IMBALANCED LEARNING TECHNIQUES: EXPERIMENTS ON NCAA COLLEGE BASKETBALL LEAGUE PLAYER STATISTICS DATASET

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EMİR GÜLER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
STATISTICS

SEPTEMBER 2022

Approval of the thesis:

IMBALANCED LEARNING TECHNIQUES: EXPERIMENTS ON NCAA COLLEGE BASKETBALL LEAGUE PLAYER STATISTICS DATASET

submitted by **EMİR GÜLER** in partial fulfillment of the requirements for the degree of **Master of Science** in **Statistics, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Özlem İlk Dağ
Head of the Department, **Statistics** _____

Prof. Dr. Barış Sürücü
Supervisor, **Statistics, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Şükrü Acıtaş
Statistics, ESTU _____

Prof. Dr. Barış Sürücü
Statistics, METU _____

Assoc. Prof. Dr. Zeynep Işıl Kalaylıoğlu Akyıldız
Statistics, METU _____

Date: 02.09.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name Last name : Emir Güler

Signature :

ABSTRACT

IMBALANCED LEARNING TECHNIQUES: EXPERIMENTS ON NCAA COLLEGE BASKETBALL LEAGUE PLAYER STATISTICS DATASET

Güler, Emir
Master of Science, Statistics
Supervisor : Prof. Dr. Barış Sürücü

August 2022, 122 pages

This study was conducted with the purpose of finding an answer to the question: "What are the state of art methods for imbalanced classification and which combinations of these methods yield best results in extremely imbalanced real-world data?" In order to accomplish the purpose, internal (algorithm-based) and external (sampling-based) imbalanced learning techniques were applied individually and in combination. The dataset used for the imbalanced classification task is National Collegiate Athletic Association (NCAA) Men's Basketball League Player Statistics Data. The players' draft status (whether the player drafted by any NBA Teams or not) was used as the target variable for binary classification. Minority : Majority ratio of the target variable is 3.39 : 96.61. F1 score was used as the main evaluation metric. It was found in the experiments that default parameters of sampling techniques do not work well with extreme imbalance. Optimum minority over majority ratio hyperparameters ranged between 0.07 to 0.11 which differs from the general advice and application where minority and majority class frequencies are matched which makes the ratio hyperparameter equal to 1. On the other hand, Cost-sensitive methods were combined with sampling methods and class weight hyperparameters

of cost-sensitive learning model which works optimally found as {class0: 1, class1: 1}, {class0: 2, class1: 1} or {class0: 3, class1: 2} contrary to the general teaching of "if class ratio is 1:9, the cost-sensitive weight hyperparameter should be the inverse of the original ratio". Lastly, probability threshold moving was applied to maximize F1 score. That way, 3 different methods in Imbalanced Learning were consolidated and better results were acquired compared to the single use of the state of art methods. Additionally, Monte Carlo simulation was applied to fortify and generalize the results obtained by real-world dataset.

Keywords: Imbalanced Classification, Imbalanced Learning, Oversampling, Cost-Sensitive, Simulation

ÖZ

DENGESİZ VERİDE ÖĞRENME: NCAA KOLEJ BASKETBOL LİGİ OYUNCU İSTATİSTİKLERİ VERİ SETİ ÜZERİNDE UYGULAMALAR

Güler, Emir
Yüksek Lisans, İstatistik
Tez Yöneticisi: Prof. Dr. Barış Sürücü

Ağustos 2022, 122 sayfa

Bu çalışma, "Dengesiz sınıflandırma için en güncel yöntemler nelerdir ve bu yöntemlerin hangi kombinasyonları aşırı dengesiz gerçek dünya verilerinde en iyi sonuçları verir?" sorusuna cevap bulmak amacıyla yapılmıştır. Amacı gerçekleştirmek için içsel (algoritma tabanlı) ve dışsal (örnekleme tabanlı) dengesiz öğrenme teknikleri ayrı ayrı ve birlikte uygulanmıştır. Dengesiz sınıflandırma görevi için kullanılan veri seti, National Collegiate Athletic Association (NCAA) Erkekler Basketbol Ligi Oyuncu İstatistikleri Verileridir. Oyuncuların draft durumu (oyuncunun herhangi bir NBA takımı tarafından draft edilip edilmediği) ikili sınıflandırma için hedef değişken olarak kullanılmıştır. Azınlık sınıf : Çoğunluk sınıf oranı 3.39 : 96.61'dir. Model performansı değerlendirme ölçütü olarak F1 skoru kullanılmıştır. Deneylerde, örnekleme tekniklerinin varsayılan hiperparametrelerinin aşırı dengesizlik durumunda iyi çalışmadığı bulunmuştur. Optimum azınlık/çoğunluk oranı hiperparametreleri 0.07 ile 0.11 arasında değişmiştir, bu da azınlık ve çoğunluk sınıfı frekanslarının eşitlendiği ve oran hiperparametresini 1'e eşit yapan genel tavsiyeden ve uygulamadan farklı olduğu saptanmıştır. Öte yandan, Maliyet duyarlı (cost-sensitive) yöntemler örnekleme

yöntemleriyle birleştirilmiş ve maliyete duyarlı öğrenme modelinin optimal olarak çalışan sınıf ağırlığı hiperparametreleri, "sınıf oranı 1:9 ise, maliyete duyarlı ağırlık hiperparametresi orijinal sınıf oranının tersi olmalıdır." genel tavsiyesinden farklı olarak {class0: 1, class1: 1}, {class0: 2, class1: 1} veya {class0: 3, class1: 2} olarak bulunmuştur. Son olarak, F1 skorunu en üst düzeye çıkarmak için olasılık eşiğini değiştirme yöntemi uygulanmıştır. Bu şekilde, çalışmada dengesiz öğrenmedeki 3 farklı yöntemi birleştirilmiş ve güncel yöntemlerin tek başına kullanımına kıyasla daha iyi sonuçlar elde edilmiştir. Bunlara ek olarak, gerçek dünya veri seti ile elde edilen sonuçları güçlendirmek ve genellemek adına Monte Carlo simülasyonu uygulanmıştır.

Anahtar Kelimeler: Dengesiz Öğrenme, Dengesiz Sınıflandırma, Örnekleme, Maliyet Duyarlı, Simülasyon

To Self-Discipline...

ACKNOWLEDGMENTS

First, from the bottom of my heart, I would like to thank to my girlfriend Burcu Koca for her unending support, love and patience.

I would like to offer my appreciation to my supervisor Prof. Dr. Barış Sürücü for the trust he put in and the support he provided.

I would like to thank to my examining committee member Assoc. Prof. Dr. Zeynep Işıl Kalaylıoglu for her illuminating guidance.

I would like to thank to my examining committee member Assoc. Prof. Dr. Şükrü Acıtaş for him to share his valuable thoughts and suggestions.

I would like to thank to my mother Şükran Güler and my father Ömer Güler for their trust, support and prays for me to succeed.

I would like to offer my thanks to my close friend Kaan Yavuzdoğan for his trust and sense of humor.

I would like to offer my thanks to Harun Demir for his fellowship and support.

I also want to offer my thanks to people who has a huge role in the making process of who I am today. Thank you, Alex Becker, for sharing your body and mind optimization techniques without which I would probably perform at only 50% of my productivity potential. Thank you, Sam Ovens, for teaching me the importance of sacrifice and focus which are the key ingredients of becoming great. Thank you, David Goggins, for teaching me that "at the end of suffering is greatness". Thank you, Jocko Willink, for teaching me that "discipline equals freedom". Lastly, Thank you, Kobe Bryant, for inspiring me to going forward at what I do every day.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xii

LIST OF FIGURES xiv

LIST OF ABBREVIATIONS xv

1 INTRODUCTION 1

2 LITERATURE REVIEW 7

3 METHODOLOGY 11

4 DATASET 45

5 EXPERIMENTS AND RESULTS 61

6 DISCUSSION AND CONCLUSION 101

REFERENCES 107

APPENDICES

LIST OF TABLES

TABLES

Table 3.1 Confusion Matrix 12
Table 4.1 Drafted Players Whose Minutes Played Percentage is Below 10 52
Table 4.2 Principal Components, the Amount of Variance They Explain and Model Performances 57
Table 5.1 Baseline Model Results on Original Dataset 62
Table 5.2 Oversampling Techniques' F1 Scores 64
Table 5.3 Oversampling Techniques with 0.1 Minority Over Majority Class Ratio F1 Scores 65
Table 5.4 Ratio Hyperparameter Comparison for Logistic Regression 66
Table 5.5 Ratio Hyperparameter Comparison for SVM's 67
Table 5.6 Undersampling Techniques' F1-Scores 69
Table 5.7 Undersampling Techniques with 0.1 Minority Over Majority Class Ratio F1 Scores 70
Table 5.8 Ratio Hyperparameter Tuning for Logistic Regression 70
Table 5.9 Ratio Hyperparameter Tuning for SVM's 71
Table 5.10 Grid Search for Hyperparameter Optimization of Combined Sampling Techniques (SMOTE+) with Logistic Regression 73
Table 5.11 SMOTENM1 Grid Search for Optimum Class Ratio Results Using Logistic Regression 74
Table 5.12 SMOTENM2 Grid Search for Optimum Class Ratio Results Using Logistic Regression 75
Table 5.13 SMOTENM3 Grid Search for Optimum Class Ratio Results Using Logistic Regression 76
Table 5.14 Best 3 Hyperparameter Combinations for Each Oversampling Technique 77

Table 5.15 Top 3 Hyperparameter Combinations for combined sampling techniques and Cost-Sensitive Logistic Regression 79
Table 5.16 Best Scores Obtained by Imbalanced Learning Techniques, Scenario 1 94
Table 5.17 Best Scores Obtained by Imbalanced Learning Techniques, Scenario 2 96
Table 5.18 Best Scores Obtained by Imbalanced Learning Techniques, Scenario 3 97
Table 5.19 Best Scores Obtained by Imbalanced Learning Techniques, Scenario 4 99

LIST OF FIGURES

FIGURES

Figure 1.1 Scatter Plot of an Imbalanced Dataset which has 1:100 IR.....3
Figure 4.1 Scree Plot of Principal Components and Explained Variance Ratios ...56
Figure 4.2 Box Plot of the Grid Search Results for the Number of Components That Yields the Best G-Mean Score 58
Figure 4.3 Box Plot of the Grid Search Results for the Number of Components That Yields the Best F1 Score.....59
Figure 5.1 Precision - Recall Curve for Model 1.....83
Figure 5.2 Probability Threshold Plot of Model 1 (10-fold Cross Validation).....84
Figure 5.3 Precision – Recall Curve for Model 2 85
Figure 5.4 Probability Threshod Plot of Model 1 (10-fold Cross Validation).....86
Figure 5.5 Precision – Recall Curve for Model 3 87
Figure 5.6 Probability Threshod Plot of Model 1 (10-fold Cross Validation).....88
Figure 5.7 Precision – Recall Curve for Model 4 89
Figure 5.8 Probability Threshod Plot of Model 4 (10-fold Cross Validation).....90
Figure 5.9 Precision – Recall Curve for Model 5 91
Figure 5.10 Probability Threshod Plot of Model 1 (10-fold Cross Validation.....92

LIST OF ABBREVIATIONS

ABBREVIATIONS

SVM's : Support Vector Machines

NM1: Near Miss 1

NM2: Near Miss 2

NM3: Near Miss 3

OSS: One-Sided Selection

SMOTE: Synthetic Minority Oversampling Technique

LR_liblin: Logistic Regression with "liblinear" solver.

LR_lbfgs: Logistic Regression with "lbfgs" solver

FGA: Field Goal Attemts

FTA: Free Throw Attempts

CHAPTER 1

INTRODUCTION

In Machine Learning language, a classification problem refers that a researcher is trying to predict the class of a given observation via predictive models. These classes can be binary (e.g., success – failure) or multiclass (e.g., win – loss – draw).
The number of observations that belong to each class is not always in equilibrium, it can vary for different datasets. These cases are called imbalanced classification. In these type of problems, the class that has the least amount of observations is called minority class, other class or classes are called majority class. Class imbalance can cause serious problems since almost all machine learning algorithms that is used for classification have the assumption of equal frequency distribution of classes. This results in classification errors especially for the minority class. Additionally, Minority class is what a researcher tries to predict in most of the cases (e.g., Fraud Detection, Churn Analysis). This fact amplifies the detrimental effects of class imbalance.

1.1 Classification in Machine Learning

The data used for classification problems are composed by observations and class labels which are assigned to each observation. Classification process includes predicting a class for observations whose class labels are unknown.

To accomplish a classification process input and output (i.e., class label) is required. This process can be exemplified as the following case of e-mail spam detection. In this case, input variables are the words used in an e-mail and the frequencies of each word, on the other hand, output variable is the predicted class label (i.e., spam, not spam). The number of classes is fixed if the problem is clearly stated as above.

Alternatively, one might want to find the probabilities of observations belonging a specific class.

Kuhn and Johnson (2013) stated that both continuous and discrete prediction is generated by classification algorithms. Continuous prediction is mostly in probability form (i.e., predicted values range from 0 to 1 and they add up to 1). On the other hand, a discrete category prediction is derived which is needed to make a decision.

In above e-mail spam filter example, the class labels are discrete (i.e., spam, not spam). If the predictive model derive that an e-mail is spam with the probability of 0.51 and another e-mail that is considered as spam by the model has a 0.98 probability of being a spam, discrete outcome will show only the class label which is "spam". Because of this kind of situations, the researcher might also want to look at the probabilities each decision has, which will increase the confidence about the decision.

A training dataset is required in order to build classification models. Training dataset is collected from the sphere of the problem which is composed by input variables and output variable. The number of observations in the training data vary from problem to problem depending on the complexity of the problem at hand. For one problem it will suffice to have hundreds of examples, on the other hand another problem may require millions of examples.

1.2 Imbalanced Classification and the Possible Causes of Class Imbalance

When a classification modelling task is at hand, the ratio of occurrences of different class labels have a substantial effect over the performance of the model. The cases where the proportion of one or more class labels are lower than the other classes are called imbalanced cases. One can face an imbalanced case in any dataframe or application, therefore one should be cognizant of the requirements of the modeling these type of data (Kuhn & Johnson, 2013).

The general way of describing the class imbalances in a dataset is to use ratio. An example to this can be an imbalanced case where the minority class to majority class ratio is 2 to 98 (2:98). Another way of showing the class imbalance is using IR (Imbalance Ratio). IR simply means dividing majority class to minority class. For a 1:100 scenario as per Figure the value for IR would be 100. Fernández et al. (2019) mentions that IR is not always an accurate measure of the complexity of the data.

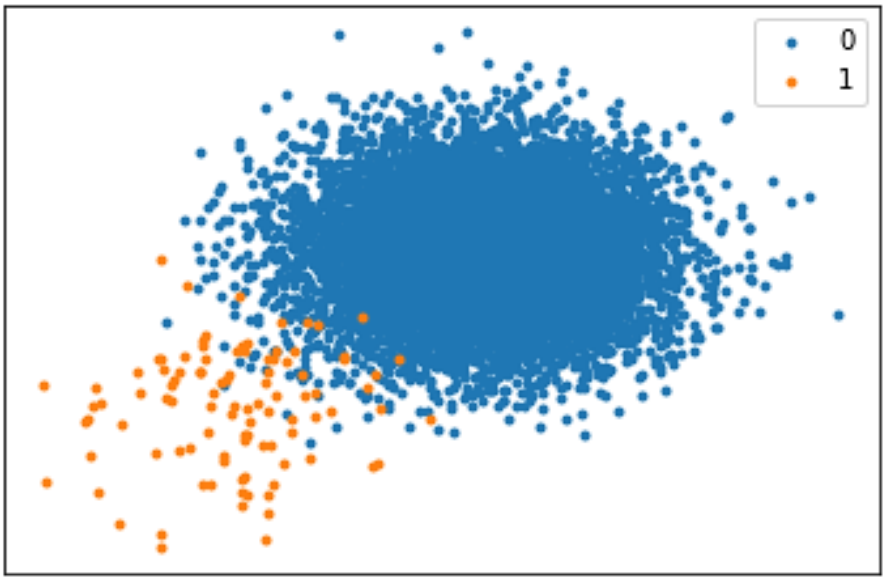


Figure 1.1 Scatter Plot of an Imbalanced Dataset which has 1:100 IR

There are two main causes of class imbalance. One of them is sampling errors. For example, it can happen that the data were collected from a small geographical area or from a small portion of time (Brownlee, 2020). These can trigger a class imbalance. Sampling errors can be fixed by wielding improved sampling techniques.

Second main cause of imbalance is the problem's intrinsic dynamics itself. It might be that a class label naturally occurs seldomly compared to the other class or classes. For example, only a small portion of the basketball players in college league end up drafted into NBA, only a small portion of the customer actions in an online banking app is fraudulent, only a small portion of the patients has a rare lethal disease or oil-spills are rarely happen.

The imbalance degree between classes vary, for one problem the imbalance might be a little skewed (e.g. 4:6), for another one there can be 1 example that belongs to the minority class for 100 examples that belongs to the majority class (1:100).

Most modern-day applications of imbalanced classification focus on imbalance ratios that range between 1:4 and 1:100. Some of the real-world problems such as fraud detection might have imbalance ratios that range between 1:1000 and 1:5000. This introduce new hardships to classification process in general since everything must be tuned to such severe cases (Krawczyk, 2016).

A slight imbalance usually does not cause a problem when approached as a standard classification problem, but more intense imbalanced cases should be treated with specialized techniques.

Standard classification algorithms assume approximately similar distribution of observations between different class labels but for most of the real-world examples, the distributions are skewed. This causes a challenge toward learning algorithms since these algorithms will favor the majority class and in fact, the minority class has much higher importance (Krawczyk, 2016).

1.3 Techniques Used to Tackle the Problem of Imbalanced Classification

There are four main methods to solve the problem of imbalance in Machine Learning:

1. Algorithm Level: These methods adapt the classification algorithm to favor the minority class. In order to accomplish this task, Thorough knowledge of both the problem domain and the classification algorithm itself is required. These methods are also named as internal methods.

2. Data Level: These methods are used to eliminate the imbalance itself by using resampling techniques. When the data is balanced, the need of adaptation of the

classification algorithm end up being unnecessary. These methods are also called external methods. The focal point of this thesis is on data level methods.

3. Cost-sensitive: These methods use both data level and algorithm level approaches. It adds costs to specific observations in data additionally it modifies the algorithm to assume higher costs of misclassification when a minority class instance predicted wrong. This renders the algorithm biased towards minority class.

4. Ensemble based: These approaches use ensemble algorithms and one of the methods described above in combination.

1.4 Aim of the Study

Since most of the real world scenarios in classification contains imbalanced datasets of varying degrees and detecting or predicting the minority/rare class has the utmost importance, testing and finding best working techniques for extreme imbalance scenarios is the main focus of this study

CHAPTER 2

LITERATURE REVIEW

In classification predictive modeling, one wants to predict dependent variable Y by building a predictive function $Y = f(X_1, X_2, ..., X_p)$ using the training data $\{\langle x_i, y_i \rangle\}_{i=1}^{n}$ (Branco et al., 2016). In classification models, Y has binary or multiclass labels. If some of the class or classes are not represented equally compared to other class or classes, this predictive process is called imbalanced classification (Brownlee, 2020).

He and Garcia, in their paper published in 2009 mentioned that any dataset which has uneven class distribution can be considered as an imbalanced dataset technically. But in the Machine Learning community, an imbalanced dataset referred to a significant or severe imbalance. Imbalance ratios of 1:100, 1:1000 or 1:10000 are not uncommon. Fernández et al. (2019) points out the reasons that can decrease the performance of a classifier: One being the intrinsic characteristics of the training dataset and the uneven class distribution.

Class imbalance can happen due to the inherent characteristics of the problem domain. For example, fraudulent credit card activities are really rare compared to the non-fraudulent credit card activities in real life. On the other hand, certain shortages in data collection (e.g. economic reasons, privacy) can also cause class imbalance (Chawla et al., 2004).

Due to the fact that the class imbalance case is common in most domains of high importance (e.g. medical diagnosis, text analysis, oil spill detection) (Sun et al., 2009) and most machine learning algorithms have a bias toward majority class when trained on an imbalanced training data and this is ignored some of the researchers

that focuses on solely on the learning process itself, a need of finding an optimum solution to the imbalance problem has arisen recently (Susan & Kumar, 2021).

The first person who approached systematically to the problem of class imbalance and examined it as a problem topic itself is Japkowicz (2000). She proved in her study with synthetically created data that the sample size is not a significant factor for classification performance, but the imbalance degree is. She also compared the effectiveness of the various oversampling and undersampling methods over the performance of the classification performance and found out that all of the methods improved the performance classification significantly.

Susan and Kumar (2021), found out in their review that out of three main techniques which are data-level (i.e. oversampling, undersampling or both), algorithm-level and cost-sensitive; data-level methods acquired best results overall.

Cost-sensitive learning conflict higher penalty for misclassified minority class samples than misclassified majority class samples (Tayal et al., 2015).

Jo and Japkowicz, in their paper published in 2004 imposed different costs (1.0 for minority, 0,1 for majority class) in order to mitigate a 1:9 imbalance.

Weiss and Provost (2001) showed in their study that the class distribution has an effect over the performance of the model. They applied different density thresholds for minority and majority classes. The ROC AUC performance metric performed better when the minority class density increased.

Susan et al. (2021) used cost-sensitive learning approach to solve the skewed distribution of nodule and non-nodule lung pictures. They achieved better accuracy scores on their proposed model than other existing techniques.

One of the most popular methods for class balancing is to use sampling methods which are comprised of undersampling, oversampling and combinations of both. One of the widely used undersampling method is Tomek Links method, which is introduced by Tomek (1976), finds pairs that belong different classes which has minimum distance, which are called Tomek Links, they are considered as a noise or

examples lay near the decision threshold. In order to balance the majority and minority classes, majority class is removed from the training data.

Jianping and Inderjeet (2003) offered Near Miss undersampling method which has 3 different versions. Version 2 performed best in their study alongside with random undersampling.

There are methods that includes undersampling into learning process. For example, Liu et al. (2008) offered two new ensemble algorithms that exploits the missing information caused by undersampling. The algorithms are named as EasyEnsamble and BalanceCascade which are elaborated upon in Methodology part of this study.

Biased Random Forests is another algorithm which was proposed by Bader-El-Den et al. (2019). In this method, algorithm does not only learn from the original training data, but also from the derived data which contains datapoints which are considered as critical region.

Dumpala et al. (2018) offered an approach which undersamples the majority class in a manner where dataset is transformed. Two samples from training data is combined under 4 clusters in accordance with the class labels (majority-majority, minority-minority, majority-minority, minority-majority). In order to decrease the density of the majority class, majority-majority cluster modified by matching majority class of one sample with majority class examples of the other sample which are reduced to the number of minority class.

(G. E. A. P. A. Batista et al., 2003) used combinations of oversampling and undersampling methods. One of them is SMOTE and Tomek Links.

CHAPTER 3

METHODOLOGY

The techniques introduced in the introduction chapter of the study are elaborated upon in this chapter. The chapter starts with evaluation metrics widely used in imbalanced learning and continues with the methods widely used for tackling the problem of imbalanced classification.

3.1 Choosing the Right Evaluation Metric for Imbalanced Learning

In Machine Learning world, there are some evaluation metrics often preferred by researchers. Most prevalent one is the "accuracy" and "error rate" metrics. The formula of the accuracy and error metrics are given below.

$$Accuracy = \frac{Correct\ \ Predictions}{Total\ \ Predictions} \tag{3.1}$$

$$Error\ \ Rate = \frac{Incorrect\ \ Predictions}{Total\ \ Predictions} \tag{3.2}$$

Also, Error Rate is the complement of the Accuracy.

$$Accuracy = 1 - Error\ Rate \tag{3.3}$$

There is another way which is often preferred by researchers for classification evaluation. The way is called confusion matrix. In this matrix, rows represent the actual classes. The columns express predicted classes. Each cell of this matrix shows the number of predictions made by the model for that intersection. An example of the confusion matrix that belong to a binary classification problem is given below.

Table 3.1 Confusion Matrix

	Positive Prediction	Negative Prediction
Positive Class	True Positive (TP)	False Negative (FN)
Negative Class	False Positive (FP)	True Negative (TN)

In above confusion matrix, all cells have specific names (e.g. True Positive). In order to explain these terms in a simpler manner, assume that 0 is assigned for the negative class and 1 is assigned for the positive class. True Positive (TP) means the prediction label is positive (i.e. 1) and real life label of the observation is also positive in the dataset. False Negative (FN) means the prediction made by the model negative (i.e. 0) and real life value for the observation is positive in the dataset. Similarly, False Positive (FP) means the predictions is Positive and real value for the same observation is negative. Lastly, True Negative (TN) means the prediction is negative and real value for the observation is also negative.

Accuracy, Error Rate can be calculated from the confusion matrix as the following:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.4}$$

$$Error\ \ Rate = \frac{FP + FN}{TP + TN + FP + FN} \tag{3.5}$$

Accuracy treats correct positive class predictions and correct negative class predictions equally. Which renders it not suitable for Imbalanced Classification scenarios.

