COBRA 6 Group Project

A Project Report submitted

for the Course

MA691: Advanced Statistical Algorithms

by

Aadi Gupta	**(180123059)**
Shashank Goyal	**(180123042)**
Tejus Singla	**(180123061)**
Vishisht Priyadarshi	**(180123053)**

under

Dr Arabin Kumar Dey

Associate Professor

Department of Mathematics

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, INDIA

November 2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The problem at hand is to use COBRA (A combined regression strategy) for classification tasks on the imbalanced dataset.

COBRA [2] is a new method for combining several initial estimators of the regression function. We implement it from scratch and convert it to a model which can be used for the classification task. Then we compare the created classifier COBRA model with the AdaBoost classification algorithm and logistic regression. We also demonstrate the performance of the COBRA model on various imbalanced datasets using different types of undersampling algorithms.

We also present a new technique, **CobraBoost**, which utilizes the COBRA and AdaBoost with the undersampling algorithms to address the class imbalance problem.

The code for the COBRA model is present at the GitHub repository. We have also created PyPI package to use the classifier COBRA model along with the various undersampling algorithms.

In the upcoming chapters, we first explain undersampling algorithms and then the classification models. This is followed by the numerical analysis on the various datasets and our observations on the performance of the discussed techniques.

Chapter 2

Undersampling Algorithms

In the undersampling step, methods are employed to balance the distribution of classes in a dataset with a skewed distribution, by downsampling the majority class.

2.1 Retaining instances of the Majority Class in the final Training Dataset

2.1.1 Near Miss Algorithms

Undersampling methods referred to as Near Miss select examples based on their distance from minority class examples. There are three versions of Near Miss, namely Near Miss - 1, Near Miss - 2, and Near Miss - 3:

1. **Near Miss - 1:**
   Majority class examples with minimum average distance to three closest minority class examples.

2. **Near Miss - 2:**
   Majority class examples with minimum average distance to three furthest minority

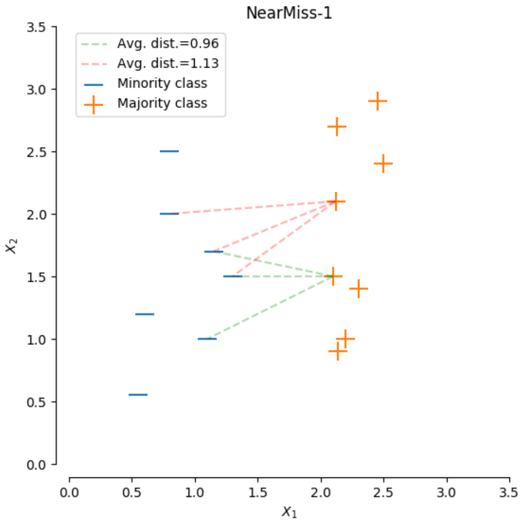

Figure 2.1: Near Miss - 1 [8]

class examples.

3. **Near Miss - 3:**

   Majority class examples with minimum distance to each minority class example.

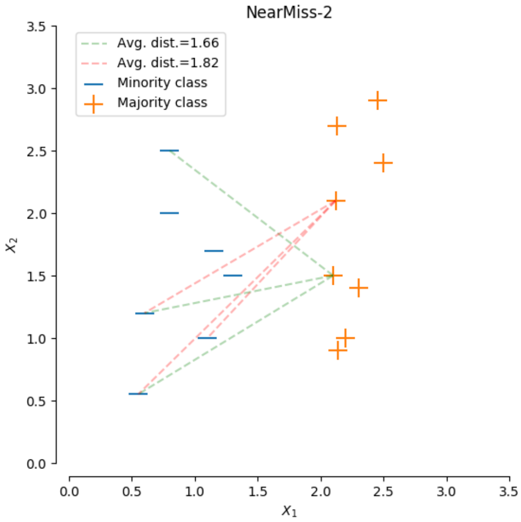

Figure 2.2: Near Miss - 2 [8]

2.1.2 Condensed KNN

As the name implies, CNN is an undersampling technique that seeks a minimal consistent set from a set of samples in which no loss in performance occurs. Using this method, the examples within the dataset are enumerated and added to the ”store” only if they cannot be classified correctly by the current contents.
The imbalanced classification method stores all samples from the minority set, while the majority set can only be classified correctly for those examples from the minority set that get incrementally added to the store.

