Major Research Project

Fitting AdaBoost Models From Imbalanced Data with Applications in College Basketball

Raymond Romaniuk, Brock University

May 2, 2023

Abstract

Data imbalance is an important consideration when working with real world data. Over/undersampling approaches allow us to gather more insight from the limited data we have on the minority class; however, there are many proposed methods. The goal of our study is to identify the optimal approach for over/undersampling to use with Adaptive Boosting (AdaBoost). Based on a simulation study, we've found that combining AdaBoost with various sampling techniques provides an increased weighted accuracy across classes for progressively larger data imbalances. The three Synthetic Minority Oversampling Technique's (SMOTE) and Jittering with Over/Undersampling (JOUS) performed the best, with the JOUS approach being the most accurate for all levels of data imbalance in the simulation study. We then applied the most effective over/undersampling methods to predict upsets (games where the lower seeded team wins) in the March Madness College Basketball Tournament.

Keywords: Imbalanced Data, Boosting Methods, AdaBoost, Over/Undersampling, College Basketball

Contents

1 Introduction

1.1 Decision Trees

Two of the most common machine learning problems, *regression* and *classification*, are members of the *supervised learning* family. Supervised learners are created using labelled training data, where the model makes a prediction for the dependent variable in the dataset and then establishes how close/far its prediction was from the true value. This difference between prediction and true value is known as the *residual* and, naturally, models are built to minimize this difference and create accurate predictions.

In this paper we will focus on solving classification problems, specifically binary classification problems. That is to say in general there can be any number of classes attributed to the dependent variable in our dataset, but we will focus on the case where there are only two classes (True/False, Yes/No, Win/Lose, etc.).

The decision tree model provides one of the most intuitive outputs in the field of machine learning. A decision tree is a combination of splitting criteria that splits the sample space into regions and provides values to be predicted (whether it be a regression or classification problem) for data points meeting the criteria of a given region. As with any machine learning model the decision tree fits a function $y = f(x)$ to the training dataset. In the case of the decision tree this function is constructed as follows,

$$f(x) = \sum_{m=1}^{M} c_m \cdot I_{R_m}(x)$$

where c_m can be estimated as,

$$\hat{c}_m = \frac{\sum_{i=1}^{n} I_{R_m}(x_i) \cdot y_i}{\sum_{j=1}^{n} I_{R_m}(x_j)}$$

given some partition of the data, R_m.

Here M is the total number of regions created by the chosen splits, $I_{R_m}(x)$ is an indicator function indicating whether an observation falls in the m^{th} region, n is the total number of observations, x_i and y_i are the x and y values corresponding to the i^{th} observation, and R_m is a partition of the X-space of the independent variables. These partitions are found recursively and are the partitions that minimize the loss of the model at a given iteration. Selecting the optimal partition each time, however, does not guarantee a global optimization of our loss, as paths that prove to make superior predictions may be dismissed due to not being the optimal partition at a given iteration. For the smallest of datasets each possible set of partitions could be checked and the global minimum loss found, but this is simply not a feasible approach for the majority of datasets. We can look at the $\hat{c}_m$ formula as finding the average value of y for the m^{th} region and thus $\hat{f}(x)$ simply predicts this average value when a new x observation falls in the m^{th} region.

Decision trees get their name from the tree-like structure that is often used to visualize them. In the case of the decision tree, the *branches* grow downward (alternatively to the upward growth found in nature) and culminate in *terminal nodes*, where there are no more splits in the given branch, known as *leaves* (again leaning into the tree theme). Within each branch we also find *decision/interior nodes* where a decision is made, based on the data, which split to follow. The last piece of terminology in a decision tree is the *root node*. The root node is the first split in our decision tree from which all of the subsequent splits follow. In the context of this project we will be using what is known as *decision stumps*. A decision stump is a decision tree with only one split, the root node, thus it only has the root node and two leaves. Figure 1 depicts an example of a decision tree for predicting whether a team will win a basketball game, given their points per game (PPG) and field goal percentage (FG%).

Now that we've introduced the structure of a decision tree the next, and most important, step is introducing how the splits in our tree are created. When fitting a decision tree we want to chose splits that enable our model to have the best possible accuracy and this leads us to the notion of the *purity* of a node. Purity, in the context of decision trees, refers to the distribution of classes within the two nodes that result from the split. Our goal is to develop a model with as pure leaf nodes as possible, preferably *pure nodes* where each node only contains observations pertaining to a single class.

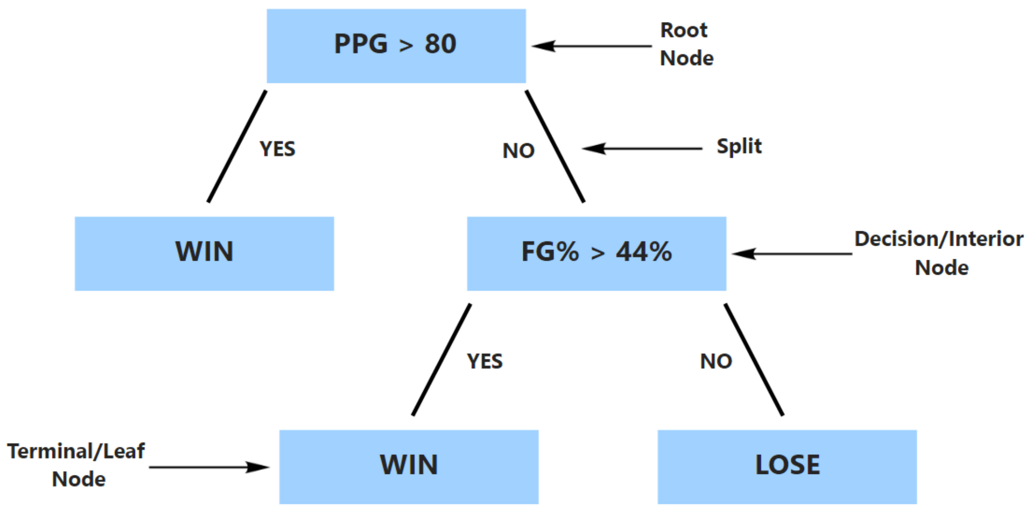

Figure 1: Visual representation of a decision tree for the outcome of a basketball game, given their offensive statistics.

For example, if we have six observations corresponding to two classes, Class A and Class B, with three observations per class and one independent variable, we'll call Variable One. Consider two possible splits:

Table 1: Split One

Node	Class A Obs	Class B Obs
1	2	1
2	1	2

Table 2: Split Two

Node	Class A Obs	Class B Obs
1	3	0
2	0	3

The option that provides us with the most pure nodes is #2, since each node only contains observations from a single class. In practice we would check the purity of nodes for all possible splits of our Variable One and we can look at our simple example as a comparison of the two best available splits. However, as mentioned this is a simple example that is conveniently perfectly separable. So, what is the procedure when our data is not separable (as likely will be the case) or all of our observations are not equally weighted?

This brings us to the measures of impurity, *Gini impurity index* and *entropy*. For this project we'll focus specifically on the Gini impurity index, but entropy could be used just as easily. The Gini impurity index and entropy are computed as follows,

$$I_G(p) = 1 - \sum_{i=1}^{k} p_i^2$$

$$I_E(p) = -\sum_{i=1}^{k} p_i \cdot \log_2 p_i$$

where p_i is the probability that a sample is of class i in a given node, if all observations have identical weight, and k is the number of classes (in our cases moving forward k = 2). If the observation weights are not equal, then p_i is calculated as $p_i = \frac{\sum_{j=1}^{n} I(y_j=i)\cdot w_j}{\sum_{j=1}^{n} w_j}$. Let's use this formula on our previous example to determine which is the optimal split.

In our first possible split each of our nodes contains two observations from one class and one observation from the other class. Plugging this information into our formula we obtain,

$$I_{G1}(p) = 1 - \sum_{i=1}^{2} p_i^2$$

$$= 1 - \left(\left(\frac{2}{3}\right)^2 + \left(\frac{1}{3}\right)^2\right)$$

$$= \frac{4}{9} = 0.4444$$

Naturally, since the distribution of classes within Node One and Node Two are identical, we can similarly obtain $I_{G2}(p) = 0.4444$. We can then sum the individual Gini values of our nodes and divide by the number of nodes we have. Doing so, we obtain an overall Gini of 0.4444 for our first potential split. Note that this averaging of the Gini values for our two nodes is actually a weighted average, however, since the total weight of the observations in each node are equal we can simply average the two Gini values.

Next we'll follow the same process for our second potential split and find,

$$I_{G1}(p) = 1 - \left(\left(\frac{3}{3}\right)^2 + \left(\frac{0}{3}\right)^2\right)$$

$$= 1 - 1 = 0$$

Thus, with identical class distributions between our two nodes, we obtain an overall Gini of 0. The optimal split occurs where the Gini value is minimized, which is the second case with completely pure nodes and a Gini of zero.