3.1.1 Failure of Accuracy in Imbalanced Learning

Consider an imbalanced dataset whose imbalance ratio is 1:99. Which means each minority class example there are 99 majority class examples. Most of the algorithms used in machine learning assumes an even data distribution which means 50:50 ratio

for a binary classification problem. Also most of them has simple learning principles like having a tendency to predict majority class when faced with uneven class distributions. In the scenario of 1:99 class distribution, mainstream algorithms often predict only majority class labels. If a researcher use accuracy as an evaluation metric, he/she will attain 0.99 accuracy with the dataset described above. In standard conditions where class labels distributed equally, 0.99 accuracy is extremely good. If a naïve researcher with an imbalanced dataset achieved 0.99 accuracy on his predictive model, he would think he did a great job, but he would be completely misled by the wrong evaluation metric which is accuracy in this case.

Since the accuracy does not separate different classes in terms of the number of correct predictions, it is not a proper way to evaluate models in imbalanced scenarios. (Fernández et al., 2018)

3.1.2 Precision

If we assign the name positive for minority class and also assign negative for majority class, Precision metric tells us the ratio of correct positive predictions to all positive predictions (i.e. True Positives plus False Positives).

The formula for precision for a binary classification is below:

$$Precision = \frac{TP}{TP + FP} \tag{3.6}$$

The principle of the precision metric simply ignores false negative class predictions and minimizing the false positives (Brownlee, 2020).

Consider an imbalanced classification scenario in which the dataset has 2:98 imbalance with 10000 total observations. 9800 of them would be negative class and 200 of them would be positive class. If a model predicts 220 observations as positive, 40 of them is not true and 180 of them is true. In a situation like that, the Precision will be:

$$Precision = \frac{TP}{TP + FP} = \frac{180}{220} = 0,82 \tag{3.7}$$

Higher the Precision, better quality of prediction the model has.

Precision is a good metric to use for classification problems where we want to minimize the number of false positives. In other words, if it is costlier to predict a class label as positive when it belongs to the negative class compared to classifying a class label as negative when it belongs to the positive class, precision metric would make a great choice.

3.1.3 Recall

Recall metric is the ratio of correct positive predictions to all observations that belong to the positive class. In this way, recall can address the incorrectly classified positive class observations.

Recall is typically used as a measure of how a classifier covers the minority class (He & Ma, 2013).

For a binary classification problem, recall is calculated as below:

$$Recall = \frac{TP}{TP + FN} \tag{3.8}$$

For an imbalanced scenario in which the imbalance ratio is 3:97 and there are 10000 total observations, which means 300 positive (minority) class examples and 9700 negative (majority) class examples, the recall of a predictive model which predicts 270 of the positive examples right and remaining 30 positive examples wrong would be:

$$Recall = \frac{270}{300} = 0,9 \tag{3.9}$$

Higher the recall score, better the prediction quality the model has.

Recall measure is a good option for the situations when we want to minimize the negative predictions which are false. If it is more important to minimize false negative predictions compared to false positive predictions, Recall would make a great choice of evaluation metric.

3.1.4 F-score

In imbalanced learning, the objective is increasing recall without decreasing prediction. Nonetheless, these goals has inherent contradictions. To be more precise, in order to increase recall, False Negative predictions should be minimize, which almost always comes with the cost of increased False Positives, increase in False Positive predictions automatically increase Precision (He & Ma, 2013).

In Imbalanced Learning, neither recall nor precision covers the whole situation. One model can have terrible recall and still have excellent precision and vice versa.

F-score combines these two measures in one measure. After finding both recall and precision, they can be combined as one under the name of F-score with the formula given below:

$$F - score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.10}$$

There are different names which are used in literature, for example F1-measure, F-measure.

F-score gives even importance to recall and precision metrics, also most common metric used for imbalanced learning (He & Ma, 2013).

Consider an imbalanced dataset in which positive class to negative class ratio is 1:100. There are 100 examples in the positive class and 9900 in the negative class.

A model predicted 150 positives and 90 of them is correct (True Positive), which means 10 positives are misclassified as negative (False Negatives) and 60 of the positive predictions are False Positives. In this case, F-score will be:

$$Precision = \frac{TP}{TP + FP} = \frac{90}{90 + 60} = 0,6 \tag{3.11}$$

$$Recall = \frac{TP}{TP + FN} = \frac{90}{100} = 0,9 \tag{3.12}$$

By using Precision and Recall metrics, the F-score of the model will be:

$$F - score = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * 0,6 * 0,90}{0,6 + 0,90} = 0,72 \tag{3.13}$$

The model has poor Precision score and a good Recall, F-score helps to balance these metrics.

3.1.5 Fbeta-Score

F-score creates a balance between precision and recall as it is stated above. For some situations, minimizing false positives are the most important, on the other hand, minimizing false negatives are still important but less. In these situations, one would want to increase the importance of precision in the metric. The situation might be vice-versa, one might prioritize to minimize false negatives and at the same time, minimizing false positives also be important albeit not the most important. Fbeta-score offers a solution to this by adding a parameter called β. The parameter will determine the balance between precision and recall. The calculation of Fbeta-score is given below:

$$F\beta = \frac{(1 + \beta^2) * Precision * Recall}{\beta^2 * Precision + Recall} \tag{3.14}$$

The value of β determines the name of the metric. For example,

For $\beta = 0,5$ The metric is called F0,5-score, this version increases the importance of precision and decrease the importance of recall

$\beta = 1$, This version of the metric actually the classic F-score which is stated above, the metric also called F1-score, this version balances the Precision and Recall.

$\beta = 2$ This version of the metric is called F2-score, it amplifies the importance of Recall and reduce the importance of Precision.

3.1.6 ROC Curves and ROC AUC

ROC Curves

ROC Curves are a popular way to define a binary classification model's performance. Y axis of the plane on which ROC Curves lay is called "True Positive Rate" another name for the metric Recall. The formula of the True Positive Rate is given below:

$$True\ Positive\ Rate = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{3.15}$$

X axis of the plane represents "False Positive Rate". False Positive Rate is False Positive predictions divided by all negative examples in the data (i.e. False Positives + True Negatives). The formula is given below

$$False\ Positive\ Rate = \frac{False\ Positives}{False\ Positives + True\ Negatives} \tag{3.16}$$

The plot actually shows accuracy of the positive class on y axis and error of negative class in x axis. As a result, the best scenario would be achieved through maximizing y axis (i.e. y=1) and minimizing x axis (i.e. x=0).

ROC Curves are created by applying new classification probability threshold values iteratively and recording True Positive Rates and False Positive Rates for each threshold value. Each recorded value represents a dot in the coordinate plane where y axis shows True Positive Rate and x axis shows False Positive Rate as stated above. These dots form a curve which starts form bottom left and ends at top right, also the peak of the curve leans towards the top left corner.

A no skill model (i.e. predicts negative every time or predicts positive every time) will form a diagonal line from bottom left corner to top right corner. Any point that falls under this line would be considered worse than a no skill classifier.

ROC Curve's does not favor predictive models that compromise minority class for the performance over majority class. This makes ROC Curve's very appealing for imbalanced learning (He & Ma, 2013).

ROC AUC

Comparing the performance of imbalanced learning models is hard just by looking at the ROC Curves. ROC AUC offers a solution to this. AUC stands for Area Under the Curve. ROC AUC value stretches from 0,00 to 1,00. The higher this metric is, the better for the classification model. ROC AUC being equal to 1,00 means perfect classifier.

ROC AUC might be the most common metric used for comparing classification models (Brownlee, 2020).

ROC AUC has shortcomings under the occasions of severe imbalance and the minority class has few examples in the data. With few examples, any misclassified example would make big differences on the True Positive Rate and True Negative Rate metrics which will lead big changes in ROC AUC.

3.1.7 Precision-Recall Curves and AUC

Precision is a metric which shows the fraction of correct positive predictions out of all positive predictions. Recall is a metric is the fraction of how many positive class members predicted correctly out of all positive class members. These metrics helps a researcher to evaluate model performance over predicting minority class.

Precision-Recall Curves (PR Curves) lay upon a coordinate plane where Precision is represented in the y axis and Recall is represented in x axis.

A perfect skill model will appear on the top right corner of coordinate plane. Which represents the point where Precision is equal to 1 and also Recall is equal to 1. A good skilled models' Precision-Recall Curve peak will lean towards top right spot. A model with no skill will be shown as a horizontal line on the plane and Precision will be equal to the ratio of minority class.

ROC Curves considers both classes in its creation. On the other hand, Precision-Recall Curves focuses only the minority class. For severely imbalanced cases, ROC curves can be misleading for this reason.

For extreme imbalanced cases, ROC Curves might give misleadingly optimistic performance for predictive models, this is why Precision-Recall Curves are offered in such cases (Branco et al., 2016).

Precision-Recall Curve AUC

The Area Under the Precision-Recall Curve is a very similar concept with ROC AUC. It summarizes the Curve which is composed by multiple threshold values as one score. A perfect classification model would score 1,00 on Precision-Recall Curve AUC.

3.1.8 Probabilistic Score Evaluation Metrics

For some classification cases, Researches might want to see the observations' probabilities of belonging a specific class. In this type of problems, one has to use appropriate evaluation metrics. Most common metrics used are Log Loss Score and Brier Score.

Log Loss Score

Log loss is a function which is known to be used in training of Logistic Regression Algorithms. The formula for a binary classification problem can be represented as below.

$$LogLoss = -((1-y)*\log(1-y) + y*\log(y)) \tag{3.17}$$

In above equation y means expected probability of belonging one class out of two. y means predicted probability of belonging corresponding class.

For a set of predictions, we will have multiple log loss values. In order to summarize all these values the average of all log loss values is taken.

$$AverageLogLoss = \frac{1}{N}*\sum_{i=1}^{N} -((1-y_i)*\log(1-y_i) + y_i*\log(y_i)) \tag{3.18}$$

A score equal to 1 is perfect score for this metric. The closer the score to the 0 the better performance the model has.

Brier Score

Brier score simply means the Mean Squared Error for probabilistic scores and built for binary classification problems. Brier score can be computed as below:

$$BrierScore = \frac{1}{N}*\sum_{i=1}^{N}(y_i - y_i)^2 \tag{3.19}$$

Brier score takes values between 0 and 1. Perfect score is 0. The model performance gets better when Brier Score gets closer to the 0.

3.2 Cross Validation for Imbalanced Learning

Model evaluation involves finding best data preparation method, best performing learning algorithm and tuning hyperparameters to increase the performance of the learning algorithm. In order to accomplish this task, two methods are mostly used. cross validation and train/test splits.

Train/test splits are effective if you have a large enough dataset which can represent the problem well. The required size of the data determined by the problem domain itself. Ideally, a train/test split should be 50/50, even though more imbalanced splits (e.g. 80/20 or 67/33) are more common (Brownlee, 2020).

Most of the time, the dataset at hand is not sufficiently large to apply train/test splits effectively. Data sampling procedures should be implemented on these types of datasets. Most commonly used one is k-fold cross validation.

k-fold cross validation process works as the following: Dataset is cut into k folds. k-1 folds are used as training data and k^{th} fold is used as test data. As expected, the procedure repeats itself for k times so each fold has opportunity to be used as a train and test set.

k-fold cross validation fails for imbalanced datasets. The reason for this can be explained with a simple scenario. Assume that one has a dataset of imbalance with ratio of 1:99. One splits this data into k folds. Some folds end up not having any minority class examples. When these folds reach their turn to be used as test split, model will be evaluated as it should only predict the majority class. This definitely cause some evaluation problems.

To solve this issue, most common practice is using stratified sampling. Stratified sampling still keeps the randomness but each fold will keep the distribution of minority class to majority class.

3.3 Data Level Methods

Most of the machine learning algorithms focus on the class distribution and concludes that minority class is not that important and develop a bias towards majority class.

Since minority class is actually the focal point of the imbalanced classification process, this is a problem. In order to avoid this problem, the most common approach is using sampling methods. In this method, one simply samples from existing observations to balance the data distribution, so common machine learning algorithms can work fine. Otherwise the common machine learning algorithms can offer misleading results since most of the algorithms built with the assumption of even class distribution.

The reasons behind why sampling methods are popular to solve imbalance issues are the following: The method is simple to understand and implement. After the data sampling is done and the data is transformed, common machine learning algorithms works fine.

Rather than waiting from the model to take of the imbalance, one can try to eliminate the class imbalance. Going on this route dissolves the base problem of class imbalance in training. (Kuhn & Johnson, 2013)

Sampling is employed only the training dataset, not for evaluation set (i.e. Test set). If one tries to evaluate a model that is trained with a sampled training set by using a sampled test set, the results will be misleadingly optimistic.

There so many sampling methods one can apply and there is no "cure it all" one. Different sampling techniques respond different for chosen learning algorithm which makes things more complicated.

Sampling techniques can be decreased into 3 main categories: Oversampling techniques, undersampling techniques, techniques combined.

3.3.1 Oversampling Techniques

Random Oversampling

One of the popular oversampling techniques is Random Oversampling, which is the easiest one to understand and implement. This method simply chooses currently existing examples from minority class with replacement and makes copies of them to increase the numbers of the minority class to reach equilibrium between minority and majority class frequencies.

Random oversampling increases the time needed for building a classification model and also have a tendency to overfitting (He & Ma, 2013).

Synthetic Minority Oversampling Technique (SMOTE)

The technique is the most popular one in terms of oversampling techniques. The method choses one random minority class example and closest minority examples to that example are found. One of the close neighbors of the first chosen example is randomly chosen. Between these two examples, a line is drawn. On randomly chosen one point of this line a synthetic sample is created.

Chawla et al. in a paper conducted in 2002 mentions that previous literature shows that majority class undersampling yields better results compared to the oversampling of the minority class. Also, combining the oversampling and undersampling methods shows no improvement in terms of classification performance over using solely undersampling. Nevertheless, on these studies the oversampling only done by copying the minority class examples. SMOTE is a different method than this.

The detailed process of SMOTE can be explained as the following steps:

The input of the algorithm contains T which represents the number of examples in the minority class, $N\%$ represents the oversampling ratio, lastly, k represents the number of neighbors which are the nearest to the minority example.

At the end of the process, $(N/100)*T$ synthetic samples are created.

The process starts choosing a random example from the minority class *T*. Then k nearest neighbors of the chosen example are calculated. One of the k neighbors is chosen randomly and the distance between the minority example and the neighbor is calculated. On the line of closest distance is where the new synthetic sample is created.

The basic formula for new synthetic samples is:

$$Synthetic = Sample + r * dif \tag{3.20}$$

Here, *Synthetic* represents the synthetic data created. *Sample* means the minority class example, *r* means a random number and *dif* means the distance between the minority example and the neighbor randomly chosen for synthetic data creation.

In the original paper of the Technique SMOTE, which is published in 2002, Chawla et al. mentions that SMOTE with an undersampling technique works better than plain undersampling with SMOTE.

Borderline SMOTE

There are some extensions to SMOTE, one of them is called Borderline SMOTE which focuses on the misclassified examples and create samples from these misclassified examples.

The examples that lays nearby to decision boundaries are the most important ones since they are more susceptible to mispredictions (Han et al., 2005).

For the reason above, Han et al. (2005) offered two new oversampling techniques called Borderline SMOTE1 and Borderline SMOTE2.

Borderline SMOTE1

Borderline SMOTE1 looks at all minority class examples and finds out the k-nearest neighbors of them (default value of k is 5 but different numbers for k can be tried). If all the neighbors of the minority class example belong to the majority class, this example will be deemed as noise and the process of Borderline SMOTE1 will not

continue for these examples. The focal point of the algorithm is the examples whose equal to or more than half number of neighbors belong to the majority class. These examples are at the stake of being misclassified by the classification algorithm. After that step, the detected examples' nearest neighbors in minority class are found. The difference between these neighbors and the detected example calculated and then multiplied with a random number between 0 and 1. The end result is the location of the synthetic sample.

Assume that the training dataset is called T, minority class examples in the training dataset is called P and majority class in the training dataset is called N. So,

$$P = \{p_1, p_2, ... p_{pnum}\}, \quad N = \{n_1, n_2, ..., n_{nnum}\} \tag{3.21}$$

In above equation pnum refers to the number of examples in the minority (i.e. positive) class and nnum refers to the majority (i.e. negative) class. SMOTE1 algorithm steps are as described below:

At the beginning of the process, for each $p_i (i = 1, 2, ..., pnum)$ in P, m nearest neighbors are calculated. Among these nearest neighbors, the examples that belong to the majority class are referred as $m' \in [0, m]$.

Under the condition of $m' = m$, each neighbor of the example p_i belongs to the majority class and p_i considered as noise and is not included in the following steps. Besides, if $m/2 \le m' \le m$ the majority class neighbors are equally frequent or outnumbers to the minority class neighbors of p_i. These examples are the ones that the process continue with because they are at the edge of misclassification and they construct a set called *DANGER*. Lastly, if $0 \le m' < m/2$, these examples are not participated to the following steps since they are not in danger of misclassification. The set called *DANGER* is composed by borderline positive (i.e. minority) class members so, $DANGER \subseteq P$ and:

$$DANGER = \{p'_1, p'_2, ... p'_{dnum}\} \quad 0 \le dnum \le pnum \tag{3.22}$$

The minority class k nearest neighbors are calculated for each p'_i in *DANGER*.

In the following step, $s*dnum$ examples are created synthetically ($s \in [1, k]$).

In order to achieve this, s neighbors are chosen from k positive class nearest neighbors of p'_i. This process is followed by difference calculation between s positive neighbors and the example $p'_i \in DANGER$. The differences denoted as

dif_j $(j = 1, 2, ..., s)$. Then, every dif_j is multiplied by a corresponding random number called r_j $(j = 1, 2, ..., s)$ which takes values between 0 and 1. The synthetic samples are created as a result of aforementioned progressive steps. The synthetic samples can be formulated as:

$$synthetic_j = p'_i + r_j * dif_j, \quad j = 1, 2, ..., s \tag{3.23}$$

The end result of repeating the process above for all $p'_i \in DANGER$ yields $s*dnum$ synthetic samples.

This process creates synthetic data between borderline minority examples and their positive (i.e. minority) class examples which strengthen the minority examples on the borderline (Han et al., 2005).

Borderline SMOTE2

Borderline SMOTE2, in addition to generating new examples between minority class and its minority neighbors, it generates synthetic examples between the example and its nearest majority class example, the only difference is the random number used for generating the synthetic example is between 0 and 0.5 this time. This make sure the created sample is closer to the minority class.

Borderline Oversampling

This technique uses Support Vector Machines (SVM's) algorithm. The method uses SVM to create a decision boundary then focuses on the minority examples that are close to the borderline because they are the ones in danger of misclassification. The

method also uses extrapolation method to increase the region of minority class where the nearest majority class neighbors of the minority instance are count for less than half of the total neighbors. This is a difference between this method and original SMOTE. Another difference between Borderline SMOTE and SMOTE original is the neighbors selected randomly to create synthetic samples in the original SMOTE but this method generates samples considering the order of the neighbors from first to k^{th}.

The step by step progress of the method is given below:

The required inputs of the method can be described as:

X: Training set

N: Level of sampling (100, 200, … percent)

K: represents the total number of nearest neighbors

m: interpolation or extrapolation threshold value

Following variables will be used for method description and it is useful to give them beforehand:

$\mathbf{SV}^+$: positive support vectors

T: Required number of artificial samples

amount: An array that is composed by the number of artificial samples to be generated corresponding to the positive examples in SV

nn: An array that is composed by corresponding k nearest neighbors of minority (positive) class examples in SV^+

First step of the sample generating process of Borderline Oversampling is finding out the required number of synthetic samples, which is denoted by *T*.

$$T = (N/100)*|X| \tag{3.24}$$

After that step, Positive support vectors (SV^+) are calculated employing SVM algorithm. The *amount* variable is calculated by distributing T in an even fashion over SV^+. Following that step, **nn** is calculated.

For every example in SV^+, which can be denoted as sv_i^+, m neares neighbors are detected. If the number of negative class examples are less than $m/2$, *amount[i]* number of examples are created on the line that connects the sv_i^+ and its k nearest neighbors (in an order that starts nearest neighbor to k^{th} nearest neighbor) by implementing the following formula:

$$x_{new}^+ = sv_i^+ + \rho(sv_i^+ - nn[i][j]) \tag{3.25}$$

In above equation $nn[i][j]$ represents j^{th} nearest positive neighbor of sv_i^+. ρ is a random number and $\rho \in [0, 1]$. For other situations where the positive nearest neighbors outnumber the negative nearest neighbors (i.e. positive neighbors count more than m/2), following interpolation formula used to create synthetic examples as per original SMOTE algorithm:

$$x_{new}^+ = sv_i^+ + \rho(nn[i][j] - sv_i^+) \tag{3.26}$$

End of the process, the final dataset X_{new} is:

$$X_{new} = X \cup \{x_{new}^+\} \tag{3.27}$$

Adaptive Synthetic Sampling (ADASYN)

ADASYN is another oversampling method which chooses examples to be oversampled by looking at their density in the space of features. If the density of an example low in an area, ADASYN chooses this example and creates synthetic samples from it. If the density is high for an example, the method opts not choose that specific example for sampling.

The main point of ADASYN is to use density distribution of the minority samples to decide how many synthetic samples will be created (He et al., 2008).

He et al. (2008) mentions that ADASYN method contributes two things to imbalance classification problems. First one is to reducing the inherent bias that comes from the data imbalance, second one is to adaptively manipulate decision boundary in favor of "difficult to classify" type of examples.

The algorithm of the ADASYN is described below:

Assume a training dataset called D_{train}, which has m observations $\{x_i, y_i\}$, $i = 1, 2, ..., m$ x_i is an observation in the feature space with n dimensions called $\boldsymbol{X}$, on the other hand, $y_i \in Y = \{1, -1\}$ represents the class labels. The number of minority class examples are referred as $m_{minority}$ and the number of majority class examples are referred as $m_{majority}$. Hence, $m_{minority} < m_{majority}$ also $m_{minority} + m_{majority} = m$

As the first step, the degree of imbalance is calculated:

$$d = m_{minority} / m_{majority} \tag{3.28}$$

Here $d \in (0,1]$.

Define d_{th} as the threshold of maximum imbalance ratio allowed, if $d < d_{th}$ then the number of required synthetic examples calculated as:

$$G = (m_{majority} - m_{minority}) * \beta \tag{3.29}$$

Above equation, $\beta \in [0, 1]$ is the parameter of the aimed balance level of minority and majority classes. $\beta = 1$ represents a fully balanced dataset.

For every x_i that belongs to the minority class, K nearest neighbors are calculated using Euclidean distance. Then the ratio referred as r_i is calculated as the following:

$$r_i = \Delta_i / K, \quad i = 1, ..., m_{minority} \tag{3.30}$$

Above equation, Δ_i represents the number of majority class examples in the K nearest neighbors of x_i, Hence, $r_i \in [0, 1]$.

r_i is normalized to derive the density distribution r_i,

$$r_i = r_i / \sum_{i=1}^{m_{minority}} r_i \tag{3.31}$$

In order to find out how many synthetic samples will be created for each x_i, below formula is employed:

$$g_i = r_i * G \tag{3.32}$$

In above equation, G represents the number of synthetic samples to be generated.

g_i synthetic examples are created for each x_i according to the following procedure:

The loop continues from 1 to g_i:

Between K nearest neighbors of x_i, one minority class neighbor x_{zi} is chosen.

Synthetic sample is created by the following formula:

$$s_i = x_i + (x_{zi} - x_i) * \lambda \tag{3.33}$$

Above equation, $(x_{zi} - x_i)$ is a vector in feature space and λ is a number randomly created and $\lambda \in [0, 1]$

The loop ends.

3.3.2 Undersampling Methods

Undersampling methods used for balancing the class distributions for imbalanced classification problems. Contrary to the oversampling methods which increase the density of the minority class via different sampling techniques, this method focuses on decreasing the minority class density in the training dataset.

Even though undersampling methods can be used only by themselves for imbalanced classification Using undersampling methods along with oversampling methods usually yields better results than using solely undersampling or oversampling.(Brownlee, 2020)

Undersampling methods can be examined under 2 branches: Methods that informs about which examples should be kept and methods that informs which examples to remove.

Near Miss Method

This method has three different versions. First version is Near Miss 1. In this version of the method the negative (i.e. majority class) examples are chosen whose average distances to the nearest 3 positive examples (i.e. minority class) are the smallest.

The second version is Near Miss 2. In this version the negative examples are chosen whose average distance to the three furthest positive class examples are the smallest.

Last version which is called Near Miss 3 chooses the negative class examples whose distances to the all positive examples are the smallest.

Condensed Nearest Neighbor (CNN) Method

This method aims to achieve a subset of training samples without losing any classification performance.

(Hart, 1967) described the algorithm with below steps:

The algorithm uses bins named *STORE* and *GRABBAG*.

At the beginning of the algorithm first example is chosen and put in the *STORE*.

After that step second example is put into the *STORE* and Nearest Neighbor algorithm applied. If the second example is classified correctly, it is then placed in *GRABBAG* otherwise it is placed in the *STORE*.