2.2 Deleting instances of the Majority Class from the final Training Dataset

2.2.1 KNN Und

Basically, the neighborhood count is used to remove instances from the majority class in the (k Nearest Neighbor) KNN Und [1]. As described below, the KNN Und method accounts for removing instances from the majority of classes based on each instance's k nearest neighbors:

1. Obtain the k nearest neighbors for $\mathrm{x_i}$ in N

2. $\mathrm{x_i}$ will be removed if the count of its neighbor is greater or equal to t

3. The process is repeated for every majority instance of the subset N

In the parameter t, you can specify the minimum number of neighbors who are members of the P (minority) subset around the given $\mathrm{x_i}$. A training set T is removed from the training set $\mathrm{x_i}$ if this count is greater than or equal to t. The valid values of t are $1 \leq t \leq k$ and undersampling becomes more aggressive as t is reduced. The negative subset N can

also contain instances from several majority classes, so the algorithm can also be used in multiclass problems.
Unlike other methods, KNN-Und is a deterministic method, since it includes no random variables. It can be considered that KNN-Und is a simple algorithm.

2.2.2 Edited Nearest Neighbor (ENN)

When using the Edited Nearest Neighbor Rule (ENN) method, the instances of the majority class whose prediction was different from how the majority class predicted it are removed from the analysis. An instance xi in N will be removed if it has more neighbors that belong to a different class. ENNs are built according to the following steps:

1. Obtain the k nearest neighbors of $\mathrm{x_i}$ in N
2. $\mathrm{x_i}$ will be removed if the number of neighbors from another class is predominant
3. The process is repeated for every majority instance of the subset N.

Using ENN, both noisy and borderline examples are removed from the decision surface, thus producing a smoother decision surface.

2.2.3 Tomek Links

Tomek Links is one of a number of Undersampling Techniques based on Condensed Nearest Neighbors (CNN). As opposed to CNN which only includes the samples that have its k nearest neighbors from the majority class that wishes to be removed, Tomek Links applies the following rule to selects the pair of observations (say, a and b) that satisfy the following criteria:

1. The observation a's nearest neighbor is b.

2. The observation b's nearest neighbor is a.
3. Observation a and b belong to a different class. That is, a and b belong to the minority and majority class (or vice versa), respectively.

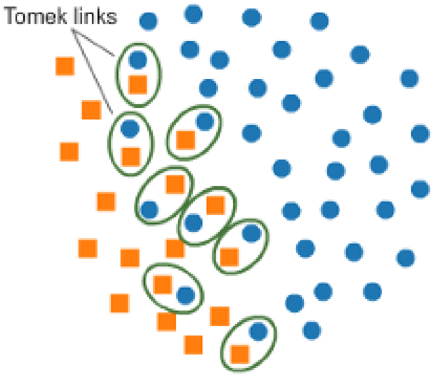

Figure 2.3: Tomek Links

In this way, the majority class data can be extracted and the minority class data can be removed by selecting the samples that have the lowest Euclidean distance (a data set representing the majority class closely associated with that of the minority class, making it ambiguous to distinct).

So in this chapter, we looked into the details of some of the undersampling algorithms that can be utilized to deal with the class imbalance problem. In the next chapters, we will look into some of the classification models and algorithms that can be utilised in our classification task concerning imbalanced data.

Chapter 3

Classification Algorithms

Based on training data, the Classification algorithm identifies the categorization of new observations. Programs use datasets or experimental observations to classify new observations into categories or groups based on the given dataset.

3.1 COBRA

A **COBRA** ensemble estimate is formed by combining several initial regression estimates. We use them as a collective indicator of the proximity between the training data and the test observation, rather than building a linear or convex optimized combination over a collection of basic estimators; $r_1, ..., r_M$.

A non-parametric and nonlinear approach to combine estimations is used in the algorithm based on a proximity criterion.
An observation, 'x', is combined with the hypothesis, $r_1, ..., r_M$, based on the predictions of the estimations $r_1, ..., r_M$ for the data. For this new observation, reliable estimation can be established if all estimators predict data values that are within a predefined margin of error, i.e., not more than ϵ away from each other. It is then calculated as an

average value based on the responses of the selected observations corresponding to this query point x. Here, we are emphasizing the fact that the average is calculated over the original results provided by the selected observations, and not over those provided by the various machines.

