3.5.3 Stop word Filtering | 30 |
| 3.5.4 K means Clustering approach | 30 |
| 3.5.5 Frequency Based approach | 30 |
| 3.5.6 Word Probability | 31 |
| 3.5.7 Term Frequency-Inverse Document Frequency | 32 |
| | |
| 4. Performance Analysis | 34 |
| 4.1 Approaches to System Developed | 34 |
| 4.2 Training Dataset | 35 |
| 4.3 Use Case Diagram | 36 |
| 4.4 Classifiers Used | 38 |
| 4.4.1 Logistics Regression | 38 |
| 4.4.2 Decision Tree Classifier | 39 |
| 4.4.3 Random Forest | 41 |
| 4.4.4 Ada Boost | 43 |

| | |
|------------------------------------|-----------|
| 4.4.5 Linear Discriminant Analysis | 45 |
| 4.5 Results and Output | 48 |
| 5.Conclusion | 49 |
| 5.1 Conclusion | 50 |
| 5.2 Future Scope | 51 |
| 5.3 Applications | 52 |
| References | 53 |

LIST OF FIGURES

| S.no | Name of Figure | Page No. |
|-------------|--|----------|
| Figure 3.1 | Distribution | 9 |
| Figure 3.2 | Text Conversion | 11 |
| Figure 3.3 | Architecture of smote | 12 |
| Figure 3.4 | Encoder-Decoder structure | 15 |
| Figure 3.5 | Pipeline using Transformer | 16 |
| Figure 3.6 | Work flow of smote | 19 |
| Figure 3.7 | Vectorization | 22 |
| Figure 3.8 | Encoder-Decoder | 22 |
| Figure 3.9 | Working Mechanism | 24 |
| Figure 3.10 | Technologies used | 36 |
| Figure 3.11 | Minimum k value | 29 |
| Figure 3.12 | Total sentimental Index | 30 |
| Figure 3.13 | Probability of particular word | 31 |
| Figure 3.14 | Sentence weight | 32 |
| Figure 3.15 | Terms frequency index | 33 |
| Figure 3.16 | Inverse terms frequency index | 33 |
| Figure 4.1 | Data distribution in smote | 36 |
| Figure 4.2 | Use Case Diagram | 37 |
| Figure 4.3 | Estimated Probability | 38 |
| Figure 4.4 | Confusion matrix for logistic regression | 39 |
| Figure 4.5 | Formula for information gain | 40 |
| Figure 4.6 | Confusion matrix for decision tree | 40 |
| Figure 4.7 | Mean squared error | 41 |
| Figure 4.8 | Distribution tree | 42 |

| | | |
|-------------|---|----|
| Figure 4.9 | Confusion matrix for random forest | 42 |
| Figure 4.10 | Confusion matrix for Ada Boost | 44 |
| Figure 4.11 | Training and testing in Ada Boost | 45 |
| Figure 4.12 | Confusion matrix for linear discriminant analysis | 47 |
| Figure 4.13 | Comparison between Different MLA | 48 |
| Figure 4.14 | Accuracy of all models | 49 |
| Figure 4.15 | Voting classifiers with weight | 49 |

ABSTRACT

An approach called SMOTE (Synthetic Minority Over-sampling Technique) was developed to deal with the issue of unbalanced datasets in machine learning. When one class has a disproportionately smaller number of examples than another, the datasets are imbalanced, which makes it challenging for machine learning models to correctly forecast the minority class.

To balance the dataset, the SMOTE algorithm creates fictitious samples from the minority class. Finding a minority class instance at random and its k closest neighbors accomplishes this. The method then generates synthetic models along each k nearest neighbors' line segment, starting with the minority instance.

SMOTE successfully increases the number of examples of the minority class in the dataset by creating synthetic instances, which enables machine learning models better to understand the patterns and traits of the minority class. Machine learning models perform better on unbalanced datasets when using SMOTE, it has been demonstrated.

Chapter 1

INTRODUCTION

1.1 Introduction

Smote algorithm stands for Synthetic Minority Oversampling technique which add fake data to our original datasets just to balance the improper ratio of minor group and provide proper accuracy while training modeling The project aims to research about imbalanced classification which involves creating a models which classify datasets that can have imbalance datasets. Working with imbalance datasets is most challenging work which is usually ignored in machine learning models so to resolve the issue of imbalance datasets which affect the performance matrix or overall accuracy of models. For solving such situation simple approach we can use is creating duplicates minor datasets which in turn act as balancing original datasets for improving its accuracy and performance measures. This approach of smote algorithm is proven to be effective as new synthetic cases from minor class are created plausible which is almost very close to features that are already existing in minor group.

In the field of Data Science and Machine Learning, there is a common problem known as imbalanced data distribution. This issue arises when the number of observations in one class is significantly higher or lower than that in other classes. Machine learning algorithms often aim to increase accuracy by reducing errors and may not take into account the class distribution. This problem is particularly prevalent in applications such as fraud detection, anomaly detection, facial recognition, and so on.

Conventional machine learning techniques such as decision trees and logistic regression tend to favor the majority class and may ignore the minority class. As a result, they are more likely to predict the majority class, leading to significant misclassification of the minority class compared to the majority class. In technical terms, if our dataset has an imbalanced data distribution, our model is more likely to predict the majority class and may overlook the minority class, leading to poor performance in the case of the minority class.

To address the problem of imbalanced data distribution, various techniques have been developed such as oversampling the minority class, undersampling the majority class, using ensemble

methods such as Random Forest, or applying cost-sensitive learning algorithms. The choice of technique will depend on the specifics of the problem and the algorithm being used.

In conclusion, it is essential to be aware of the problem of imbalanced data distribution and to take the appropriate steps to address it. By applying the right techniques, we can improve the performance of machine learning algorithms and achieve better results, especially in cases where the minority class is crucial.

1.2 Problem Statement

In a scenario where you are tasked with detecting health insurance fraud, it's common to encounter a class imbalance, where only 1 out of every 100 insurance claims is fraudulent, while the rest are non-fraudulent. As a result, a simple binary classifier model could easily predict all outcomes as 0, indicating non-fraudulent claims, and achieve a high accuracy of 99%. However, in situations like this, where the class distribution is skewed, the accuracy metric can be biased and may not be the best metric to use for evaluation.

The issue of when unbalanced data is a problem in machine learning. The question points out that there is a lot of literature in machine learning that discusses class imbalance and the challenges it poses for classification algorithms, including probabilistic models. The premise of the question is that this literature suggests that imbalanced datasets with a significant difference in the number of positive and negative classes cause problems for machine learning algorithms and that balancing the dataset by restoring a 50/50 split between classes should be the goal.

It is true that class imbalance can pose challenges for some machine learning algorithms, particularly those that rely on a decision boundary that is sensitive to the distribution of the classes. When the positive class is underrepresented, the algorithm may be biased towards the negative class, leading to poor performance in predicting the positive class. However, it is not always necessary or desirable to balance the dataset by equalizing the number of positive and negative examples.

In fact, balancing the dataset can sometimes lead to overfitting or other problems. There are many techniques available for dealing with class imbalance, such as oversampling the minority class, undersampling the majority class, or using cost-sensitive learning algorithms. The choice of technique will depend on the specifics of the problem and the algorithm being used.

Overall, it is important to be aware of the potential problems posed by class imbalance and to consider the appropriate techniques for dealing with it, but there is no one-size-fits-all solution. The goal should be to achieve the best possible performance on the problem at hand, rather than to restore an arbitrary balance between classes.

1.3 Objectives

Binary classification problems can become difficult when the dataset is unbalanced, meaning that one class has significantly more samples than the other. This asymmetry can cause problems for both classifier training and evaluation, leading to biased results in favor of the majority class. To address this challenge, various metrics have been proposed, such as the Brier score and the area under the receiver operating characteristic curve (AUC), which are designed to place greater emphasis on correctly classifying minority samples. However, these metrics alone may not be sufficient to address the underlying problem of imbalanced data.

One solution to imbalanced data is to use data balancing techniques such as SMOTE, which stands for Synthetic Minority Oversampling Technique. SMOTE generates synthetic samples from the minority class by creating new samples that are similar to existing minority samples. This helps balance the dataset and can improve the performance of classifiers, especially weak learners like SVM.

oversampling the minority class, SMOTE allows SVM to learn from a more representative set of samples and improve its ability to classify both minority and majority samples. This can lead to better overall performance and more accurate evaluation metrics.

Overall, SMOTE is a powerful technique for addressing the challenges of imbalanced data in binary classification problems and can improve the performance of weak learners like SVM.

1.4 APPROACH TO SOLVE THIS PROBLEM OF UNBALANCE DATASET

The paper proposes an approach to construct classifiers from imbalanced datasets, where the distribution of the classes is not balanced, and one class is underrepresented compared to the other. The proposed method combines over-sampling of the minority class and under-sampling of the majority class to improve classifier performance. The cost of misclassifying an abnormal or interesting example as a normal example is higher than the cost of the reverse error, making it necessary to improve the sensitivity of the classifier to the minority class. The paper shows that under-sampling of the majority class alone is not sufficient to improve classifier performance and that a combination of over-sampling of the minority class and under-sampling of the majority class can achieve better performance in ROC space. The method of over-sampling the minority class involves creating synthetic examples of the minority class. The experiments were performed using C4.5, Ripper, and Naive Bayes classifiers. The proposed method was evaluated using the area under the Receiver Operating Characteristic curve (AUC) and the ROC convex hull strategy.

The paper concludes that the proposed approach is effective in improving classifier performance in imbalanced datasets, and the combination of over-sampling of the minority class and under-sampling of the majority class is more effective than varying the loss ratios in Ripper or class priors in Naive Bayes. The proposed method can be useful in real-world applications where the dataset is imbalanced, and the cost of misclassification is high.

1.5 Organization

- Chapter 1: Introduction of the project, covering the main idea and purpose.
- Chapter 2: Details on system development, including project design, applied models, and formulas used.
- Chapter 3: Literature research, discussing papers and journals from reputable sources of machine learning and neural networks.
- Chapter 4: Results and analysis, presenting the outcomes generated by the most accurate and productive models and examining them.
- Chapter 5: Conclusion, summarizing the project's results and discussing its future scope.