Notice that in our example each observation is weighted equally, meaning that each observation carries the same importance (since we have six observations each one has a weight of $\frac{1}{6}$). Unequal weights may be useful for training a model where error's in prediction of a specific class lead to significant repercussions. An example of this is predicting whether someone has a certain disease or does not have the disease. It would be considerably more important that we limit the error of predicting that someone does not have the disease when they actually do (a Type I error), so that treatment can start immediately. Conversely it may be incredibly inconvenient to be told you have a disease that you don't have, but this error doesn't pose as significant a risk to an individuals health as not diagnosing someone as diseased. Thus, in this medical example we would increase the weights of the observation for individuals with this disease and decrease the weights of observations for individuals without the disease. Another issue, since there are likely many more healthy individuals than diseased individuals, is that we may have imbalanced data and 90% of our observations are for healthy people (we will introduce strategies to combat this imbalance later in the paper).

Let's return to our previous example, but this time updating the weights attributed to our observations such that the three observations from Class One account for 75% of the total weight and observations from Class Two account for only 25% of the total weight. Previously both classes accounted for 50% of the total weight. Table 3 shows the previous weights for each observation in each class versus the new weights. Note that each class contains three observations and the sum of weights between classes always sums to 100%.

Table 3: Breakdown of Observation Weights

Class	Original Weight	New Weight
1	16.67%	25%
2	16.67%	8.33%

We'll use the case where the split did not perfectly separate our classes to demonstrate how the new Gini values are computed. Node One contains two observations from Class One and one observation from Class Two. From Table 1 we see that an individual observation from Class One has a new weight of 25% and an observation from Class Two has an new weight of 8.33%. Previously to calculate the Gini we were able to use the count of the observations of each class within a given node, but this time we will be using the sum of each classes weights divided by the sum of all the weights in the node as follows,

$$I_{G1}(p) = 1 - \sum_{k=1}^{2} \left(\frac{\sum_{i=1}^{3} w_i \cdot I\left(y_i = k\right)}{\sum_{i=1}^{3} w_i} \right)^2$$

$$= 1 - \left(\left(\frac{0.25 + 0.25}{0.25 + 0.25 + 0.0833} \right)^2 + \left(\frac{0.0833}{0.25 + 0.25 + 0.0833} \right)^2 \right)$$

$$= 1 - (0.8572)^2 - (0.1428)^2$$

$$= 1 - 0.7348 - 0.0204 = 0.2448$$

Unlike the case with equal weights, we no longer have identical weight distributions within both nodes. The Gini value for Node Two is found following the same procedure as Node One with different weights.

$$I_{G2}(p) = 1 - \left(\left(\frac{0.25}{0.25 + 0.0833 + 0.0833} \right)^2 + \left(\frac{0.0833 + 0.0833}{0.25 + 0.0833 + 0.0833} \right)^2 \right)$$

$$= 1 - (0.6001)^2 - (0.3999)^2$$

$$= 1 - 0.3601 - 0.1599 = 0.48$$

This time the weight of our observations between the nodes are no longer equal with 58.33% of the total weight in Node 1 and the remaining 41.67% in Node 2. Thus our new overall Gini value is calculated as,

$$I_G(p) = 0.5833 \cdot I_{G1}(p) + 0.4167 \cdot I_{G2}(p)$$

$$= 0.5833 \cdot 0.2448 + 0.4167 \cdot 0.48$$

$$= 0.3428$$

Notice that despite having the same distribution of observations between nodes as our equally weighted example, our Gini value has decreased from 0.4444 to 0.3428. We would still obtain a Gini value of zero for our separable case and it would remain the optimal split despite whether observations have their weights adjusted or not.

1.2 Boosting

Boosting is a member of a family of machine learning methods known as *ensemble methods*, which combine a number of *base learners* to create one final model. The base learner used is usually a *weak learner*, who's accuracy is slightly better than chance, and an ensemble method uses a number of these weak learners, with different strengths and weaknesses, and creates predictions by committee. A model using M weak learners can be thought of as follows,

$$\hat{f}\left(x\right) = g\left(\hat{f_1}\left(x\right), \hat{f_2}\left(x\right), ..., \hat{f_M}\left(x\right)\right)$$

where the final model is a function of the weak learners. Essentially we are "boosting" the effectiveness of these weak learners by combining them to create a single strong learner.

It is difficult to simultaneously fit M weak learners, so we use boosting algorithms to do so sequentially. A meta model is a model of a set of models that gives the user a broader view of how the set of models is developed. The meta model of a boosting algorithm is the weighted sum of its weak learners,

$$\hat{f}\left(x\right) = \sum_{m=1}^{M} \hat{\alpha}_m \cdot \hat{f_m}\left(x\right)$$

A general boosting algorithm is run as follows, where G_m is the m^{th} base learner,

1. Initialize f_0 as $f_0 = 0$.
2. Set $m = 1$
3. Compute the $\hat{\beta}$ and $\hat{\Theta}$ that satisfy the minimization of the loss function below,

$$\left(\hat{\beta}_m, \hat{\Theta}_m\right) = \underset{\beta,\Theta}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y_i, f_{m-1}\left(x_i\right) + \beta \cdot G_m\left(x_i|\Theta\right)\right)$$

4. Set $f_m\left(x\right) = f_{m-1}\left(x\right) + \hat{\beta}_m \cdot G_m\left(x|\hat{\Theta}_m\right)$
5. Repeat steps 3 and 4 for $m = 2, ..., M$ and set the final model as $\hat{f}\left(x\right) = f_M\left(x\right)$

1.3 AdaBoost

Adaptive Boosting, known as AdaBoost, is the first boosting method with the ability to adapt to the intricacies of the weak learners and was developed by Freund and Schapire [7]. AdaBoost is known as one of the best "out of the box" classifiers, meaning that with little, if any, intervention by the user, it will create a robust model with high accuracy for its relative simplicity.

AdaBoost follows the general idea outlined in the *Boosting* section, creating a number of weak learners where the observations inputted for training have their weights adjusted iteratively based on the correctness of previous predictions. Weak learners with higher accuracy will also have more influence on the predictions of our final strong learner than less accurate weak learners.

The AdaBoost algorithm can be run as follows [8]:

1. Initialize the observation weights as $w_i = \frac{1}{N}$, where w_i is the weight of the i^{th} observation and N is the total number of observations in the training set
2. Fit a weak learner to the training dataset with the corresponding weights for each observation
3. Compute the error of the new weak learner as

$$err_m = \frac{\sum_{i=1}^{N} w_i \cdot I\left(y_i \neq G_m\left(x_i\right)\right)}{\sum_{i=1}^{N} w_i}$$

   where m denotes the m^{th} weak learner, y_i represents the class of the i^{th} observation, $G_m\left(x_i\right)$ represents the classification made by the m^{th} weak learner on the i^{th} observation and $I\left(y_i \neq G_m\left(x_i\right)\right)$ is an indicator function that takes the value of 1 when a prediction is incorrect and 0 otherwise

4. Next calculate the amount of influence that the weak learner should have on the final classifications made by the strong learner using

$$\alpha_m = log\left(\frac{1 - err_m}{err_m}\right)$$

5. Using the current weights along with the α_m we update the weight of each observation with the following formula

$$w_i^{m+1} = w_i^m \cdot exp\left(\alpha_m \cdot I\left(y_i \neq G_m\left(x_i\right)\right)\right)$$

6. Repeat Steps 2 - 5, until the specified stopping criteria is reached. The stopping criteria can be a specific number of weak learners or a threshold of the change in prediction error between iterations. Finally, make predictions with

$$sign\left(\sum_{m=1}^{M} \alpha_m \cdot G_m\left(x\right)\right)$$

   since AdaBoost is a perceptron machine learning algorithm for the problem of binary classification.

Table 4: AdaBoost Example Dataset

Class	X$_1$	X$_2$
-1	64	44
-1	97	90
1	46	8
1	67	96
-1	13	100
1	78	33
-1	72	18
1	62	58
1	65	71
-1	66	73
-1	45	70
-1	8	50
1	5	13
-1	99	13
-1	49	45

Using the small dataset in Table 4, we will run the AdaBoost algorithm as an example of AdaBoost's effectiveness. Three weak learners will be fit in this example and we will compare the accuracies of each weak learner to that of the final strong learner.

In this example decision stumps will be used as our weak learner and we'll fit $M = 3$ of them. This dataset has $N = 15$ observations, so each observation will have an initial weight of $\frac{1}{15}$.

We'll run through the first iteration of the algorithm by hand to illustrate its usage.