The process continues until the last example in the training set is classified using the examples in the *STORE*. Again, if the example is classified correctly, it is placed in the *GRABBAG*, otherwise it is placed in *STORE*.

After the whole training dataset is passed through once, the aforementioned process continues with the examples in *GRABBAG*.

The loop continues unless one of the two following conditions are met:

1. All examples in the *GRABBAG* are placed into the *STORE*. This means the final subset is equal to the dataset at the beginning of the process.
2. All examples in the *GRABBAG* iterated and no transfers happened into the *STORE*.

The examples in the *STORE* is the new training dataset and the examples in the *GRABBAG* is discarded.

Tomek Links Method

Tomek Links Undersampling method chooses examples which are both nearest neighbors and have different class labels. After detection the majority class observation is deleted. The idea behind is that these kind of observations makes hard to set a decision boundary. Algorithm works as following:

Consider two observations in a training dataset which are referred as O_i and O_j. These two examples belong to the different class labels. Also, $d(O_i, O_j)$ represents the distance between two observations. If there are no other example O_a where $d(O_i, O_a) < d(O_i, O_j)$ or $d(O_j, O_a) < d(O_i, O_j)$, O_i and O_j are called Tomek Link. If two examples are Tomek Link, either these examples are a noise, or they are

borderline examples. Tomek links as an Undersampling technique removes only the majority class observation from the training data. Tomek links method is used as a data cleaning method too. In this version of usage, both minority and majority class examples are removed.

Edited Nearest Neighbors (ENN) Method

ENN is another undersampling method which implements 3 nearest neighbor rule to detect misclassified samples. The detected majority class samples that are misclassified end up being deleted which reduced the density of the majority class examples in the training data. Alternatively, the method can be applied on minority class too. In this case, the minority class examples which are misclassified by its 3 nearest neighbors are detected first. Then, the majority class examples in these three neighbors are deleted from the training data. This method is offered by Wilson in his paper which is publised in 1972.

One Sided Selection Method (OSS)

This method is a combination of CNN and Tomek Links. The method is proposed by Kubat and Matwin, in the paper published in 1972. The method first applies Tomek Links algorithm to reduce the borderline ambiguous majority examples. Then CNN algorithm applied in purpose of reducing the density of the majority class without hurting the classification performance.

Neighborhood Cleaning Rule (NCR)

This method combines CNN method with ENN method.

Laurikkala (2001) who is the person that propose the NCR mentions that the major downside of the OSS method is that the CNN has a tendency to add noisy examples to the training dataset. The classification performance does not only depend on the class distribution, details like noise is also important. In order to decrease the noise, the method employs ENN method in combination with CNN method.

The algorithm works as following:

1. The training dataset *T* is split into Positive class (*P*) and Negative Classes (*N*).
2. Using ENN algorithm, the noisy data is detected. The noisy data points which belongs to the Negative classes (*N*) is placed into a set called A_1.
3. For each negative class N_i in *N:*

   If $x \in N_i$ belongs 3-NN of misclassified *y* value that belongs to the Positive Class (*P*) and $|N_i| \leq 0.5 * |P|$:

   The example *x* will be added to a set called A_2.
4. Final version of the training data will be $T_{final} = T - (A_1 \cup A_2)$

3.3.3 Oversampling and Undersampling Combined

Oversampling algorithms creates synthetic data examples to increase the density of the minority class and undersampling methods delete examples from majority class to decrease the density of the majority class. Main methods that belong to the oversampling and undersampling categories are mentioned above. These methods can be implemented individually with success. However, using a combination of these methods can yield better results. (Brownlee, 2020).

The possible combinations can be listed starting from the most naïve ones: Random undersampling and random oversampling. Different ratios for the two techniques can be defined. For example, an imbalanced dataset with the imbalance ratio of 1:100 can be oversampled until the minority to majority class ratio becomes 20:80 and following this step by applying an undersampling method to make the distribution completely balanced (i.e. 50:50).

One of the most popular oversampling method is called SMOTE. The method is proposed by Chawla et al. (2002) and in the paper they proposed using SMOTE with an undersampling technique.

SMOTE can be combined with a variety of undersampling methods, SMOTE – Random Undersampling is one of the ways to do it.

On the other hand, there are some techniques that are proven to work well. The paper of Batista, Gustavo et al. (2016) has good method combinations. Some of them are: Combining SMOTE with Tomek Links method, combining SMOTE with ENN, combining CNN and Tomek links.

3.4 Cost-Sensitive Methods

Most classification algorithms assume that, the misclassification of the positive class is the same with misclassification of the negative class. This is not the case for most real world problem domains. (Thai-Nghe et al., 2010)

The example that a cancer patient (positive class) is diagnosed as non-cancer (false negative) is a much bigger problem than a non-cancer patient (negative class) is diagnosed as a cancer patient (false positive). Similarly, if a terrorist that carries a bomb belong to positive class, it is much of a bigger problem to misclassify the positive class example and let the terrorist walk away than misclassifying an innocent person (negative person) and search him/her (Sammut & Webb, 2010).

3.4.1 Cost-Sensitive (Weighted) Logistic Regression

Logistic Regression algorithm is a great algorithm for classification tasks. As for the other well-known classification algorithms, it does not work at its best when there is an imbalance between class labels. The base idea of finding the best suited logistic regression model is to minimize the log loss function. The algorithm is given below.

$$\min \sum_{i=1}^{n} -(\log(yhat_i) * y_i + \log(1 - yhat_i) * (1 - y_i)) \tag{3.34}$$

The above algorithm iteratively changes the coefficients until it finds the minimum log loss function. The above loss function gives equal weights to classes. Different

weights can be assigned for classes in order to mitigate the class imbalance effect. The above formula can be modified into a weighted log loss function as below.

(3.35)

High weight value means high importance and require more update on the coefficients. Low weight values referred as less importance and cause less modifications over the coefficients.

Three options of action are available for determining the class weights. First one is asking an expert of the problem domain. Second one is using hyperparameter tuning in order to find what works best for the specific data at hand. Last one is using what works best in general (e.g. the inverse of the class distribution).

3.4.2 Cost-Sensitive Decision Trees

Decision tree algorithm uses nodes to split the data. Each node is evaluated by the criterion chosen (e.g. entropy). The aim of the algorithm is to attain pure nodes, a node that has only one class examples are referred as "pure". The algorithm is simple and effective for balanced datasets. On the other hand, when used for imbalanced cases, this is not the case. Deriving pure nodes are not that hard for the algorithm since the majority class dominates the minority class and if your nodes contain all majority class examples the evaluation criterion would give good results although the algorithm almost completely ignores the minority class.

Cost-Sensitive Decision Trees eliminate this issue by manipulating the criterion by assigning weights for class labels. This method is offered by Ting, in his paper published in 2002.

In order to mitigate the negative effects of imbalance on the minority class, low weight can be assigned to majority class and high weight can be assigned to the minority class.

The weights can be determined by an expert of the problem domain, parameter tuning by grid search, or using inverse of the density of the classes in the dataset as weights.

3.4.3 Cost-Sensitive SVM's

SVM's is seen as a modified version of Perceptron algorithm. The perceptron finds a hyperplane that separates the classes in the training data. SVM's added the feature of Support Vectors which maximizes the margin between closest instance of each class to the hyperplane. SVM's function is given below.

$$\max(w^*, b^*) \frac{2}{\|w\|} \quad such \ that \quad y_i(\vec{w} * \vec{X} + b) \geq 1 \tag{3.36}$$

or

$$\min(w^*, b^*) \frac{\|w\|}{2} \quad such \ that \quad y_i(\vec{w} * \vec{X} + b) \geq 1 \tag{3.37}$$

Here, w is the vector which is a vector and it is perpendicular to the hyperplane.

Aforementioned formula belongs to the Hard Margin SVM's which works for linearly separable datasets. Finding linearly separable datasets in real life is not quite possible. This is why Soft Margin SVM's are used in most of the cases. The optimization function for Soft Margin SVM is given below.

$$\min(w^*, b^*) \frac{\|w\|}{2} + c \sum_{i=1}^{n} \zeta_i \tag{3.38}$$

For correctly classified samples ζ will be 0 and for incorrectly classified samples ζ will be equal to the distance of example to its class's corresponding support vector.

SVM's uses kernel tricks in order to make non-linearly separable data into separable one. The feature space transformed by using different kernel tricks so a hyperplane can be drawn which cannot be drown in the original feature space. Most commonly used kernels are linear, radial and polynomial transforms. In some situations, data is not linearly separable even with kernel tricks. For these situations soft margin SVM function which is given above is used.

In above formula soft margin regularization parameter is called c, it controls the balance between maximizing the margin between support vectors and minimizing examples that are misclassified (Fernández et al., 2018).

If c set to 0, that would be a hard margin which would not allow any violations of misclassification. When the value assigned for c increases, the margin gets softer and allows more misclassifications but increase the margin as a trade-off.

Even though SVM's are effective in balanced data, it is susceptible to give suboptimal performance for imbalanced data (He & Ma, 2013).

Parameter c which is described above is used as a penalty term to tune the algorithm in accordance with the imbalance rate. The weights of the classes are equal for each class so the softness of the margin is equal for classes as default and that favors the majority class. (Brownlee, 2020).

To accommodate this issue, higher weight can be assigned to minority class which makes the margin softer and lower weight can be assigned for majority class which in turn makes the margin to be harder for majority class.

This will cause the majority class to be stricter about misclassifications and do not allow misclassified minority class examples to the majority class side. On the other hand, minority class side will be more flexible about misclassified examples of majority class to be on the minority class side

By assigning different weights for different data points Weighted SVM algorithm learns the decision boundary in accordance with the relative density of the data points in dataset. (Yang & Song, 2007)

3.4.4 Weighted XGBoost Algorithm

Extreme Gradient Boosting (XGBoost) is a really powerful Tree boosting method which is widely used in Machine Learning problems. The algorithm is developed by Chen and Guestrin (2016). It is a union of decision trees algorithm in which new trees correct the errors of the previously built trees. This process continue until no improvements can be offered by new trees.

The algorithm provides state-of-art performance on many classification tasks and its scalability to all types of classification scenarios is the most important aspect of this algorithm (T. Chen & Guestrin, 2016).

The algorithm has variety of hyperparameters which should be tuned in order to get the best results possible. One of these parameters is the parameter where we can set the weight of positive class. In the algorithm, gradient term indicates how steep is the loss function which the algorithm aims to minimize. If gradient is high, that shows high error, If the gradient is low, the error is low. The parameter of assigning weight to positive class actually emphasizes the importance of positive class when increased. The model gives more importance to the positive class since the increase in the parameter will result higher gradient which means higher error to correct. Too much increase in the weight parameter can cause overfitting of the model where the prediction quality of majority class or both suffers as a result.

The parameter is set to 1 as default. The logical approach for the parameter value is to use the inverse of the class distribution

3.5 Algorithm Level Methods

Algorithm level methods are also called as internal methods. Under this category, there are methods referred as threshold moving, probability calibration, ensemble algorithms and one-class classification.

3.5.1 Threshold Moving

Most of the classification algorithms calculates the probabilities and uses thresholds to turn these probabilities into class labels. Default thresholds used by the predictive models are as given below:

$$\begin{aligned} &Predicted\ \ probability < 0.5 \rightarrow class\ 0 \\ &Predicted\ \ probability \geq 0.5 \rightarrow class\ 1 \end{aligned} \tag{3.39}$$

In order to use predicted probabilities in the best possible manner, above thresholds falls short for the following reasons:

1. Some algorithms, because of their nature, give uncalibrated probabilities,
2. Severe class distribution of the dependent variable,
3. The metric used in the training phase does not match with the metric used in the testing phase,
4. The misclassification costs are not equal between positive and negative class.

Provost (2000) mentioned that it would be a mistake to use the predicted class labels produced by standard machine learning algorithms without adjusting the threshold. In order to resolve aforementioned problems, moving threshold to find the optimal one is a required step.

Using other methods, for example sampling, without trying the set optimum threshold can be misleading, threshold moving simply uses original dataset and moves the threshold to an optimum place where the minority class is recognized and predicted better (He & Ma, 2013)

The algorithm of threshold moving is as follows: Model is fitted using training data. After that, the probabilities of belonging a class is predicted using test dataset. Following that step the candidate thresholds are used to convert the probabilities calculated in the previous step. Class labels are evaluated with an appropriate metric and highest scoring threshold is chosen.

3.5.2 Probability Calibration

Many Machine Learning algorithms can predict probability scores of belonging a specific class. Unfortunately, most of these probabilities are not well calibrated. Some of the models give optimistic probabilities, on the other hand, some of them give pessimistic ones. This problem is amplified when the class imbalance is present in the dataset. Hence, calibrating probabilities is a good practice especially for Imbalanced classification.

Calibrated probabilities show the real likelihood of the class membership of an example. To give an example, assume that one mail account has received 50 mails and 4 of them are spam. That means 8% of the mails are spam. This is the case when probabilities are calibrated. Calibrated probabilities set a good foundation for classification (Brownlee, 2020).

Some Machine Learning algorithms provides calibrated probabilities, these are the ones that use probabilistic framework. Logistic Regression, Artificial Neural Networks, LDA, Naïve Bayes, can be counted in this category.

On the other hand, some algorithms only predict class labels or uncalibrated probability scores. SVM's, k- Nearest Neighbors (KNN), Decision Trees can be counted in this category.

One of the most common ways to scale probabilities are Platt scaling, which uses Logistic Regression for Probability calibration. The other one is Isotonic Regression, which implements Weighted least squares Regression for probability calibration.

3.6 Ensemble Algorithms

3.6.1 Bagging for Imbalanced Classification

Bagging name comes from Bootstrap Aggregation. The algorithm is a part of Ensemble algorithm because it builds and ensembles different decision trees in order

to make a better prediction than a simple decision tree. The algorithm uses Bootstrapping to attain different samples which is used to build different decision trees. As a last step, each model in the ensemble makes predictions on a new sample and the average of the predictions is used as the prediction of the bagging classifier (Brownlee, 2020).

The Bootstrapping to create new samples and aggregation of the new models to the ensemble continues until the peak prediction performance is achieved which is the point that yields no additional improvement by model aggregation. This peak point is decided by using a validation set.

Since the bagging algorithm applies bootstrap to create samples, the class imbalance negatively affects the number of examples chosen from the minority class. One of the most commonly used method is oversampling the minority class or undersampling the majority class examples prior to the bagging algorithm.

3.6.2 Random Forest Classifier for Imbalanced Classification

Random Forests can be thought as an improvement of bagging algorithm. The algorithm has significant similarities with bagging algorithm. The sampling creation via bootstrap exists in the Random Forest Algorithm. The difference lies in random selection of different subsets of features for each model. This renders the correlation of the created models low. The prediction of the Random forest attained by the average of the all tree models that are built throughout the process by the algorithm. (Kuhn & Johnson, 2013).

In imbalanced datasets, it is highly likely that minority class is represented poorly or not represented at all in bootstrap samples, which automatically results poor prediction performance over minority class examples (C. Chen et al., 2004).

There are some techniques to solve this issue. One of them is using weighted Random Forest. This category has 2 subcategories. One of them is using class weights based on training data imbalance ratio, other one is giving different class

weights for each bootstrap sample. The logic behind the sample focused weighting is each bootstrap sample contains different minority to majority ratio. Other technique is using under or over sampling on training data before the usage of the algorithm.

3.6.3 Easy Ensemble for Imbalanced Learning

The Easy Ensemble algorithm is proposed by Liu et al. in their paper published in 2008. They also mentioned that even though undersampling is effective for class imbalance, some of the potential value is lost by ignored majority class examples. In order to prevent this, Easy Ensemble algorithm is offered by the authors. The algorithm works as following:

Assume that minority class examples in training dataset is called P and the majority class examples in the training data is called T. The algorithm uses all examples in P and randomly samples from T. The subset taken from T is shown as T' and number of items in T' is equal to the number of items in P. AdaBoost algorithm is used to train models. All models are averaged to get the Easy Ensemble model performance.

3.7 One Class Classification

Outliers in a dataset usually causes problems. The mean and variance are directly affected by existence of outliers. Also removing outliers before building predictive models are also suggested.

There are several methods to detect outliers in a dataset. One of them is called One Class Classification. This classifiers are trained with non-outlier or normal datasets and new instances are classified as normal or anomaly(outlier) later on. Compared to the binary classification, main differences can be counted as:

1. Positive (minority) class is named as anomaly(outlier) and encoded as "-1" instead of "0"

2. Negative (majority) class is called as normal instances and encoded as “1” as per in the binary classification.
3. Only majority class included in the training part.

This method is suitable for the problems in which minority class examples are so rare that including in the training of a binary classification model would be trivial. On the other hand, if minority class lacks a structure, and composed by noisy or very little clusters, one class learning also works well.

A big disadvantage of using one class learning on imbalanced classification problems is discarding all useful information in minority class. This makes the method not feasible for some problem domains (Fernández et al., 2018).

CHAPTER 4

DATASET

The dataset used for the study obtained from www.barttorvik.com. It is an internet site about college basketball statistics. The dataset was obtained as .csv format. The dataset consist of college basketball players from 2008 to 2022. The default version of the dataset contains 63 features and 70610 observations.

4.1 Variables

All variables in NCAA College Basketball League Player Statistics Dataset is given below.

1. **Player Name**

Name of the player

2. **Team**

Team of the player

3. **Conference**

The name of the conference. Conference is a collection of the teams.

4. **Games Played**

Total games played by the player.

5. **Minutes Percentage**

Minutes played by the player divided by the total time played by the team in the season. It shows what percentage of the whole season the player was on the court.

6. **Offensive Rating (ORtg)**

How many points a player possibly produce when he attempts.

7. Usage (usg%)

It shows the estimated percentage of the team plays that a player used. The formula is as follows:

$$usg\% = 100 * ((FGA + 0.44 * FTA + TOV) * (TmMP / 5)) / (MP * (TmFGA + 0.44 * TmFTA + TmTOV)) \quad (4.1)$$

8. Effective Field Goal Percentage (eFG%)

the formula is:

$$eFG\% = \left(FG + 0.5 * 3P\right) / FGA \quad (4.2)$$

Assume player A scored 10 points by 2 two point field goals and 2 three point field goals, on the other hand, Player B scored 10 points by 5 two point field goals. Both of the players scored 10 points so the Effective Field Goal Percentage for both players is %50.

9. True Shooting Percentage (TS%)

The formula of the metric is:

$$TS\% = PTS / (2 * TSA) \quad (4.3)$$

10. Offensive Rebound Percentage (ORB%)

The percentage of the potential offensive rebounds the player may grab while he was on the court. The formula was given below.

$$ORB\% = 100 * (ORB * (TmMP / 5)) / (MP * (TmORB + OppDRB)) \quad (4.4)$$

11. Defensive Rebound Percentage (DRB%)

The percentage of the potential defensive rebounds the player may grab while he was on the court. The formula was given below.

$$DRB\% = 100 * (DRB * (TmMP / 5)) / (MP * (TmDRB + OppORB)) \quad (4.5)$$

12. Assist Percentage (AST%)

It is a prediction of the team scores which the player assisted while he was on the court. The formula is as follows:

$$AST\% = 100 * AST / (((MP / (Tm MP / 5)) * Tm FG) - FG) \quad (4.6)$$

13. Turnover Percentage (TOV%)

Predicted turnover count per 100 possessions. Formula is as follows:

$$TOV\% = 100 * TOV / (FGA + 0.44 * FTA + TOV) \quad (4.7)$$

14. Free Throws Made (FT)

Free throws the player made in the season.

15. Free Throw Attempts (FTA)

Free throw attempts of the player in the season.

16. Free Throw Percentage (FT%)

Free throws made divided by free throw attempts.

17. Two Points Made (2P)

Two point field goals made by the player.

18. Two Point Attempts (2PA)

Two point field goal attempts of the player

19. Two Point Percentage (2P%)

Two points made divided by two point attempts

20. Three Points Made (3P)

Three point field goals scored by the player

21. Three Point Attempts (3PA)

Three point field goal attempts by the player.

22. Three Point Percentage (3P%)

Three point field goals made divided by three point attempts.

23. Block Percentage (BLK%)

A prediction of the percentage of the opponent field goals blocked by the player while he was on the court. The calculation of this metric is as follows:

$$BLK\% = 100 * (BLK * (Tm\,MP / 5)) / (MP * (Opp\,FGA - Opp\,3PA)) \quad (4.8)$$

24. Steal Percentage (STL%)

Percentage of the opponent possessions that end with a steal by the player. The formula is given below:

$$STL\% = 100 * (STL * (Tm\,MP / 5)) / (MP * Opp\,Poss) \quad (4.9)$$

25. Free Throw Rate (FTR)

Free Throw Rate is Free Throw Attempts divided by Field Goal Attempts.

26. Year in College

It is a categorical variable which can take values of Freshman, Sophomore, Junior, Senior.

27. Height

Height of the player.

28. Points Over Replacement Per Adjusted Game (Porpag)

Porpag is a metric to predict how many more points a given player can score compared to a hypothetical substitute. The formula of the metric is given below:

$$porpag = (ORtg - 88) * Poss\% * Min\% * 65 \quad (4.10)$$

In this equation, 88 is the expected rating and 65 is the constant for possessions.

29. Adjusted Offensive Efficiency (AdjOE)

An estimate of the points per 100 possessions a team would score against the defense of an average division 1 team

30. Personal Foul Rate (pfr)

Personal fouls per 40 minutes

31. Year

Year of the season.

32. Player ID (pid)

ID's that are assigned for each unique player.

33. Assist / Turnover (ast/tov)

Assists of the player divided by turnovers of the player.

34. Rimmade

The number of shots made around the rim.

35. Rimmade + Rimmiss

The number of shots made around the rim plus the number of shots missed around the rim.

36. Midmade

Number of shots made from midrange

37. Midmade + Midmiss

Number of shots made from midrange plus number of shots missed from midrange

38. Rimmade / (Rimmade + Rimmiss)

Attained by dividing "Rimmade" by "Rimmade plus Rimmiss".

39. Midmade / (Midmade + Midmiss)

Attained by dividing the number of shots made by the player by total midrange shot attempts by the player.

40. Dunksmade

Number of dunks made from the player

41. Dunksmiss + Dunksmade

Number of dunks which are missed by the player plus the number of dunks that are made by the player

42. Dunksmade / (Dunksmade + Dunkmiss)

Number of dunks that are made by the player divided by the total dunk attempts of the player.

43. Pick

Shows the players draft status. 1 for drafted 0 for not drafted. If this variable equals to 1, it means the player is drafted in his career, it does not mean he is drafted at the specific season.

44. Defensive Rating (drtg)

For 100 possessions, how many points are allowed by the player.

45. Adjusted Defensive Rating (adrtg)

For 100 possessions, how many points are allowed to the league average.

46. Defensive Points Over Replacement Per Adjusted Game (dporpag)

Defensive approach to the Points Over Replacement Per Adjusted Game metric.

47. Stops

Stops are blocks, steals, defensive rebounds plus the likelihood of turnovers and misses forced by the individual which did not ended up as a steal or a block.

48. Box Plus/Minus (bpm)

Points above the average league player for 100 possessions.

49. Offensive Box Plus/Minus (obpm)

Offensive version of the box plus/minus.

50. Defensive Box plus/minus (dbpm)

Defensive version of box plus/minus.

51. Game Box Plus/minus (gbpm)

New version of Box Plus/minus.

52. Minutes Played (mp)

Total minutes played by the player.

53. Offensive Game Box Plus/Minus (ogbpm)

New version of the offensive box plus/minus.

54. Defensive Game Box Plus/Minus (dgbpm)

New version of the defensive box plus/minus.

55. Offensive Rebounds (oreb)

Total offensive rebounds of the player.

56. Defensive Rebounds (dreb)

Total defensive rebounds of the player.

57. Total Rebounds (treb)

Summation of offensive rebounds and defensive rebounds.

58. Assist (ast)

Total assists the player made.

59. Steal (stl)

Total steals the player made.

60. Block (blk)

Total blocks the player made.

61. Points (pts)

Total points the player made.

62. Position (Pstn)

Position of the player (e.g. point guard, center)

63. Threes Per 100 Possessions (TPA / 100)

Three point attempts of the player per 100 Possessions.

4.2 Data Preprocessing

Data preprocessing is the most time-consuming part in any Machine Learning problem. The data preprocessing steps taken in this study were given below.

4.2.1 Minutes Played Percentage Threshold

As stated in the Features section, Minutes Played Percentage (Min_per) feature is a measure of percentage of the total match time in which the player was on the court. If the value of this feature is low (e.g. 3) that means the player was not played in this season much. This can cause some statistics of the player to be unnaturally high or low (e.g. Offensive Rating, Usage, etc.). In addition to this, players who are drafted have played at least %10 percent of the total season. There are only 2 exceptions to this.

Which can be seen in below table:

Table 4.1 Drafted Players Whose Minutes Played Percentage is Below 10

Player Name	Minutes Played %	Games Played	Drafted
Michael Porter Jr.	4.1	3	Yes
James Wiseman	5.5	3	Yes

Michael Porter Jr. was injured and got a back surgery. Therefore he missed most of the season. Due to his promising high school career, he was drafted regardless of the injury.