3.1.1 Intuition behind COBRA

Let D_n be the training sample of the model whose values are $(X_1, Y_1), ..., (X_n, Y_n)$. D_n is composed of i.i.d. random variables taking their values in $R^d \times R$, and distributed as an independent prototype pair (X,Y) satisfying $EY^2 < \infty$ (with the notation $X = (X_1, ..., X_d)$). Using the Euclidean metric as standard, R^d possesses the properties of time as well as space

With the data D_n we are aiming to consistently estimate the regression function $r^*(x) = E[Y|X = x]$, $x \epsilon R^d$. First, the original data set D_n is split into two data sequences $D_k = (X_1, Y_1), ..., (X_k, Y_k)$ and $D_l = (X_{k+1}, Y_{k+1)}, ..., (X_n, Y_n)$, with $l = n - k \geq 1$. The elements of D_l are given the names $(X_1, Y_1), ..., (X_l, Y_l)$ for ease of notation. The notation used here is slightly erroneous, because the same letter is used for D_k and D_l subsets - but since the context is clear, it should not pose any problems.

Let's say we're asked to estimate $r**$ based on a collection of $M \geq 1$ competing candidates $r_{k,1}, ..., r_{k,M}$ to estimate r^*. Based on only the first subsample D_k, these basic estimators—basic machines—are generated. Any of the researcher's favorite tools may be used, including linear regression, kernel smoothing, support vector machines, latent variables, neural networks, naive bayes, or random forests. In this paper, we consider the number of basic machines M to be a fixed number

If $r_k = (r_{k,1}, ..., r_{k,M})$ is the collection of basic machines, then T_n is the collective estimator

$$T_n(r_k(x)) = \sum_{i=1}^{l} W_{n,i}(x) Y_i, \;\; x \, \epsilon \, R^d \tag{3.1}$$

where the random weights $W_{n,i}(x)$ take the form

$$W_{n,i}(x) \;\; = \frac{\mathbf{1}_{\cap_{m=1}^{M} |r_{k,m}(x) - r_{k,m}(X_i)| \leq \epsilon_l}}{\sum\limits_{j=1}^{l} \mathbf{1}_{\cap_{m=1}^{M} |r_{k,m}(x) - r_{k,m}(X_i)| \leq \epsilon_l}} \tag{3.2}$$

0/0 (by convention) equals 0 in this definition, and ϵ_l is some positive parameter. Our regression collective uses a unique weighting scheme which is both distinctive and of little apparent significance. We see that T_n is a local averaging estimator in the following way:

Basically, it is the unweighted average of all these Y_i's such that X_i is near the query point, which is the predicted value for $r^*(x)$.

Specifically, "close" means that in the sample D_l, at each X_i point, the output generated by each basic machine is within an ϵ_l - distance of the output generated by that basic machine at that point. The corresponding outcome Y_i is included in the average when a basic machine evaluated at X_i is close to the basic machine evaluated at query point x.

3.2 AdaBoost

This algorithm uses Boosting by redistributing weights to each instance of the ensemble, with higher weights assigned to incorrectly classified instances. It is called Adaptive Boosting because the weights are re-assigned to each instance each round. By using boosts, supervised learning can reduce both bias and variance. The first decision tree is made based on the incorrectly classified record in the first model. As the first decision tree is made, it makes a number of decisions trees. The task of creating a second model involves sending only these records as input, after which we specify how many base learners we want to create. Enhancing methods work by training predictors sequentially, each attempting to correct its predecessor.

Step 1: Initialize the sample weights Each sample of AdaBoost is assigned a weight indicating its importance in terms of classification in the first step. The initial weights for all samples are identical (1 divided by the number of samples).

Step 2: Each feature of the data is put into a decision tree, and then we classify and evaluate the result. Then, we classify the data using each decision tree. After this, we examine how well each tree predicted the training samples. The next tree in the forest is the tree that performed the best at classifying the training samples.

Step 3: To determine the importance of the tree in the final classification, we use the determination formula. Once we have chosen a decision tree, we use the a formula to find the impact it has.