1. First, weights are initialized as $\frac{1}{15}$
2. Next, with the Gini impurity index we find the optimal split of the equally weighted problem. With Python, we find the optimal split at $X_1 = 6.5$ having a Gini value of 0.2296
3. With our optimal split we can now compute the error of this weak learner. This is done as follows,
$$err_m = \frac{\sum_{i=1}^{N} w_i \cdot I\left(y_i \neq G_m\left(x_i\right)\right)}{\sum_{i=1}^{N} w_i}$$
$$= \frac{5}{15} = 0.3333$$
4. The influence this weak learner should have on our final model is,
$$\alpha_m = \log\left(\frac{1 - err_m}{err_m}\right)$$
$$= \log\left(\frac{1 - \frac{1}{3}}{\frac{1}{3}}\right) = 0.6931$$
5. Next we update the weights for use in the next iteration, with an example of the difference between an observation predicted correctly and one predicted incorrectly below,
$$w_1^{m+1} = w_1^m \cdot exp\left(\alpha_1 \cdot I\left(y_i \neq G_1\left(x_i\right)\right)\right)$$
$$= \frac{1}{15} \cdot exp\left(0.6931 \cdot 0\right) = 0.0667$$
$$w_3 = \frac{1}{15} \cdot exp\left(0.6931 \cdot 1\right) = 0.1333$$
Using this method to update the weights does not guarantee that they still sum to one, so we will standardize them
$$w_1 = \frac{0.0667}{0.0667 + ... + 0.0667} = \frac{0.0667}{1.3333} = 0.05$$

$$w_3 = \frac{0.1333}{1.3333} = 0.1$$

6. Finally, we would iterate through the algorithm two more times and stop once we reach our specified stopping criteria of $M = 3$

Running through the entire algorithm we obtain a strong learner with an accuracy of 73.33%, notably the three decision stumps used to create this final model only had accuracies of 66.67%, 53.33% and 66.67%, respectively. Table 5 shows the α value, split and accuracy of each weak learner, while Table 6 displays how the weights differ between each iteration.

Table 5: Summary of AdaBoost Weak Learners

Weak Learner	α_m	Split	Accuracy
1	0.6931	$X_1 \leq 6.5$	66.67%
2	0.1335	$X_1 \leq 87.5$	53.33%
3	0.6931	$X_2 \leq 10.5$	66.67%

Table 6: Weight Adjustments By Iteration

Obs	Initial	Adj. 1	Adj. 2
1	6.67%	5%	7.41%
2	6.67%	5%	3.70%
3	6.67%	10%	7.41%
4	6.67%	10%	7.41%
5	6.67%	5%	7.41%
6	6.67%	10%	7.41%
7	6.67%	5%	7.41%
8	6.67%	10%	7.41%
9	6.67%	10%	7.41%
10	6.67%	5%	7.41%
11	6.67%	5%	7.41%
12	6.67%	5%	7.41%
13	6.67%	5%	3.70%
14	6.67%	5%	3.70%
15	6.67%	5%	7.41%

Now that we've introduced the main concepts that will be explored throughout this paper we will move on to a literature review regarding strategies for handling imbalanced datasets. Later these strategies will be implemented to study their effectiveness on different types of imbalanced data.

2 Literature Review

In many datasets the class we are interested in predicting is the minority class. An example is trying to predict whether an individual has a rare disease or not. This dataset would contain many observations where the disease is not present and few where it is. We could create an accurate model by predicting all individuals to not have the disease, but that would not provide any information about the most important class, the diseased. In this section we'll review the current literature about sampling methods and performance measures to try to solve this problem.

2.1 Sampling

To combat the problem of an imbalanced dataset one potential solution is to attempt to balance the dataset. This can be acheived in two ways. The first is *undersampling* where we select a subset of the observations from the majority class. Here we continue to sample from the majority class, removing the sampled observation from the dataset, until there are a similar number of observations from both classes. By undersampling we are throwing away valuable data and we would always prefer to have more data rather than less. Thus, this solution can be a problem, especially when we are working with a small dataset.

The second potential solution is known as *oversampling*. To balance the majority and minority classes we can sample the minority class, with replacement, until we have a balanced dataset. Here instead of removing data, as happens in undersampling, we make our dataset larger. Although sampling these observations that already exist will necessarily lead to duplicates in our dataset, with datasets having a larger disparity between majority and minority classes containing more duplicates. This can lead to an overfit model, and a decrease in the models accuracy on new data, especially if the difference between the number of minority and majority observations is large.

Below we will introduce a number of over/undersampling techniques, including techniques that create a new data point out of the existing observations available from the minority class.

2.1.1 Random Undersampling (RUS)

As mentioned *random undersampling* (RUS) is the process of sampling the data related to the majority class in the dataset, and removing these sampled observations from the dataset. We continue to sample and "delete" observations from the majority class until there is an equal number of observations from both classes.

As mentioned this is feasible, but not ideal for small datasets since we do not have the luxury of being able to discard data. When we randomly delete observations from our majority class we have no indication of how important the deleted observation is to determining the unknown decision boundary of the data. Thus deleting observations may make it more difficult to train a robust model and ultimately we would prefer more data rather than less.

2.1.2 Random Oversampling (ROS)

Random oversampling (ROS), on the other hand, expands the number of observations in our dataset rather than diminishes it. Oversampling entails sampling from the minority observations, with replacement, until the number of observations of our majority and minority classes are equivalent.

Random oversampling can lead to duplicate observations of the minority class. The greater the difference between the number of the majority and minority observations the more duplicates we will have. Since there is only a finite number of minority observations in our dataset we are unable to create any unique new minority observations and are confined to sampling only from those that are available. We will introduce some techniques to create unique *synthetic data* points for the minority class later on using the minority observations that are available.

2.1.3 Tomek Link Majority Undersampling (TLMU)

The idea of *Tomek links* was introduced in 1976 by Ivan Tomek [21]. A Tomek link is found by first using the k-Nearest Neighbour (kNN) algorithm and determining the nearest neighbour ($k = 1$). Once each observations nearest neighbour has been found we define a Tomek link as the observation whose nearest neighbour is of the opposing class.

Although ultimately unknown, the purpose of finding these links is to attempt to determine a decision boundary encompassed in the data. We undersample by deleting observations from the majority class that are involved in a Tomek link. This will decrease the number of majority class observations, but does not guarantee that our dataset will be balanced. Instead as mentioned the idea is to remove ambiguous points along the theorized boundary making the data more separable.

Depending on the dataset this may not be a useful technique as this approach will not provide much help if the dataset is not separable.

2.1.4 Synthetic Minority Oversampling Technique (SMOTE)

The first technique we'll introduce for creating new minority class observations is known as the *Synthetic Minority Oversampling Technique* (SMOTE) [3]. SMOTE was developed specifically to combat the data imbalance problem by creating new observations for the minority class using the available minority class observations in the dataset.

Using the features available for each of our minority observations we perform kNN to find the k most similar minority observations. We then randomly select one of the nearest neighbours for each observation and create a new artificial data point, x_{new} as follows [9],

$$x_{new} = x_i + (\hat{x}_i - x_i) \cdot \delta$$

where x_i are the features of the i^{th} minority class observation, $\hat{x}_i$ is the the randomly selected nearest neighbour of the i^{th} minority class observation and δ is a random number in the range [0,1].

2.1.5 SMOTE with Tomek Links (SMOTE-TL)

The *SMOTE with Tomek links* [2] method combines an undersampling and oversampling method to produce the benefits that we find in both methods on their own. Here we'll have the advantage of creating more minority data using SMOTE, while also, hopefully, creating a more distinguishable decision boundary between the two classes.

To perform this method we first follow the SMOTE algorithm and create enough artificial minority observations to balance the dataset. Once we have our balanced dataset we can find the Tomek links that exist and, instead of deleting only the observation from the majority class, we'll delete both observations and keep the dataset balanced.

2.1.6 SMOTE with Edited Nearest Neighbour (SMOTE-ENN)

SMOTE with Edited Nearest Neighbour (SMOTE-ENN) [1] is a combination of SMOTE for oversampling and ENN for undersampling. To use ENN [22] we perform kNN (usually with $k = 3$) and determine whether the observation in question belongs to the same class as the predominant class displayed by its neighbours. If the classes do not match then each of the observations in the neighbourhood are removed from the dataset.

This method provides a more aggressive approach to creating clear decision boundaries than using Tomek links, however for small datasets this may be problematic as we are removing more valuable observations from our analysis.

2.1.7 Jittering with Over/Undersampling (JOUS)

The *Jittering with Over/Undersamping* (JOUS) method was created explicitly for use in boosting binary classification trees known as JOUS-Boost. As indicated in the name the idea of JOUS is to oversample the minority class while adding small "jitters" to the samples, such that we don't create duplicate observations as occurs with simple random oversampling.

Mease et. al [10] define these jitters in the data as taking a sample of the Uniform$(-v \cdot \sigma_j, v \cdot \sigma_j)$ distribution where σ_j is the standard deviation of the j^{th} variable in the dataset and v is a tuning parameter that can be optimized by the user. The jitter for each variable is then added to the random sample and the unique new minority observation is completed.

2.2 Performance Measures

A second topic we'll explore in an attempt to combat the imbalanced data problem is adjusting the error calculation in the AdaBoost algorithm to account for the imbalance. From the AdaBoost algorithm outlined previously the error can be rewritten as $1 - Accuracy$ since it is the fraction of observations predicted incorrectly.