James Wiseman on the other hand, deemed ineligible to play in NCAA since his family accept 11,500 dollars from the Penny Hardaway (Who became the head coach of the Memphis Tigers team in future) when James Wiseman was a junior at high school. After that NCAA give him 12 games suspension (each game removes 1000

dollars that he received when he was at high school). After that NCAA demanded 11.500 dollars donation to charity in order to play again for Memphis again. He reported that he will stop his college career and prepare with an agent to NBA DRAFT 2020. This is why he has played only 3 games in NCAA. He Picked by Golden State Warriors in 2020 as a 1st round 2nd order pick.

Other than aforementioned drafted players, there are no other drafted players who played less than 10 percent of the season. Additionally, it is a fact that majortiy of the drafted players played above a certain threshold, and under a certain threshold of minutes played percentage, no players are not drafted. there are lots of players whose minutes played percentage is lower than 1. Due to the fact that this can lead algorithms to mistakenly classify players who has low amounts of minutes played percentage (e.g. lower than 5 percent) as not drafted and classify players with higher amounts of minutes played percentage (e.g. more than 90 percent) as drafted. Even though this would not be completely wrong, there are high level players who were seen as a future NBA player got a minor injury and missed a proportion of the season which end up decreasing his minutes played percentage. In order to prevent algorithms to shortcut the learning process, a threshold of 10 percent is added to the dataset. In other words, players whose minutes played percentage is below 10 is ditched from the data except the players in table 4.1. since the drafted players are really rare in the dataset. Loss of observations belong to the minority class cannot be accepted.

4.2.2 Missing Values in Year and Height Variables

There were 35 players in total whose year or height information have been missing. The correct information is collected from the internet and imputed for the observations that lack this information under the variables mentioned.

4.2.3 Height Transformation

Height values of the players were shown as 6-3 (6 feet 3 inches) or 7-1 (7 feet 1 inches) in a string format. When the format changed to the integer format the Python thought 6-3 is a subtraction and gives the result of 3 as the players height. In order to correct this issue, all height transformed into inches. Since 1 foot is equal to 12 inches, 6 feet 3 inches were encoded as 75 inches in the dataset.

4.2.4 Assist / Turnover Missing Data Imputation

15 observations had had missing data in the Assist / Turnover variable. Assist and Turnover totals acquired from the internet and ratio of them were imputed to corresponding missing values.

4.2.5 Dropping the variables that has Missing Variables

Below variables deleted from the dataset since they contain too much missing data:

- Rimmade,
- Rimmade + Rimmiss,
- Midmade,
- Midmade + Midmiss,
- Rimmade / (Rimmade + Rimmiss),
- Midmade / (Midmade + Midmiss),
- Dunksmade,
- Dunksmiss + Dunksmade,
- Dunksmade / (Dunksmade + Dunkmiss).

4.2.6 Label Encoding Applied to Year in College Feature

Year in college is an ordinal categorical feature which has 4 labels (i.e. Freshman, Sophomore, Junior, Senior). Label encoding was applied to these labels since the variable is not nominal, but ordinal.

4.2.7 Standardization and Min-Max Feature Scaling of the Data

For all the experiments, 3 versions of the dataset were used: First version was the dataset with standardized features, in other words, features are transformed into Z scores with below formula.

$$Z = \frac{x - \mu}{\sigma} \tag{4.11}$$

Second version was normalized versions of the data features. The min-max feature scaling normalization formula is given below:

$$x_{norm} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{4.12}$$

For the third version of the input data, Principal Component Analysis is applied and the dimension of the training dataset is reduced. The steps taken throughout Principal Component Analysis is given below.

4.2.8 Principal Component Analysis

The scree plot which shows principal components on the X axis and explained variance on Y axis is given in Figure 5.1 below.

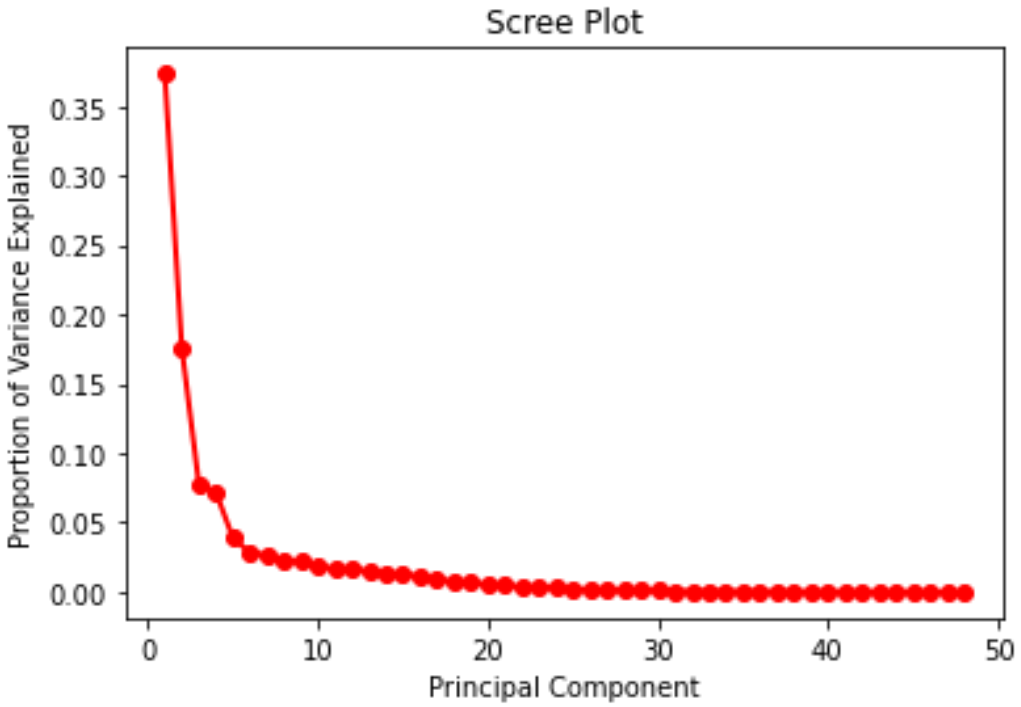


Figure 4.1 Scree Plot of Principal Components and Explained Variance Ratios

By looking the scree plot, it can be seen that after 6th principal component, the additional variance explanation gets lower and lower, on the other hand, from 6th to 20th component there is a visible decline which may be a sign of losing valuable information if one cuts the Principal Components at 6. The big picture can be seen from scree plot but in order to be more precise, numerical values of the variance explanation should be checked.

In order to see more detail about variance explanation of the principal components, Table 4.2 was created. Since there are 48 variables in the training set and after 20th component the increase of variance explanation is significantly low, the rows of the table only extended to 20th component.

Table 4.2 Principal Components, the Amount of Variance They Explain and Model Performances

PC	Exp.Var.	Cum.Exp.Var.	LogReg F1	LogReg G-Mean
PC1	0.37358675	0.37358675	0.360	0.495
PC2	0.17590321	0.54948996	0.373	0.509
PC3	0.07818494	0.62767491	0.407	0.534
PC4	0.07203243	0.69970734	0.421	0.547
PC5	0.03912875	0.73883609	0.426	0.553
PC6	0.02835085	0.76718693	0.419	0.550
PC7	0.02661743	0.79380437	0.452	0.585
PC8	0.02262434	0.8164287	0.476	0.604
PC9	0.02158208	0.83801078	0.537	0.657
PC10	0.01825058	0.85626136	0.537	0.659
PC11	0.01707665	0.87333801	**0.560**	**0.673**
PC12	0.01629209	0.8896301	**0.615**	**0.719**
PC13	0.01521112	0.90484122	**0.672**	**0.762**
PC14	0.01357958	0.91842081	0.679	0.768
PC15	0.01238499	0.9308058	0.677	0.766
PC16	0.01177007	0.94257588	0.692	0.778
PC17	0.00851607	0.95109194	0.686	0.776
PC18	0.00806393	0.95915588	0.696	0.785
PC19	0.00655405	0.96570993	0.706	0.791
PC20	0.005598	0.97130794	0.702	0.789

In this table, PC13 and earlier components explains 90% of the variance in the training data. Also, grid search is applied using Logistic Regression to find out critical thresholds for components. Stratified K fold Cross Validation is used for evaluation of the models created. It seems that there are significant increases in F1 score and G-mean score around PC13 and after that Threshold the increase is not significant. In Figure 4.2. and Figure 4.3 the fluctuation in G-Mean score and F1

Score can be seen more clearly. The performance of the Logistic Regression model increases until it reaches PC13. From that point on, increases happen but in a very slow manner. The number of components decided as 13 after these analyses.

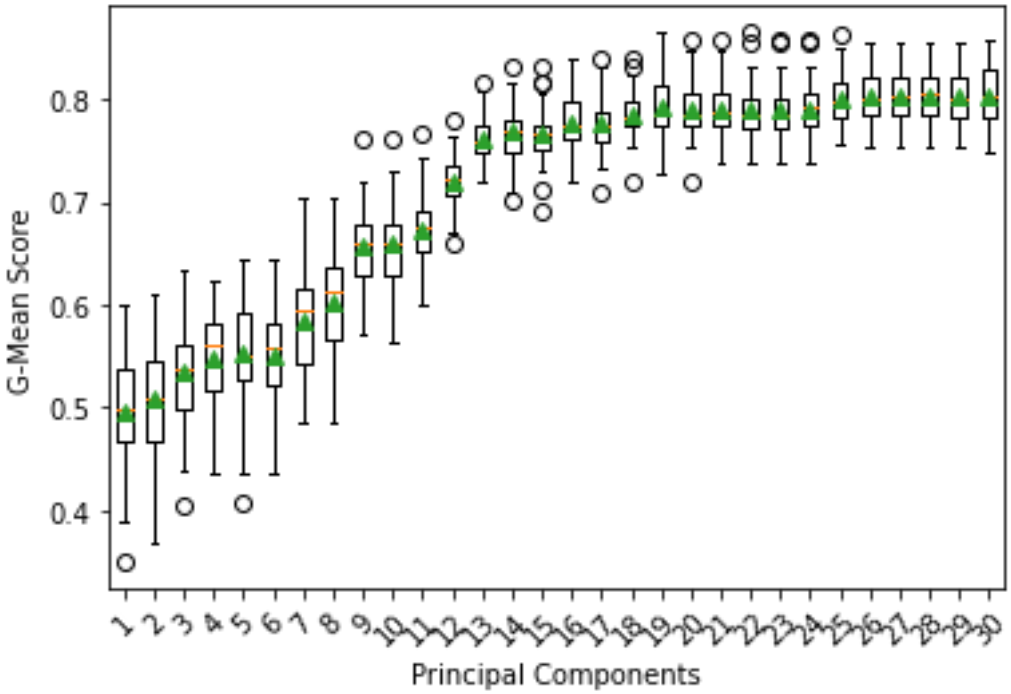


Figure 4.2 Box Plot of the Grid Search Results for the Number of Components That Yields the Best G-Mean Score

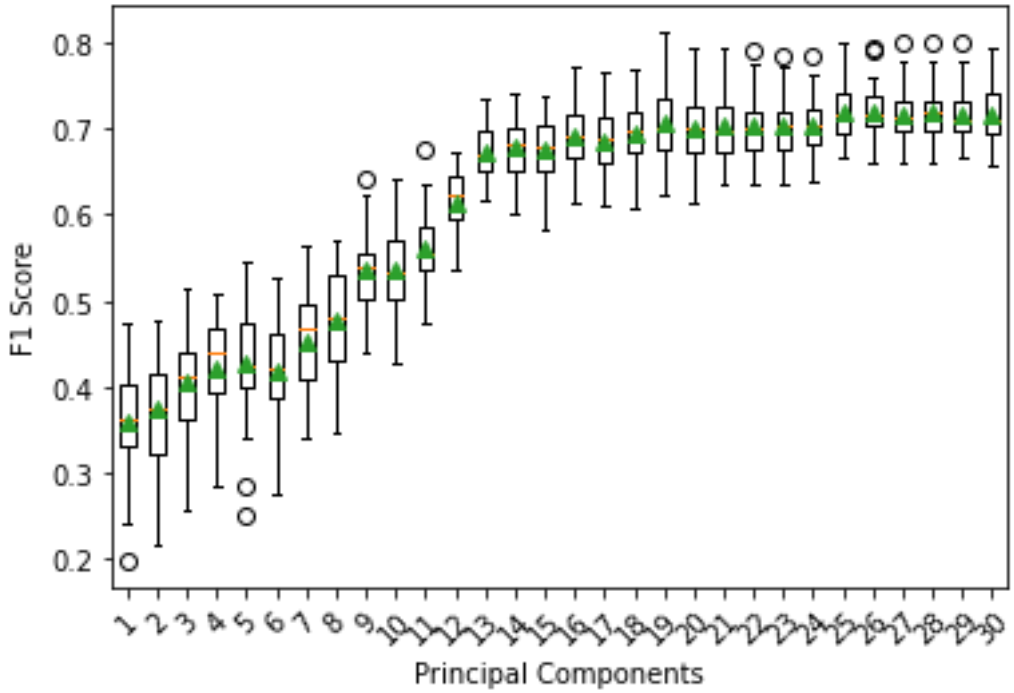


Figure 4.3 Box Plot of the Grid Search Results for the Number of Components That Yields the Best F1 Score

CHAPTER 5

EXPERIMENTS AND RESULTS

5.1 Baseline Model Performances

In this chapter, the methods which are elaborated upon on methodology part are tested upon the NCAA Player Statistics dataset.

There are many machine learning algorithms one can use for variety of machine learning classification problems. Most common ones are Logistic Regression (LR_liblin: liblinear solver Logistic Regression and LR_lbfgs: lbfgs solver Logistic regression), SVM's (SVM_lin: SVM's linear kernel and SVM_rbf: SVM's rbf kernel), Decision Trees (D.Tree), Naïve Bayes (N.Bayes), K-Nearest Neighbors (KNN), Random Forest (R.Forest), Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Bagged Decision Trees (BAG), Extra Trees (ET), XGBOOST (XGB) etc. Before applying techniques dedicated upon imbalanced learning, the baseline model performances are evaluated to have a foundation to build upon. F1 score, G-Mean score and Accuracy Scores are calculated and given in Table 5.1. These scores were chosen since F1 score is the metric which is widely used in imbalanced learning and it is the harmonic mean of 2 popular metrics (i.e. precision and recall) that measures the positive class prediction quality. G-Mean is used because it is seen as an alternative metric for imbalanced cases where two classes has equal importance. Accuracy is given because it is the most popular metric in Machine Learning even though it is not suitable for imbalanced learning. All scores were attained by Repeated Stratified 10-fold Cross Validation with 3 repeats. Below results are attained from Original Dataset, no standardization, normalization is applied on the data.

Table 5.1 Baseline Model Results on Original Dataset

	Standardized			Normalized (min-max scaling)			PCA		
Alg.	F1	G-M	Acc.	F1	G-M	Acc.	F1	G-M	Acc.
LR_liblin	**0.723**	0.805	**0.983**	0.700	0.778	0.982	0.672	0.759	**0.981**
LR_lbfgs	**0.723**	0.807	**0.983**	0.700	0.778	0.982	**0.675**	0.764	**0.981**
SVM_lin	0.717	0.804	**0.983**	0.705	0.781	**0.983**	0.670	0.755	**0.981**
SVM_rbf	0.696	0.770	0.982	0.700	0.770	**0.983**	0.669	0.750	**0.981**
N.Bayes	0.314	0.895	0.861	0.314	0.895	0.861	0.571	0.738	0.972
D.Tree	0.594	0.767	0.971	0.586	0.783	0.971	0.486	0.696	0.965
R.Forest	0.602	0.694	0.979	0.613	0.686	0.979	0.574	0.665	0.978
KNN	0.580	0.673	0.978	0.619	0.717	0.978	0.557	0.658	0.977
LDA	0.668	0.855	0.975	0.705	0.855	0.975	0.637	0.780	0.976
QDA	0.319	**0.897**	0.864	0.319	**0.897**	0.864	0.565	**0.821**	0.964
BAG	0.646	0.745	0.981	0.653	0.736	0.980	0.561	0.660	0.975
ET	0.630	0.718	0.980	0.637	0.719	0.979	0.508	0.584	0.976
XGB	0.720	0.806	**0.983**	**0.720**	0.806	**0.983**	0.648	0.751	0.979

Above table shows 13 machine learning algorithms as rows and 3 evaluation metrics as columns. Evaluation metrics are calculated for 3 version of the dataset. First one is Standardized data, second one is Normalized data and the last one is 13 Principal Components of the dataset (out of 48 features).

F1 score is the harmonic mean of Precision and Recall score. Both of the metrics are related with positive class. Above plot results shows that the highest F1 score belongs to the Logistics Regression (both linear solver and lbfgs solver type) algorithm which is 0.723. XGBoost F1 score is clolse to that score which is 0.720. The performance comparision of all models for standardized features with box plots can be seen in Appendix A.

The reason Naïve Bayes algorithm offer comparatively low F1 score is that the Naïve Bayes uses Prior probabilities which in this case is the distribution of the classes (Class0 = 96.611% and Class1 = 3.389%). Since the prior probability is multiplied with the probability of *datapoint given class label* which is called likelihood to find posterior probability which can be declared as *class label given datapoint*. To conclude, datapoints in the training dataset with class label 0 starts the training process with a huge disadvantage because of the fact that prior probability of minority class is very low and this low probability is a multiplier which decrease the overall probability of belonging to class 0 dramatically. This explains low F1 score of the Naïve Bayes algorithm.

G-Mean score is the geometric mean of sensitivity and specificity. In other words, it is the geometric mean of the positive class accuracy and negative class accuracy. This is a metric popular for imbalanced classification since a great prediction performance on the negative class does not make up for a poor positive class prediction performance as per the accuracy score. Highest score belongs to Quadratic Discriminant Analysis (QDA) which is 0.897 slightly surpasses the score of Naïve Bayes which is 0.895.

Normalized features yielded similar results with standardized features for most of the algorithms. Regardless, there are differences for some algorithms. For example, Logistic Regression performed better in terms of F1 score with standardized coefficients than normalized coefficients. KNN (K=5) on the other hand, performed better on the normalized coefficients in terms of F1 and G-Mean scores. However, the best performing models for the specific dataset (e.g. Linear regression, SVM) performs better on standardized dataset. Hence, the remaining parts of the experiments, standardized features opted over normalized features.

13 principal components were chosen with Principal Component Analysis which is detailed on the Data Preprocessing part of the study and tested with same algorithms. The time required for algorithms to learn significantly dropped but the prediction performances also dropped. This means reduced model of 13 Principle components

is not a feasible for robust model building purposes. The Experiments were continued with standardized features.

All results in Table 5.1 were given in Appendix A-I as box plots.

5.2 Oversampling and Undersampling Methods

On this part of the study, Oversampling, Undersampling and combinations of both is applied.

5.2.1 Oversampling Methods Performance Comparison

In this part of the study, oversampling method performances are compared. The algorithms used for this part is Logistics Regression, SVM's and XGBoost Since they were the best performing models on previous part.

The most basic form of oversampling is random oversampling in which the minority class examples oversampled randomly. The random oversampling F1 score for Logistic Regression is 0.562.

Table 5.2 Oversampling Techniques' F1 Scores

	SMOTE	BLSMOTE	SVMSMOTE	ADASYN
LR	0.596	0.591	0.621	0.561
SVM_lin	0.575	0.575	0.613	0.546

In above table, performances of most popular oversampling techniques are presented. SMOTE, Borderline SMOTE (BLSMOTE), Borderline SMOTE SVM (SVMSMOTE), ADASYN techniques are used and each of them represented as columns in the table. The rows of the tables contain learning algorithms which performed best in previous section (Table 5.1). Intersections of learning algorithm and oversampling methods F1 scores were shown. F1 score was chosen for

evaluation because the metric is widely used for imbalanced classification cases and put positive class (i.e. minority class) more importance compared to other metrics like G-Mean or accuracy.

By looking at above scores, it can be said that all oversampling techniques yielded worse performance in terms of F1 score compared to Logistics Regressions default F1 score which is 0.723 and SVM's default score of F1 which is 0.717 (Table 5.1). This might be happened due to overfitting because the minority class density is 3,389% which makes the imbalance severe. Default versions of the oversampling techniques creates synthetic samples with different approaches to match the density of the minority class with majority class and there are 726 examples in the minority class and 20696 examples in the majority class. So that means default oversampling techniques creates synthetic samples whose total count is 19.970 which can cause overfit and poor performance on test datasets. Thankfully, the algorithms allow to change the minority / majority ratio before transforming the dataset. If overfit theory is true, lower minority / majority ratio should yield better scores. In Table 5.3 Same learning techniques and same algorithms were used but minority over majority class ratio parameter was set to 0.1 instead of the default score which is 1.

Table 5.3 Oversampling Techniques with 0.1 Minority Over Majority Class Ratio F1 Scores

	SMOTE	BLSMOTE	SVMSMOTE	ADASYN
LR	**0.737**	0.732	0.733	**0.737**
SVM_lin	0.734	0.728	0.731	0.727

Above table shows new F1 scores after the minority over majority class ratio parameter is modified from 1 to 0.1. The results are much better compared to the default ratio parameter F1 scores. Also, all of the oversampling techniques provided better results compared to the default scores of each learning algorithm. For Logistic Regression algorithm, SMOTE and ADASYN provided best F1 scores which is 0.737 for both oversampling techniques. On the other hand, SVM's with linear

kernel algorithm performed best with a dataset oversampled using SMOTE. The F1 score of SVM's with SMOTE is 0.734. In general, best prediction performance come from Logistics Regression with an F1 score of 0.737.

Finding the Best Ratio Hyperparameter Value for Oversampling Techniques

In this part of the study, the best oversampling parameter of class ratio was detected for 2 best performing learning algorithms for the NCAA Player Stats dataset which are Logistic Regression and SVM's.

Oversampling Methods Hyperparameter Tuning for Logistic Regression

Changing the minority / majority ratio from 1 to 0.1 increased the performance of all oversampling techniques in Chapter 5.2.1. In this part of the study, grid search is used to detect best hyperparameter for prediction performance of the model. Lowest ratio is determined as 0.05 since 0.036 is the original ratio of the minority class and oversampled ratio cannot go below that. Upper limit of minority over majority ratio determined as 0.20. Same 4 oversampling techniques were used as per in chapter 5.2.1 and they are placed as column headers in table 5.4. Logistic Regression is used for grid search and results obtained can be seen in in below table.

Table 5.4 Ratio Hyperparameter Comparison for Logistic Regression

	SMOTE	BLSMOTE	SVMSMOTE	ADASYN
0.05	0.7356	0.7353	0.7335	0.7363
0.06	0.7395	0.7385	0.7390	0.7435
0.07	0.7420	**0.7411**	**0.7403**	**0.7441**
0.08	**0.7434**	0.7398	0.7397	0.7429
0.09	0.7430	0.7362	0.7332	0.7369
0.10	0.7391	0.7361	0.7352	0.7354
0.11	0.7362	0.7282	0.7299	0.7278
0.12	0.7313	0.7256	0.7266	0.7213
0.13	0.7301	0.7182	0.7266	0.7181

Table 5.4 (continued)

0.14	0.7265	0.7162	0.7208	0.7159
0.15	0.7232	0.7130	0.7192	0.7080
0.16	0.7157	0.7100	0.7168	0.7051
0.17	0.7136	0.7041	0.7127	0.6976
0.18	0.7142	0.7003	0.7135	0.6948
0.19	0.7095	0.6992	0.7077	0.6913
0.20	0.7054	0.6965	0.7084	0.6862

Best minority over majority ratio for SMOTE observed to be 0.08, with this parameter Logistic Regression achieved the F1 score of 0.7434. Borderline SMOTE, Borderline SMOTE SVM and ADASYN algorithms performed best at the ratio of 0.07. Borderline SMOTE and Logistic Regression achieved the F1 score of 0.7411, Borderline SMOTE SVM and Logistic Regression achieved 0.7403 F1 score. ADASYN algorithm with the ratio parameter equal to 0.07 makes the Logistic Regression achieve the F1 score of 0.7441. All results were attained by applying 10-fold cross validation with 3 repeats. Highest scores are shown in bold numbers.

Oversampling Methods Hyperparameter Tuning for SVM's

Grid search is applied to find best hyperparameter value for minority over majority class ratio. The experiment results were given in Table 5.5 below.

Table 5.5 Ratio Hyperparameter Comparison for SVM's

	SMOTE	BLSMOTE	SVMSMOTE	ADASYN
0.05	0.7359	0.7373	0.7392	0.7329
0.06	0.7400	0.7397	**0.7415**	0.7394
0.07	0.7363	**0.7414**	0.7342	0.7406
0.08	**0.7437**	0.7380	0.7325	**0.7417**
0.09	0.7422	0.7389	0.7314	0.7351
0.10	0.7379	0.7290	0.7290	0.7311

Table 5.5 (continued)

0.11	0.7331	0.7278	0.7293	0.7212
0.12	0.7275	0.7195	0.7274	0.7152
0.13	0.7242	0.7147	0.7220	0.7090
0.14	0.7223	0.7106	0.7178	0.7076
0.15	0.7200	0.7069	0.7178	0.6962
0.16	0.7116	0.6993	0.7165	0.6952
0.17	0.7065	0.6976	0.7146	0.6863
0.18	0.7066	0.6970	0.7096	0.6842
0.19	0.7029	0.6914	0.7090	0.6839
0.20	0.6972	0.6882	0.7045	0.6778

Linear Kernel SVM's were used to obtain the results shown in the table. F1 scores are attained using 10-fold cross validation with 3 repeats.