Chapter 2

LITERATURE REVIEW

| | |
|------------------------------------|---|
| <i>Title</i> | Automatic Text Summarization Approaches[1] |
| <i>Authors</i> | Ahmad T. Al-Taani (Ph.D., MSc, BSc) Professor of Computer Science (Artificial Intelligence) Faculty of Information Technology and Computer Sciences Yarmouk University, Jordan. |
| <i>Year of Publications</i> | August 2017 |
| <i>Publishing Details</i> | International Conference on Infocom Technologies and Unmanned Systems (ICTUS'2017) |
| <i>Summary</i> | <p>Automated Text summarization systems are important in many aspects in a language like natural language processing. ATS creates the summary of given document which save time and resources. There are single and multi-document text summary. Only one document is extracted in case of single document summarization whereas group of documents is selected in multi document summarization.</p> <p>On other hand, mathematics techniques makes the extractive summarization language independent to theoretical ways. In this analysis, we tend to the thought of the utilization of extractive summarization methodology. There are two content-based summaries i.e. - generic and query-based summaries. In the generic summarization system if the user doesn't have knowledge about text then information measure equal level in information. Whereas in query-based summarization, before starting of the summarization technique, the topic is verified of the initial text. The system extracts</p> |

| | |
|--|---|
| | <p>that knowledge from the provision text and presents it defines. There are three main approaches to summarization i.e.- statistical, the graph-based, and machine learning approaches. The other approach is a clustering approach.</p> <p>In statistical approaches, researchers are based upon sentence ranking and the important sentences are selected from the given document, regarded as the important summary compression ratio.</p> <p>Graph-based approaches concentrate on the semantic analysis and relationship among sentences. The graph-based approach is used in the representation for text inside documents.</p> <p>Machine learning approaches helps in producing summary by applying machine learning algorithms. This approach deals with the summarization process as a classification problem. Based on the characteristics sentences are divided for summary.</p> |
|--|---|

| | |
|-----------------------------|---|
| <i>Title</i> | Framework of automatic text summarization using Reinforcement learning |
| <i>Authors</i> | Seonggi Ryang, Graduate school of Information science and technology, University of Tokyo Takeshi Abekawa, National institute of informatics |
| <i>Year of Publications</i> | August 2012 |

| | |
|-----------------------|---|
| <p>Summary</p> | <p>Well organized summary is generated of single and multiple documents. Multi-document summarization has become very important part of our daily lives as there is lot of information about one particular topic so it becomes very difficult to read. Summary of document helps to easily understand about the topic and important information is generated. The extractive approach is used which is popular for document summary. Summary is generated by selecting words and sentences from the provided document because it is difficult to guarantee the linguistic quality. Marginal relevance (MMR) is used which is used to score every textual unit and take out the highest score. Greedy MMR algorithm is also used but due to its greediness they don't take into account the whole quality. Global inference algorithm is also used for summary. However, these algorithms create lot of problem in formulation of integer linear programming for scoring and the time complexity is very hard. So</p> |
| | <p>Reinforcement Learning (ASRL), where the summary is generated within framework and scores the function of summary. The method is used and adapts to problem with automatic summarization in natural way. Sentence compression is also adapted as action of framework. ASRL is evaluated which is comparable with the state of ILP-style taking rouge score into consideration. Evaluation is done on basis of execution time. State space is searched efficiently for sub optimal solution underscore functions and the score function, and produce a summary whose score denotes the expectation of the score of the same features' states. The quality of summary only depends on score function.</p> |

| | |
|------------------------------------|--|
| <i>Title</i> | Text Summarization Techniques: A Brief Survey[3] |
| <i>Authors</i> | Mehdi Allahyari, Seyedamin Pouriye, Mehdi Assefi, Saeid Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, Krys Kochut |
| <i>Year of Publications</i> | November 2017 |
| <i>Publishing Details</i> | (IJACSA) International Journal of Advanced Computer Science and Applications |
| <i>Summary</i> | <p>This paper surveys the distinctive procedures for synopsis and portrays the adequacy and inadequacies of the diverse strategies. Content outline helps in shortening a content report into a packed form keeping all the imperative data. Programmed content synopsis is the undertaking of delivering a short rundown while protecting key data</p> <p>Substance and by and large meaning. There are numerous precedents like web crawlers create pieces as the sneak peeks of the archives, Different models helps in incorporating the new sites which helps in delivering packed portrayals of news points more often than not as features to encourage perusing. Programmed content synopsis is troublesome and non paltry errand. Luhn et al presented a technique to choose acclaimed sentences from the content utilizing highlights,</p> |

Chapter 3

SYSTEM DEVELOPMENT

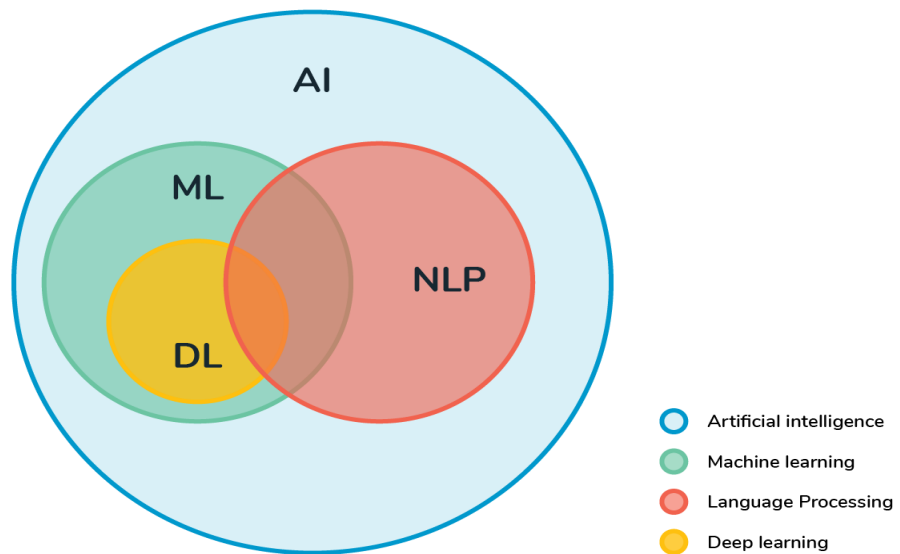


Figure 3.1: Distribution

3.1 Natural Language Processing

- Human success is due to our ability to communicate and share information.
- Oral communication led to painting on walls and caves, which led to the development of language.
- NLP (natural language processing) is a method of communicating with an intelligent system using natural language.
- NLP is a part of computer science and AI that deals with human language.

- NLP can be used to organize massive chunks of text data, automate tasks, and solve problems.
- Examples of NLP tasks include automation, summarization, machine translation, named entity recognition, sentiment analysis, speech recognition, and topic segmentation.
- Only 21% of available data is in structured form; most is unstructured text data.
- NLP can be used for spell-checking, keyword search, extracting information, and sentiment analysis.
- NLP is used in speaker recognition and voice assistants like Siri, Google Assistant, and Cortana.

Natural language understanding (NLU) and the difficulties that machines face when trying to understand language. It explains that there are three types of ambiguities that can make language processing challenging: lexical ambiguity, grammatical ambiguity, and referential ambiguity.

Lexical ambiguity occurs when a word or phrase has multiple potential meanings, such as the word "match" in the sentence "she is looking for a match." It could mean a sports match or a romantic partner, for example.

Grammatical ambiguity occurs when a sentence can be interpreted in multiple ways based on the structure or placement of words. For instance, in the sentence "the chicken is ready to eat," it's unclear whether the chicken is prepared for the speaker to eat or if it's prepared to eat something else.

Referential ambiguity arises when pronouns are used to refer to something without specifying exactly what it is. For example, in the sentence "the boy told his father about the theft, and he was very upset," it's not clear who is upset - the boy, the father, or the thief.

To address these challenges, the passage suggests using the Natural Language Toolkit (NLTK), a platform for building Python programs that work with human language data. NLTK provides easy-to-use interfaces for working with text processing libraries and resources, such as WordNet, which contains lexical data for English.

The passage also explains the process of tokenization, which involves breaking a complex sentence into smaller units called tokens. This process helps to understand the meaning of each word in a sentence and produce a structured representation of the input sentence

3.2 Extractive Summarization Approach

Data cleaning is a crucial step in any Machine Learning project as it ensures that the data fed to an ML model is properly formatted and error-free. Since data can come in various formats and be spread across different systems, it is essential to prepare it appropriately before using it for machine learning. However, data preparation is often a tedious and time-consuming process, with some surveys indicating that data scientists spend up to 80% of their time on this step. Nevertheless, it is also the most important step since the quality of the input data significantly impacts the model's performance.

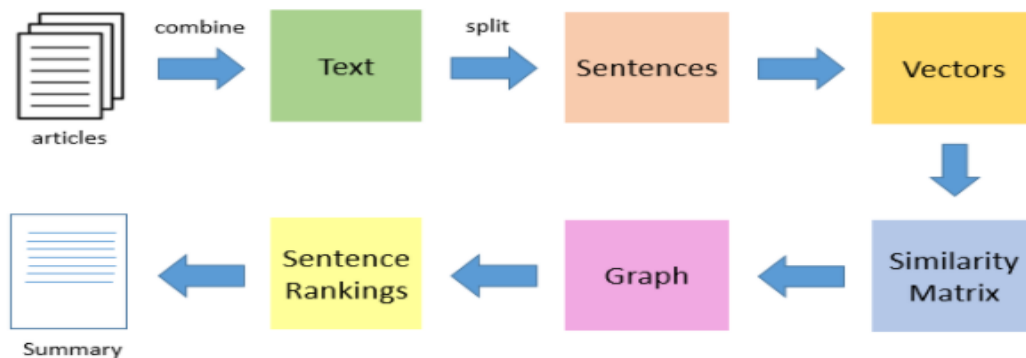


Figure 3.2: Text conversion

Python provides several packages to simplify the data preparation process, including Custom Transformers used in conjunction with Pipelines. In this article, we will explore what custom transformers are and how to code them in a pipeline for mean encoding and shirt-sizing.

3.2.1 Architecture for Smote Algorithm

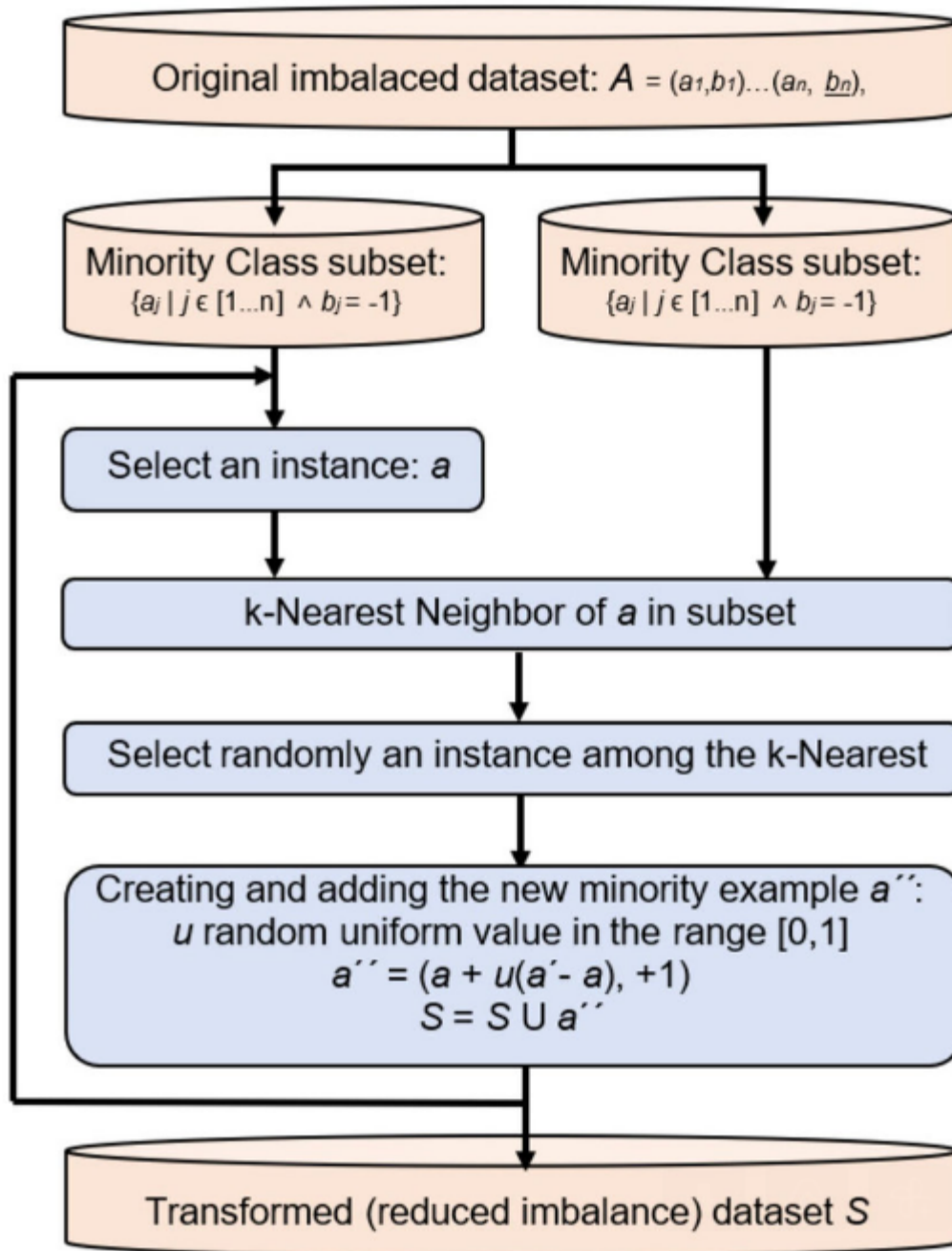


Figure 3.3: Architecture of smote

3.2.2 Transformer and pipelining

If you have experience working with machine learning problems, you are likely familiar with

transformers in Python. These transformers can be utilized to clean, reduce, expand or generate features for your data. The fit method is used to learn parameters from a training set, and the transform method applies these transformations to unseen data. Some examples of predefined transformers include StandardScaler, LabelEncoder, Normalize, etc.

