CLASSIFICATION WITH CLASS IMBALANCE PROBLEM: A REVIEW

SMAI PROJECT REPORT

Megha Bose

2019111021

Nirmal Manoj

2019111011

Samyak Jain

2019101013

Varul Srivastava

2019111015

Monsoon 2021

Contents

1 Problem Statement

Classification approaches often assume that the training set is evenly distributed over all the classes considered. However, the distribution of examples across the known classes might be biased or skewed. The distribution may vary from a slight bias to a severe imbalance where the training set for one or more classes, called the majority, might far surpass the training set of the minority class. Often, the minority class is the class we are interested in.

For instance, in a medical diagnosis of a rare disease, it is critical to identify a rare medical condition among the normal populations. Any errors in such a diagnosis could severely affect the patients' well-being. Thus, it is crucial for the classification model to achieve a higher identification rate on the rare occurrences, i.e., the minority class, in datasets. Such imbalance can also be seen in fraud detection, outlier detection, intrusion detection, conversion prediction, etc.

Multiple types of approaches have been adopted to produce decent results to handle classification with such imbalanced datasets. The selected paper studies trends and advancements in class imbalance learning and classification.

2 Handling Class Imbalance

2.1 Methods of class handling

Our project focuses on the following major approaches used to tackle class imbalance classification. We aim to implement these and compare their performance on different types of datasets. The approaches can be classified into two major categories:

1. Algorithm-level
2. Data preprocessing

Data approaches mainly revolve around variations of oversampling and undersampling.

2.2 Metrics

One of the most common metrics is Accuracy. It is defined as :

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

However, notice that if $TN + FP$ is 100 times more than $TP + FN$, then a very good classifier would be which classifies all datapoints as Negative-class. This is the problem with using accuracy as a metric with imbalanced classes.

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

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

$$F_1 = 2.\frac{precision \times recall}{precision + recall}$$

To resolve this issue we can either have weighted evaluation of TP and TN, or use other metrics like Precision, Recall and F1-score. We also represent the model performance on datasets using Confusion Metric. Other than this, we also have ROC and AUC score.

3 Theoretical Analysis

3.1 SMOTE

Earlier attempts to learn on imbalanced classes observed increased accuracy for under-sampling techniques but not so much for oversampling. Chawla et al. (2002a) proposes an oversampling-technique which involves creation of synthetic examples of minority class rather than the previously done oversampling with replacement. They also take Precision, Recall, ROC and AUC as accuracy metrics. Synthetic Minority Over-sampling Technique (SMOTE) is thus a data-preprocessing technique that involves minority-oversampling through aggregation of k-nearest data-points of a randomly selected minority class data point.

SMOTE is an over-sampling approach in which the minority class is over-sampled by creating new synthetic examples. The synthetic examples are generated by taking each minority class sample and introducing synthetic examples along the line segments joining them with k minority class nearest neighbors.

1. Neighbors from the k nearest neighbors are randomly chosen depending upon the amount of over-sampling required.
2. If the amount of over-sampling needed is 300%, three neighbors from the five nearest neighbors are chosen and one sample is generated in the direction of each.
3. Synthetic samples are generated by taking the difference between the feature vector under consideration and its nearest neighbor.
4. We multiply this difference by a random number 'ratio between 0 and 1 and add it to the feature vector.

3.2 Borderline-SMOTE

The paper Han et al. (2005) proposes two methods improving SMOTE, where it over samples only the borderline examples. The borderline examples of the minority class are more easily misclassified than those ones far from the borderline. The idea of the borderline-SMOTE methods stems from this observation. It only over-sample the borderline examples of the minority class, while SMOTE and random over-sampling augment the minority class through all the examples from the minority class or a random subset of the minority class. Experiments we conducted(described below) indicate that SMOTE-borderline methods behave better than SMOTE in most datasets. This validates the efficiency of SMOTE-borderline methods.

3.2.1 Algorithm

Unlike SMOTE, methods of borderline-SMOTE only oversample or strengthen the borderline minority examples. First, we find out the borderline minority examples; then synthetic examples are generated from them and added to the original training set.

Suppose that the whole training set is T, the minority class is P and the majority class is N, and

$$P = \{p_1, \ldots, p_{pnum}\}, N = \{n_1, n_2, \ldots, n_{nnum}\},$$

where $pnum$ and $nnum$ are the number of minority and majority examples respectively.