The idea we're proposing is to update the accuracy measure contained within the error to a performance measure that is more resistant to imbalanced data. Below we'll introduce a group of performance measures and hopefully determine whether there are any gains to be made by adjusting the AdaBoost error measure.

2.2.1 Accuracy

Accuracy is the standard quantity used to determine the effectiveness of a models predictions. It is the number of correctly predicted observations divided by the total number of observations and is calculated as:

$$Acc = \frac{\sum_{j=1}^{k}\sum_{i=1}^{n} f(i,j) \cdot I(y_i = G(x_i))}{n}$$

For balanced datasets simply using accuracy is reasonable, however if we have an imbalanced dataset using accuracy could be misleading. In a binary classification problem, if only 1 in 100 observations is from the minority class we could obtain an accuracy of 99% by predicting the majority class every time. This would not teach us anything about the minority class and if this model were deployed and used on new data there may not be the same 100:1 distribution dropping our accuracy with every minority class observation.

To attempt to combat this issue we'll try some alternative statistics to accuracy and see if we can make any improvement on our predictions.

2.2.2 Sensitivity

For a binary classification problem the classes can be labelled as either positive or negative, where the positive case, for the sake of our imbalanced dataset research, is defined as the minority class and the negative case is the majority class. Labelling the classes as positive or negative for balanced datasets is not a significant issue (either could be positive or negative with proper explanation) as long as each class is consistently labelled.

When our model makes predictions on data we observe the models performance using a *confusion matrix*. A confusion matrix contains four values, labelled true positive (TP), false positive (FP), true negative (TN) and false negative (FN). A prediction that is a TP occurs when our model predicts the observation to be in the positive class when that observation is actually in the positive class. Likewise a TN is when we predict the observation is in the negative class and it is actually in the negative class. Alternatively, a FP is when we predict the observation is in the positive class, but it is actually in the negative class, and a FN is when we predict the observation is in the negative class, but it is actually in the positive class. Table 7 demonstrates an example of a confusion matrix.

Table 7: Confusion Matrix Example

		Predicted	Class
		Positive	Negative
True	Positive	TP	FN
Class	Negative	FP	TN

For consistency we'll write TP, FP, TN and FN in terms of the functions $f(i,j)$ and $C(i,j)$.

$$TP = \sum_{i=1}^{n} f(i,1) \cdot C(i,1)$$

$$FP = \sum_{i=1}^{n} f(i,1) \cdot C(i,-1)$$

$$TN = \sum_{i=1}^{n} f(i,-1) \cdot C(i,-1)$$

$$FN = \sum_{i=1}^{n} f(i,-1) \cdot C(i,1)$$

where $f(i,j) = 1$ if observation i is predicted to be in class j and $C(i,j) = 1$ if observation i is actually in class j.

The next six statistics are created using the TP, FP, TN and FN values gathered from our models predictions. The idea is to see if all six will follow the same pattern of performance or if any will differ from the rest of the group.

The first statistic from this group is *sensitivity*, where

$$Sens = \frac{TP}{TP + FN}$$

Sensitivity is the true positive rate and demonstrates how well our model did predicting observations in the positive class.

2.2.3 Specificity

Next is *specificity*, which is the opposite of sensitivity, and is the true negative rate, indicating how well we did predicting observations of the negative class.

$$Spec = \frac{TN}{FP + TN}$$

2.2.4 Geometric Mean

The *geometric mean* is created using the model's sensitivity and specificity as follows,

$$GM = \sqrt{Sens \cdot Spec}$$

2.2.5 Precision

Precision is the accuracy of our predictions that the observation is a member of the positive class (minority class).

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

2.2.6 Recall

Recall is just the true positive rate and is identical to sensitivity.

$$Rec = \frac{TP}{TP + FN}$$

2.2.7 Mean F-Score

The *mean F-score* takes the precision and recall creating the statistic as,

$$MFS = \frac{\sum_{i=1}^{k} \frac{2 \cdot Rec(i) \cdot Prec(i)}{Rec(i) \cdot Prec(i)}}{k}$$

where $Prec\,(i)$ and $Rec\,(i)$ are the precision and recall of the i^{th} class.

2.2.8 Kappa Statistic

The *kappa statistic* [4] is a measure of the agreement between variables and takes into account the potential for this agreement having occured by chance. Additionally, the kappa statistic can be used as a measure of classifier performance, which is how we will employ it.

The kappa statistic is defined as,

$$\kappa = \frac{Acc - P\,(E)}{1 - P\,(E)}$$

where,

$$P\,(E) = \frac{\sum_{m=1}^{k} \left(\left[\sum_{j=1}^{k} \sum_{i=1}^{n} f\,(i,m) \cdot C\,(i,j) \right] \cdot \left[\sum_{j=1}^{k} \sum_{i=1}^{n} f\,(i,j) \cdot C\,(i,m) \right] \right)}{n^2}$$

with $P\,(E)$ representing the probability that the agreement between variables is due to chance [6].

2.2.9 Macro Average Arithmetic (Weighted Average)

The *macro average arithmetic* is the average of the model's accuracy between classes. Unlike the standard accuracy this measure is less susceptible to problems caused by imbalanced data. Thus if we came across the scenario outlined in the accuracy section, with one observation from the minority class and 99 from the majority, where the majority class was predicted everytime our macro average arithmetic value would be half that of our accuracy since our minority class offers no accuracy.

The macro average arithmetic value is calculated as,

$$MAA = \frac{\sum_{j=1}^{k} \frac{\sum_{i=1}^{n} f(i,j) \cdot C(i,j)}{n_j}}{k}$$

2.2.10 Macro Average Geometric

The *macro average geometric* measure similarly utilizes the class-wise accuracies, however instead of summing over the class accuracy it takes the product followed by the k^{th} root as follows,

$$MAG = \sqrt[k]{\prod_{j=1}^{k} \cdot \frac{\sum_{i=1}^{n} f(i,j) \cdot C(i,j)}{n_j}}$$

3 Simulation Study

3.1 Data Generation

For our simulation study we will generate the dependent variable using a logistic link function. This means we will be computing $P(y_i = -1 \mid x_i)$. The generation of our y's is done as follows

$$P(y_i = -1 \mid x_i) = \frac{e^{\mathbf{x}\cdot\beta}}{1 + e^{\mathbf{x}\cdot\beta}}$$

where β is a vector of the regression coefficients, with β_0 being an adjustable parameter to produce datasets with a desired imbalance or lack of imbalance, while β_1 through β_5 were set to 1 for simplicity and y is found by generating a random number k from the Uniform$(0,1)$ distribution and determined as follows

$$y_i = \begin{cases} -1 \text{ if } P(y_i = -1 \mid x_i) \leq k, \\ 1 \text{ otherwise} \end{cases}$$

Here $\mathbf{x}$ is an n×5 matrix representing our generated data and simulated under the following distributions:

$$x_1 \sim \text{Normal}(0,1)$$

$$x_2 \sim \text{Normal}(0,5)$$

$$x_3 \sim \text{Normal}(0,0.1)$$

$$x_4 \sim \text{Normal}(-3,1)$$

$$x_5 \sim \text{Normal}(5,0.5)$$

By default 1500 observations will be generated for each dataset, but we will also explore how limiting and expanding the number of observations impacts the effectiveness of our methods.

3.2 Comparison of Sampling Method Performance

To test the effectiveness of our sampling methods we'll compare the performance of our seven sampling methods with the performance of using AdaBoost without altering the dataset. To do this we will create 50 datasets, with 1500 observations, for each of our seven possible data imbalance situations (50/50, 60/40,..., 95/5). We will use each of our sampling methods to minimize the data imbalance, and with these datasets determine the accuracy and MAA (weighted average) of the predictions made using AdaBoost. Figure 2 shows the average accuracy for each method and data imbalance, while Figure 3 shows the average MAA.

Upon inspection of Figure 2 we see that the most accurate technique for the balanced data (50/50, 60/40, 70/30) is JOUS, followed by a cluster of methods that include the three SMOTE techniques. However, these methods are not the most accurate for the imbalanced data (90/10, 95/5). TLMU and Adaboost with no sampling method significantly outperform all the other methods. This is peculiar since we would expect combining AdaBoost with our sampling methods to provide a better model than using AdaBoost alone, and if AdaBoost is already the optimal method then using these additional sampling methods is a waste of computing power.

Figure 3, however, provides us with an explanation for what we were seeing in Figure 2. When we compare the weighted averages of each method, TLMU and AdaBoost with no sampling method perform the worst for all data imbalances. This is not surprising since these are the only two methods that return a dataset that is still imbalanced, so what looks like exceptional performance in Figure 2 is shown to be a bias towards predicting the majority class in Figure 3, since the weighted average controls for model accuracy between classes.