In this article, we will be discussing custom transformers in Python. These are transformers that are tailored to meet specific needs of a machine learning project. They can be designed to implement any custom logic or operations on the data. Custom transformers can help streamline the data preparation process and make it more efficient.

Transformers are an essential part of the machine learning workflow in Python, as they can be used to clean, reduce, expand, or generate features. The fit method of a transformer is used to learn parameters from a training set, while the transform method applies these transformations to unseen data. Predefined transformers such as StandardScaler, LabelEncoder, and Normalize are available in Python, but custom transformers can also be created.

Custom transformers are user-defined transformers that can be tailored to specific needs. They can be used to perform tasks that may not be available in the pre-existing transformers, or to customize the transformation process to suit the requirements of a particular project. By creating custom transformers, users can have more control over the data transformation process and can tailor the process to the specific needs of their project.

In this article, we will delve into the details of creating custom transformers in Python and demonstrate their use in a machine learning pipeline.

Transformers in Python are an essential tool for data preparation in machine learning projects. They can be used to clean, reduce, expand or generate features from data. The fit method is used to learn parameters from a training set, and the transform method is used to apply these transformations to unseen data.

While Python provides many predefined transformers such as StandardScaler, LabelEncoder,

and Normalize, custom transformers can also be created. Custom transformers allow data scientists to create their own feature engineering techniques that are specific to their problem domain. These transformers can be created using Python's scikit-learn library, which provides a base class for transformers called BaseEstimator.

Creating custom transformers can be beneficial in several ways. For example, they can help to:

- Automate data preparation tasks
- Reduce the amount of code required for data preparation
- Create reusable code for data preparation across different projects

In summary, custom transformers are a powerful tool for data preparation in machine learning. By creating your own feature engineering techniques, you can automate data preparation, reduce code, and create reusable code across different projects.

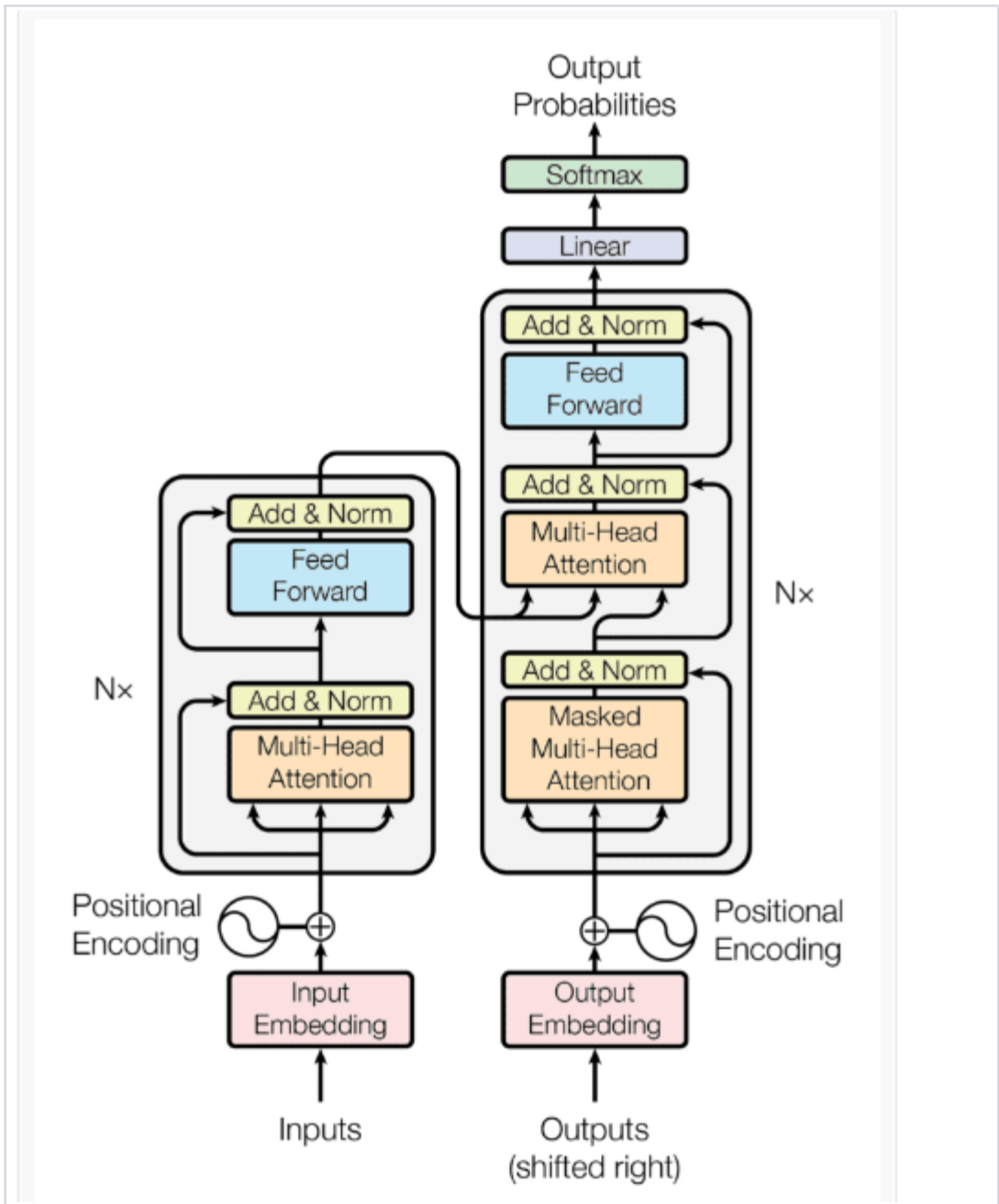


Figure 3.4: Encoder- Decoder architecture of transformer

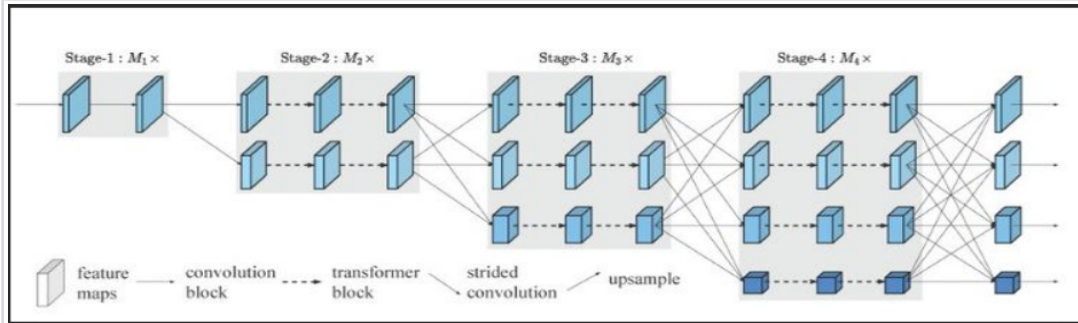


Figure 3.5: Pipeline using Transformer

3.3 SMOTE

In the field of machine learning and data science, imbalanced data distribution occurs when there is a significant difference in the number of observations between classes. This issue is commonly encountered in applications such as fraud detection, anomaly detection, and facial recognition. However, many machine learning algorithms tend to focus on reducing errors and increasing accuracy, without considering class distribution. This can result in biased models that only predict the majority class and have poor performance on the minority class.

Standard machine learning techniques, such as decision trees and logistic regression, often have a bias towards the majority class and ignore the minority class, leading to classifications of the minority class. In technical terms, imbalanced data can cause the model to have low recall for the minority class.

To address this issue, there are several techniques available to handle imbalanced data. Two widely used algorithms are Synthetic Minority Over-sampling Technique (SMOTE) and Near Miss Algorithm. These algorithms can help balance the class distribution and improve model performance on the minority class.

SMOTE, which stands for synthetic minority oversampling technique, is a widely used method for addressing the issue of imbalanced datasets. The goal of SMOTE is to balance the class distribution by oversampling the minority class. This is achieved by creating synthetic examples of the minority class by replicating them.

To generate these synthetic training examples, SMOTE performs linear interpolation between existing minority class instances. Specifically, it randomly selects one or more of the k-nearest neighbors for each minority class example and generates synthetic examples along the line segments connecting them. This results in a set of new, synthetic training examples that are added to the minority class.

Once the oversampling process is complete, the data is reconstructed and can be used for training classification models. This allows for better performance on imbalanced datasets by ensuring that the minority class is adequately represented.

Overall, SMOTE is a powerful tool for addressing the challenge of imbalanced datasets in machine learning. By generating synthetic examples of the minority class, it can help improve the performance of classification models and ensure that all classes are adequately represented in the training data.

SMOTE is an oversampling technique that generates synthetic minority class instances by interpolating between pairs of instances from the minority class. To generate a new synthetic instance, SMOTE selects one of the k nearest neighbors at random and calculates the difference between the feature values of the selected neighbor and the current instance. The difference is then multiplied by a random number between 0 and 1 and added to the feature values of the current instance. This process is repeated until the desired number of synthetic instances is generated.

The key advantage of SMOTE is its ability to balance class distribution without overfitting. However, there are some limitations to the method. For instance, SMOTE may produce noisy samples when the minority class overlaps significantly with the majority class or when outliers are present in the data.

3.3.1 WORKING PROCEDURE OF OVERSAMPLING TECHNIQUE

To begin oversampling, a total number of observations, N , is chosen. Usually, N is chosen to achieve a 1:1 binary class distribution, but it can be adjusted based on specific needs. The oversampling process starts by randomly selecting a positive class instance. Then, the K-nearest neighbors (KNNs), typically set to 5 by default, of this instance are identified. From these KNNs, N instances are randomly chosen to generate synthetic instances.

To create these synthetic instances, the difference in distance between the feature vector of the positive class instance and its neighbors is calculated using a distance metric. Next, a random value between 0 and 1 is multiplied by this difference, and the result is added to the feature vector of the positive instance. This process generates new synthetic instances that can be used to augment the original dataset.

3.3.2 METHODOLOGY

Identify the column(s) containing the minority class: First, you need to identify the column(s) in your dataset that contains the minority class. You can do this by counting the number of samples in each class using the Counter function from the collections module and then identifying the class(es) with the smallest number of samples. Once you have identified the column(s) containing the minority class, you can proceed with applying SMOTE to those column(s) only.