3.2.1.1 Borderline-SMOTE1

Step 1: For every $p_i (i = 1, 2, .., pnum)$ in the minority class P, calculate its m nearest neighbours from the whole training set T. The number of majority examples among the m nearest neighbours is denoted by $m'(0 <= m' <= m)$.

Step 2: If $m' = m$, that is, all the m nearest neighbours of p_i are majority examples, p_i is considered to be noise and is not operated in the following steps. If $m/2 <= m' < m$, namely the number of p_i's majority nearest neighbours is larger than the number of its minority ones, p_i is considered to be easily misclassified and put into a set DANGER. If $0 <= m' < m/2$, p_i is safe and does not participate in the following steps.

Step 3: The examples in DANGER are the borderline data of the minority class P. We can set:

$$DANGER = \{p'_1, p'_2, .., p'_{dnum}\},\ where\ 0 <= dnum <= pnum$$

For each example in DANGER, we calculate its k nearest neighbours from P.

Step 4: In this step, we generate $s * dnum$ synthetic positive examples from the data in DANGER, where s is an integer between 1 and k.

For each p'_i, we randomly select s nearest neighbours from it's k nearest neighbours in P. Firstly, we calculate the differences, dif_j $(j = 1, 2, .., s)$ between p'_i and its s nearest neighbours from P, then multiply dif_j by a random number r_j $(j = 1, 2, ..., s)$ between 0 and 1. Finally, s new synthetic minority examples are generated between p'_i and its nearest neighbours:

$$Synthetic_j = p'_i + r_j * dif_j,\ j = 1, 2, .., s$$

We repeat the above procedure for each p'_i in DANGER and can attain s_{dnum} synthetic examples. This step of the procedure is similar to the original SMOTE, which we have discussed earlier.

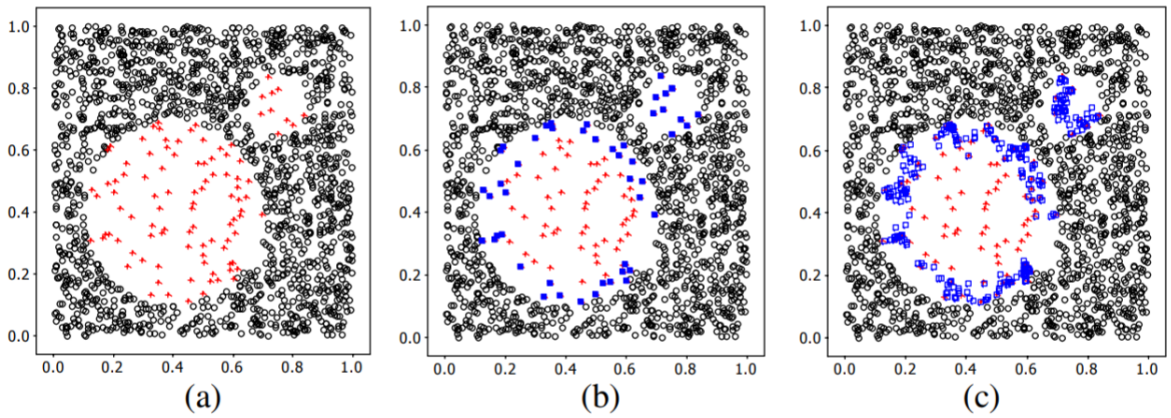
3.2.1.2 Borderline-SMOTE2

Borderline-SMOTE2 generates synthetic samples from each sample in DANGER. But along with considering the positive nearest neighbours from P, it also considers its nearest negative neighbours from N. The difference between the p'_i and its nearest negative neighbour is multiplied by a random number between 0 and 0.5. As a result, the newly generated samples are closer to the minority class.

3.2.2 Visualization

The following figure from the original paper shows a visualization of the data oversampled using this method.

Figure (a) shows the original dataset. Figure (b) shows all the borderline samples found by borderline-SMOTE. New synthetic samples are generated through those borderline examples of the minority class. The synthetic examples are shown in figure (c) with hollow squares. We can easily see that the borderline is strengthened by the algorithm.



3.3 SMOTE-IPF

Sáez et al. (2015) proposes an extension of SMOTE through an iterative ensemble-based noise filter called Iterative-Partitioning Filter (IPF), which can overcome the problems produced by noisy and borderline examples in imbalanced datasets. The experiments are carried out both on a set of synthetic datasets with different levels of noise and shapes of borderline examples as well as real-world datasets.