SMOTE oversampling technique provided best results for SVM's algorithm when ratio hyperparameter is equal to 0.08. The F1 score when the SMOTE ratio hyperparameter set to 0.08 is 0.7437.

Borderline SMOTE algorithm performed best when the ratio hyperparameter set to 0.07. The best F1 score is SVM's attained with Borderline SMOTE is 0.7414.

Borderline SMOTE SVM oversampling algorithm provided best performance when the ratio parameter set to 0.06. F1 score attained by this hyperparameter is 0.7415.

ADASYN algorithm worked best when the hyperparameter of minority over majority class ratio set to 0.08. The F1 score attained by this hyperparameter is 0.7417.

In general, Oversampling algorithms performed best when they are set between the range of [0.6, 0.8] for both Logistic Regression and SVM's.

5.2.2 Undersampling Methods Performance Comparisons

In this part of the study, most popular and effective undersampling algorithms are tested with learning algorithms that performed best on raw dataset which are Logistic Regression which is denoted as LR and SVM's with linear kernel which is denoted as SVM_lin. Undersampling algorithms used for experiments are Near Miss 1 algorithm which is denoted as NM1, Near Miss 2 algorithm which is denoted as NM2, Near Miss 3 algorithm which is denoted as NM3, Tomek Links algorithm which is denoted as TomekL, CNN, OSS, NCR and ENN. All oversampling algorithms are placed as columns in Table 5.6.

Table 5.6 Undersampling Techniques' F1-Scores

	NM1	NM2	NM3	TomekL	CNN	OSS	NCR	ENN
LR	0.449	0.280	0.631	0.722	0.732	0.723	0.729	0.726
SVM_lin	0.528	0.292	0.678	0.728	0.728	0.727	0.731	0.732

When F1 scores oversampling methods with 2 different algorithm is examined, it can be said that only Tomek Links, CNN, OSS and NCR algorithms yielded better or similar results compared to the baseline results of Linear Regression and SVM's. This is because well performing algorithms mentioned above has a limited undersampling budget due to the nature of the algorithms itself and the ratio hyperparameter of minority over majority cannot be manipulated. Poor performing algorithms on the other hand, has the property of minority over majority ratio manipulation. Since the default rate is 1 (i.e. minority class examples equal to majority class examples), 20696 samples are decreased to 726 samples which equals to a lot of valuable information loss. The undersampling algorithms which allows to tune the minority over majority ratio are Near Miss 1, Near Miss 2, Near Miss 3. The ratio parameter was set to 0.1 (the default parameter is 1) and the F1 scores after this tuning shared in Table 5.7

Table 5.7 Undersampling Techniques with 0.1 Minority Over Majority Class Ratio F1 Scores

	NM1	NM2	NM3	TomekL	CNN	OSS	NCR	ENN
LR	0.712	0.708	0.656	0.724	0.732	0.726	0.729	0.726
SVM_lin	0.717	0.711	0.698	0.728	0.728	0.727	0.731	0.732

Near Miss 1 and Near Miss 2 undersampling algorithms' F1-scores are drastically improved and Near Miss 3 algorithm's F1 score is slightly improved after ratio parameter tuning.

Finding the Best Ratio Hyperparameter Value for Undersampling Techniques

In this part of the study, best minority class over majority class ratio hyperparameter searched via applying grid search on undersampling techniques that offers the possibility to change the ratio hyperparameter.

Undersampling Methods Hyperparameter Tuning for Logistic Regression

In order to find best hyperparameter ratio for Near Miss algorithms, grid search was applied. The results were shared in Table 5.8 below.

Table 5.8 Ratio Hyperparameter Tuning for Logistic Regression

	NM1	NM2	NM3
0.05	0.7174	0.7175	0.7168
0.06	0.7178	0.7172	0.7168
0.07	**0.7190**	**0.7181**	0.7224
0.08	0.7170	0.7146	**0.7301**
0.09	0.7162	0.7103	0.7070
0.10	0.7121	0.7075	0.6738
0.11	0.7044	0.6947	0.6410
0.12	0.7002	0.6817	0.6173
0.13	0.6948	0.6728	0.6016

Table 5.8 (continued)

0.14	0.6960	0.6579	0.5852
0.15	0.6942	0.6431	0.5692
0.16	0.6868	0.6296	0.5516
0.17	0.6768	0.6149	0.5398
0.18	0.6786	0.6047	0.5305
0.19	0.6765	0.5890	0.5197
0.20	0.6693	0.5733	0.5106

Near Miss algorithms' minority / majority ratio hyperparameter performances checked via grid search between the range of [0.05, 0.20]. Optimum Hyperparameter value for Near Miss 1 and Near Miss 2 is detected as 0.07 which yielded the F1 scores of 0.7190 and 0.7181 respectively. On the other hand, best hyperparameter for Near Miss 3 undersampling algorithm found to be 0.08 which provided the F1 score of 0.7301.

Undersampling Methods Hyperparameter Tuning for SVM's

Ratio hyperparameter tuning is performed with grid search using SVM's algorithm (linear kernel). The experiment results were given in Table 5.9.

Table 5.9 Ratio Hyperparameter Tuning for SVM's

	NM1	NM2	NM3
0.05	**0.7208**	0.7195	0.7221
0.06	0.7201	0.7188	0.7221
0.07	0.7202	0.7206	**0.7247**
0.08	0.7185	**0.7212**	0.7198
0.09	0.7184	0.7179	0.6895

Table 5.9 (continued)

0.10	0.7130	0.7141	0.6591
0.11	0.7050	0.7014	0.6241
0.12	0.7056	0.6813	0.6004
0.13	0.7027	0.6707	0.5890
0.14	0.7092	0.6562	0.5760
0.15	0.7070	0.6450	0.5646
0.16	0.7016	0.6251	0.5514
0.17	0.6969	0.5980	0.5419
0.18	0.6999	0.5761	0.5343
0.19	0.6991	0.5596	0.5224
0.20	0.7037	0.5465	0.5154

Near Miss 1 algorithm performed best when ratio hyperparameter was set to 0.05 which provided 0.7208 F1 score. Near Miss 2 algorithm performed best when the ratio hyperparameter was set to 0.08 which gave the F1 value of 0.7212. Lastly, Near Miss 3 Algorithm performed best when the hyperparameter of minority over majority ratio was set to 0.07.

5.2.3 Performance of Combinations of Oversampling and Undersampling Techniques

In this part of the study, combinations of oversampling and undersampling technique performances were tested.

Hyperparameter optimization for SMOTE + Tomek, SMOTE + ENN, SMOTE + OSS, SMOTE + NCR Algorithms

SMOTE + Tomek Links (SMOTETomek), SMOTE + ENN (SMOTEENN), SMOTE + OSS (SMOTEOSS) and SMOTE + NCR (SMOTENCR) algorithms will

be evaluated in this section. Since Random Oversampling and Near Miss algorithms has their own ratio parameters, these were separated from other algorithms.

Table 5.10 Grid Search for Hyperparameter Optimization of Combined Sampling Techniques (SMOTE +) with Logistic Regression

	SMOTETomek	SMOTEENN	SMOTEOSS	SMOTENCR
0.05	0.7334 (0.027)	0.7202 (0.029)	0.7328 (0.023)	**0.7256 (0.031)**
0.06	0.7380 (0.029)	**0.7232 (0.022)**	**0.7420 (0.027)**	0.7202 (0.031)
0.07	**0.7409 (0.029)**	0.7190 (0.028)	0.7382 (0.026)	0.7132 (0.033)
0.08	0.7393 (0.025)	0.7143 (0.028)	0.7411 (0.033)	0.7098 (0.031)
0.09	0.7407 (0.029)	0.6989 (0.030)	0.7379 (0.026)	0.7067 (0.033)
0.10	0.7375 (0.031)	0.6914 (0.030)	0.7381 (0.030)	0.7016 (0.030)
0.11	0.7366 (0.028)	0.6823 (0.029)	0.7366 (0.030)	0.6935 (0.031)
0.12	0.7327 (0.032)	0.6780 (0.031)	0.7333 (0.031)	0.6925 (0.029)
0.13	0.7292 (0.030)	0.6737 (0.030)	0.7325 (0.030)	0.6862 (0.030)
0.14	0.7268 (0.032)	0.6683 (0.029)	0.7235 (0.031)	0.6840 (0.031)
0.15	0.7219 (0.031)	0.6619 (0.031)	0.7220 (0.034)	0.6835 (0.031)
0.16	0.7214 (0.033)	0.6550 (0.032)	0.7169 (0.033)	0.6778 (0.031)
0.17	0.7173 (0.033)	0.6488 (0.031)	0.7156 (0.032)	0.6729 (0.030)
0.18	0.7112 (0.033)	0.6470 (0.030)	0.7125 (0.033)	0.6708 (0.033)
0.19	0.7118 (0.035)	0.6406 (0.031)	0.7074 (0.032)	0.6654 (0.032)
0.20	0.7084 (0.032)	0.6362 (0.034)	0.7052 (0.030)	0.6626 (0.031)

SMOTETomek algorithm performed best with an F1 score of 0.7409 when the minority over majority ratio hyperparameter was set to 0.07. SMOTE, BLSMOTE and ADASYN algorithms surpassed this score slightly with ratio parameters of 0.08, 0.07 and 0.07 respectively.

SMOTEENN algorithm provided the highest F1 score (0.7232) with Logistic Regression when ratio hyperparameter was set to 0.06.

SMOTEOSS algorithm gave the best results when the ratio parameter was set to 0.06. The corresponding F1 value is 0.7420.

SMOTENCR performed best when the hyperparameter of minority class ratio set to 0.05. The F1 value Logistic Regression achieved after hyperparameter tuning is 0.7256.

Hyperparameter Optimization for SMOTE + Near Miss-1 Algorithm

Near Miss methods allow setting minority over majority class ratio. All Near Miss methods are combined with SMOTE algorithm and two-dimensional grid search tables were used for each SMOTE + Near Miss method.

SMOTE + Near Miss-1 (SMOTENM1) algorithm used as a preprocessing step of Logistic regression and grid search is applied to find best class ratio hyperparameter for both SMOTE and Near Miss-1. The results which were obtained given in Table 5.11 below.

Table 5.11 SMOTENM1 Grid Search for Optimum Class Ratio Results Using Logistic Regression

		Near Miss-1 Hyperparameter						
		0.05	0.06	0.07	0.08	0.09	0.10	0.11
SMOTE Hyperparameter	0.04	**0.7231**	07215	0.7231	0.7221	0.7216	0.7193	0.7161
	0.05		0.7344	**0.7373**	0.7325	0.7372	0.7332	0.7306
	0.06			0.7367	0.7410	**0.7443**	0.7408	0.7407
	0.07				**0.7426**	0.7425	0.7421	0.7417
	0.08					0.7421	0.7411	**0.7433**
	0.09						**0.7424**	0.7389
	0.1							0.7381

The best result was attained by setting SMOTE minority / majority ratio to 0.06 and setting Near Miss-1 ratio hyperparameter to 0.09. Combined algorithm increases the minority class / majority class ratio to 0.06 via SMOTE and increase minority /

majority ratio further to 0.09 by undersampling majority class using Near Miss-1 method.

Hyperparameter Optimization for SMOTE + Near Miss 2 Algorithm

SMOTE which is an oversampling method and Near Miss-2 which is an undersampling method combined for this part of the study. The algorithm named as SMOTENM2.

Near Miss-2 algorithm has minority over majority ratio hyperparameter as Near Miss-1 algorithm. Cross grid search was applied between SMOTE ratio parameter and Near Miss-2 algorithm and the results were given in Table 5.12 below.

Table 5.12 SMOTENM2 Grid Search for Optimum Class Ratio Results Using Logistic Regression

		Near Miss-2 Hyperparameter						
		0.05	0.06	0.07	0.08	0.09	0.10	0.11
SMOTE Hyperparameter	0.04	0.7233	**0.7245**	0.7235	0.7242	0.7185	0.7134	0.7126
	0.05		**0.7346**	0.7326	0.7324	0.7335	0.7302	0.7279
	0.06			**0.7412**	0.7401	0.7385	0.7387	0.7401
	0.07				0.7438	0.7437	**0.7443**	0.7414
	0.08					0.7433	0.7453	**0.7454**
	0.09						**0.7416**	0.7386
	0.1							0.7383

The best F1 score Logistic Regression attained with SMOTENM2 combined sampling algorithm is 0.7454 which is achieved by setting the ratio parameter for SMOTE to 0.08 and setting the ratio hyperparameter of Near Miss-2 algorithm to 0.11.

Hyperparameter Optimization for SMOTE + Near Miss 3 Algorithm

SMOTE which is an oversampling method and Near Miss-3 which is an undersampling method combined for this part of the study. The algorithm named as SMOTENM3. Both SMOTE and Near Miss-3 algorithms have minority class over majority class ratio hyperparameter which can be tuned for optimum performance. Grid search was applied on these 2 hyperparameters in order to find the best combination of hyperparameters. Logistic Regression learning algorithm used since it gave the best prediction performance alongside with SVM's throughout the study. Between Logistic Regression and SVM's, there are no big difference in F1 scores and the decision made that using only one of them is sufficient. Obtained F1 scores for different hyperparameters given in Table 5.13 below.

Table 5.13 SMOTENM3 Grid Search for Optimum Class Ratio Results Using Logistic Regression

		Near Miss-3 Hyperparameter						
		0.05	0.06	0.07	0.08	0.09	0.10	0.11
SMOTE Hyperparameter	0.04	**0.7251**	0.7242	0.7217	0.7278	0.7305	0.7051	0.6792
	0.05		0.7324	**0.7338**	0.7330	0.7305	0.7398	0.7265
	0.06			0.7384	0.7400	0.7402	**0.7407**	0.7148
	0.07				**0.7384**	0.6754	0.6060	0.5602
	0.08					**0.7048**	0.6303	0.5703
	0.09						**0.7067**	0.6403
	0.1							**0.7071**

Best performance for each SMOTE hyperparameter was written with bold numbers. In general, it can be said that version SMOTENM3 provided lover performance compared to SMOTENM1 and SMOTENM2. Highest F1 score Logistic Regression achieved with SMOTENM3 is 0.7407 which is achieved by setting SMOTE ratio hyperparameter to 0.06 and setting Near Miss-3 algorithm hyperparameter to 0.10.

5.3 Incorporating Cost-Sensitive Methods

Cost-sensitive methods change the weight given to the class labels so learning process is manipulated to give more importance to the classes. Finding best weights by intuition can be challenging. In order to find best weight hyperparameters, grid search was applied to the best performed algorithms on previous parts.

5.3.1 Combining Cost-Sensitive Methods with Oversampling Algorithms

All oversampling algorithms that were used in previous chapters were combined with cost-sensitive methods. Oversampling methods' hyperparameters and learning algorithm's class weight hyperparameter tuned via grid search cross validation. Table 5.13 contains the best 3 F1 score results for each Oversampling Technique. Oversampling technique and cost-sensitive Logistic Regression combined via Pipeline function in Python library called imbalanced-learn.

Table 5.14 Best 3 Hyperparameter Combinations for Each Oversampling Technique

	Rank	F1 Score	Class Weights	Minor / Major
SMOTE	1	**0.7474**	Class0: 3, Class1: 2	0.11
	2	0.7464	Class0: 4, Class1: 1	0.28
	3	0.7441	Class0: 7, Class1: 3	0.15
BLSMOTE	1	0,7426	Class0: 2, Class1:1	0.1
	2	0.7417	Class0:2, Class1:1	0.11
	3	0.7412	Class0:3, Class1:2	0.09
SVMSMOTE	1	0.7437	Class0:2, Class1:3	0.05
	2	0.7414	Class0:4, Class1:1	0.22
	3	0.7411	Class0:2, Class1:1	0.18
ADASYN	1	0.7416	Class0:7, Class1:3	0.16
	2	0.7409	Class0:2, Class1:1	0.15

Table 5.14 (continued)

	3	0.7406	Class0:2, Class1:1	0.12

Above table consists of the results which belongs to 4 oversampling techniques and they were placed as rows in the table. For each oversampling technique, 3 highest F1 score were written. The Rank column shows the order of the best 3 predictions. F1 score column shows the mean F1 score which is derived from 10-fold cross validation. Class Weight column shows the Cost-Sensitive Logistic Regression's weight hyperparameter. Minor / Major shows the ratio hyperparameter of SMOTE algorithm.

The F1 score of SMOTE and Logistic Regression achieved at class weight hyperparameter {0:3, 1:2} and ratio hyperparameter 0.11 is 0.7474 which were the highest F1 score until this point of the experiments.

SVMSMOTE algorithm's best F1 score (0.7437) is attained when the class weight hyperparameter was set to {0:2, 1:3} and minority / majority hyperparameter was set to 0.05. 2nd best F1 score which is 0.7414 was attained by setting the class weight hyperparameter to {0:4, 1:1} and by setting the ratio hyperparameter to 0.22. 3rd best F1 score is which is 0.7411 was attained by setting the class weight hyperparameter to {0:2, 1:1} and by setting the ratio hyperparameter to 0.18.

BLSSMOTE algorithm's best F1 score which is 0.7426 attained when the class weight hyperparameter was set to {0:2, 1:1} and minority / majority hyperparameter was set to 0.1. 2nd best F1 score which is 0.7417 was attained by setting the class weight hyperparameter to {0:2, 1:1} and by setting the ratio hyperparameter to 0.11. 3rd best F1 score is 0.7412 was attained by setting the class weight hyperparameter to {0:3, 1:2} and by setting the ratio hyperparameter to 0.09.

ADASYN algorithm's best F1 score which is 0.7416 attained when the class weight hyperparameter was set to {0:7, 1:3} and minority / majority hyperparameter was set to 0.16. 2nd best F1 score which is 0.7409 was attained by setting the class weight

hyperparameter to {0:2, 1:1} and by setting the ratio hyperparameter to 0.15. 3rd best F1 score is 0.7406 was attained by setting the class weight hyperparameter to {0:2, 1:1} and by setting the ratio hyperparameter to 0.12.

5.3.2 Combining Cost-Sensitive Methods with Combined Sampling Algorithms

Combined sampling algorithms combine oversampling and undersampling methods. SMOTETomek, SMOTEENN, SMOTENCR, SMOTENM1 and SMOTENM2 combined sampling algorithms were used in compliance with cost-sensitive Logistic Regression. Grid search Cross Validation technique used to evaluate best hyperparameter for both sampling technique and weight hyperparameter.

3 best F1 scores were given in Table 5.15 for each combined sampling technique below.

Table 5.15 Top 3 Hyperparameter Combinations for combined sampling techniques and Cost-Sensitive Logistic Regression

	Rank	F1 Score	Class Weights	Minor / Major
SMOTETomek	1	0.7460	Class0: 2, Class1: 1	0.13
	2	0.7455	Class0: 2, Class1: 1	0.15
	3	0.7430	Class0: 2, Class1: 3	0.07
SMOTEENN	1	0.7390	Class0: 10, Class1: 1	0.13
	2	0.7368	Class0: 10, Class1: 1	0.16
	3	0.7358	Class0: 10, Class1: 1	0.15
SMOTEOSS	1	0.7448	Class0: 7, Class1: 3	0.19
	2	0.7439	Class0: 3, Class1: 2	0.1
	3	0.7439	Class0: 2, Class1: 1	0.16
SMOTENCR	1	0.7419	Class0: 7, Class1: 3	0.06
	2	0.7419	Class0: 10, Class1: 1	0.18
	3	0.7416	Class0: 10, Class1: 1	0.19

Table 5.15 (continued)

SMOTENM1	1	0.7466	Class0: 1, Class1: 1	SMOTE: 0.07 NM1: 0.08
	2	0.7452	Class0: 1, Class1: 1	SMOTE: 0.07 NM1: 0.1
	3	0.7452	Class0: 1, Class1: 1	SMOTE: 0.07 NM1: 0.11
SMOTENM2	1	**0.7473**	Class0: 1, Class1: 1	SMOTE: 0.07 NM2: 0.09
	2	0.7462	Class0: 1, Class1: 1	SMOTE: 0.07 NM2: 0.11
	3	0.7458	Class0: 1, Class1: 1	SMOTE: 0.06 NM2: 0.11

SMOTETomek algorithm's best F1 score found as 0.7460 which is the second highest F1 score until this stage of the study. This F1 score was attained when the class weight hyperparameter was set to {0:2, 1:1} and minority / majority hyperparameter was set to 0.13. 2nd best F1 score which is 0.7455 was attained by setting the class weight hyperparameter to {0:2, 1:1} and by setting the ratio hyperparameter to 0.15. 3rd best F1 score is 0.7430 was attained by setting the class weight hyperparameter to {0:2, 1:3} and by setting the ratio hyperparameter to 0.07.

SMOTEENN algorithm's best F1 score is 0.7390 which was attained when the class weight hyperparameter was set to {0:10, 1:1} and minority / majority hyperparameter was set to 0.13. 2nd best F1 score which is 0.7368 was attained by setting the class weight hyperparameter to {0:10, 1:1} and by setting the ratio hyperparameter to 0.16. 3rd best F1 score is 0.7358 which was attained by setting the class weight hyperparameter to {0:10, 1:1} and by setting the ratio hyperparameter to 0.15.

SMOTEOSS algorithm's best F1 score is 0.7448 which was attained when the class weight hyperparameter was set to {0:7, 1:3} and minority / majority hyperparameter was set to 0.19. 2nd best F1 score which is 0.7439 was attained by setting the class weight hyperparameter to {0:3, 1:2} and by setting the ratio hyperparameter to 0.1. 3rd best F1 score found as 0.7439 which was attained by setting the class weight hyperparameter to {0:2, 1:1} and by setting the ratio hyperparameter to 0.16.

SMOTENCR algorithm's best F1 score is 0.7419 which was attained when the class weight hyperparameter was set to {0:7, 1:3} and minority / majority hyperparameter was set to 0.06. 2nd best F1 score which is 0.7419 was attained by setting the class weight hyperparameter to {0:10, 1:1} and by setting the ratio hyperparameter to 0.18. 3rd best F1 score found as 0.7416 which was attained by setting the class weight hyperparameter to {0:10, 1:1} and by setting the ratio hyperparameter to 0.19.

SMOTENM1 algorithm allows to change both SMOTE and Near Miss-1 (NM1) algorithms' ratio parameter so the grid search applied for 3 different hyperparameters this time. First hyperparameter is SMOTE ratio hyperparameter, second one is Near Miss-1 ratio hyperparameter and the last one is the class weight hyperparameter of cost-sensitive Logistic Regression. Best F1 score attained after the grid search is 0,7466 which was attained when class weight hyperparameter was set to {0:1, 1:1} and SMOTE ratio hyperparameter was set to 0.07 and NM1 ratio hyperparameter was set to 0.08. 2nd best F1 score (0.7452) was reached when the weight class hyperparameter was set to {0:1, 1:1}, SMOTE ratio hyperparameter was set to 0.07 and NM1 ratio parameter was set to 0.1. 3rd best F1 score (0.7452) was achieved by setting the weight class hyperparameter to {0:1, 1:1}, SMOTE ratio hyperparameter to 0.07 and NM1 ratio hyperparameter to 0.11.

SMOTENM2 algorithm used in coherence with cost-sensitive Logistic Regression. As per SMOTENM1, 3 hyperparameter grid search was applied. Best F1 score attained after the grid search is 0,7473 which was attained when class weight hyperparameter was set to {0:1, 1:1} and SMOTE ratio hyperparameter was set to 0.07 and NM1 ratio hyperparameter was set to 0.09. 2nd best F1 score (0.7462) was

reached when the weight class hyperparameter was set to {0:1, 1:1}, SMOTE ratio hyperparameter was set to 0.07 and NM1 ratio parameter was set to 0.11. 3rd best F1 score (0.7458) was achieved by setting the weight class hyperparameter to {0:1, 1:1}, SMOTE ratio hyperparameter to 0.06 and NM1 ratio hyperparameter to 0.07.

5.4 Probability Threshold Moving for Final Results

Probability threshold moving was applied for 5 best performing combination of methods in order to decide the final predictive model. Stratified train test split was used

5.4.1 Model 1 - SMOTENM2 + Cost Sensitive Logistic Regression I

SMOTENM2 algorithm hyperparameters consist of SMOTE ratio hyperparameter which was set to 0.07 and NM2 ratio hyperparameter which was set to 0.09. Cost-sensitive Logistic Regression class weight hyperparameter was set to {0:1, 1:1} This combination without threshold moving provided the F1 score of 0.7473. After the optimum probability threshold grid search, optimum threshold is found as 0.531995 which provided the F1 score of 0.752. Since F1 score is the harmonic mean of Precision and Recall metrics, Precision and Recall Curve were given to show the balance between two metrics and the threshold point where the highest F1 score achieved also shown in the graph with a black dot. Precision – Recall Curve is given below.