Split the dataset into features and labels: Before applying SMOTE, you need to split your dataset into features and labels. Features refer to the input variables that are used to predict the label, while labels refer to the output variable that you want to predict. In your case, the features are the columns A, B, D, E, F, G, H, I, and K, while the label is the column C.

Normalize the features: It is generally a good practice to normalize the features before applying SMOTE. Normalization ensures that the features are on the same scale, which can improve the performance of machine learning models. You can use the preprocessing.normalize() function from scikit-learn to normalize the features. In your code, you have already done this by normalizing each feature column separately using L1 normalization.

Apply SMOTE on the minority class: Once you have identified the column(s) containing the minority class and normalized the features, you can apply SMOTE to only those columns using the SMOTE() function from the imblearn.over_sampling module. The sampling_strategy parameter should be set to the desired ratio of the number of samples in the

minority class to the number of samples in the majority class after applying SMOTE.

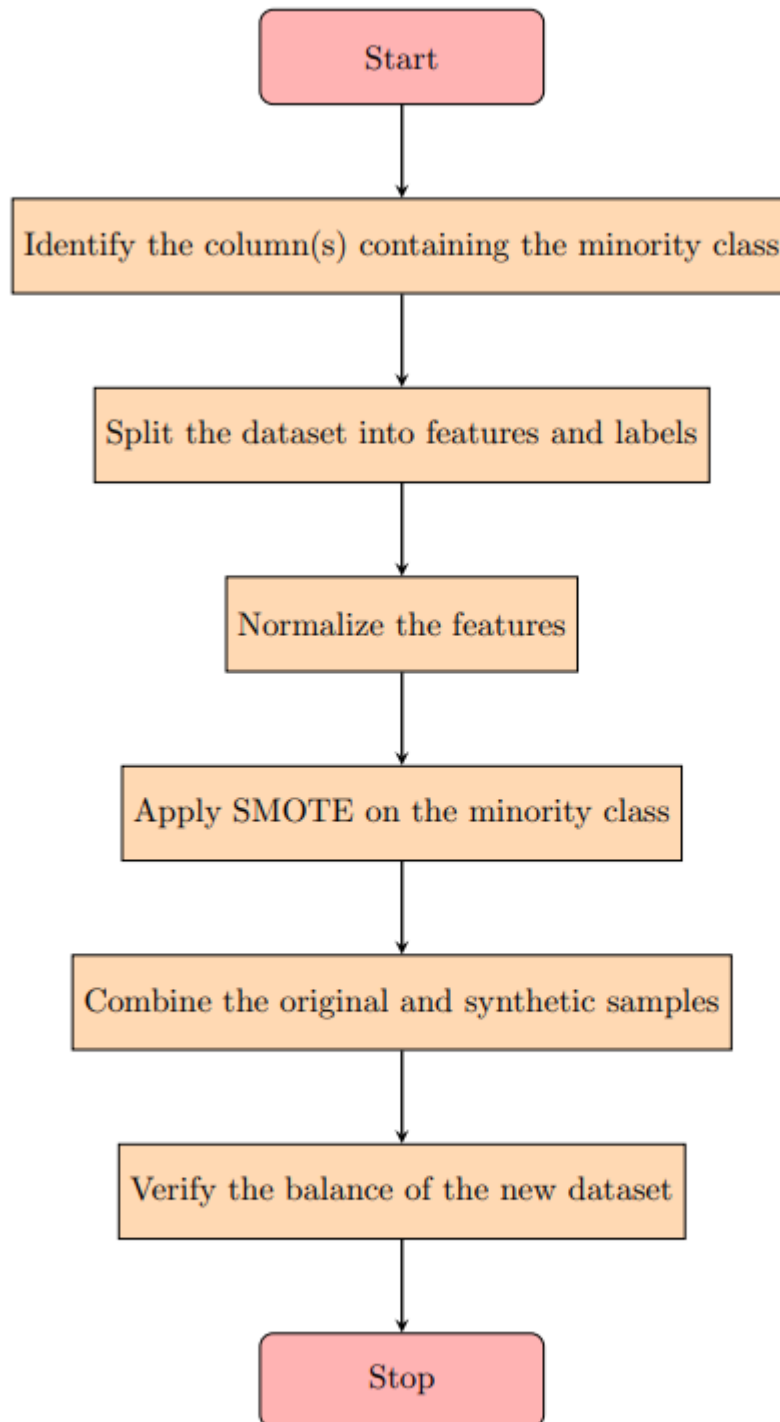


Figure 3.6: Work flow of smote

Combine the original and synthetic samples: After applying SMOTE, you will have a new set of synthetic samples for the minority class. You need to combine these synthetic samples with the original samples to create a new balanced dataset. You can use the `concat()` function from pandas to combine the original and synthetic samples.

Verify the balance of the new dataset: Finally, you should verify the balance of the new dataset to ensure that the number of samples in each class is roughly equal. You can use the Counter function to count the number of samples in each class.

In summary, to apply SMOTE on the column(s) containing the minority class, you need

3.3.3 STEPS INVOLVE WHILE IMPLEMENTING

SMOTE (Synthetic Minority Over-sampling Technique) is a widely used method for dealing with imbalanced datasets. The steps involved in SMOTE are as follows:

Step 1: Identify the minority class samples in the dataset.

Step 2: For each minority class sample, select k nearest neighbors (usually $k=5$) from the minority class samples.

Step 3: For each selected nearest neighbor, generate synthetic examples by interpolating between the original sample and the nearest neighbor. Specifically, for each nearest neighbor, a new synthetic example is generated as follows:

$$x' = x + \text{rand}(0, 1) * (\text{neighbor} - x)$$

where x is the original minority class sample, neighbor is a randomly selected nearest neighbor of x , and $\text{rand}(0,1)$ is a random number between 0 and 1.

Step 4: Repeat steps 2 and 3 until the desired number of synthetic examples is generated.

Step 5: Combine the original minority class samples and the synthetic examples to create a balanced dataset.

The generated synthetic examples are added to the minority class to increase its representation in the dataset. This helps in improving the performance of machine learning models by reducing the bias towards the majority class.

3.3.4 TF-IDF

TF-IDF stands for term frequency-inverse document frequency, and it is a commonly used technique in natural language processing (NLP) for determining the relevance of a term within a document or a corpus of documents.

TF measures how frequently a term occurs in a document, as you mentioned. It is calculated by dividing the number of times a term appears in a document by the total number of terms in the document. TF values increase as the number of occurrences of the term within the document increases, making it an indicator of the term's importance or relevance within the document.

However, simply using TF to measure term relevance has some limitations, as certain terms may appear frequently in a document without necessarily carrying much meaning or significance. This is where IDF comes into play. IDF measures the rarity of a term across all documents in a corpus, and it is calculated by taking the logarithm of the total number of documents in the corpus divided by the number of documents containing the term. Terms that appear in many documents have a low IDF score and are considered less important than terms that appear in only a few documents.

By combining TF and IDF, we can calculate a TF-IDF score that takes into account both the term's frequency within a document and its rarity across the corpus. This score can be used to determine the relevance of a term within a document or to compare the relevance of different terms across multiple documents.

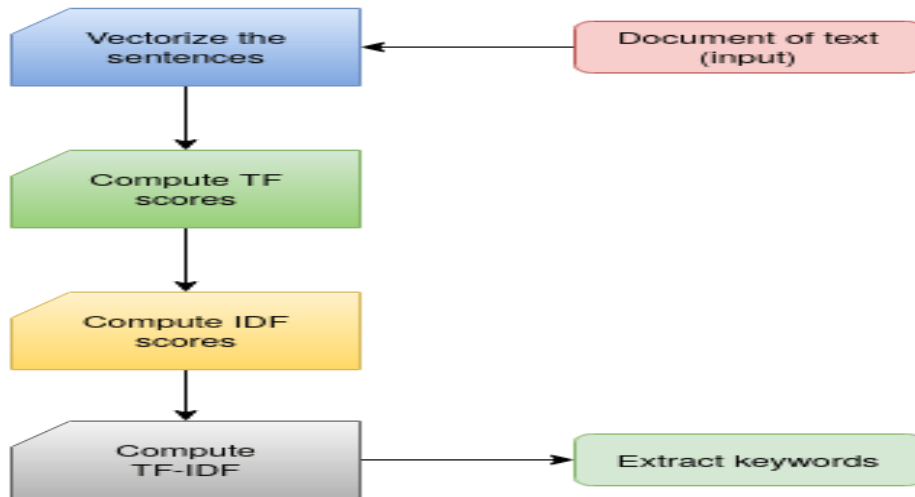


Figure 3.7: Vectorization

3.3.5 Sequence-to-Sequence Modeling

We have built a sequence to sequence modeling. The seq2seq model involves sequential information. Our main objective in this project was to build a text summarizer in which the input will be a long sequence of words and the resultant output would be a short summary.

So we have modeled this as a Many-toMany Seq-2-Seq model architecture. In creation of this project we have taken two major components of Sequence-to-Sequence modeling into action that are:-

1. Encoder
2. Decoder

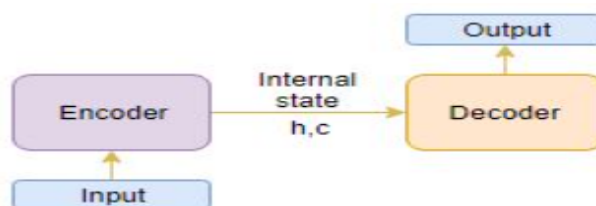


Figure 3.8: Encoder- Decoder

3.3.6 The Encoder-Decoder Architecture of transformer

The Transformer architecture is a popular neural network architecture that is commonly used in natural language processing (NLP) tasks, such as machine translation, text generation, and language modeling.

The Transformer architecture consists of two main components: the encoder and the decoder. The encoder takes an input sequence and maps it to a sequence of continuous representations, while the decoder takes the output of the encoder and generates an output sequence.

In the encoder, the input sequence is first transformed into a sequence of embeddings, which are continuous vector representations of the input tokens. These embeddings are then processed through multiple layers of self-attention and feedforward neural networks to create a sequence of context vectors. These context vectors capture the relationships between the different tokens in the input sequence.

In the decoder, the output of the encoder is used as input along with the previous output token to generate the next output token. The decoder also uses self-attention and feedforward layers to process the input and generate the output sequence. During training, the decoder is trained to predict the next token in the output sequence, given the input and previous output tokens.

Overall, the Transformer architecture has been shown to be highly effective in NLP tasks, achieving state-of-the-art results in several benchmarks. Its encoder-decoder structure and use of self-attention allow it to capture complex dependencies between the input and output sequences and generate high-quality translations and text generations.