The paper claims that:

1. The SMOTE algorithm fulfills a dual function: it balances the class distribution and it helps to fill in the interior of sub-parts of the minority class.
2. The IPF filter removes the noisy examples originally present in the dataset and also those created by SMOTE. Besides this IPF cleans up the boundaries of the classes, making them more regular.

According to the ranking in the smote_variants repo, SMOTE-IPF is one of the top 10 best performing oversampling techniques.

3.3.1 Mid-Evaluation

For the mid-eval submission, we made the code required to oversample any binary class or multi-class dataset using SMOTE-IPF. The code for this is given in two python notebook files, in the folder code/smote_ipf in our repository. For this phase, we use two known classical datasets, Iris and Wine, to show the performance of SMOTE-IPF in binary class and multi-class scenarios. The notebook creates a scatterplot visualization showing the original dataset vs. oversampled comparison in a side-by-side manner.

3.3.2 Algorithm

3.3.2.1 Motivation

Apart from imbalance ratio between classes, there are other relevant issues that leads to the degradation of the performance of the classifiers:

1. Presence of small disjuncts: The minority class can be decomposed into many sub-clusters with very few examples in each one, surrounded by majority class examples. This is a source of difficulty for most learning algorithms in detecting enough of these sub-concepts.
2. Overlapping between classes: There are often some examples from different classes with very similar characteristics, in particular if they are located in the regions around decision boundaries between classes. These examples refer to overlapping regions of classes.

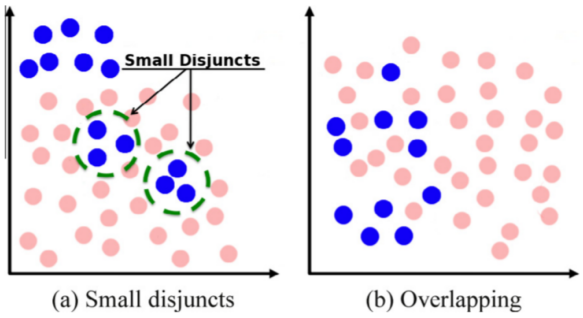


Figure 1

The examples can be split into three categories.

1. Safe examples are placed in relatively homogeneous areas with respect to the class label.
2. Borderline examples are located in the area surrounding class boundaries, where either the minority and majority classes overlap or these examples are very close to the difficult shape of the boundary – in this case, these examples are also difficult as a small amount of the attribute noise can move them to the wrong side of the decision boundary.
3. Noisy examples are individuals from one class occurring in the safe areas of the other class. Accordingly, they could be treated as examples affected by class label noise. Notice that the term noisy examples will be further used in this paper in the wider sense of, in which noisy examples are corrupted either in their attribute values or the class label.

The examples belonging to the last two categories often do not contribute to a correct class prediction. Therefore, one could ask whether removing them (all or the most difficult misclassification parts) could improve classification performance.

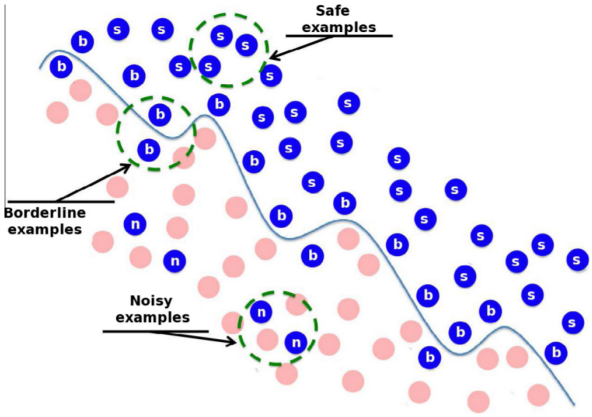


Figure 2

SMOTE-IPF uses ensemble based noise filter to achieve this goal.

3.3.2.2 Thought process behind the SMOTE-IPF algorithm

Two different generalizations of SMOTE are 'change-direction' and 'filtering-based' methods. However, 'change-direction' method has several drawbacks. However, change direction methods has several drawbacks when we are dealing with imbalanced datasets that suffer from noisy and borderline examples:

1. Noisy and borderline examples could be removed from the data to improve the final performance but 'change-direction' methods do not allow this option since their only function is to create new positive examples.
2. The creation of new positive examples, although directed towards specific parts of the input space, may be erroneous because it is based on data with noisy examples. This fact shows, once gain, the need to introduce a cleaning phase after the creation of examples.