Step 4: In the following decision tree, weight the samples so that the errors made by the preceding decision tree are taken into consideration. Using the following formula, we increase the weights associated with the samples that the current tree incorrectly classified:

$$NewSampleWeight = SampleWeight * e^{Performance} \tag{3.3}$$

Using the same formula, we use a negative performance value for correctly classified records. This reduces the weight for records that have been correctly classified when compared to those that have been incorrectly classified. The formula goes:

$$NewSampleWeight = SampleWeight * e^{-Performance} \tag{3.4}$$

If previous stumps misclassified samples, they should be allocated larger sample weights, and samples that it correctly classified should be allocated smaller sample weights.

Step 5: Form a new dataset a random sample weight is selected between 0 and 1 for each pocket on a roulette table. After that, we generate a new dataset with the same size as the original. If a random number falls in a specific slice of the distribution, we place the sample under that slice. Due to their higher weights, the incorrectly classified samples are more likely to fall under their slice than the other samples. Therefore, misclassified samples will appear multiple times in the new dataset. If we go back to the step where we compare the predictions made by each decision tree, the tree with the highest score will have correctly classified the samples incorrectly classified by the previous one.

Step 6: As a result, you will continue to perform the steps 2 through 5 until you have selected the correct number of estimators (i.e. number of hyperparameters).

Step 7: Predicting data not included in the training set is possible using a forest of decision trees.

As part of the AdaBoost model, the trees in the forest are asked to classify the sample. Once the trees are separated in groups based on their decisions, we sum up the significance of every tree within each group. By calculating the sum of the groups, the forest as a whole makes its final classification.

3.3 Logistic Regression

With Logistic Regression, we don't directly fit a line onto the data as in linear regression. Instead, we analyze the relationship between multiple existing independent variables. In these cases, we compute a sigmoid function of X (that is a weighted sum of the input features) instead of fitting a linear regression model. We can then calculate the amount of probability that an observation belongs in one of two categories.
In mathematics, sigmoid functions are defined as follows:

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{3.5}$$

From the maximum likelihood estimation method, the logistic regression cost function is called log loss:

$$LogLoss = \frac{1}{N}\sum_{i=1}^{N} -(y_i * log(\hat{Y_i}) + (1-y_i) * log(1-\hat{Y_i})) \tag{3.6}$$

MLE's primary objective is to find the values of our parameters that maximize the likelihood function. The S-shaped line can be fitted to our data in a variety of ways when training a Logistic Regression model. Calculating the parameters of the model (the weights) can be done using an iterative optimization algorithm like Gradient Descent or we can use probabilistic methods like Maximum Likelihood. Our model is ready for some prediction once one of these methods has been used to train it.

3.4 CobraBoost

We propose a new approach, CobraBoost for dealing with the class imbalance problem. The CobraBoost algorithm combines the COBRA technique with the AdaBoost.

In the AdaBoost, instead of calling the *weak learners* for updating the weights, we utilise COBRA for this step. This means we find an ensemble estimate using the weak learners which is further utilised in the calculation of loss and in turn, weights. Further we can also use the discussed undersampling algorithms as an intermediate step to improve the performance.

The CobraBoost algorithm is a hybrid boosting algorithm which also utilizes the bagging features of the COBRA. As a part of our analysis, we found that the CobraBoost outperforms the Cobra and AdaBoost when applied to the training data set.

We provide a brief pseudo-code for training the data using the CobraBoost in Algorithm 1.

Algorithm 1 CobraBoost

```
for i = 1 to n do
    $w_i^{(1)} = \frac{1}{n}$
end for
for t = 1 to T do
    Fit weak learners using COBRA to minimise the objective function:
    $\epsilon_t = \frac{\sum_{i=1}^{n} w_i^{(t)} I(f_t(x_i) \neq y_i)}{\sum_i w_i^{(t)}}$        where $I(f_t(x_i) \neq y_i) = 1$ if $f_t(x_i) \neq y_i$ and 0 otherwise
    $\alpha_t = \frac{\epsilon_t}{1-\epsilon_t}$
    for i = 1 to n do
        $w_i^{(t+1)} = w_i^{(t)}.e^{\alpha_t I(f_t(x_i) \neq y_i)}$
    end for
end for
```

Hence, with this we conclude our discussion on the classification algorithms. In the next chapter, we will perform numerical analysis of these algorithms along with the undersampling algorithms.

Chapter 4

Results and Observations

In this chapter, we evaluate the performance of our implementation of classifier COBRA model with and without using the discussed undersampling algorithms. We also compare the performance of CobraBoost with different models including Cobra. Our implementation of Cobra classifier is inspired from the Pycobra [5]. The datasets chosen are well known and publicly available at UCI Machine Learning Repository [4].