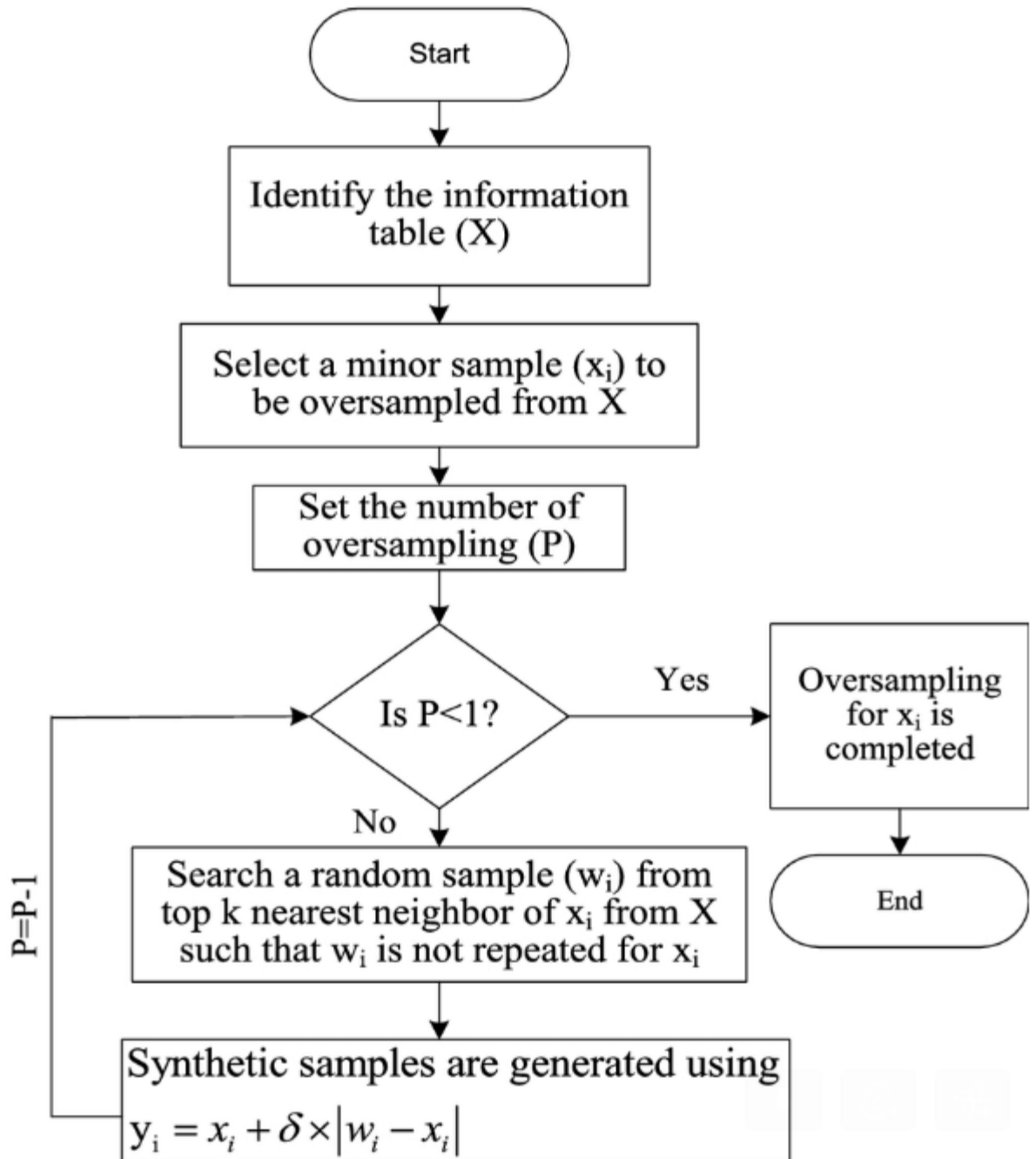


Figure 3.9: Working Mechanism

3.4 Technologies Used

3.4.1 Keras

Keras is a popular deep learning framework developed in 2015, known for its ease of use and flexibility. It provides a wide range of pre-built functions, layouts, and layers, making it straightforward to build neural networks without having to start from scratch. Keras simplifies the process of building and training neural networks, making it an efficient tool for researchers and practitioners alike.

3.4.2 Tensorflow

TensorFlow Hub is a library that facilitates the publication, discovery, and sharing of reusable machine learning models in the form of modules. A module is essentially a pre-trained piece of a TensorFlow graph, including its weights and assets, that can be used for transfer learning. Transfer learning is a machine learning technique that involves leveraging knowledge gained from one task to improve performance on another related task. With TensorFlow Hub, developers and researchers can easily incorporate pre-trained models into their own applications, saving time and resources that would otherwise be required to train their own models from scratch. The modules available on TensorFlow Hub cover a wide range of applications, including natural language processing, computer vision, and audio processing, among others. By using pre-trained modules, developers can quickly create powerful machine learning applications without needing to have deep expertise in each individual domain. Overall, TensorFlow Hub is a valuable resource for the machine learning community, providing a centralized platform for sharing and reusing pre-trained models, and enabling the development of more accurate and efficient machine learning applications.

3.4.3 Scikit-learn

Scikit-learn is a library for ML. It performs various dimensional reduction methods such as PCA(Principal Component analysis), various regression algorithms as well as its also perform regressions

3.4.4 Pandas

Pandas is a platform for data management in the data framework. It contains data analysis tools written with Python, it has numerous tools for testing missing,resize data structures, etc.



Figure 3.10: Technologies used

3.4.5 Imblearn

Class imbalance is a common issue in machine learning where we have an unequal proportion of classes. For instance, in a dataset, we may have 500 records of the 0 class and only 200 records of the 1 class, which is called class imbalance. This can cause the model to be biased towards the majority class and affect the precision and recall of the model. To address this issue, we can use Imblearn techniques, which are specifically designed to handle class imbalance. These techniques can either upsample the minority class or downsample the majority class to achieve a balanced dataset. Upsampling involves increasing the number of samples in the minority class by randomly replicating them. This is done until the number of samples in the minority class matches the number of samples in the majority class. On the other hand, downsampling involves reducing the number of samples in the majority class by randomly removing samples. This is done until the number of samples in the majority class matches the number of samples in the minority class. To implement these techniques, we can use the Imblearn library, which provides several functions for upsampling and downsampling.

For example, we can use the `RandomOverSampler` function for upsampling and the `RandomUnderSampler` function for downsampling. In conclusion, Imblearn techniques are essential for handling class imbalance issues in machine learning. By using these techniques, we can create a balanced dataset and ensure that our model does not get biased towards the majority class. Imblearn techniques are methods used to address imbalanced data sets, where one class has significantly more data points than the other. This can lead to biased predictive models that perform poorly on the minority class. The two main approaches to address this issue are upsampling and downsampling. Upsampling involves generating synthetic data points for the minority class to increase its representation and match the ratio of the majority class. Downsampling involves reducing the number of data points in the majority class to match the minority class. By using these imblearn techniques, we can create a more balanced data set that enables our predictive models to generalize well and make accurate predictions for both classes. It's important to note that while upsampling and downsampling can be effective techniques, they need to be used carefully to avoid overfitting or underfitting the data.

3.4.6 Sklearn

Scikit-Learn is a Python library for machine learning that is available free of charge. It offers a broad range of algorithms for supervised and unsupervised learning, including classification, regression, clustering, and dimensionality reduction. The library is constructed using well-known Python libraries like NumPy and SciPy, and it integrates easily with other popular libraries such as Pandas and Seaborn.

3.4.7 Numpy

NumPy is a Python programming language library, which adds support for large, multi-sided arrays and matrices, as well as a large collection of advanced mathematical functions to work on these components.

3.5 Mathematical Model Development

3.5.1 Term Frequency

The word frequency is a very important factor. TF (term frequency) represents the frequency with which a word appears in a document (usually a press function such as square root or logarithm used) to calculate the word usually. A word that indicates the boundaries of sentences in a document is based on punctuation such as (. (, “, [, {, etc.) and divided into sentences. These sentences are nothing out of the ordinary tokens.

3.5.2 Keyword Frequency

Keywords are high-frequency words that occur frequently in a document or corpus. To identify keywords, the document is first preprocessed by removing stopwords, punctuations, and other noise words. Then, the frequency of each remaining word is calculated, and the words with the highest frequencies are selected as keywords.

3.5.3 Stop word filtering

When analyzing text, there are certain words, such as 'the', 'and', 'exist', and 'on', that appear frequently but do not provide much meaning to the document. While these words are useful for searching, they are not typically the words users search for when asking questions.

3.5.4 K means clustering approach

The process of oversampling is used to balance datasets with imbalanced class distributions. Traditional cluster-based oversampling algorithms can lead to issues such as the loss of samples from certain classes and the creation of new boundary samples. To address these concerns, a new cluster-based oversampling algorithm called KNSMOTE has been proposed

in a recent paper.

The KNSMOTE algorithm works by clustering the original dataset using the k-means algorithm. Once the dataset has been clustered, the algorithm identifies "safe samples" by using the class discriminant function to filter out any potentially problematic samples. The safe samples are then used to oversample the minority class in a way that preserves the original distribution of samples.

One of the advantages of KNSMOTE is that it reduces the likelihood of losing important samples from the minority class while also minimizing the creation of new boundary samples. This can help to improve the overall accuracy of machine learning models that are trained on imbalanced datasets.

It's important to note that while the KNSMOTE algorithm is an improvement over traditional cluster-based oversampling algorithms, it is not a one-size-fits-all solution. The effectiveness of any oversampling method will depend on the specific characteristics of the dataset being used. Therefore, it's always a good idea to evaluate different oversampling methods and choose the one that works best for the particular problem at hand.

Given the observation set (x_1, x_2, \dots, x_n) , where each observation is a real d -dimensional vector, the addition of k means the marked division into k sets ($k \leq n$) $S = \{S_1, S_2, \dots, S_k\}$ to reduce the total number of squares within a set (WCSS):

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \sum_{x_j \in S_i} \|x_j - \mu_i\|^2$$

Figure 3.11: Minimum k value

Emotional polarity by category is a crucial issue in sentiment analysis. To address this, a database of over 5.1 million product reviews from Amazon.com across four categories was analyzed. The max-entropy POS tagger was used to tokenize the sentences, and a Python program was employed to expedite the process. Words that express contradictions, such as 'no', 'not', and 'some', were included in the lexicon, and the negation of adjectives and verbs was utilized to identify phrases. Various classification models were selected by class, including Naïve Bayesian, Random Forest, Logistic Regression, and Support Vector Machine.

With the component determination, Pang and Lee recommended that the deliberate sentences be eliminated by material result. Propose a text-sharing interaction that is natural recognizing emotional substance utilizing little cuts. Gann et al. chosen 6,799 tokens dependent on Twitter information, where every token is given passionate focuses, which is TSI , which shows itself as a positive token or a negative token.

The total sentiment index is calculated by :-

$$TSI = \frac{p - \frac{tp}{tn} \times n}{p + \frac{tp}{tn} * n}$$

Figure 3.12: Total sentimental index

p : number of times a token appears in positive review

n : number of times a token appears in negative review

$\frac{tp}{tn}$: ratio of total number of positive to negative reviews

3.5.5 Frequency Analysis

To prioritize activities based on a summary of content, it is important to identify keywords that appear frequently and compare them with other words in the text. This allows for the identification of important sentences based on word repetition. Many summary systems use this method for extracting sentences. Two popular approaches use repetition as a key step in the summarization process: word frequency and sentence similarity based on repeated terms.

3.5.6 Word Probability

It was expected that one of the most complex ways to use repetition was done by which includes the absurd repetition of a word, that is, by combining the whole word event archive. However, actions are strongly influenced by the length of the report. One way to get a report duration change is to process the word opportunities.

The following gives the probability of particular word :-

$$f(w) = \frac{n(w)}{N} \tag{1}$$

Where:

$n(w)$ = The frequency count of the word w in the document
 N = The total number of words in the document

Figure 3.13: probability of particular word

The findings of the test were transmitted based on a man-made framework showing that people, in general, will use word repetition to determine the key archive studies. In the case of a summary of the framework, misuse the word opportunities to make frames. The Sum Basic framework begins to process the word as it may originate from a database. Each sentence considers the weight of the sentence as volume voice opportunities. Excellent points are taken on the basis of sentence weight.

$$\text{weight}(S_j) = \frac{\sum_{w \in S_j} f(w)}{|\{w | w \in S_j\}|}$$

Figure 3.14: Sentence weight.