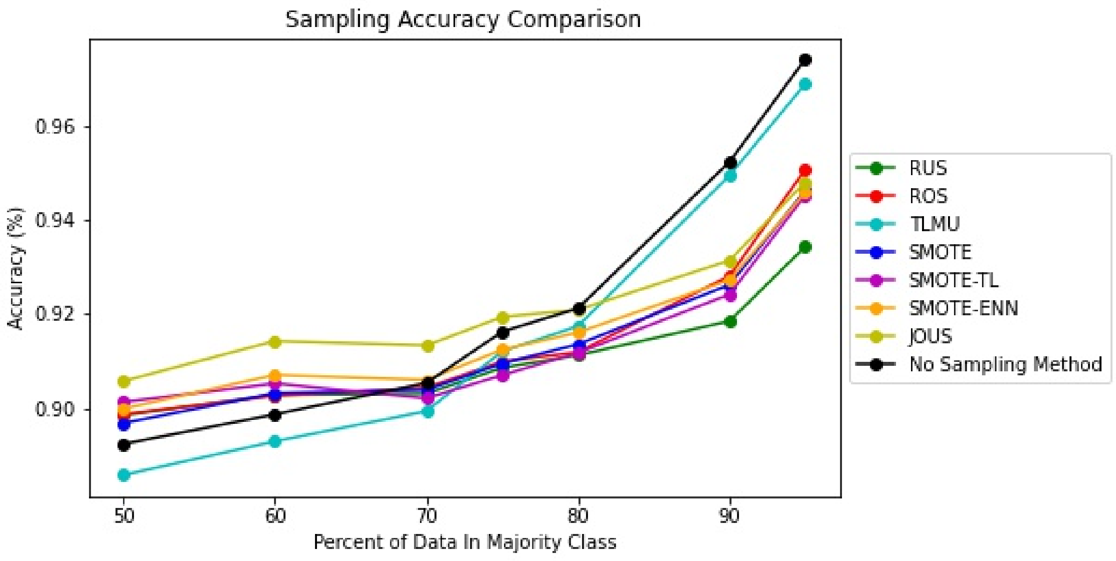

Figure 2: Comparison of the mean accuracies of each sampling method for increasing data imbalance.

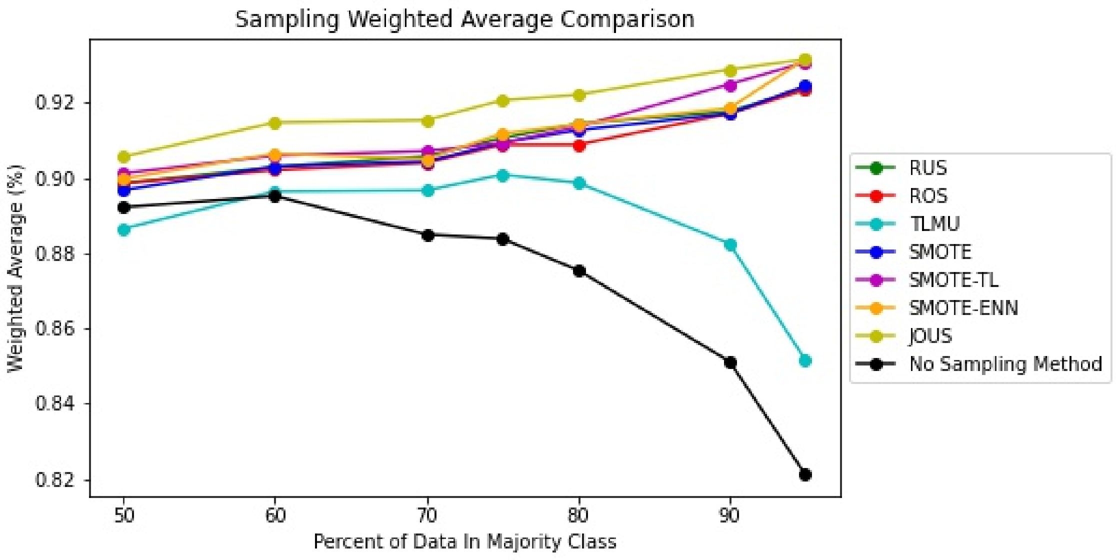

Figure 3: Comparison of the mean weighted averages of each sampling method for increasing data imbalance.

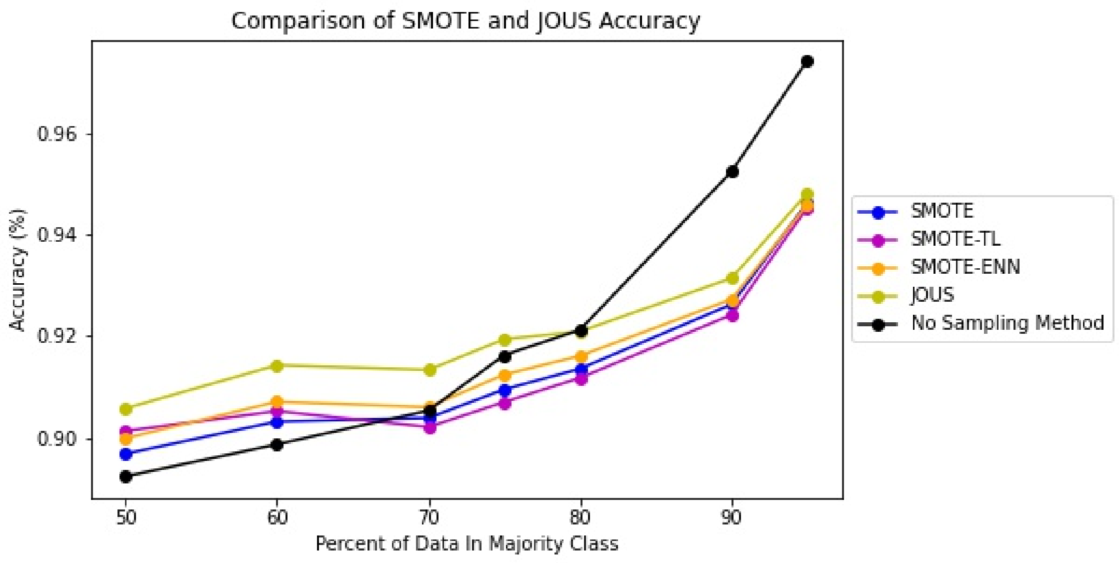

Figure 4: Comparison of the mean accuracies of JOUS and the three SMOTE methods for increasing data imbalance with not using any techniques.

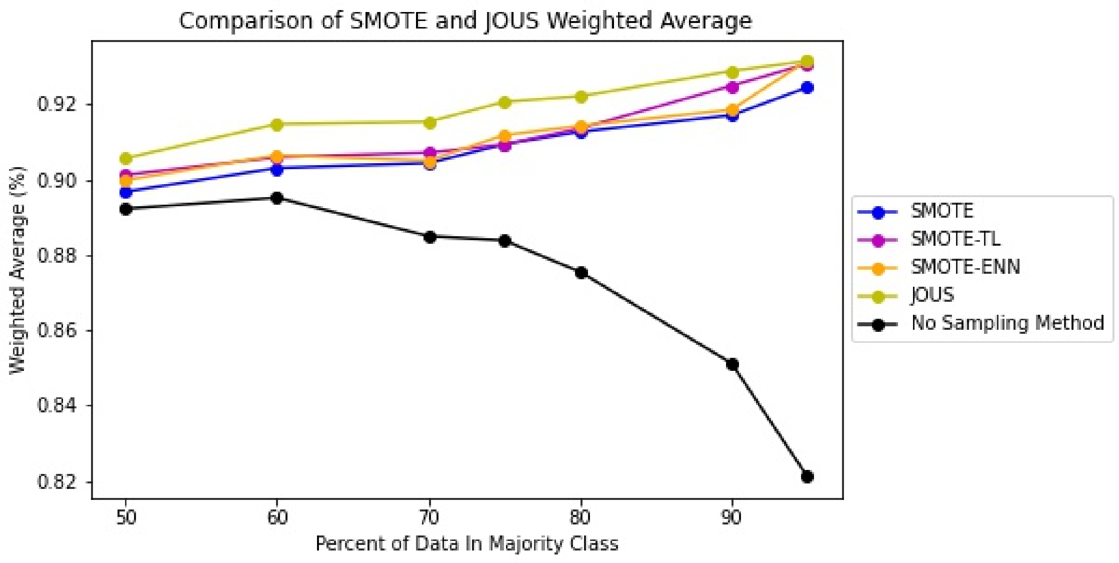

Figure 5: Comparison of the mean weighted averages of JOUS and the three SMOTE methods for increasing data imbalance with not using any techniques.

The consistent top performers between Figure 2 and 3 are JOUS and the three SMOTE variations. Figure 4 and 5 compare JOUS, SMOTE-ENN, SMOTE-TL and SMOTE with the use of no sampling technique. While JOUS slightly outperforms the SMOTE methods, it is not surprising that these are the top four best performing techniques. All four techniques create new *artificial* minority observations to balance the dataset. SMOTE-TL and SMOTE-ENN both use undersampling to remove observations and create a more separable dataset and improve model performance on the decision boundary. This leads to them both outperforming SMOTE, with respect to weighted average. Table 8 shows the mean accuracy and weighted average for each method and data imbalance combination.