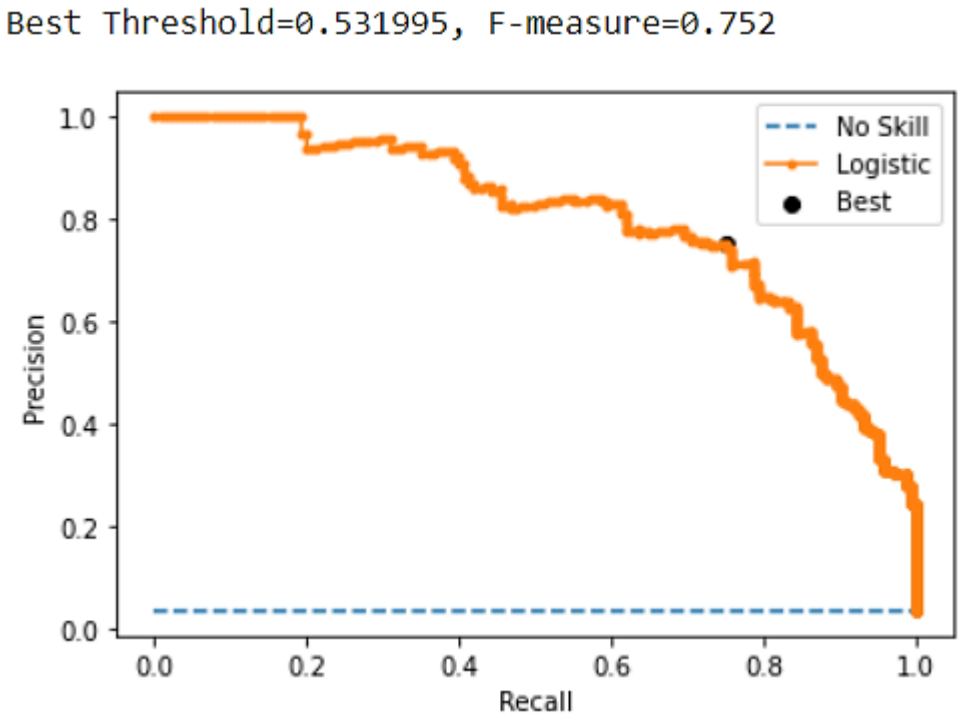


Figure 5.1 Precision - Recall Curve for Model 1

Above results attained from a model which is trained by 0.80 of dataset and tested by 0.20 of the dataset. In order to be more precise, below results were attained by 10-fold cross validation with 10 repeats which accumulates to 100 models. The probability threshold was found as 0.52 which is shown with dotted line in Figure 5.2. Also, F1, recall and precision scores is shown with red, green and blue lines respectively.

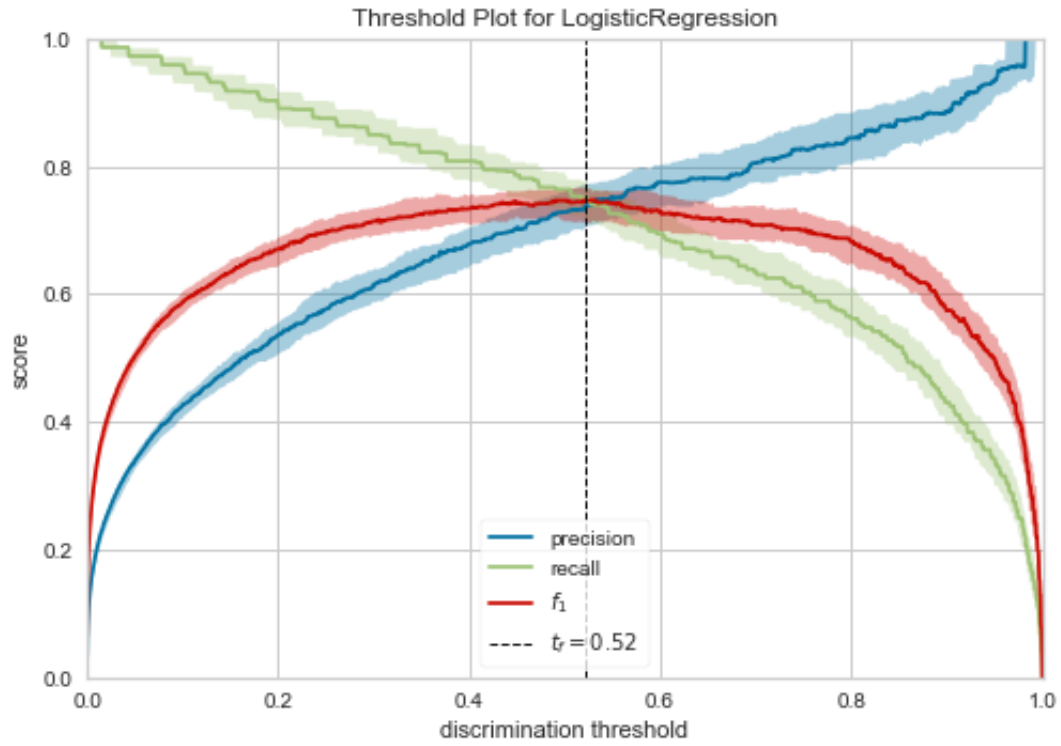


Figure 5.2 Probability Threshold Plot of Model 1 (10-fold Cross Validation)

The threshold found with 10-fold Cross Validation is close to the threshold found with train-test split. The inverse relationship between precision and recall metrics can be clearly seen in Figure 5.2.

5.4.2 Model 2 - SMOTENM2 + Cost-Sensitive Logistic Regression II

Same Combination with model one but different hyperparameters used. SMOTE ratio hyperparameter was set to 0.07 and NM2 algorithm hyperparameter was set to 0.11 lastly, class weight hyperparameter was set to {0:1, 1:1}. With these hyperparameters and threshold moving, the combined algorithm gave the F1 score of 0.7500 which is higher than the performance of the same algorithm without threshold moving. Best probability threshold was found as 0.508798. The Precision - Recall Curve of the algorithm was given below.

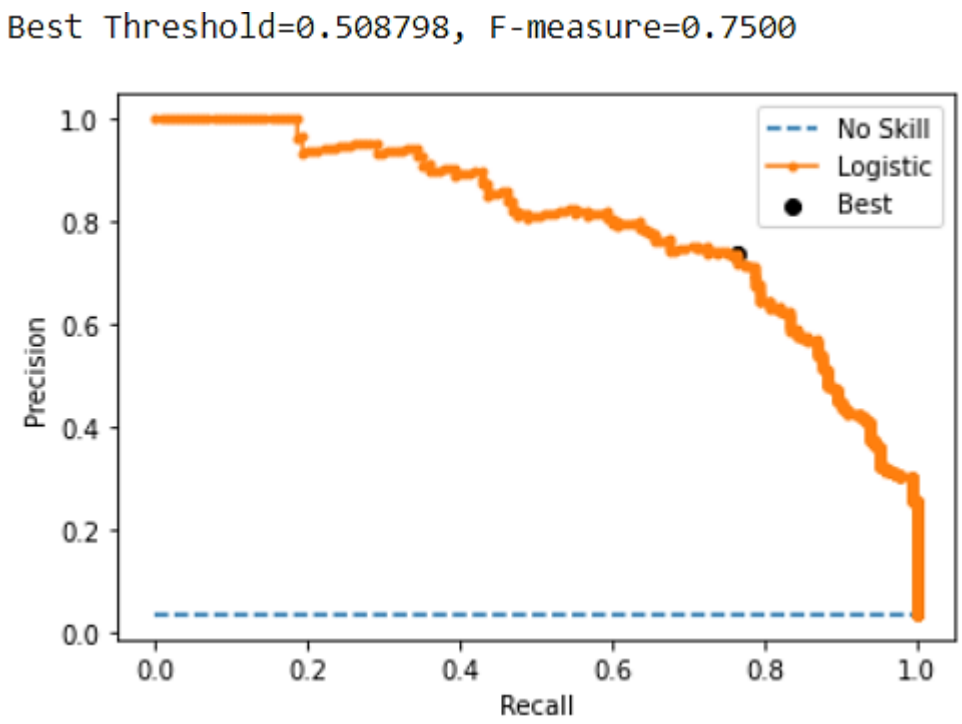


Figure 5.3 Precision – Recall Curve for Model 2

Above results attained from a model which is trained by 0.80 of dataset and tested by 0.20 of the dataset. In order to be more precise, below results were attained by 10-fold cross validation with 10 repeats which accumulates to 100 models. The probability threshold was found as 0.51 which is shown with dotted line in Figure 5.4. Also, F1, recall and precision scores is shown with red, green and blue lines respectively.

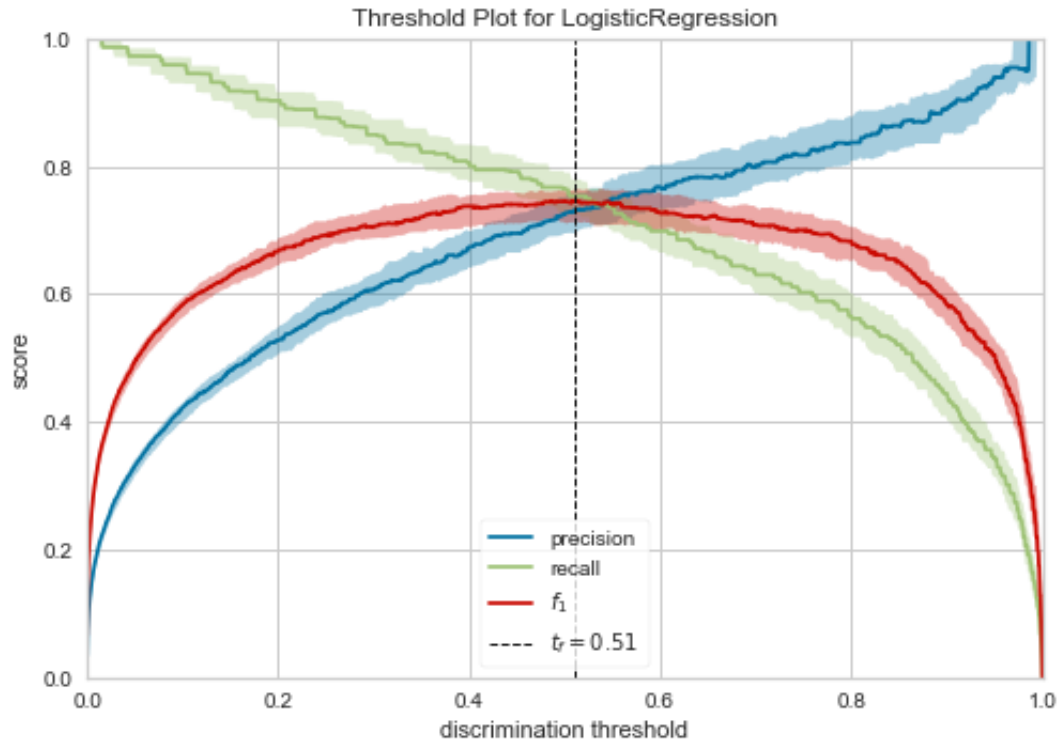


Figure 5.4 Probability Threshod Plot of Model 1 (10-fold Cross Validation)

The threshold found with 10-fold Cross Validation is close to the threshold found with train-test split. The fact that precision and recall metrics works as a trad-off can be clearly seen in Figure 5.4.

5.4.3 Model 3 – SMOTENM1 + Cost-Sensitive Logistic Regression

For this combined algorithm, SMOTE ratio hyperparameter was set to 0.07 and NM2 algorithm hyperparameter was set to 0.08 lastly, class weight hyperparameter was set to {0:1, 1:1}. With these hyperparameters and threshold moving, the combined algorithm gave the F1 score of 0.7575 which is higher than the performance of the same algorithm without threshold moving. Best probability threshold was found as 0.491407. The Precision - Recall Curve of the algorithm was given below.

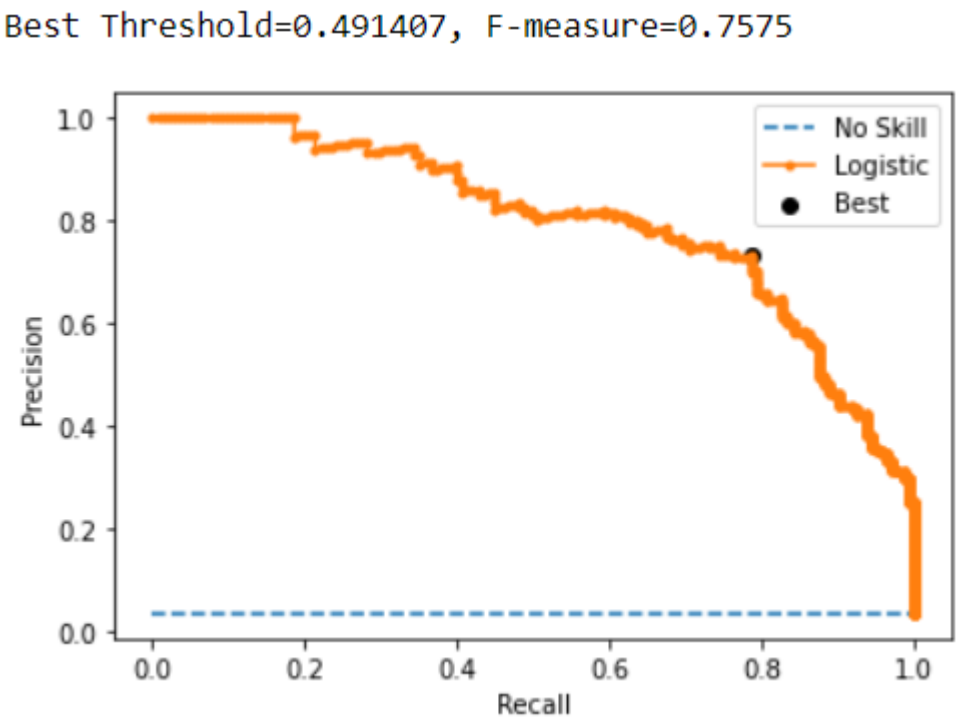


Figure 5.5 Precision – Recall Curve for Model 3

Above results attained from a model which is trained by 0.80 of dataset and tested by 0.20 of the dataset. In order to be more precise, below results were attained by 10-fold cross validation with 10 repeats which accumulates to 100 models. The probability threshold was found as 0.51 which is shown with dotted line in Figure 5.6. Also, F1, recall and precision scores is shown with red, green and blue lines respectively.

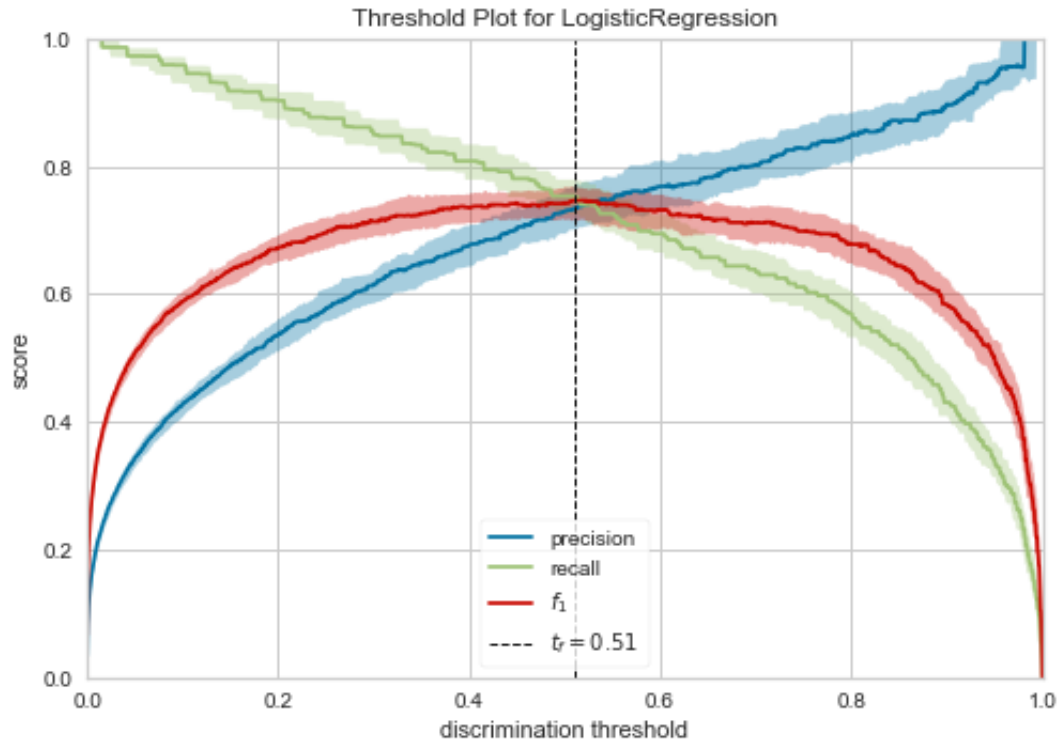


Figure 5.6 Probability Threshod Plot of Model 1 (10-fold Cross Validation)

Also for model 3, the optimum threshold value is around 0.50.

5.4.4 Model 4 - SMOTETomek + Cost-Sensitive Logistic Regression

For this combined algorithm, SMOTETomek ratio hyperparameter was set to 0.13 and class weight hyperparameter was set to {0:2, 1:1}. With these hyperparameters and threshold moving, the combined algorithm gave the F1 score of 0.7551 which is higher than the performance of the same algorithm without threshold moving. Best probability threshold was found as 0.478086. The Precision - Recall Curve of the algorithm was given below.

```
Best Threshold=0.478086, F-measure=0.7551
```

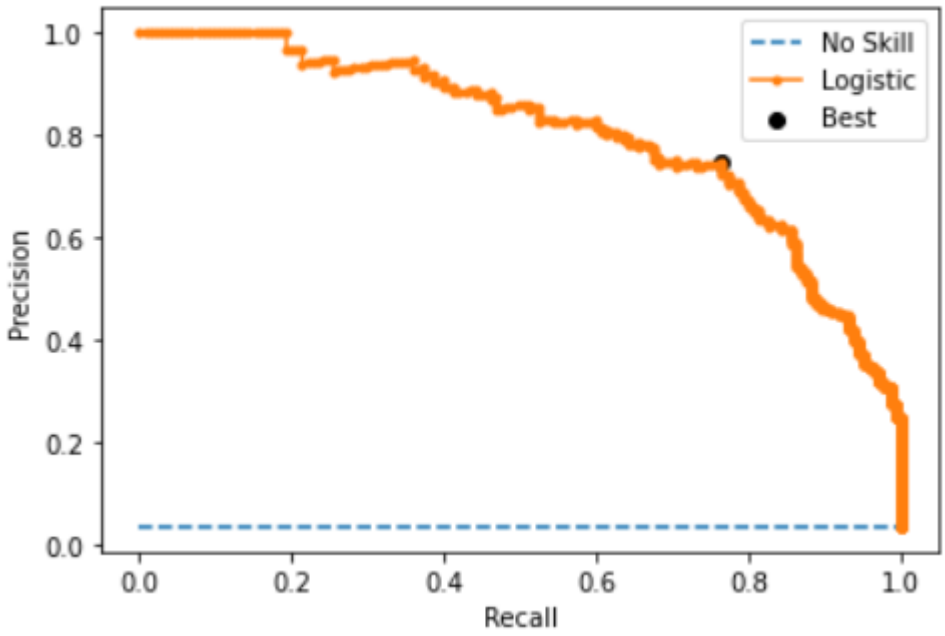


Figure 5.7 Precision – Recall Curve for Model 4

Above results attained from a model which is trained by 0.80 of dataset and tested by 0.20 of the dataset. In order to be more precise, below results were attained by 10-fold cross validation with 10 repeats which accumulates to 100 models. The probability threshold was found as 0.47 which is shown with dotted line in Figure 5.8. Also, F1, recall and precision scores is shown with red, green and blue lines respectively.

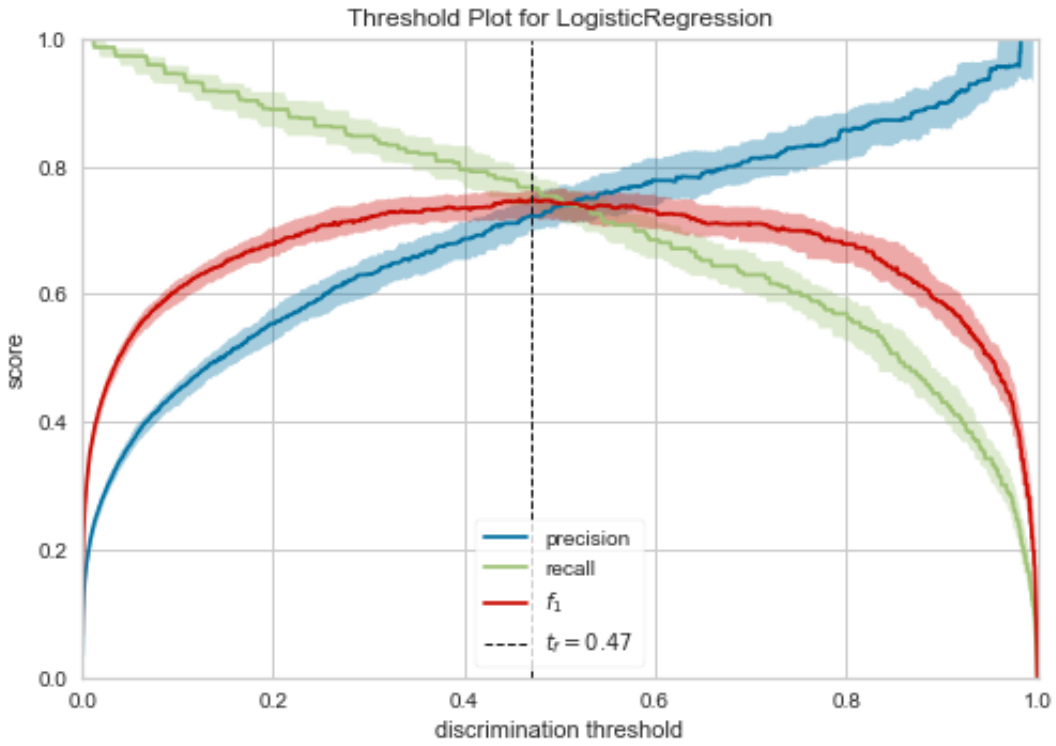


Figure 5.8 Probability Threshod Plot of Model 4 (10-fold Cross Validation)

The threshold attained by train-test split is really close with the threshold attained by 10-fold cross validation with 10 repeats. The bands around the lines show interquartile range of the scores.

5.4.5 Model 5 – SMOTE + Cost-Sensitive Logistic Regression

For this combined algorithm, SMOTETomek ratio hyperparameter was set to 0.11 and class weight hyperparameter was set to {0:3, 1:2}. With these hyperparameters and threshold moving, the combined algorithm gave the F1 score of 0.7568 which is

higher than the performance of the same algorithm without threshold moving. Best probability threshold was found as 0.521922. The Precision - Recall Curve of the algorithm was given below.

```
Best Threshold=0.521922, F-measure=0.7568
```

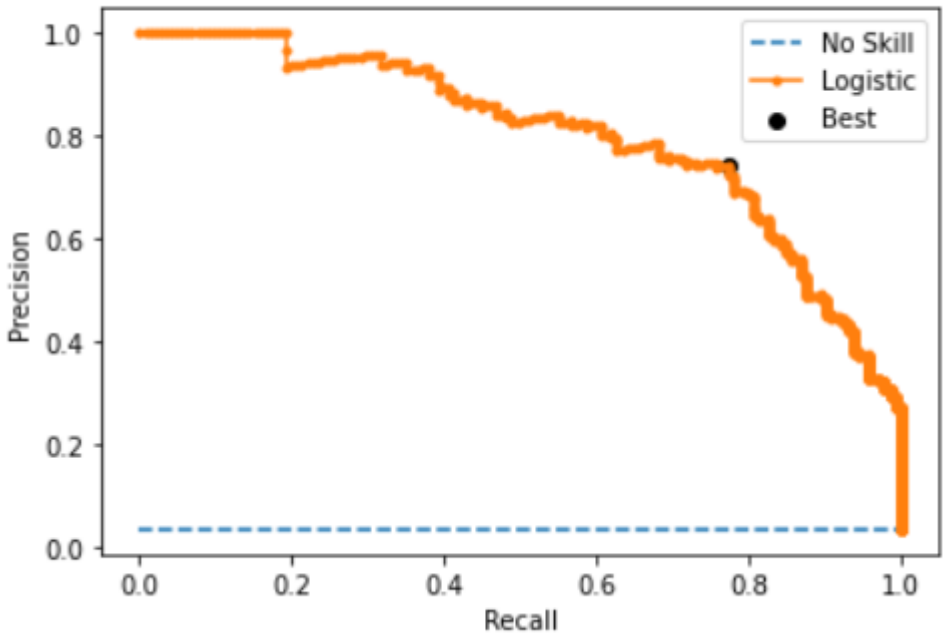


Figure 5.9 Precision – Recall Curve for Model 5

Above results attained from a model which is trained by 0.80 of dataset and tested by 0.20 of the dataset. In order to be more precise, below results were attained by 10-fold cross validation with 10 repeats which accumulates to 100 models. The probability threshold was found as 0.50 which is shown with dotted line in Figure 5.10. Also, F1, recall and precision scores is shown with red, green and blue lines respectively.