Methods with SMOTE that use additional cleaning seem to be more appropriate to deal with the imbalanced problems with noisy and borderline examples. Such methods follow two postulates:

1. Class distribution has to be transformed to be balanced to some degree, in order to support the learning of classifiers.
2. The most difficult noisy and borderline examples should be removed from the training data since they are often the most difficult for learning.

For handling the task in the first postulate, we can use SMOTE. SMOTE can fill the interior of minority class sub-parts with synthetic minority class examples. This is a positive fact since in imbalanced datasets with a considerable quantity of borderline examples, minority class clusters are usually defined by those with an emptier interior. Nevertheless, class clusters may not be well defined in cases where some majority class examples might be invading the minority class space. The opposite can also be true, since interpolating minority class examples can expand the minority class clusters, introducing artificial minority class examples too deeply into the majority class space. This additional minority noise is also caused by the blind over-generalization of SMOTE-based techniques of looking for nearest neighbors from the minority class only. Both situations could introduce additional noise into datasets. These problems are solved using SMOTE with additional under-sampling, such as with ENN.

For handling the task in the second postulate, we need specific and more powerful methods designed to eliminate mislabeled examples when datasets have a considerable number of such examples. A group of methods that address this problem is ensemble-based noise filters. SMOTE-IPF is created as an extension of SMOTE with an ensemble based filter named IPF. IPF is supposedly responsible for removing noisy examples originally present in the dataset as well as those created by SMOTE. IPF also tries to clean up class boundaries, making them more regular.

3.3.2.3 Basic idea of the algorithm

SMOTE and IPF must be applied, one after the other to the imbalanced dataset in the correct order in order to obtain reasonable final results. IPF is designed to deal with standard classification dataset. Hence, applying IPF first carries the risk of removing all the examples from the minority class, which may be seen as noisy examples because they are underrepresented in the dataset.

In short, the crucial idea of the SMOTE-IPF is based on two claims:

1. The SMOTE algorithm fulfills a dual function: it balances the class distribution and it helps to fill in the interior of sub-parts of the minority class.
2. The IPF filter removes the noisy examples originally present in the dataset and also those created by SMOTE. Besides this, IPF cleans up the boundaries of the classes, making them more regular.

Borderline-SMOTE may over-strength the boundary zone. It could be problematic for studying data with noisy and borderline examples. Hence, replacing SMOTE with such algorithms may not be useful, although SMOTE in SMOTE-IPF can be replaced with any of the change-direction generalizations.

3.3.2.4 Algorithm in detail

Part 1: SMOTE

In our comparative study, we use SMOTEChawla et al. (2002b) with varying parameters of k (number of neighbors) and the proportion of $positive : negative$ samples. But in the original paper, both classes are balanced to 50% with $k = 5$. There are no other changes to the classical SMOTE algorithm.

Part 2: IPF Noise Filter

Noise filters are preprocessing mechanisms designed to detect and eliminate noisy examples in the training set. The result of noise elimination in preprocessing is a reduced training set which is then used as an input to a machine learning algorithm. From the various experiments conducted by the authors(of SMOTE-IPF) using different ensemble-based filters, the performance of Iterative-Partitioning Filter(IPF) has been notably good. The paper of SMOTE-IPF has an entire section dedicated to the hypothesis for why IPF shows better behavior than other filters. We will briefly touch upon the critical hypothesis later. Here, let's look at the algorithm in detail.

IPF removes noisy examples in multiple iterations until a stopping criterion is reached. The iterative process stops when, for several consecutive iterations k, the number of identified noisy examples in each of these iterations is less than a percentage p of the size of the original training dataset. Initially, the method starts with a set of noisy examples $A = \emptyset$.

The basic steps of each iteration are as follows:

1. Split the current training dataset E into n equal-sized subsets.
2. Build a classifier with the C4.5 algorithm over each of these n subsets and use them to evaluate the whole current training dataset E.
3. Add to A the noisy examples identified in E using a voting scheme (consensus or majority).
4. Remove the noisy examples: $E = E \setminus A$.

Two voting schemes can be used to identify noisy examples: consensus and majority. The former removes an example if it is misclassified by all the classifiers, whereas the latter removes an example if it is misclassified by more than half of the classifiers.

3.3.3 Why IPF?

1. Experimental results showed that IPF outperforms the rest of the methods. It particularly stands out with attribute noise datasets, which is the most common type of noise and, in this case, it is also the most disruptive one due to it affecting both classes. In contrast, class noise only affects the majority class.
2. Iterative elimination of noisy examples implies that the examples removed in one iteration do not influence detection in subsequent ones, resulting in a more accurate noise filtering.
3. Ensemble nature of IPF enables it to collect predictions from different classifiers, which may provide a better estimation of difficult, noisy examples, as opposed to collecting information from a single classifier only.