3.5.8 Term Frequency-Inverse Document Frequency

Frequency of terms in text is commonly used to extract useful information from a large corpus of documents. However, it is important to consider if all frequently appearing keywords are equally important. For example, a collection of news articles about an earthquake disaster will naturally contain the word "earthquake" in all of them. Therefore, to determine the significance of a term, tf-idf reduces the weight of a term based on its frequency in the entire document collection, rather than just its frequency within a single document. This approach has made tf-idf one of the most widely used techniques for extractive summarization. In mathematical Terms Frequency can Be defined IN the following way :-

$$tf_{i,j} = \frac{n_{i,j}}{\sum n_j} \quad ($$

Where:

$n_{i,j}$ represents the frequency count of the word i document j .

Figure 3.15: Terms Frequency index

Each word is separated and matched by a combined number of different words in text j . The word is used to measure a number similar to the word probabation Number given in Statement 1. The IDF of a word is processed as:-

$$idf_i = \log \frac{|D|}{|\{d | t_i \in d\}|}$$

Figure 3.16: Inverse Terms Frequency index

Chapter 4

Performance Analysis

4.1 Analysis Of System Developed

Comparing the Accuracy of our Machine learning project imbalance dataset to balance dataset Model having Sentiment Analysis with five of the algorithms namely Logistic Regression Algorithm accuracy 78% ,Decision tree 83%,Random forest 83.1%, Ada boost 77.9% and Linear Discriminant analysis 78.9%.

Tabular flow of performance matrix

| S.No. | NAME OF CLASSIFIERS | PERFORMANCE MEASURE |
|--------------|----------------------------------|----------------------------|
| 1. | LOGISTIC REGRESSION | 0.788 |
| 2. | DECISION TREE | 0.830 |
| 3. | RANDOM FOREST | 0.831 |
| 4. | ADA BOOST | 0.779 |
| 5. | LINEAR DISCRIMINANT ANALY | 0.789 |

4.2 Training Dataset

This dataset is constructed by a phenomenal team by snap what is this snap this particular data set we can use freely for research for research and education we can make use of this particular data set this is actually it is related to if we speak strictly it is related to amazon fine food products now if we observe carefully how many how let me see top top level details of this particular data set how many reviews are there there are 5 lakhs it must be taken into consideration that this is only sample not entire data set there are five lakh sixty eight thousand four fifty four reviews and two lakhs fifty six thousand fifty nine gave these mini reviews this and number of products seventy four thousand two fifty eight this is in between 1999 to 2012 almost maybe it's approximately.

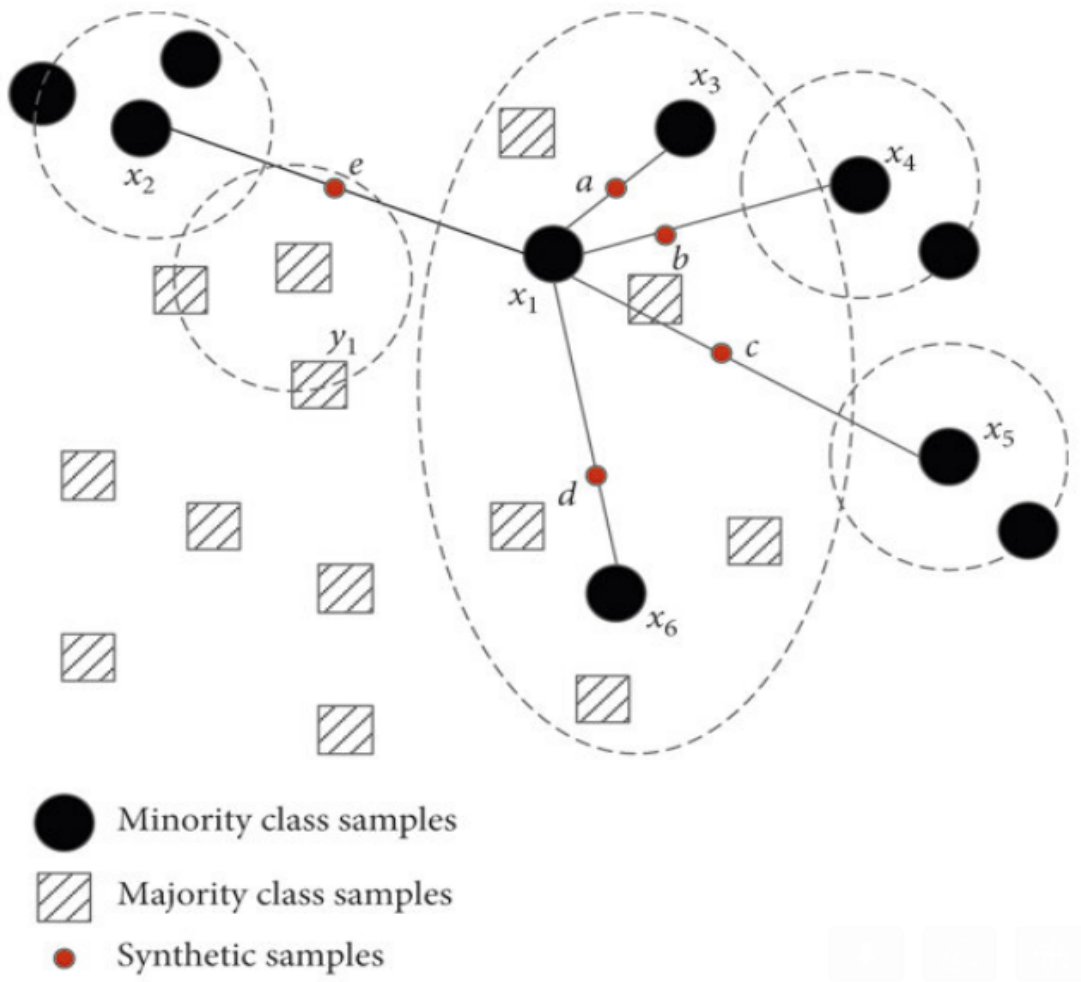


Figure 4.1: Data distribution in smote

4.3 Use Case Diagram

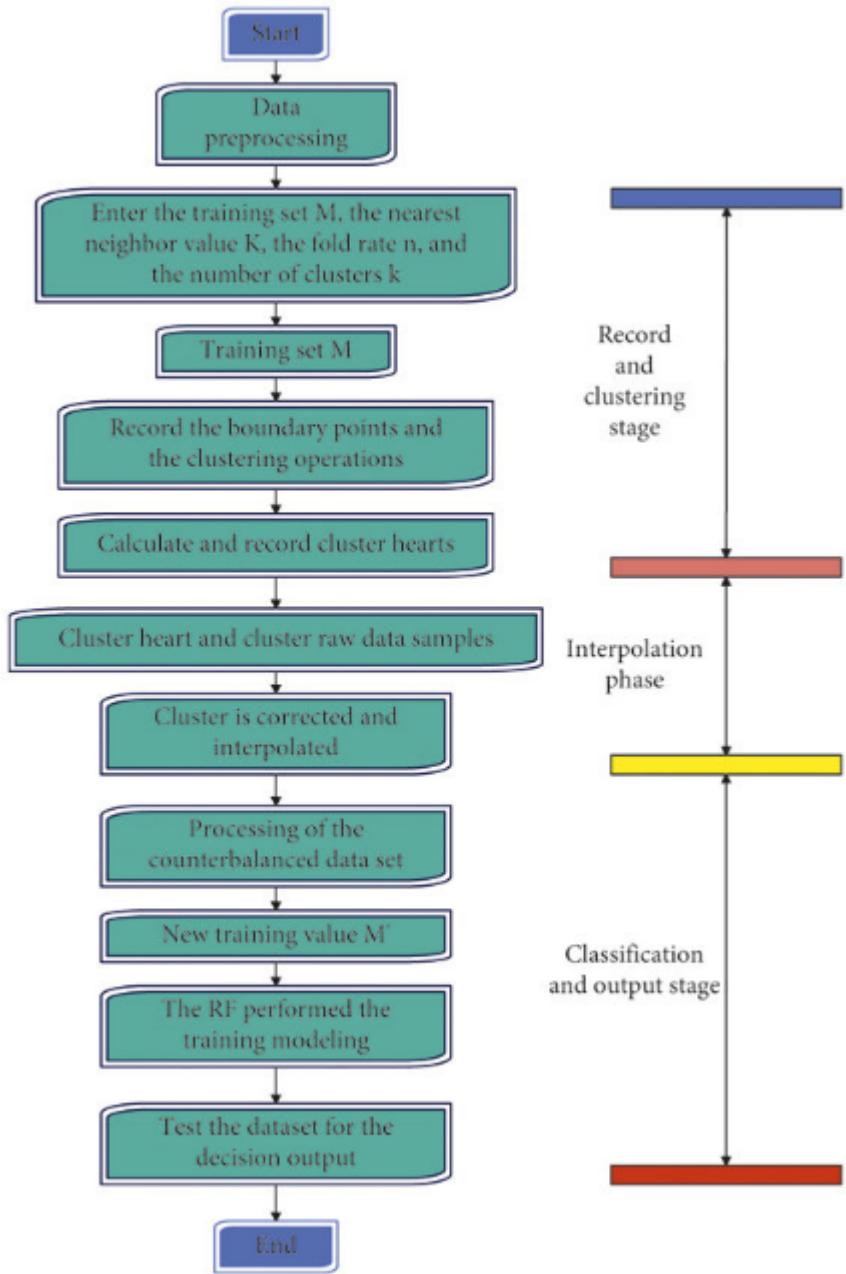


Figure 4.2: Training and testing in smote

4.4 CLASSIFIERS USED

4.4.1 LOGISTICS REGRESSION

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1.**
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems.**
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

$$P = \frac{e^{a+bX}}{1 + e^{a+bX}}$$

Figure 4.3: Estimated probability

- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

- o Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification.

```
Confusion Matrix
[[2656  529]
 [ 534 1312]]

Accuracy
0.7887099980123236
```

Figure 4.4: Confusion matrix for logistic regression

4.4.2 DECISION TREE CLASSIFIER

Decision Tree is a popular supervised learning algorithm used for solving both classification and regression problems. It represents a tree-like structure where the internal nodes represent the features, branches represent the decision rules, and each leaf node represents the outcome. This algorithm is mostly used for solving classification problems.

There are two types of nodes in the decision tree: Decision Node and Leaf Node. Decision nodes are used for making decisions and have multiple branches, while leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the tests

are performed based on the features of the given dataset.

Number of classes: C

Number of data points: N

Number of data points of class i : N_i

$$I_G = \sum_{i=1}^C \frac{N_i}{N} \left(1 - \frac{N_i}{N} \right)$$

true
class

wrong
prediction

Figure 4.5: Formula for information gain

```
Confusion Matrix
```

```
[[2594  591]  
 [ 260 1586]]
```

```
Accuracy
```

```
0.830848737825482
```

Figure 4.6: Confusion matrix for decision tree

4.4.3 RANDOM FOREST

Random forest is a machine learning algorithm that utilizes an ensemble of decision trees to make predictions. The algorithm generates a large number of decision trees, each of which produces a class prediction. The random forest then selects the class with the most votes as the final prediction for the model. By using an ensemble of decision trees, random forest can improve prediction accuracy and reduce overfitting.