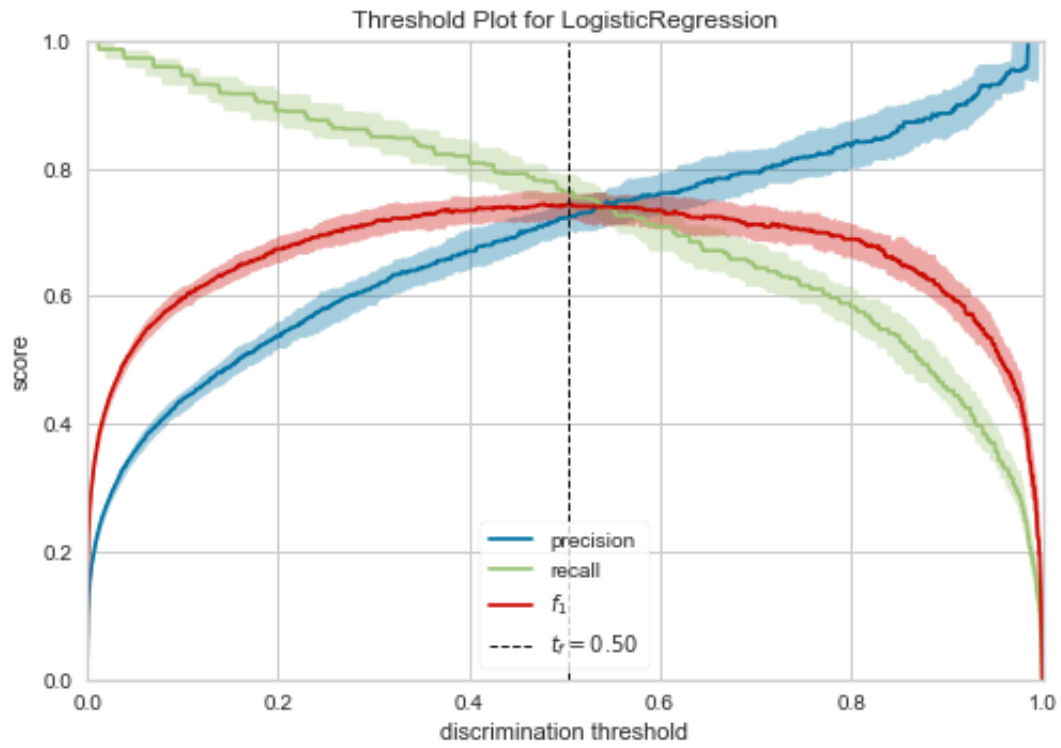


Figure 5.10 Probability Threshod Plot of Model 1 (10-fold Cross Validation)

The threshold found is close to the threshold found by train-test split. For 5 best performing models, the probability thresholds found to be close to 0.50 with slight differences.

5.5 Monte Carlo Simulation

Monte Carlo (MC) simulation was applied in order to test the previously applied methods on different simulated datasets. For this part of the study, following 3 different scenarios were used.

- 1:99 minority over majority ratio with 20 independent variables
- 40:60 minority over majority ratio 20 independent variables
- 20:80 minority over majority ratio 20 independent variables
- 1:99 minority over majority ratio 4 independent variables

First 1:99 scenario is used to represent an extreme imbalance scenario and this scenario is the closest one to the real-world data used in the study which has 3.39:96.61 imbalance ratio. Second scenario which has 40:60 imbalance ratio represents a mild imbalance scenario. Third scenario created to have an imbalance ratio that is between scenario 1 and 2. Last scenario which has the same imbalance scenario with the first one (i.e. 1:99) but only 4 independent variables instead of 20.

Aforementioned scenarios applied on the five best predictive models mentioned on previous part of the study. Since 2 out of 5 models are same combination with different hyperparameters (SMOTENM2+ Cost-Sensitive Logistic Regression) the number of unique models that performed best is actually 4. These are:

- SMOTENM2 + Cost-Sensitive Logistic Regression
- SMOTENM1 + Cost-Sensitive Logistic Regression
- SMOTE + Cost-Sensitive Logistic Regression
- SMOTETomek + Cost-Sensitive Logistic Regression

The best minority:majority ratio hyperparameters and best class weight hyperparameters for aforementioned predictive models found by grid search and then compared with best hyperparameters found for other scenarios and the real world data case. Also, each scenario's best scores were compared with the baseline score attained by Logistic Regression only to determine the effectiveness of the imbalanced learning techniques over the scenario created.

Monte Carlo Simulation applied with repeated stratified k fold cross validation which can be found in the library called scikit-learn. 10 fold and 50 repeats which counts for 500 different train test splits for each grid search.

5.5.1 Scenario 1 – 1:99 Imbalance Ratio with 20 Independent Variables

The baseline score obtained with Logistic Regression for this scenario is 0.8770. Table 5.16 shows the best F1-scores acquired for scenario 1.

Table 5.16 Best Scores Obtained by Imbalanced Learning Techniques, Scenario 1

	Rank	F1 Score	Class Weights	Minor / Major
SMOTE	1	0.8929	Class0: 4, Class1: 1	0.08
	2	0.8924	Class0: 2, Class1: 1	0.05
	3	0.8924	Class0: 1, Class1: 1	0.03
SMOTETomek	1	0.8985	Class0: 3, Class1: 2	0.04
	2	0.8978	Class0: 3, Class1: 2	0.03
	3	0.8975	Class0: 4, Class1: 1	0.08
SMOTENM1	1	**0.9162**	Class0: 7, Class1: 3	SMOTE: 0.02 NM1: 0.08
	2	0.9139	Class0: 4, Class1: 1	SMOTE: 0.06 NM1: 0.13
	3	0.9138	Class0: 4, Class1: 1	SMOTE: 0.05 NM1: 0.10
SMOTENM2	1	0.9014	Class0: 1, Class1: 1	SMOTE: 0.02 NM2: 0.03
	2	0.9013	Class0: 1, Class1: 1	SMOTE: 0.03 NM2: 0.04
	3	0.9013	Class0: 4, Class1: 1	SMOTE: 0.07 NM2: 0.08

All predictive models with the hyperparameters which are shown in tables outperformed the baseline score which is 0.8770. Best performance achieved by SMOTENM1 + Cost-Sensitive Logistic Regression model which is 0.9162. this score is approximately 4% better compared to the baseline score. With NCAA dataset, approximately 2.5% increase in F1-score was achieved. This shows that with extreme imbalance scenarios as per scenario 1 dataset and NCAA dataset, achieving big jumps in F1 score is not possible without introducing more examples for minority class. This is also caused by the nature of the F1-score since it is the harmonic mean

of Precision and Recall metrics which are working in an inversely correlated way as mentioned in this study.

The increase in performance when imbalanced learning techniques introduced for extremely imbalanced datasets is solidified by using Monte Carlo Simulation as an addition to the real-world data experiment.

The default minority : majority hyperparameter is 1 which makes the density of the minority class equal to the density of the majority class. By the results of the experiments done over the NCAA dataset, it was claimed that default hyperparameters do not work well for extreme imbalance scenarios. The Monte Carlo simulation for scenario one also supported that claim and optimum hyperparameters for minority : majority ratio hyperparameter found to be slightly above to the original minority : majority ratio of the dataset.

Similar class weight hyperparameters were found compared to the NCAA dataset experiments. The general advice is to give more class weight to the minority class in Machine Learning Literature but both real-world data experiments and Monte Carlo simulation showed that optimum weight hyperparameters give more weight to the majority class.

5.5.2 Scenario 2 – 40:60 Imbalance Ratio with 20 Independent Variables

This scenario is created to simulate a mild imbalance scenario. The baseline F1-score obtained by Logistic Regression is 0.9444 which is higher than the baseline score of scenario 1. This can be explained by the difference in the imbalance ratios since in scenario 1 only 200 out of 20000 examples belongs to the minority class. On the other hand, there are 8000 out of 20000 examples for minority class in scenario 2. This is a sign that in scenario 2, the minority class is represented better. Which is the main reason of higher baseline F1-score. The highest scores obtained in scenario 2 is given below.

Table 5.17 Best Scores Obtained by Imbalanced Learning Techniques, Scenario 2

	Rank	F1 Score	Class Weights	Minor / Major
SMOTE	1	0.9460	Class0: 3, Class1: 2	0.71
	2	0.9457	Class0: 2, Class1: 1	0.86
	3	0.9456	Class0: 3, Class1: 2	0.8
SMOTETomek	1	**0.9465**	Class0: 2, Class1: 1	0.93
	2	0.9460	Class0: 2, Class1: 1	0.88
	3	0.9460	Class0: 2, Class1: 1	0.83
SMOTENM1	1	0.9460	Class0: 3, Class1: 2	SMOTE: 0.69 NM1: 0.72
	2	0.9458	Class0: 3, Class1: 2	SMOTE: 0.67 NM1: 0.72
	3	0.9458	Class0: 3, Class1: 2	SMOTE: 0.69 NM1: 0.75
SMOTENM2	1	0.9458	Class0: 3, Class1: 2	SMOTE: 0.70 NM2: 0.71
	2	0.9457	Class0: 3, Class1: 2	SMOTE: 0.69 NM2: 0.70
	3	0.9457	Class0: 3, Class1: 2	SMOTE: 0.68 NM2: 0.70

SMOTETomek + Cost-Sensitive Logistic Regression combination worked best for scenario 2.

The baseline F1-score of 0.9444 is could not be surpassed even 1% for scenario two. This is happened because the imbalance between minority class and majority class for scenario 2 is very slight. It can be concluded that the effectiveness of the imbalance learning techniques loses its effectiveness when the degree of imbalance is lessened.

It is also can be seen that weight hyperparameter on most of the cases optimized at 0:3, 1:2 which is actually the original ratio of the data at hand. Additionally, it is important to state that minority : majority class ratio for scenario is 0.40 / 0.60 = 0.66. Especially for SMOTENM1 and SMOTENM2 predictive models' ratio hyperparameters were optimized at a very close point to 0.66.

5.5.3 Scenario 3 – 20:80 Imbalance Ratio with 20 Independent Variables

Scenario 3 showcases a balance between scenario 1 (extreme imbalance) and scenario 2 (mild imbalance) again 20 features and 20000 examples are used for this scenario. 16000 of the examples belong to the majority class and 4000 of them belong to the minority class. The baseline F1-score is 0.931. The best results that were obtained is given below.

Table 5.18 Best Scores Obtained by Imbalanced Learning Techniques, Scenario 3

	Rank	F1 Score	Class Weights	Minor / Major
SMOTE	1	0.9323	Class0: 3, Class1: 2	0.41
	2	0.9322	Class0: 2, Class1: 1	0.55
	3	0.9321	Class0: 3, Class1: 2	0.40
SMOTETomek	1	0.9323	Class0: 7, Class1: 3	0.47
	2	0.9322	Class0: 2, Class1: 1	0.53
	3	0.9321	Class0: 2, Class1: 2	0.42
SMOTENM1	1	**0.9329**	Class0: 1, Class1: 1	SMOTE: 0.30 NM1: 0.33
	2	0.9329	Class0: 1, Class1: 1	SMOTE: 0.29 NM1: 0.36
	3	0.9329	Class0: 1, Class1: 1	SMOTE: 0.29 NM1: 0.33

Table 5.18 continued

SMOTENM2	1	0.9326	Class0: 1, Class1: 1	SMOTE: 0.28 NM2: 0.30
	2	0.9324	Class0: 1, Class1: 1	SMOTE: 0.29 NM2: 0.31
	3	0.9322	Class0: 1, Class1: 1	SMOTE: 0.26 NM2: 0.28

All models surpassed the baseline model performance but very slightly (only 3rd decimal changed). That strengthens the conclusion that specifically in extreme imbalance scenarios, the importance of imbalanced learning techniques become more prominent. Best performing model was found to be SMOTENM1 + Cost-Sensitive Logistic Regression. It is important to note that, the minority : majority ratio for scenario 3 is 20 / 80 = 0.25. The optimum ratio hyperparameters do not far away from this value especially for SMOTENM1 and SMOTENM2 models. Considering that the default ratio hyperparameter is 1 which makes the density of minority and majority class equal, it can be said that default hyperparameters offer results which are far less optimal.

5.5.4 Scenario 4 – 1:99 Imbalance Ratio with 4 Independent Variables

This scenario is created to examine the results when the number of independent variables is decreased. The baseline F1-score for this scenario is 0.976. Best results attained by Monte Carlo simulation can be seen below.

Table 5.19 Best Scores Obtained by Imbalanced Learning Techniques, Scenario 4

	Rank	F1 Score	Class Weights	Minor / Major
SMOTE	1	0.9845	Class0: 4, Class1: 1	0.08
	2	0.9845	Class0: 4, Class1: 1	0.09
	3	0.9845	Class0: 4, Class1: 2	0.1
SMOTETomek	1	0.9845	Class0: 4, Class1: 1	0.06
	2	0.9845	Class0: 4, Class1: 1	0.08
	3	0.9845	Class0: 4, Class1: 1	0.09
SMOTENM1	1	**0.9872**	Class0: 2, Class1: 3	SMOTE: 0.04 NM1: 0.17
	2	0.9872	Class0: 2, Class1: 3	SMOTE: 0.03 NM1: 0.20
	3	0.9872	Class0: 1, Class1: 1	SMOTE: 0.06 NM1: 0.18
SMOTENM2	1	0.9869	Class0: 3, Class1: 2	SMOTE: 0.04 NM2: 0.08
	2	0.9845	Class0: 3, Class1: 2	SMOTE: 0.04 NM2: 0.09
	3	0.9845	Class0: 7, Class1: 3	SMOTE: 0.05 NM2: 0.13

Baseline score is surpassed by all predictive models given in Table 5.19. The difference between scenario 1 and 4 is that the results are much higher in scenario 4 even though the number of examples and the imbalance ratio is same. This is caused by the difference in the number of independent variables. Scenario 1 has 20 features as per in scenario 2 and 3 but scenario 4 has only 4 independent variables. This decreases the complexity of the dataset and acquiring high scores are easier on less complex datasets.

CHAPTER 6

DISCUSSION AND CONCLUSION

This study is conducted under the scope of finding an answer to the question: "What are the state of art methods for imbalanced classification and which combinations of these methods yields best results in extremely imbalanced real-world data?" The real-world data used for this study chosen from the field of basketball. The dataset is composed by the National Collegiate Athletic Association Men's Basketball League player statistics from 2008 up to 2022. The dataset contains 21422 unique players. 20696 of them do not drafted by NBA Teams and 726 of them are drafted by the NBA Teams. The draft status of the player was the target variable and other 48 player statistics variable was used as predictor variables. These numbers are the numbers achieved after preprocessing steps.

The target variable, which shows the draft status of the college basketball player has 2 classes. Drafted and non-drafted. By looking at numbers above, it can be calculated that only 3.389% of the players were drafted. On the other hand, remaining 96.611% belongs to the majority class which causes an extreme imbalance between class labels.

Before doing any experiments to find out which method or combinations of methods works best, the evaluation metric which was used throughout the study had been chosen with the help of literature and critical thinking. Accuracy is one of the most common evaluation metrics in the world of data science but in occasions of extreme class imbalance, it gives unreliable results. A no skill model which only predicts the

majority class would score 0.966 in accuracy with the dataset used in this study. This made finding another metric a necessity. Since most of the problem domains which contain imbalanced class labels inherently like fraud detection, e-mail spam detection, anomaly detection, oil spill detection, etc. most important class label is the minority class, using a metric focuses on the minority class was aimed in the study. Precision and Recall metrics are a good way to show how a model predict the minority class. He and Ma in their book published in 2013 mentioned that increasing recall without decreasing prediction is the main focus of imbalanced learning but these two metrics often times have an inverse relationship.

F1 score is the harmonic mean of Precision and Recall metrics which was chosen as the evaluation metric of the study. In order to mitigate the adverse effects of class imbalance, there are some techniques developed in literature. These techniques were used by themselves and in compliance with each other to find the best combination for the extremely imbalanced real-world dataset used in this study. First experiments were done by using baseline models in order to find out which default learning model fits best to the current form of the dataset. Logistic Regression and SVM provided the best results in these experiments. The F1 score of Logistic Regression and SVM was 0.723. Following experiments used oversampling and undersampling techniques also their combinations. The default hyperparameters of oversampling techniques provided significantly less scores than the scores obtained by using no additional sampling technique. For example, Logistic Regression baseline score dropped from 0.723 to 0.596. Most of the sources in literature offers oversampling techniques to balance the minority and majority class in order to achieve better prediction performance which does not match with the results obtained. Then a grid search was applied to find best hyperparameters. Best minority / majority ratio hyperparameter for oversampling techniques were found between 0.07 and 0.08. Compared to the default hyperparameter of class ratio which is 1.0, optimum ratio hyperparameters are significantly less. Weiss and Provost (2001) achieved similar findings with different imbalanced datasets. In the study, the optimum minority / majority ratio found closer to the original ratio of classes which is quite different from ratio of 1

where minority and majority class frequencies matched. This can be concluded by this experiment that for extreme imbalance scenarios as in this study, default ratio hyperparameter which makes the frequency of each class equal does not work. Same approach was applied for the undersampling techniques which allow one to change the class ratio hyperparameter like versions of Near Miss algorithm. Optimum ratio hyperparameter was found between 0.07 and 0.08 again. Also, combinations of Oversampling and Undersampling also worked best with ratio hyperparameters between the range [0.06, 0.11]. These results ensure that grid search for optimum hyperparameters of Sampling techniques have utmost importance.

Following phase of the experiments, sampling strategies aforementioned combined with cost-sensitive learning. In this part, grid search applied to find the best class weight hyperparameter for learning algorithm and best class ratio hyperparameter for sampling techniques. The general advice in literature is to use inverse values of original class distribution to balance class importance in predictive modelling. But the best performing models in the experiment showed that best combination of hyperparameters shows that even though sampling algorithm's optimized ratio hyperparameters were between 0.07 to 0.13 which still indicates an imbalanced distribution, the corresponding optimum class weight hyperparameters found as {0:1, 1:1}, {0:2, 1:1}, {0:3, 1:2}, etc. which are offering a balanced weighting with more weight on majority side and different from the general advice in the literature.

Lastly, in order to find the best probability threshold, probability threshold grid search is applied for 5 highest scoring models of the study. 2 of the best methods are SMOTENM2 algorithm which overlaps with the result which is attained from the original paper Jianping and Inderjeet published (2003) in which Near Miss algorithm were offered. In that article, the best performing version of Near Miss algorithm was also version 2. Optimum probability thresholds found around 0.50 and using optimum thresholds offered slight increase in F1 scores. The study consolidated 3 different approaches (sampling, cost-sensitive, threshold moving) in Imbalanced Learning and acquired better results than single use of the above methods.

In addition to the real-world dataset experiments, Monte Carlo simulation is applied in order to fortify and generalize aforementioned findings. In order to achieve this, 4 different scenario is simulated. For each scenario 20000 examples were used and Repeated stratified 10 fold cross validation with 50 repeats was applied which adds up to 500 different train test splits for each scenario. Scenario 1 represents an extreme scenario (1:99), scenario 2 is a balanced scenario with slight imbalance (40:60), scenario 3 is a medium imbalance which lays between scenario 1 and 2 (20:80). Last scenario is an extreme imbalance scenario but instead of 20 independent variables like first three scenario has, last scenario has only 4 independent variables.

Scenario 1 and 4 has the same imbalance ratio which is 1:99. Both of the scenarios yield very similar findings with real-world data.

The statement of default ratio hyperparameters which equalize both classes do not work well for extreme imbalance cases is supported by the results obtained from Monte Carlo simulations of scenario 1 and 4, which simulates extreme imbalance cases. Optimum minority : majority hyperparameters found to be between 0.08 to 0.2 where the default hyperparameter is equal to 1. This also supports the statement made in this study that small increases in minority : majority are enough to optimize the predictive performance of the imbalanced classification model.

The general advice for cost sensitive learning class weight hyperparameter is to use the inverse weights of the original minority : majority ratio as it was stated in the study. Extremely imbalanced real-world dataset experiments did not supported that approach, additionally the optimum weight hyperparameters put more weight on the majority class and less on minority class and sometimes a balanced class weight distribution found to be the best choice.

The contributions of the study can be listed as follows:

- First scientific study that uses NCAA College Basketball league Player Statistics Dataset

- The study proposed combining SMOTE, Near Miss, Cost-Sensitive Learning and Threshold Moving and achieved successful results for extreme imbalanced classification.
- The study proved that default minority : majority ratio which is 1 does not work well especially with imbalanced cases. This is an overlooked aspect in Imbalanced Learning.
- The study proved wrong the general advise of setting cost-sensitive learning class weight hyperparameters according to the inverse of the minority : majority ratio. (setting majority weight 2 and minority weight 8 for 20:80 minority : majority ratio)

REFERENCES

Bader-El-Den, M., Teitei, E., & Perry, T. (2019). Biased Random Forest for Dealing with the Class Imbalance Problem. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(7), 2163–2172. https://doi.org/10.1109/TNNLS.2018.2878400

Batista, G. E. A. P. A., Bazzan, A. L. C., & Monard, M. C. (2003). Balancing Training Data for Automated Annotation of Keywords: a Case Study. *In Proceedings of the Second Brazilian Workshop on Bioinformatics, January*, 35–43. http://www.cs.waikato.ac.nz/

Batista, G. E., Prati, R. C., & Monard, M. C. (2004). A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9842 LNCS*(1), 31–42. https://doi.org/10.1007/978-3-319-45378-1_4

Branco, P., Torgo, L., & Ribeiro, R. P. (2016). A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys*, *49*(2), 1–50. https://doi.org/10.1145/2907070

Brownlee, J. (2020). Imbalanced Classification with Python. *Machine Learning Mastery*, 463.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence*, *30*(2), 321–357. https://doi.org/10.1002/eap.2043

Chawla, N. V., Japkowicz, N., & Kotcz, A. (2004). Editorial: Special Issue on Learning from Imbalanced Data Sets. *ACM SIGKDD Explorations Newsletter*, *6*(1), 1–6. https://doi.org/10.1145/1007730.1007733

Chen, C., Liaw, A., & Breiman, L. (2004). Using Random Forest to Learn

Imbalanced Data. *Discovery*, *1999*, 1–12.

Chen, T., & Guestrin, C. (2016). *XGBoost : A Scalable Tree Boosting System*.

Dumpala, S. H., Chakraborty, R., & Kopparapu, S. K. (2018). *A Novel Data Representation for Effective Learning in Class Imbalanced Scenarios*. 2100–2106.

Fernández, A., García, S., Galar, M., & Prati, R. C. (2018). *Learning from Imbalanced Data Sets (2018, Springer International Publishing).pdf*.

Han, H., Wang, W.-Y., & Mao, B.-H. (2005). *Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning*. https://doi.org/10.1007/978-3-319-68385-0_20

Hart, P. (1967). *The condensed nearest neighbor rule (Corresp.)* (pp. 1966–1967).

He, H., Bai, Y., Garcia, E. A., & Li, S. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In IEEE International Joint Conference on Neural Networks, 2008. *IJCNN 2008.(IEEE World Congress on Computational Intelligence) (Pp. 1322– 1328)*, *3*, 1322–1328.

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Computer Society*, *807*(9), 81–110. https://doi.org/10.1007/978-3-030-04663-7_4

He, H., & Ma, Y. (2013). Imbalanced learning: Foundations, algorithms, and applications. In *Imbalanced Learning: Foundations, Algorithms, and Applications*. https://doi.org/10.1002/9781118646106

Japkowicz, N. (2000). The Class Imbalance Problem: Significance and Strategies. *Proceedings of the 2000 International Conference on Artificial Intelligence*, 111--117.

Jianping, Z., & Inderjeet, M. (2003). kNN Approach to Unbalanced data Distributions: A Case Study Involving Information Extraction. *Proceedings of Workshop on Learning from Imbalanced Datasets.*, *126*.

Jo, T., & Japkowicz, N. (2004). Class imbalances versus small disjuncts. *ACM SIGKDD Explorations Newsletter*, *6*(1), 40–49.

Krawczyk, B. (2016). Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, *5*(4), 221–232. https://doi.org/10.1007/s13748-016-0094-0

Kubat, M., & Matwin, S. (1972). Adressing the Curse of Imbalanced training Sets: One-Sided Selection. *IEEE Transactions on Systems, Man and Cybernetics*.