3.3.3.1 Hypothesis

The properties of IPF seem to be well adapted to the removal of noisy and borderline examples, implying an advantage over other noise filters. Most of the noise filters combined with SMOTE, such as ENN or TL, have a noise identification based on distances among examples to their nearest neighbors, considering their classes. This issue, although overlooked in the literature, maybe a vital drawback: since SMOTE is based on the distance to the nearest neighbors to create a new positive example, such synthetic examples introduced by SMOTE, although noisy, are highly likely not to be identified by filters based on distances to

the nearest neighbors. Using a noise identification method based on more complex rules, such as IPF, enables one to group larger quantities of examples with similar characteristics, although exceptions exist that will be considered to be noise, avoiding the problem as mentioned earlier and detecting noisy samples quickly.

3.4 Undersampling algorithms

Undersampling techniques remove examples from the training dataset that belong to the majority class in order to better balance the class distribution, such as reducing the skew from a 1:100 to a 1:10, 1:2, or even a 1:1 class distribution. This is different from oversampling that involves adding examples to the minority class in an effort to reduce the skew in the class distribution.

The simplest undersampling technique involves randomly selecting examples from the majority class and deleting them from the training dataset. This is referred to as random undersampling. Although simple and effective, a limitation of this technique is that examples are removed without any concern for how useful or important they might be in determining the decision boundary between the classes. This means it is possible that useful information will be deleted.

There are many different methods to choose from. These can be divided into methods that select what examples from the majority class to keep, methods that select examples to delete, and combinations of both approaches.

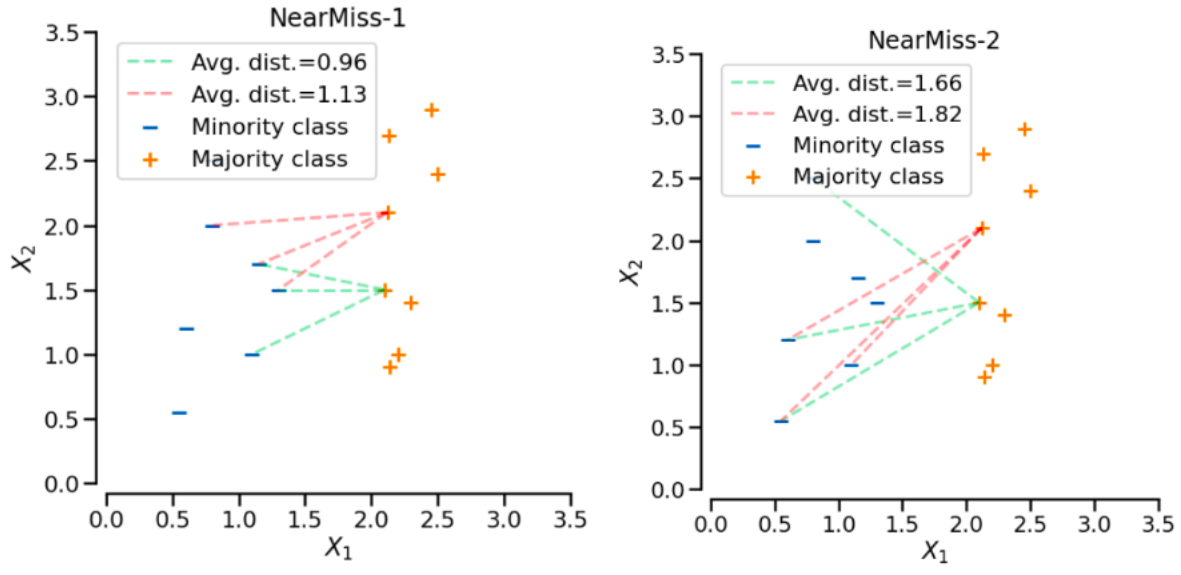
NearMiss and condensed nearest neighbor are methods that select examples to keep. Tomek links is a method that selects examples to delete. One-sided selection is Tomek links followed by Condensed Nearest Neighbor Rule, which becomes a combination of keep and delete methods.