Random forests are a type of machine learning algorithm that use an ensemble of decision trees to improve the accuracy of predictions. Unlike a single decision tree, a random forest generates many decision trees, and the final output is the average prediction made by all the trees.

The effectiveness of random forests lies in the fact that each tree is built using a random subset of the available features, and the trees are trained independently of each other. This randomness results in low correlation between the individual trees, which is important for the ensemble to work effectively.

If the individual trees in the random forest are highly correlated, they will make similar errors, which can cause the ensemble to perform poorly. Therefore, it's crucial to use a diverse set of features and ensure that the trees are not overly dependent on any particular feature.

$$MSE = \frac{1}{N} \sum_{i=1}^N (f_i - y_i)^2$$

Where N is the number of data points,
 f_i is the value returned by the model and
 y_i is the actual value for data point i .

Figure 4.7: Mean squared error

Another important prerequisite for random forests to perform well is the presence of a signal in the data. If there is no signal or pattern in the data, random forests will not be able to improve prediction accuracy beyond random guessing.

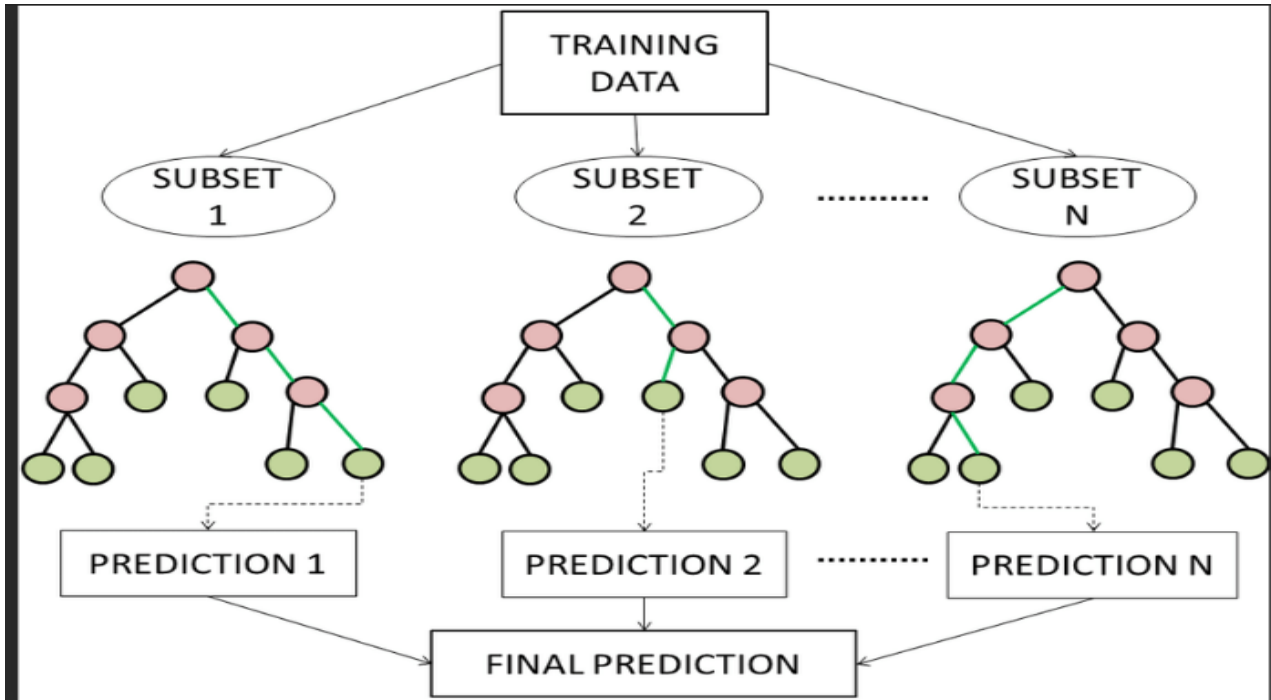


Figure 4.8: Distribution tree

In summary, the effectiveness of random forests depends on the diversity of the features used to train the individual trees and the low correlation between the trees' predictions. As long as these prerequisites are met, random forests can be a powerful tool for improving the accuracy of machine learning models.

```
Confusion Matrix
[[2595  590]
 [ 259 1587]]

Accuracy
0.8312462731067383
```

Figure 4.9: Confusion matrix for random forest

4.4.4 ADA BOOST

AdaBoost is an Ensemble Method in Machine Learning that uses Boosting, a technique that aims to reduce both bias and variance in supervised learning. The Adaptive Boosting aspect of AdaBoost refers to how weights are assigned to each instance, with higher weights given to incorrectly classified instances. The algorithm works by sequentially growing learners, with each subsequent learner built on previously grown learners. The idea is to convert weak learners into strong ones. AdaBoost follows the same principle as boosting, but with some differences.

Boosting is a machine learning technique that combines multiple weak models to create a strong model. The AdaBoost algorithm is a specific type of boosting that uses decision stumps, which are decision trees with a single split, as the weak models.

The AdaBoost algorithm starts by assigning equal weights to all training samples. It then trains a decision stump on the data and evaluates its performance. The samples that the stump misclassifies are assigned a higher weight, while the correctly classified samples are assigned a lower weight. This process is repeated for a predetermined number of iterations, with each iteration creating a new decision stump that focuses on the misclassified samples from the previous iteration.

In each iteration, the weights of the samples are adjusted based on the errors made by the previous stump, and the next stump is trained to focus on the samples that were previously misclassified. The final model is a weighted combination of all the decision stumps, with each stump's weight determined by its accuracy on the training data.

AdaBoost is a powerful algorithm that is widely used in machine learning applications such as face recognition, object detection, and speech recognition. It is known for its ability to handle complex data and produce highly accurate results.

In Python, implementing the AdaBoost algorithm is relatively easy and can be done in just a few lines of code. To use the AdaBoost classifier, you need to import it from the sci-kit learn library. Before applying AdaBoost to any dataset, you should first split the data into training and testing sets. The training data should include both the input and output data.

Once you have your training data, you can use it to train your AdaBoost model. After training the model, you can use it to predict the results on the test data. The test data should only include the input data, and the model will predict the output.

To evaluate the accuracy of the model, you can compare the actual output of the test data with the output predicted by the model. This will help you determine how well your model is performing and how much accuracy can be achieved based on the problem statement. For example, in medical problems, accuracy should be above 90%, while a 70% accuracy is generally considered good.

AdaBoost is a popular ensemble technique that can be used for both classification and regression problems. While it can be used for both types of problems, it is often used for classification problems. AdaBoost improves model accuracy by using a weight-assigning technique after each iteration, which sets it apart from other boosting algorithms.

It is often recommended to try decision trees and then random forest before implementing AdaBoost, as the accuracy of the model tends to increase in this sequence. AdaBoost is an excellent algorithm for improving model accuracy and is a popular choice for many data scientists.

```
Confusion Matrix
[[2666  519]
 [ 591 1255]]

Accuracy
0.7793679189028027
```

Figure 4.10: Confusion matrix for Ada Boost



Algorithm Adaboost - Example

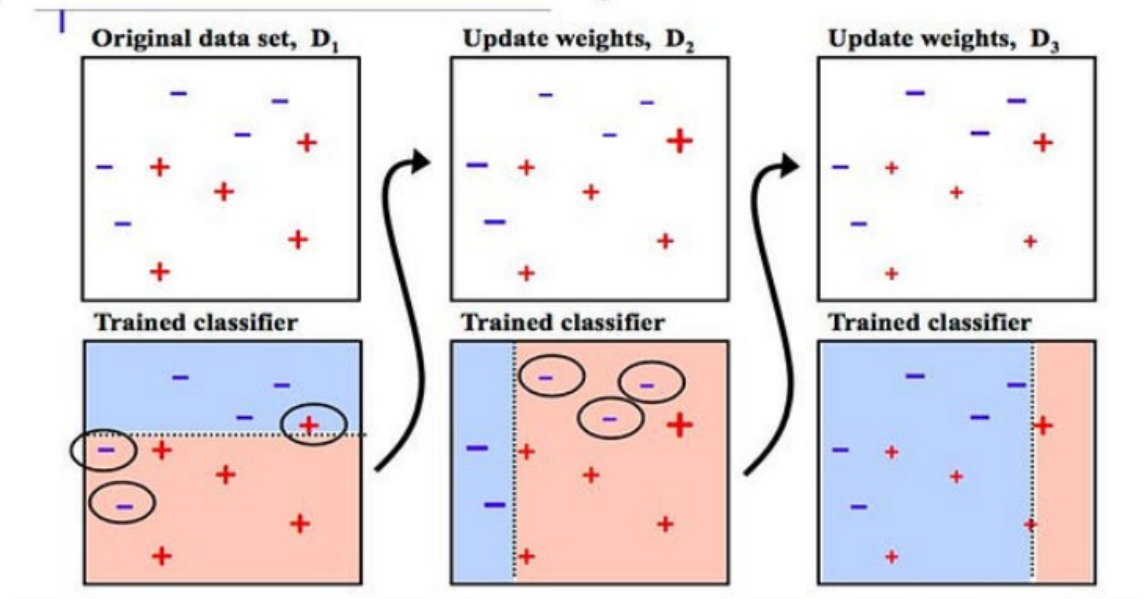


Figure: 4.11: Training and testing in Ada Boost

4.4.5 LINEAR DISCRIMINANT ANALYSIS

Linear Discriminant Analysis (LDA) is a commonly used supervised learning algorithm for dimensionality reduction and classification problems. It is a statistical technique that analyzes and models the differences between the classes of the target variable by projecting the original data into a lower-dimensional space.

The goal of LDA is to find the linear combination of features that maximally separates the classes while minimizing the within-class variance. This is done by maximizing the ratio of the between-class variance to the within-class variance.

Compared to logistic regression, which is a commonly used binary classification algorithm, LDA can handle multi-class classification problems. LDA assumes that the data follows a normal distribution and that the variance-covariance matrix is the same across all classes.

To use LDA for classification, the algorithm first estimates the parameters of the model based on the training data. Then, it applies the learned transformation to the test data to project it

into the lower-dimensional space. Finally, it predicts the class of the test data based on the discriminant function, which is a linear combination of the features that separates the classes.

To prepare the data for LDA, it is important to ensure that the data is properly scaled and that any categorical variables are converted into numeric values. Additionally, LDA assumes that the data is normally distributed and that the variance-covariance matrix is the same across all classes, so it is important to check for these assumptions before applying the algorithm.

There are also several extensions to LDA, such as Quadratic Discriminant Analysis (QDA) and Regularized Discriminant Analysis (RDA), which relax some of the assumptions of LDA and can provide improved performance in certain situations.

HOW DOES IT WORKS

Linear Discriminant Analysis (LDA) is a supervised learning algorithm that is used for classification tasks and dimensionality reduction. It works by finding a linear combination of features that best separates the classes in the data.

In the case of a 2D plane, LDA can find a straight line (known as the decision boundary) that separates the two classes of data points. By projecting the data onto this line, LDA can transform the 2D plane into a 1D plane, which can be more easily visualized and analyzed.

The goal of LDA is to maximize the separation between the classes while minimizing the variance within each class. By doing so, it can effectively reduce the dimensionality of the data while still preserving the discriminatory information between the classes.

Overall, LDA is a powerful tool for reducing the dimensionality of high-dimensional data while preserving the important features that are necessary for accurate classification.