Table 8: Sampling Performance By Data Imbalance

	Imbalance	50/50	60/40	70/30	75/25	80/20	90/10	95/5
Method	**Measure**							
No Method	Accuracy	89.23%	89.86%	90.53%	91.62%	92.12%	95.24%	97.43%
	MAA	89.22%	89.51%	88.49%	88.38%	87.55%	85.12%	82.12%
RUS	Accuracy	89.84%	90.28%	90.30%	90.86%	91.12%	91.85%	93.43%
	MAA	89.85%	90.29%	90.57%	91.05%	91.43%	91.76%	92.38%
ROS	Accuracy	89.86%	90.26%	90.43%	90.99%	91.19%	92.81%	95.07%
	MAA	89.86%	90.20%	90.39%	90.87%	90.88%	91.70%	92.34%
TLMU	Accuracy	88.57%	89.29%	89.93%	91.20%	91.74%	94.95%	96.90%
	MAA	88.65%	89.64%	89.66%	90.08%	89.87%	88.25%	85.16%
SMOTE	Accuracy	89.68%	90.31%	90.38%	90.95%	91.35%	92.62%	94.62%
	MAA	89.67%	90.30%	90.43%	90.93%	91.26%	91.70%	92.44%
SMOTE-TL	Accuracy	90.13%	90.52%	90.20%	90.70%	91.17%	92.42%	94.53%
	MAA	90.12%	90.59%	90.71%	90.92%	91.35%	92.48%	93.07%
SMOTE-ENN	Accuracy	89.98%	90.70%	90.60%	91.24%	91.61%	92.72%	94.59%
	MAA	89.98%	90.64%	90.50%	91.17%	91.42%	91.85%	93.15%
JOUS	Accuracy	90.57%	91.42%	91.33%	91.94%	92.09%	93.14%	94.81%
	MAA	90.56%	91.47%	91.53%	92.06%	92.20%	92.88%	93.14%

3.3 Altering AdaBoost

3.3.1 Adjusting the Error Calculation

The first alteration to AdaBoost that we propose is to update how the error term in the AdaBoost algorithm is calculated by changing it from accuracy to one of the other previously introduced performance measures. Upon testing this, however, we did not obtain any change in our model performance from the standard application of AdaBoost. This is likely because the error measure mainly impacts the weight that a specific weak learner receives in the final ensemble model, and has less to do with creating a more diverse group of weak learners that would result in a different overall final model.

3.3.2 Forcing Minority Prediction

Next, we'll explore if altering how our decision stumps are created can improve the effectiveness of AdaBoost on imbalanced datasets. The alteration we are proposing is to force all possible decision stumps to have a majority leaf node and a minority leaf node. Previously this was not the case as a highly imbalanced dataset, with equal weighting among observations, would likely return a decision stump who's leaves both predicted observations to be of the majority class. This idea stems from the desire to create decision stumps that differ from the standard approach, thus altering observation weights within the AdaBoost algorithm, and hopefully producing an increased prediction accuracy.

We will test the effectiveness of this AdaBoost alteration across seven data imbalances, and with three different dataset sizes (500, 1500 and 5000) with the data being simulated as outlined in the previous section. Figure's 6 through 8 compare the performance of our altered AdaBoost to that returned using Python's scikit-learn package for 50/50, 60/40 and 70/30 imbalances.

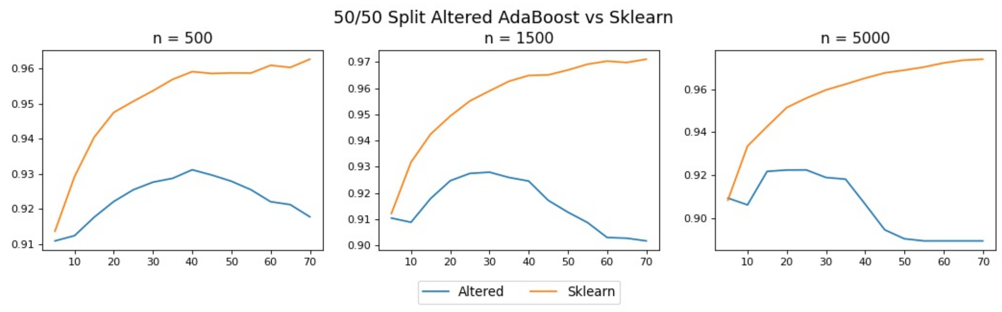

Figure 6: Comparison of the altered AdaBoost and scikit-learn's AdaBoost for a 50/50 majority/minority split and varying dataset sizes.

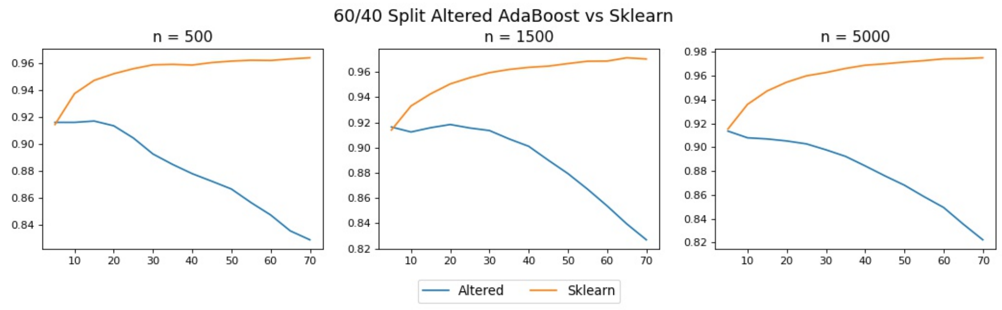

Figure 7: Comparison of the altered AdaBoost and scikit-learn's AdaBoost for a 60/40 majority/minority split and varying dataset sizes.

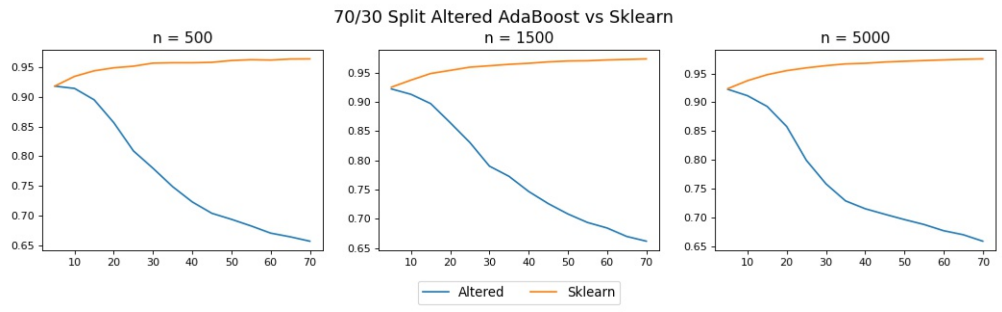

Figure 8: Comparison of the altered AdaBoost and scikit-learn's AdaBoost for a 70/30 majority/minority split and varying dataset sizes.

As we can see the scikit-learn AdaBoost outperforms our altered AdaBoost for each dataset size and almost all M values for the three imbalances. The 75/25, 80/20, 90/10 and 95/5 imbalances have performances that closely match those obtained by the 70/30 imbalance in Figure 8. We see that when $M = 5$ our altered AdaBoost occasionally outperforms scikit-learn's AdaBoost, but this improvement in performance is not significant. Thus, we can conclude that the altered AdaBoost model does not provide any improvement when predicting imbalanced data compared to the standard AdaBoost.

4 Application

4.1 March Madness

March Madness is a tournament like no other. Men's Division I College Basketball contains 363 teams from across the USA with 68 of the best teams reaching the national tournament. Teams are classified in two categories: automatic and at-large bids. There are 32 conferences nationwide and the winner of each of the conferences attains an "automatic" invitation to the tournament. The at-large bids are given to the best 36 teams that did not win their conferences championship. All conferences are not made equal, so the disparity between a team from a power conference like the Big 12 and (currently the weakest overall conference according to Ken Pomeroy [15]) the Northeast Conference is staggering. Successfully predicting March Madness entails finding the signal in the noise that can predict potential upsets by weaker teams over perennial powers. With 63 games in total, the number of possible bracket permutations is approximately 9.2 quintillion, making the tournament almost unpredictable.

4.2 March Madness Data

The data for this project comes from a plethora of sources, including ESPN (Entertainment and Sports Programming Network) [5], Fox Sports [20], College Basketball Reference [17], KenPom (Ken Pomeroy) [15] and the Associated Press [16]. The variable of interest is a binary variable that indicates whether the weaker team (underdog) won a given game, with a value of 1 if they do and -1 if they don't. The dataset contains approximately 120 variables ranging from win-loss breakdowns (home, away, neutral site, top 25) to scoring statistics (field goals, free throws, etc.) and advanced statistics (efficiencies, strength of schedule, pace).