4.1 Performance of Undersampling Algorithms

In this section, we compare the performance of the COBRA model with respect to different undersampling algorithms on various datasets. In the Table 4.1, we provide the characteristic details of these datasets:

Table 4.1: Dataset Characteristics

Dataset	**Number of majority class instances**	**Number of minority class instances**
Red Wine Quality	1500	18
Car Evaluation	1728	65
Ecoli	336	35
Abalone	4177	62
Nursery	12960	328

Table 4.2: Performance of different Undersampling Algorithms using COBRA

Dataset	**Undersampling method**	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Red Wine	Without undersampling	0.99561	0.99780	0.80556	0.89143
	Near Miss - 1	0.9975	0.99874	0.88888	0.94061
	Near Miss - 2	0.99499	0.99748	0.77777	0.87403
	Near Miss - 3	0.9975	0.99874	0.88888	0.94061
	Condensed KNN	0.98437	0.81769	0.88225	0.84874
	KNN Und	0.99311	0.746553	0.69444	0.71955
	Edited KNN	0.99437	0.99717	0.75	0.85611
Car evaluation	Without undersampling	0.97280	0.85325	0.72603	0.78452
	Near Miss - 1	0.98668	0.98259	0.83042	0.90012
	Near Miss - 2	0.98958	0.99464	0.86174	0.92343
	Near Miss - 3	0.98668	0.98487	0.82901	0.90025
	Condensed KNN	0.95138	0.75463	0.79730	0.77538
	KNN Und	0.98495	0.89108	0.91002	0.90045
	Edited KNN	0.98900	0.98422	0.86144	0.91874
Ecoli	Without undersampling	0.91369	0.64042	0.64044	0.64043
	Near Miss - 1	0.91071	0.62271	0.66488	0.64311
	Near Miss - 2	0.90476	0.75577	0.69660	0.72498
	Near Miss - 3	0.91667	0.81057	0.67466	0.73641
	Condensed KNN	0.81845	0.66033	0.75830	0.70593
	KNN Und	0.85119	0.70070	0.89088	0.78443
	Edited KNN	0.53571	0.53858	0.55196	0.54519
Abalone	Without undersampling	0.98515	0.49258	0.5	0.49626
	Near Miss - 1	0.97127	0.53050	0.58827	0.55789
	Near Miss - 2	0.61149	0.50271	0.50893	0.50580
	Near Miss - 3	0.63581	0.49615	0.41801	0.45374
	Condensed KNN	0.98156	0.49255	0.49817	0.49534
	KNN Und	0.98372	0.52393	0.50721	0.51543
	Edited KNN	0.49139	0.50592	0.51151	0.50870
Nursery	Without undersampling	0.97469	0.48735	0.5	0.49359
	Near Miss - 1	0.97469	0.61241	0.50296	0.55232
	Near Miss - 2	0.93927	0.65239	0.79958	0.71853
	Near Miss - 3	0.97469	0.48734	0.5	0.49359
	Condensed KNN	0.97561	0.68794	0.52423	0.59503
	KNN Und	0.97469	0.48734	0.5	0.49359
	Edited KNN	0.49097	0.52311	0.73145	0.60998

Observations

1. As can be observed from Table 4.2 **F1-score** for 'Red Wine', 'Car Evaluation' and 'Ecoli' datasets is fairly higher when we use various 'Undersampling Algorithms' (with COBRA as classifier) as compared to **F1-score** calculated for the same without undersampling.

2. Also undersampling does not appear to give any advantage on **F1-score** when it comes to 'Abalone' and 'Nursery' datasets.

4.2 Performance of Models

4.2.1 Without Undersampling

In this section, we evaluate and compare the performance of different models on various datasets without doing undersampling.