Logistic regression is a popular algorithm used for binary classification, but it can face difficulties when dealing with multiple classes, especially if the classes are well-separated. In contrast, Linear Discriminant Analysis (LDA) is an efficient algorithm for both binary and multi-class classification problems.

LDA is a dimensionality reduction technique that can be used to preprocess data and reduce the number of features in the dataset, just like Principal Component Analysis (PCA). This can significantly reduce the computational cost of running algorithms on large datasets.

LDA is also commonly used in face detection algorithms. In the Fisherfaces method, LDA is used to extract the most useful information from different faces, which is then combined with

Eigenfaces to produce an effective face recognition system. This technique has been used in various real-world applications, including security systems and digital image processing.

Overall, LDA is a versatile and powerful technique that can be used in various applications, including classification, data preprocessing, and face recognition. It is a valuable tool for data scientists and machine learning engineers who are working with high-dimensional datasets and need to reduce the dimensionality of their data while preserving the most important features.

```
Confusion Matrix
[[2652  533]
 [ 527 1319]]

Accuracy
0.789306300934208
```

Figure 4.12:Confusion matrix for Linear discriminant analysis



4.5 RESULT AND OUTPUTS

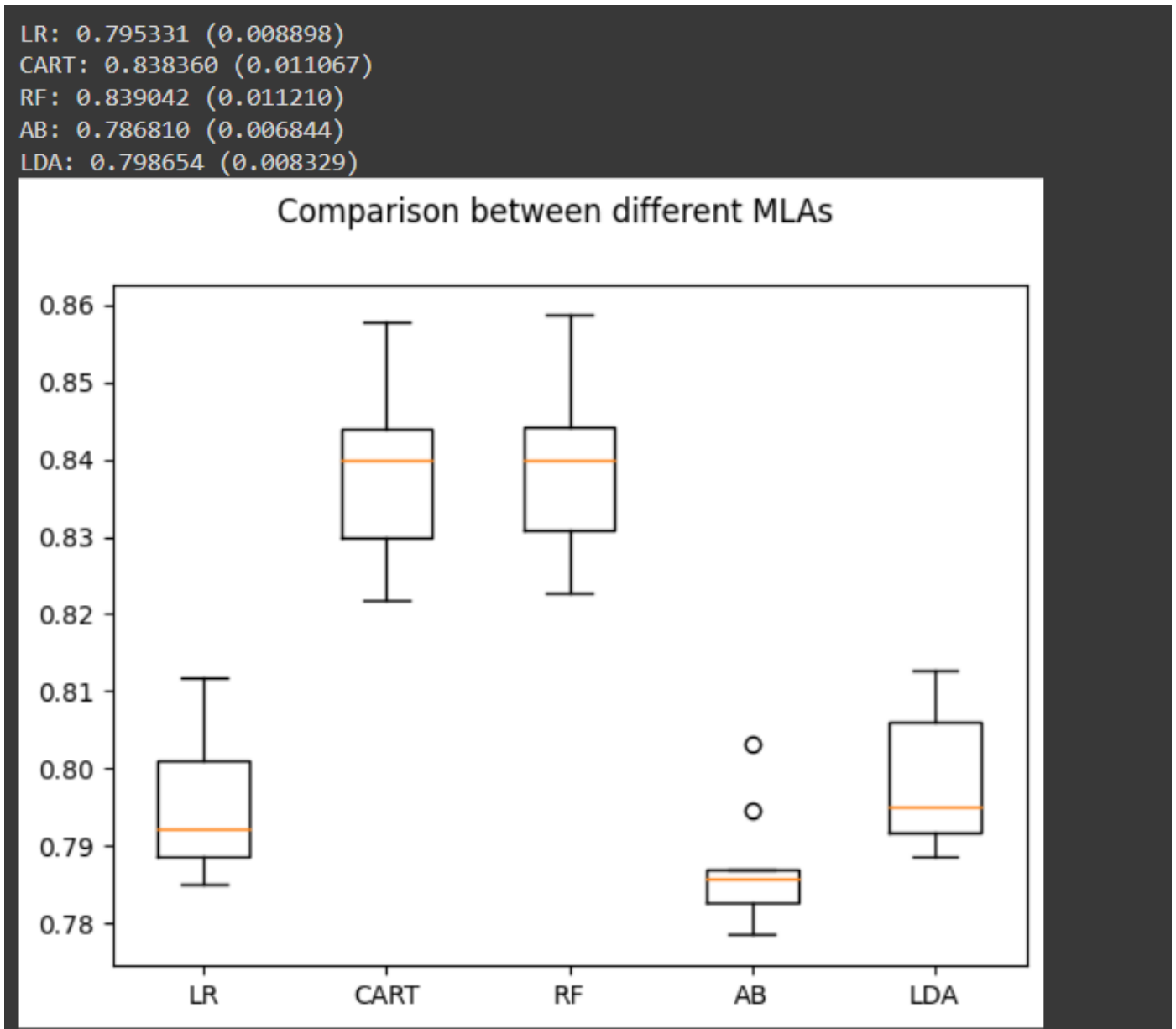


Figure 4.13: Comparison between different MLA

```
Model Accuracy:-

Logistic Regression: 78.87%
Decision Tree: 83.08%
Linear Discriminant Analysis: 78.93%

Averaging Method:-
Random Forest: 83.12%

Boosting Method:-
AdaBoost: 77.94%

Voting Classifiers:-
Voting Classifier without Weights: 82.35%
Voting Classifier with Weights: 82.27%
```

Figure 4.14: Accuracy of all models applied

```
# Voting Classifier with weights
vc1 = VotingClassifier(estimators=[('logreg',logreg),('dectree',dectree),('ranfor',ranfor),('abc',abc),('lda',lda)],
                      voting='soft', weights=[2,1,2,1,2])
vc1.fit(x_train, y_train)
```

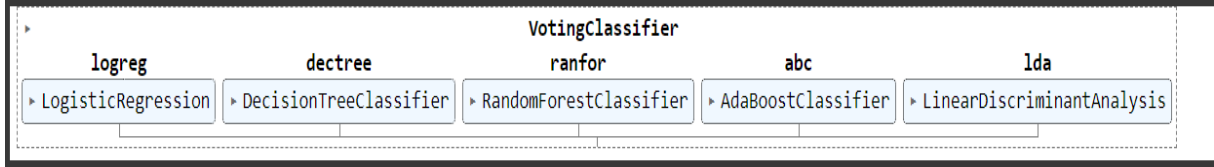


Figure 4.15: Voting classifiers with weights

Chapter 5

CONCLUSION

5.1 Conclusion

Imbalanced datasets are a common challenge in machine learning, where the distribution of samples across different classes is skewed. This often results in poor performance of classifiers on the minority class, which may be of greater interest in many real-world scenarios. Oversampling is one of the effective methods to address class imbalance, where synthetic samples are generated to balance the dataset. However, oversampling methods may suffer from several issues such as noisy samples, biased boundary samples, and lack of diversity in synthetic samples.

To address these issues, a new oversampling method called GA-SMOTE has been proposed in this paper. GA-SMOTE is an improved version of the popular oversampling method SMOTE, which uses a genetic algorithm (GA) to optimize the noise processing scheme. The GA-SMOTE algorithm assigns different sampling weights to each sample, with higher weights assigned to samples closer to the decision boundary. This ensures that the synthetic samples are generated in regions where they are most needed.

Furthermore, the GA-SMOTE algorithm divides the raw dataset into multiple sub-clusters using K-means clustering, and applies an intra-cluster neighborhood triangular sampling method to improve the diversity of synthetic samples. This approach helps to ensure that the synthetic samples generated in each sub-cluster are representative of the underlying data distribution.

Experiments conducted on several benchmark datasets demonstrate that GA-SMOTE outperforms five other state-of-the-art oversampling methods in terms of classification performance on imbalanced datasets. Overall, GA-SMOTE provides a promising solution to

address the challenges of class imbalance in machine learning, by combining the strengths of SMOTE and GA to improve the quality and diversity of synthetic samples.

5.2 Future Scope

Artificial intelligence and big data classification technology have provided significant help in medical auxiliary diagnosis research. However, the class-imbalance problem in medical big data often affects the classification performance of standard learning algorithms. The SMOTE algorithm is commonly used to address this issue by generating sample points randomly. However, its application is often limited by the marginalization of data and blind parameter selection.

To address this problem, a new improved SMOTE algorithm based on the Normal distribution is proposed in this paper. The algorithm generates sample points that are closer to the center of the minority sample, with a higher probability of avoiding the marginalization of the expanded data. The proposed algorithm is tested on imbalanced datasets from Pima, WDBC, WPBC, Ionosphere, and Breast-cancer-wisconsin, and its classification performance is found to be better than that of the original SMOTE algorithm.

Moreover, the paper analyzes the parameter selection of the proposed algorithm and finds that the classification performance is best when appropriate parameters are selected to maintain the distribution characteristics of the original data. Overall, the proposed algorithm provides an effective solution to the class-imbalance problem in medical big data classification, and its improved performance can aid in more accurate auxiliary diagnosis.

5.3 Applications

In data science, it is common to encounter imbalanced datasets in classification problems such as sentiment analysis, medical imaging, and predictive analytics. An imbalanced dataset is one

where there is an unequal distribution of instances or data points among the different classes. This means that one class may have significantly fewer instances than the others, resulting in an imbalance between the classes.

The class with fewer instances is referred to as the minority class, while the class with a larger number of instances is known as the majority class. Imbalanced datasets can pose a challenge to machine learning algorithms because they tend to be biased towards the majority class, leading to poor performance in predicting the minority class.

To overcome this challenge, several techniques can be used, including oversampling the minority class, undersampling the majority class, or using a combination of both. Other methods include using cost-sensitive learning algorithms that assign higher misclassification costs to the minority class or using ensemble techniques that combine multiple models to improve performance.

It is essential to address imbalanced datasets in classification problems as it can impact the accuracy and reliability of the model's predictions. By applying appropriate techniques to handle imbalanced datasets, data scientists can improve the performance of their machine learning models and make more accurate predictions.

REFERENCES

- [1]. A. Srivastava, "SMOTE: Synthetic Minority Over-sampling Technique for Imbalanced Data," *Towards Data Science*, Aug. 2019.
- [2]. C. Cortes, M. Mohri, and A. Rostamizadeh, "Learning Nonlinear Combinations of T kernels by Composing Kernels," arXiv:1106.1813 [cs.LG], Jun. 2011.
- [3]. M. M. Ahmed and R. Al-Otaibi, "Robust Semantic Segmentation Using CNNs for Land Use Classification," *2020 IEEE Saudi International Electronics, Communications and Photonics Conference (SIECPC)*, Riyadh, Saudi Arabia, 2020, pp. 1-5, doi: 10.1109/SIECPC51257.2020.9326603.
- [4]. B. Fernández-Navarro, M. J. del Jesus, F. Herrera, "On the use of MapReduce for imbalanced big data using Random Forest," *Expert Systems with Applications*, vol. 105, pp. 146-164, Apr. 2018.
- [5]. J. Wu, X. Zhang, Y. Shi and H. Qi, "Sentiment Classification Based on Deep Learning in Microblogging," *2020 IEEE 7th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, Nanjing, China, 2020, pp. 85-90, doi: 10.1109/CCIS49450.2020.9094859.
- [6]. A. E. M. Ali, "An Overview of Deep Learning for Curious People," *2021 IEEE 13th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, Dalian, China, 2021, pp. 478-482, doi: 10.1109/IHMSC51987.2021.9453215.