Each row in the dataset contains data about a single game with statistics available for both the underdog and their opponent. These games were contested during the 2018 and 2019 tournaments, and each tournament had 63 games. Thus, in total the dataset has 126 rows with the underdog winning approximately 32% of the time. The testing data comes from the 2022 tournament. Table 9 shows a subset of the data for the 2019 championship game where the favourite Virginia defeated Texas Tech.

Table 9: 2019 Championship Game Sample

Upset	Underdog	Seed	Wins	Losses	Favourite	Seed	Wins	Losses
-1	Texas Tech	3	26	6	Virginia	1	29	3

4.3 Predicting the 2022 March Madness Tournament

Now that we've established which methods work well in our controlled simulation study, we can test their performance on our March Madness data. First, we'll use 10-fold cross-validation on our 2018 and 2019 training data to find the optimal M-value for each of our models. Figure 9 and 10 show how the training models performed at different M-values for both accuracy and weighted average. Table 10 shows the optimal M for each model as selected by the weighted average.

Now that we've found the optimal models we can train them on the entire training dataset and use them to make predictions on the games played in the 2022 tournament. Table 11 shows the the testing accuracy and testing weighted average for each of the nine models.

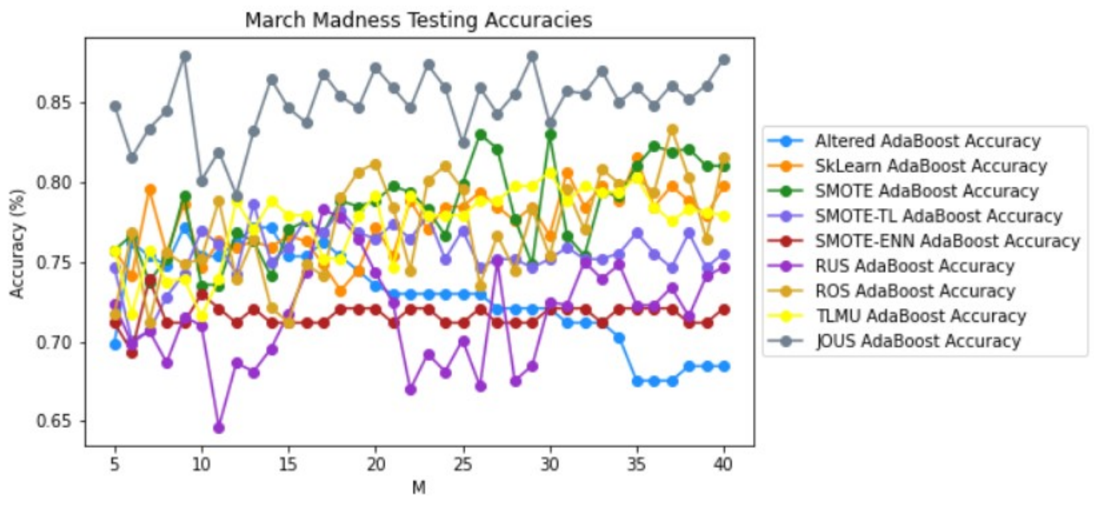

Figure 9: Average model training accuracy over the 10-folds for increasing M.

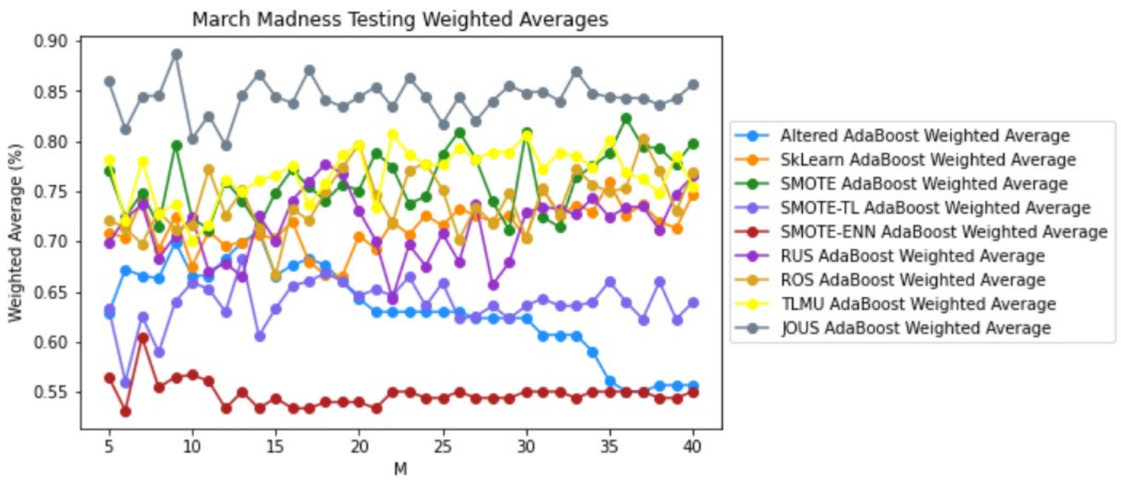

Figure 10: Average model training weighted average over the 10-folds for increasing M.

Table 10: Optimal Models Selected By Training Weighted Average

Model	Weighted Average	M
JOUS	88.70%	9
SMOTE	82.31%	36
TLMU	80.70%	22
ROS	80.34%	37
RUS	77.78%	18
AdaBoost Only	75.96%	35
Altered AdaBoost	71.01%	14
SMOTE-TL	68.35%	13
SMOTE-ENN	60.50%	7

Table 11: Accuracy & Weighted Average of Optimal Models On Testing Data

Model	Accuracy	Weighted Average
JOUS	55.56%	55.95%
SMOTE	55.56%	54.76%
TLMU	63.49%	59.52%
ROS	52.38%	52.38%
RUS	60.32%	55.95%
AdaBoost Only	57.14%	54.76%
Altered AdaBoost	52.38%	50.00%
SMOTE-TL	57.14%	55.95%
SMOTE-ENN	57.14%	47.62%

Looking at the table we see a significant decrease in the performance of all models, with models obtaining weighted averages between 12 and 33 percentage points lower than they did in training. One reason for this may be that the 2018 and 2019 tournaments are not representative of the results in the 2022 tournament and we can't prove that there is a link between underdog performance between tournaments. The notion that our training data does not model our testing data well may be relevant to this scenario, as the 2022 tournament contains 13 significant upsets (where an underdog beats a team that is ranked five seeds or higher than themselves). These 13 significant upsets tie the 2022 tournament with the 1985 tournament for the most such upsets in a single tournament all-time.

Another reason that our performance may have diminished, relative to our simulation study, is that our March Madness data does not follow anything close to a normal distribution. In our simulation study every variable was generated from a normal distribution, but real world data, like college basketball outcomes, is messy and there is a negligible likelihood that the variables each follow a normal distribution. Ultimately more research would have to be done into how our models react to data generated from different distributions and the introduction of correlated variables.

A non-statistics explanation for the poor performance on the 2022 tournament testing data is the implementation of NIL (name, image, likeness) and transfer portal between our training and testing data. The current NIL rules were implemented July 1, 2021 [11], and allow student-athletes to make money using their personal brand. Previously student-athletes were unable to monetize their talents, with the exception of scholarships, but now schools and boosters (team supporters) can pay athletes to play for the school. Table 12 shows the top 10 college basketball NIL valuations from On3 [14].

Table 12: Top 10 College Basketball NIL Valuations

Rank	Athlete	School	Valuation
1	Bronny James	Uncommitted	$7.4M
2	Hansel Emmanuel	Uncommitted	$1.4M
3	Jared McCain	Duke	$1.2M
4	Bryce James	Uncommitted	$1.2M
5	Shaqir O'Neal	Texas Southern	$1M
6	Jalen Wilson	Kansas	$926K
7	Trayce Jackson-Davis	Indiana	$897K
8	Brandon Miller	Alabama	$870K
9	Robert Dillingham	Kentucky	$854K
10	Zach Edey	Purdue	$829K

The second big change was the introduction of the one-time transfer rule in April 2021 [18]. Previously, if a student-athlete wanted to transfer schools they would be required to sit out for a season before playing for the new school, and the only transfers eligible to play immediately were graduate transfers. This rule change permits student-athletes a one-time transfer with automatic eligibility regardless of the number of years they've spent at their current school. This has created a greater level of parity in college basketball with high level recruits unhappy with playing time at top schools transferring to smaller schools and making instant impacts. The transfer portal has been a major point of discussion this year as fans, coaches and schools try to determine the best way to use it. For example, this past season Michigan State head coach Tom Izzo denounced the transfer portal and brought in no new transfers [13], while Kansas State has brought in ten transfers in the past two years [12] and didn't have a single player this year that originally committed to the school. Kansas State's team of transfers was picked to finish last in the Big 12 ahead of this season [19], while in actuality they finished third and reached the Elite Eight.