Kuhn, M., & Johnson, K. (2013). Applied Predictive Modeling with Applications in R. In *Springer* (Vol. 26). http://appliedpredictivemodeling.com/s/Applied_Predictive_Modeling_in_R.pdf

Laurikkala, J. (2001). Improving identification of difficult small classes by balancing class distribution. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *2101*(June 2001), 63–66. https://doi.org/10.1007/3-540-48229-6_9

Liu, X., Wu, J., & Zhou, Z. (2008). *Exploratory Undersampling for Class-Imbalance Learning*. 1–14.

Provost, F. (2000). Machine learning from imbalanced data sets 101. *Proceedings of the AAAI'2000 Workshop on …*, 3. https://www.aaai.org/Papers/Workshops/2000/WS-00-05/WS00-05-001.pdf%5Cnpapers://1c40c143-2a6e-4e94-8c17-c5bc9ae73d7e/Paper/p11435

Sammut, C., & Webb, G. I. (2010). Encyclopedia of Machine Learning. In *Encyclopedia of Machine Learning*. https://doi.org/10.1007/978-0-387-30164-8

Sun, Y., Wong, A. K. C., & Kamel, M. S. (2009). Classification of imbalanced

data: A review. *International Journal of Pattern Recognition and Artificial Intelligence*, *23*(4), 687–719. https://doi.org/10.1142/S0218001409007326

Susan, S., & Kumar, A. (2021). The balancing trick: Optimized sampling of imbalanced datasets—A brief survey of the recent State of the Art . *Engineering Reports*, *3*(4). https://doi.org/10.1002/eng2.12298

Susan, S., Sethi, D., & Arora, K. (2021). CW-CAE: Pulmonary Nodule Detection from Imbalanced Dataset using Class-Weighted Convolutional Autoencoder. *Advances in Intelligent Systems and Computing*, *1166*(August), 825–833. https://doi.org/10.1007/978-981-15-5148-2_71

Tayal, A., Coleman, T. F., & Li, Y. (2015). RankRC: Large-Scale Nonlinear Rare Class Ranking. *IEEE Transactions on Knowledge and Data Engineering*, *27*(12), 3347–3359. https://doi.org/10.1109/TKDE.2015.2453171

Thai-Nghe, N., Gantner, Z., & Schmidt-Thieme, L. (2010). Cost-sensitive learning methods for imbalanced data. *Proceedings of the International Joint Conference on Neural Networks*, 0–7. https://doi.org/10.1109/IJCNN.2010.5596486

Ting, K. M. (2002). An instance-weighting method to induce cost-sensitive trees. *IEEE Transactions on Knowledge and Data Engineering*, *14*(3), 659–665. https://doi.org/10.1109/TKDE.2002.1000348

Tomek, I. (1976). *Two Modifications of CNN*. 769–772.

Weiss, G. M., & Provost, F. (2001). The effect of class distribution on classifier learning: an empirical study. *Technical Report ML-TR-44, Department of Computer Science, Rutgers University*, 1–26. https://pdfs.semanticscholar.org/45ca/1d5528a4e5beb5616c1ec822901be2de1d59.pdf%0Ahttp://storm.cis.fordham.edu/~gweiss/papers/ml-tr-44.pdf

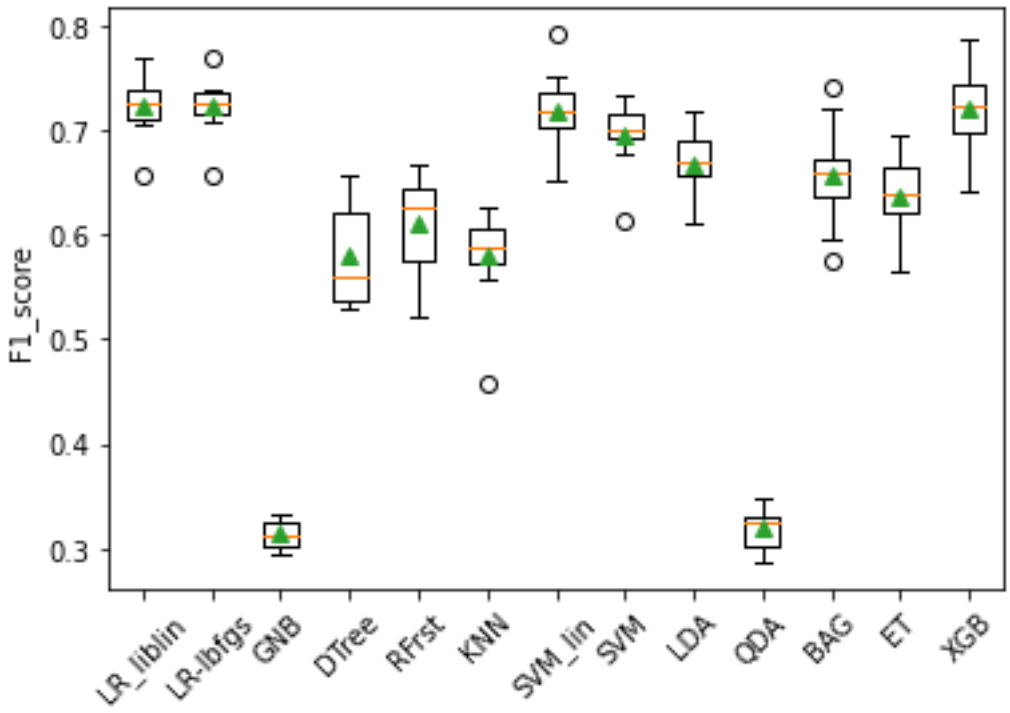
Wilson, D. L. (1972). Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. *IEEE Transactions on Systems, Man and Cybernetics*, *2*(3), 408–

421. https://doi.org/10.1109/TSMC.1972.4309137

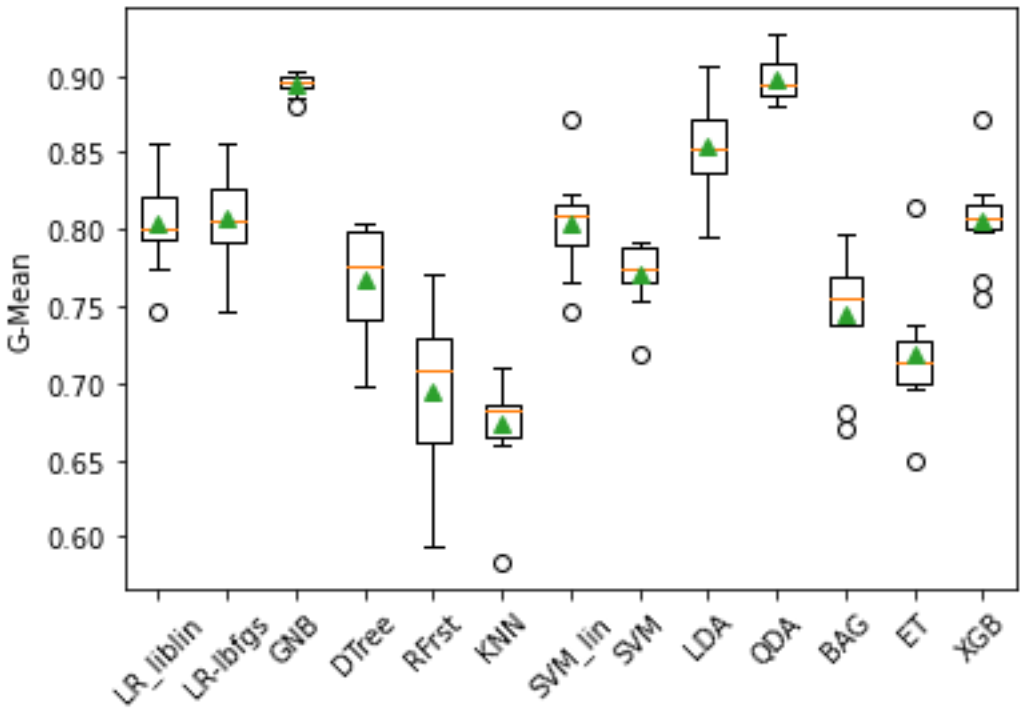
Yang, X., & Song, Q. (2007). *Weighted Support Vector Machine for Data Classification*. *December 2013*. https://doi.org/10.1109/IJCNN.2005.1555965

APPENDICES

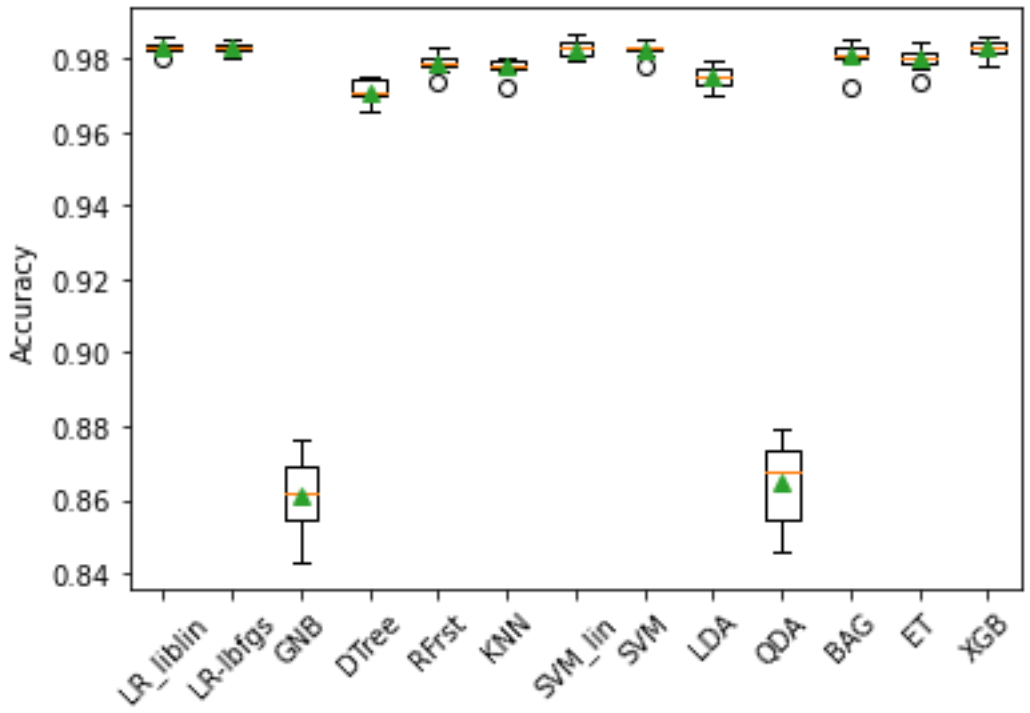
A. Cross Validation F1 Score Box Plots of Baseline Models with Standardized Variables



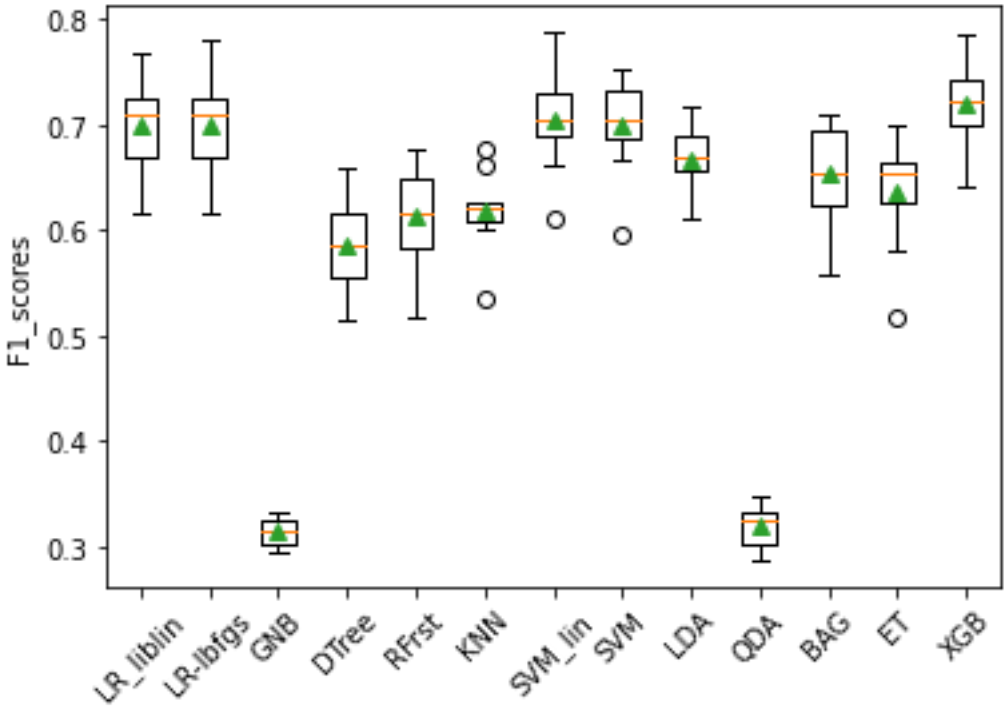
B. Cross Validation G-Mean Scores of Baseline Models with Standardized Variables



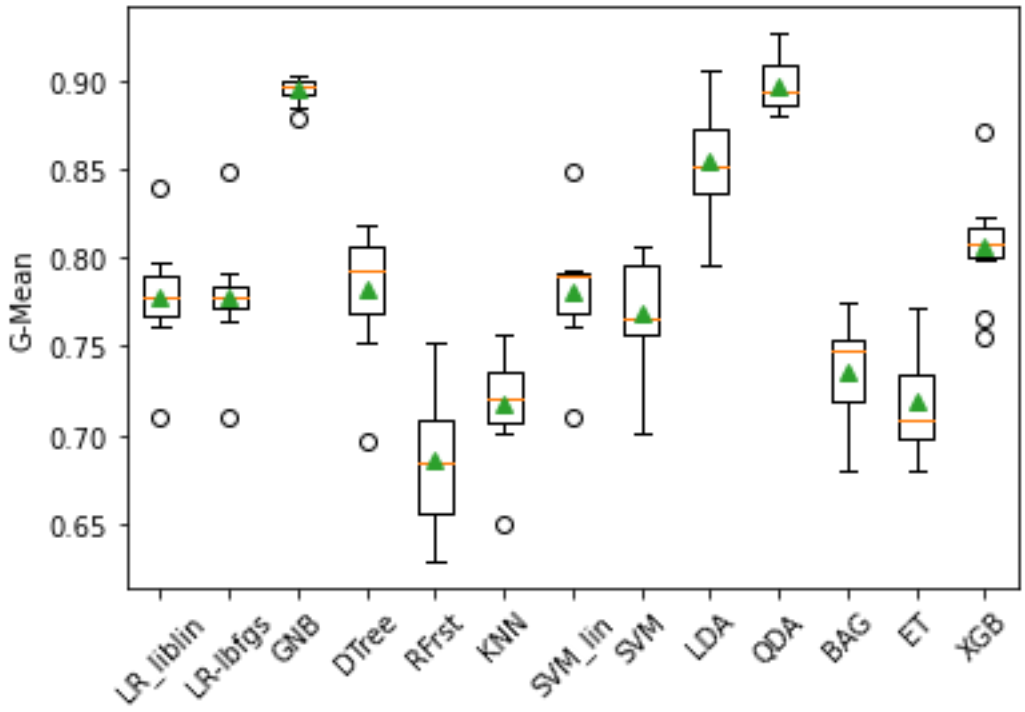
C. Cross Validation Accuracy Scores of Baseline Models with Standardized Variables



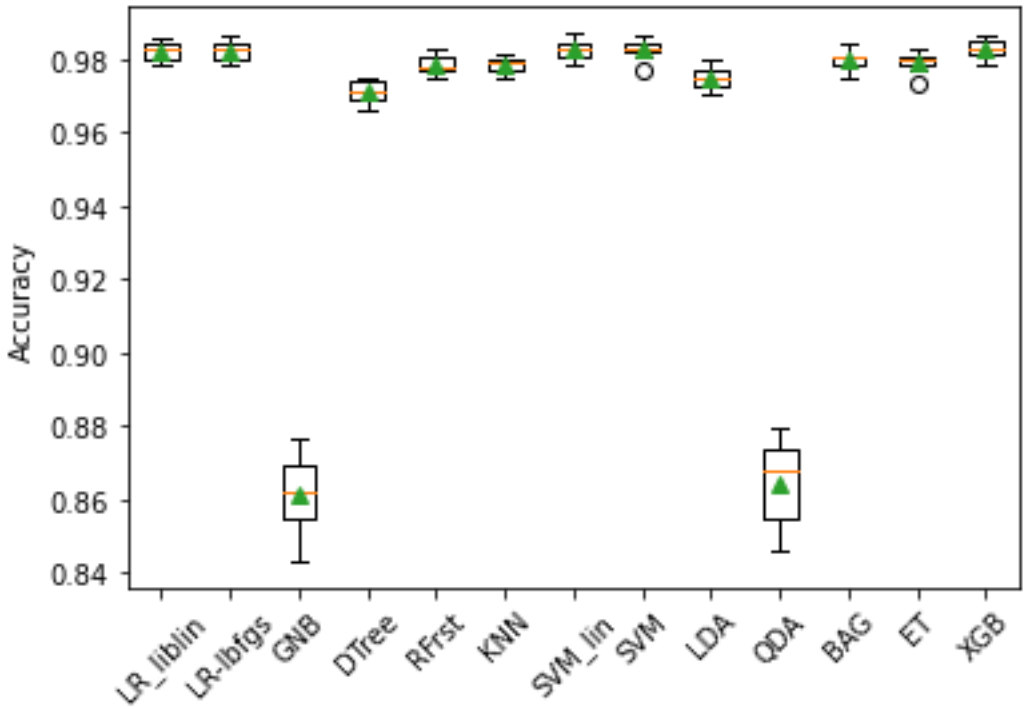
D. Cross Validation F1 Scores of Baseline Models with Normalized Variables



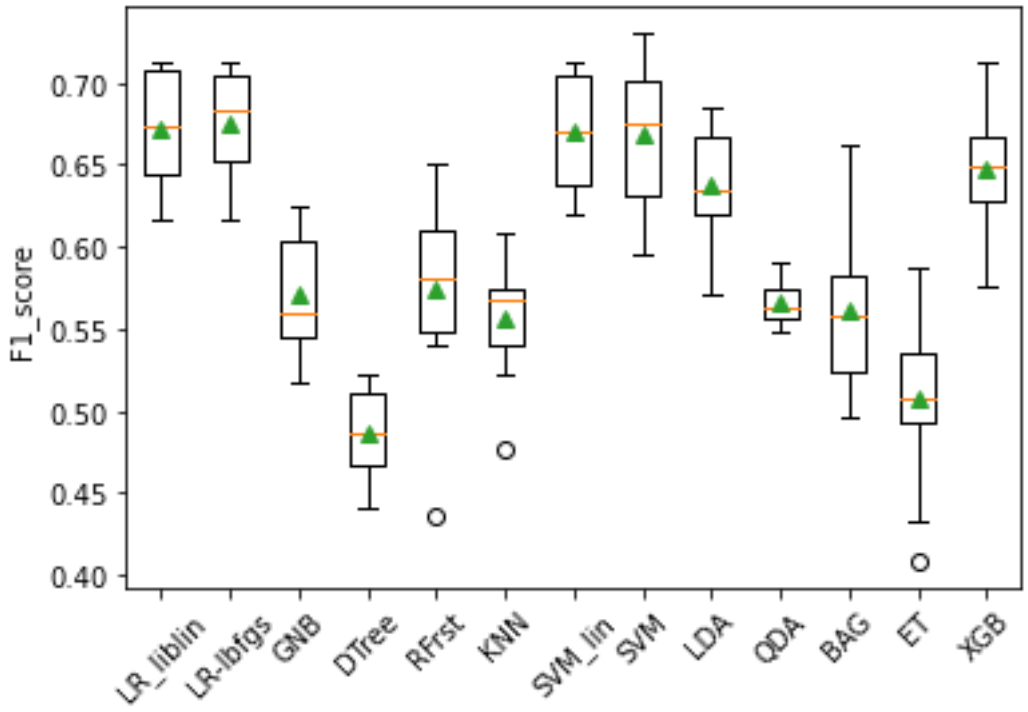
E. Cross Validation G-Mean Scores of Baseline Models with Normalized Variables



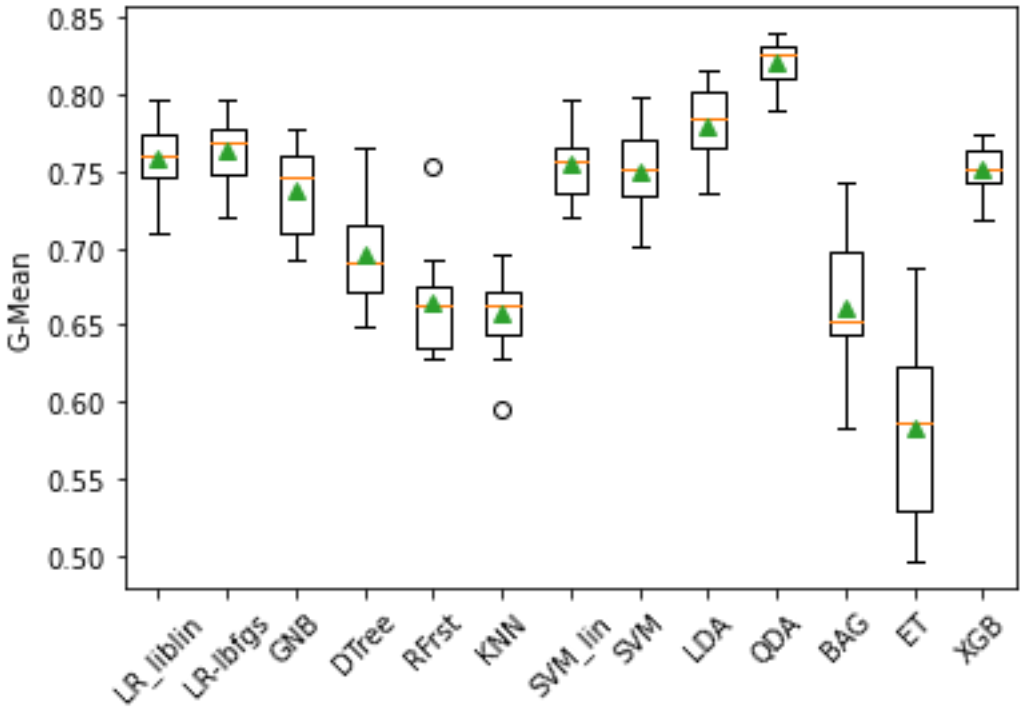
F. Cross Validation Accuracy Scores of Baseline Models with Normalized Variables



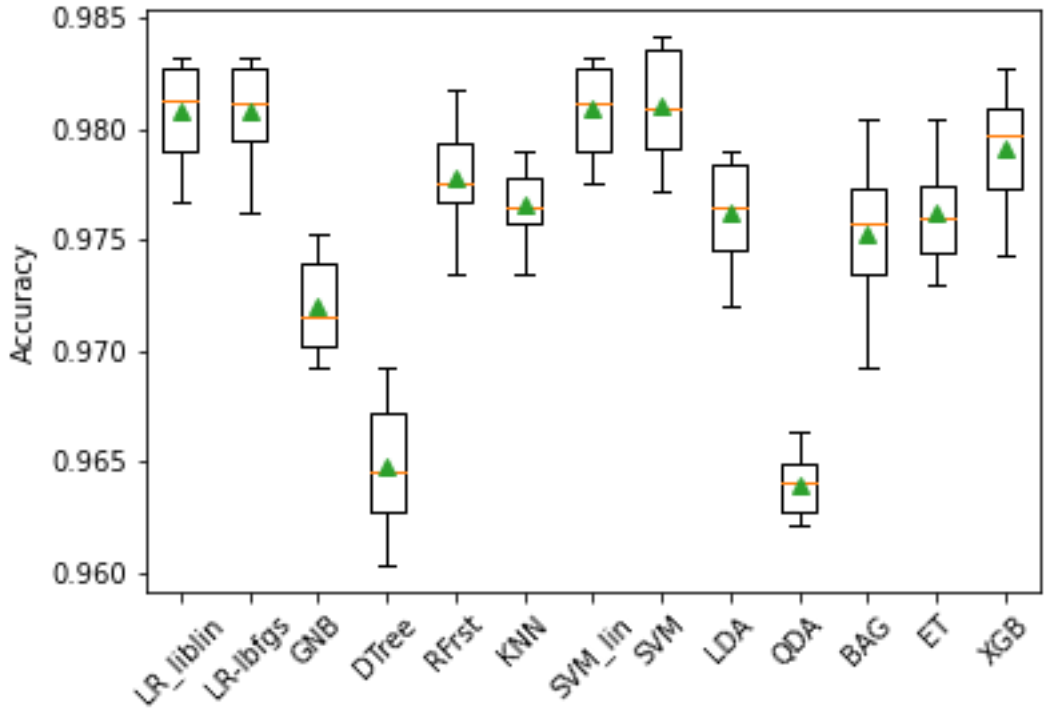
G. Cross Validation F1 Scores of Baseline Models with 13 Principal Components



H. Cross Validation G-Mean Scores of Baseline Models with 13 Principal Components



I. Cross Validation Accuracy Scores of Baseline Models with 13 Principal Components



J. Sampling Techniques Performances with Logistic Regression Algorithm