Table 4.3: Comparison between models on different dataset without undersampling

Dataset	Classifier	Accuracy	Precision	Recall	F1-Score
Red Wine	Logistic Regression	0.99812	0.99906	0.91667	0.956089
	AdaBoost	0.98937	0.744681	0.52778	0.61774
	Cobra	0.99561	0.99780	0.80556	0.89143
Car evaluation	Logistic Regression	0.98032	0.90619	0.79707	0.84814
	AdaBoost	0.97917	0.90059	0.78145	0.83680
	Cobra	0.97280	0.85325	0.72603	0.78452
Ecoli	Logistic Regression	0.92262	0.79732	0.77014	0.78349
	AdaBoost	0.92559	0.80638	0.78403	0.79505
	Cobra	0.91369	0.64042	0.64044	0.64043
Abalone	Logistic Regression	0.98348	0.55970	0.53092	0.54493
	AdaBoost	0.98180	0.663710	0.54596	0.59910
	Cobra	0.98515	0.49258	0.5	0.49626
Nursery	Logistic Regression	0.98919	0.90546	0.86677	0.88569
	AdaBoost	0.98781	0.90710	0.82893	0.86626
	Cobra	0.97469	0.48735	0.5	0.49359

4.2.2 With Undersampling

In this section, we evaluate and compare the performance of different models on various datasets using undersampling. Along with this, CobraBoost algorithm is evaluated and compared with different algorithms.

Table 4.4: Comparison between models on different dataset using Near miss v-3

Dataset	**Classifier**	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Red Wine	Logistic Regression	0.99812	0.96812	0.94413	0.95597
	AdaBoost	0.98812	0.49437	0.49968	0.497012
	Cobra	0.9975	0.99874	0.88888	0.94061
	CobraBoost	0.99875	0.99937	0.94444	0.97113
Car evaluation	Logistic Regression	0.97801	0.84897	0.83999	0.84445
	AdaBoost	0.98032	0.87368	0.84119	0.85713
	Cobra	0.98668	0.98487	0.82901	0.90025
	CobraBoost	0.98727	0.92024	0.92578	0.92299
Ecoli	Logistic Regression	0.91964	0.79151	0.78072	0.78608
	AdaBoost	0.92315	0.52806	0.61944	0.57012
	Cobra	0.91667	0.81057	0.67466	0.73641
	CobraBoost	0.92262	0.82029	0.71466	0.76385

We can observe from Table 4.4, ‘CobraBoost’ algorithm performs better on the datasets listed above when compared to ‘Cobra’ based on **F1-score**.

Chapter 5

Conclusion

We studied several undersampling algorithms and a new ensemble technique, COBRA. We also analysed the existing techniques like AdaBoost and RUSBoost, and utilized their concepts to propose a new technique, CobraBoost. Several numerical analysis of the different classification algorithms, including CobraBoost, were performed on well-known datasets.

From our analysis, we can finally conclude the following points which summarize our findings:

- From the observations recorded in Section 4.1, it can be deduced that various undersampling algorithms work well in the case where data is **not** highly skewed (i.e., highly imbalanced).

  Undersampling Algorithms can underperform when the classes are highly imbalanced which can be seen in the case of 'Abalone' (62 : 4177) and 'Nursery' (328 : 12960) datasets.

- From the observations recorded in Section 4.2.2, it can be inferred that Boosting Algorithms (CobraBoost) generally perform better than Bagging Algorithms (COBRA) in the case where overfitting does not occur.

Bibliography

[1] Marcelo Beckmann, Nelson Ebecken, and Beatriz Lima. A knn undersampling approach for data balancing. *Journal of Intelligent Learning Systems and Applications*, 7:104–116, 11 2015.

[2] Gérard Biau, Aurélie Fischer, Benjamin Guedj, and James D. Malley. Cobra: A combined regression strategy. *Journal of Multivariate Analysis*, 146:18–28, Apr 2016.

[3] Marcus A. Brubaker. Adaboost. https://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/AdaBoost.pdf. Accessed: 12th November, 2021.

[4] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[5] Benjamin Guedj and Bhargav Srinivasa Desikan. Pycobra: A python toolbox for ensemble learning and visualisation, 2019.

[6] Jason Brownlee. Undersampling Algorithms for Imbalanced Classification. Machine Learning Mastery. https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/. Accessed: 12th November, 2021.

[7] Ajinkya More. Survey of resampling techniques for improving classification performance in unbalanced datasets, 2016.

[8] Prashant Banerjee. Data Preprocessing Project - Imbalanced classes problem. GitHub. Accessed: 12th November, 2021.

[9] Chris Seiffert, Taghi M. Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Rusboost: Improving classification performance when training data is skewed. In *2008 19th International Conference on Pattern Recognition*, pages 1–4, 2008.