4.4 Predicting the 2023 March Madness Tournament

To supplement our predictions on the games from the 2022 tournament, we also tried to predict the 2023 tournament and submitted a bracket for each of our models into the ESPN Bracket Challenge. Table 13 shows the number of correct predictions, percentage of the 20,056,273 models that the given model was better than, and the number of points earned (points double each round, maximum 1920 points) by each model. Figure 11 depicts the performance of our best model (ROS) in bracket form.

Table 13: 2023 March Madness Model Performance

Model	Correct Predictions	Top	Points
ROS	33	95.4%	690
Personal	35	88.4%	560
Altered AdaBoost	30	49.9%	440
SMOTE-TL	29	40.5%	420
SMOTE-ENN	30	40.5%	420
AdaBoost Only	31	36.00%	410
RUS	22	23.9%	380
TLMU	27	17.5%	360
SMOTE	27	12.5%	340

5 Conclusion

In this paper we found that almost all the sampling methods that we introduced, when combined with an ensemble method like AdaBoost, outperformed AdaBoost on its own for varying levels of data imbalance in terms of weighted average. However, the success in our simulation study unfortunately did not lead to similar results when applied to March Madness data, and each of our models underperformed.

As noted our simulation study only used uncorrelated variables generated from normal distributions. In the future it would be interesting to see how our models would perform if this were to change. It would be beneficial to try to develop data that more closely follows the structure of the March Madness data. Additionally more research could be done combining our sampling methods with different machine learning models, since we only combined them with AdaBoost.

Lastly, with the new college basketball landscape created by NIL and the transfer portal, it may be beneficial to test the methods outlined in this paper using only seasons contested with these new rules. Determining whether training our models exclusively on the 2022 tournament improves performance when predicting the 2023 tournament would be a good first step. Every sport has different eras which has lead to the introduction of era adjusted statistics. College basketball seems to be entering into a new era, so taking this into account will likely be necessary for future tournaments.

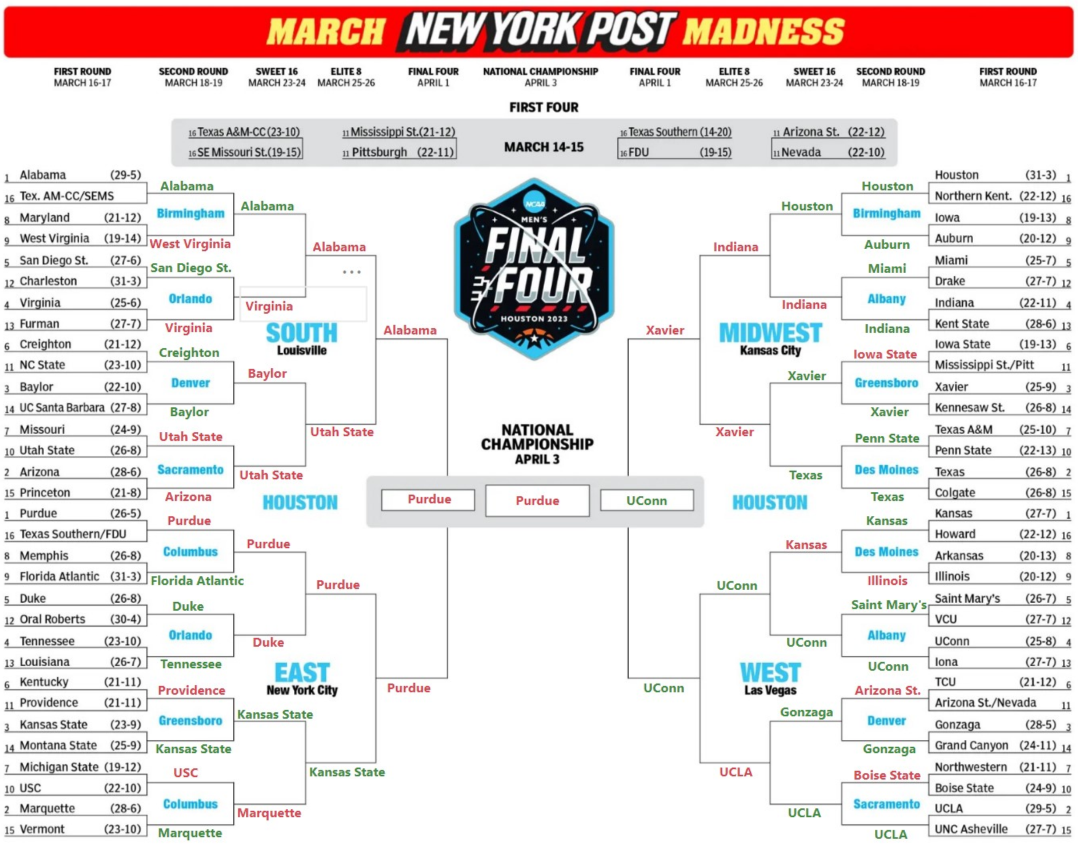

Figure 11: ROS 2023 March Madness Bracket.

6 References

[1] Gustavo E. Batista, Ronaldo C. Prati, and Maria Carolina Monard. “A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data”. In: *ACM SIGKDD Explorations Newsletter* 6.1 (2004), pp. 20–29. DOI: 10.1145/1007730.1007735.

[2] Gustavo E. A. P. A. Batista, Ana L. C. Bazzan, and Maria Carolina Monard. “Balancing Training Data for Automated Annotation of Keywords: a Case Study”. In: *Proceedings of the Second Brazilian Workshop on Bioinformatics* (2003), pp. 35–43.

[3] N. V. Chawla et al. “SMOTE: Synthetic Minority Over-Sampling Technique”. In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 321–357. DOI: 10.1613/jair.953.

[4] Jacob Cohen. “A Coefficient of Agreement for Nominal Scales”. In: *Educational and Psychological Measurement* 20.1 (1960), pp. 37–46. DOI: 10.1177/001316446002000104.

[5] Entertainment and Sports Programming Network. *2022-23 men’s College Basketball Standings*. 2023. URL: https://www.espn.com/mens-college-basketball/standings.

[6] C. Ferri, J. Hernández-Orallo, and R. Modroiu. “An Experimental Comparison of Performance Measures for Classification”. In: *Pattern Recognition Letters* 30.1 (2009), pp. 27–38. DOI: 10.1016/j.patrec.2008.08.010.

[7] Yoav Freund and Robert E Schapire. “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”. In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139. DOI: 10.1006/jcss.1997.1504.

[8] Trevor Hastie, Jerome Friedman, and Robert Tisbshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2017.

[9] Qiujie Li and Yaobin Mao. “A Review of Boosting Methods for Imbalanced Data Classification”. In: *Pattern Analysis and Applications* 17.4 (2014), pp. 679–693. DOI: 10.1007/s10044-014-0392-8.

[10] David Mease et al. “Boosted Classification Trees and Class Probability/Quantile Estimation”. In: *Journal of Machine Learning Research* (2007), pp. 409–439.

[11] Dan Murphy. *Everything you need to know about the NCAA’s Nil Debate*. Sept. 2021. URL: https://www.espn.com/college-sports/story/_/id/31086019/everything-need-know-ncaa-nil-debate.

[12] On3. *2022 Kansas State Basketball Transfer Portal*. 2023. URL: https://www.on3.com/college/kansas-state-wildcats/transfer-portal/wire/basketball/2022/.

[13] On3. *2022 Michigan State Basketball Transfer Portal*. 2023. URL: https://www.on3.com/college/michigan-state-spartans/transfer-portal/wire/basketball/2022/.

[14] On3. *On3 nil 100*. 2023. URL: https://www.on3.com/nil/rankings/player/nil-100/.

[15] Ken Pomeroy. *2023 Pomeroy College Basketball Ratings*. 2023. URL: https://kenpom.com/.

[16] Associated Press. *NCAA college basketball rankings: AP top 25 basketball poll*. 2023. URL: https://apnews.com/hub/ap-top-25-college-basketball-poll.

[17] Sports Reference. *2022-23 men’s college basketball conference standings*. 2023. URL: https://www.sports-reference.com/cbb/seasons/men/2023-standings.html.

[18] On3 Staff Report. *What is the NCAA transfer portal? everything you need to know*. Dec. 2022. URL: https://www.on3.com/transfer-portal/news/ncaa-transfer-portal-everything-you-need-to-know/.

[19] Big 12 Sports. *Baylor chosen big 12 men’s basketball preseason favorite*. Oct. 2022. URL: https://big12sports.com/news/2022/10/11/baylor-chosen-big-12-mens-basketball-preseason-favorite.aspx.

[20] Fox Sports. *NCAA college basketball news, scores, Rankings amp; Stats*. 2023. URL: https://www.foxsports.com/college-basketball.

[21] Ivan Tomek. “Two Modifications of CNN”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6.11 (1976), pp. 769–772. DOI: 10.1109/tsmc.1976.4309452.

[22] Dennis L. Wilson. “Asymptotic Properties of Nearest Neighbor Rules Using Edited Data”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-2.3 (1972), pp. 408–421. DOI: 10.1109/tsmc.1972.4309137.