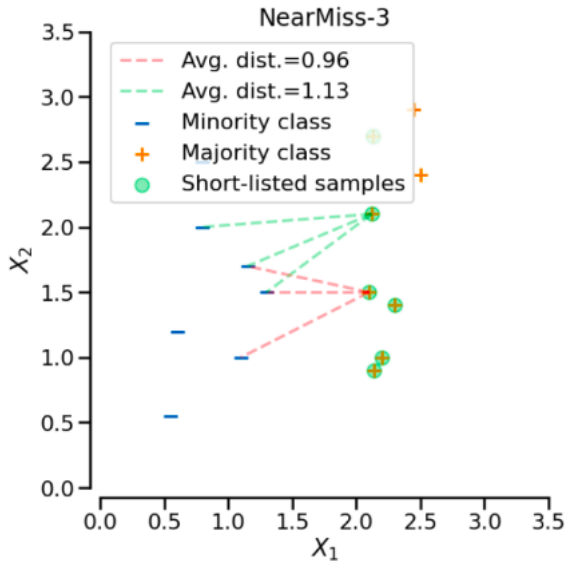
3.4.1 Near Miss

There are three versions of the technique, named NearMiss-1, NearMiss-2, and NearMiss-3.

NearMiss-1 selects examples from the majority class that have the smallest average distance to the three closest examples from the minority class. NearMiss-2 selects examples from the majority class that have the smallest average distance to the three furthest examples from the minority class. NearMiss-3 involves selecting a given number of majority class examples for each example in the minority class that are closest.

Here, distance is determined in feature space using Euclidean distance or similar.





NearMiss-3 can be divided into 2 steps. First, nearest neighbors is used to short-list samples from the majority class (i.e. correspond to the highlighted samples in the following plot). Then, the sample with the largest average distance to the k nearest neighbors is selected.

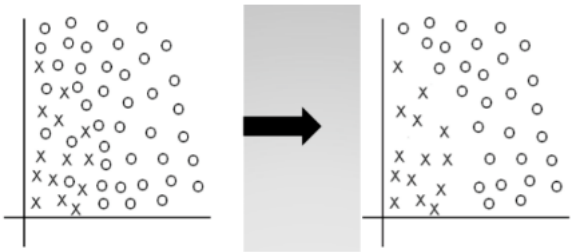
3.4.2 Tomek Links

It was proposed by by Tomek (1976).

1. Find pairs of points that are different classes, they form a Tomek link if there is no closer example to either point.
2. Remove the majority example in the pair.
3. Effectively widens the decision boundary between majority and minority.

The simplified pseudocode is shown below.

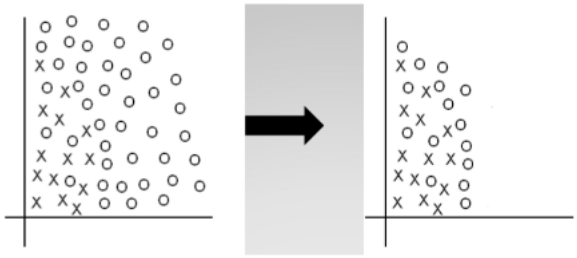
```
For each example 'a':
    For each example 'b'
        Calculate distance between examples

For each minority example 'm':
    Find the closest majority example 'n'.
    Is 'm' the closest point to 'n'?
        Remove majority example.
```



3.4.3 Condensed Nearest Neighbour

It was proposed by by Hilborn and Lainiotis (1967).

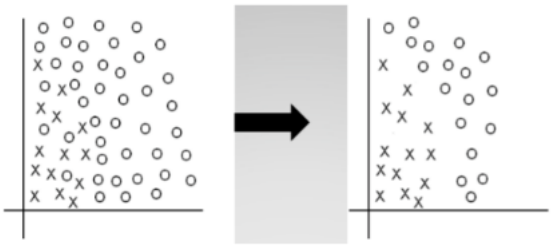
1. Removes all points, and adds them back in as required to correctly predict the examples with a kNN classification where k=1.
2. It removes majority class examples that are distant from the decision border.



3.4.4 One-sided selection

It was proposed by by Kubát and Matwin (1997).

The following illustration shows how the undersampling works in one-sided selection. The algorithm is a combination of Tomek links and condensed nearest neighbor.



3.5 Balanced Random Forest - Ensemble

3.5.1 Standard Random Forest

Like bagging, random forest involves selecting bootstrap samples from the training dataset and fitting a decision tree on each. The main difference is that all features (variables or columns) are not used; instead, a small, randomly selected subset of features is chosen for each bootstrap sample. This has the effect of de-correlating the decision trees (making them more independent), and in turn, improving the ensemble prediction.

Random forest is very effective on a wide range of problems, but the performance of the standard algorithm is not great on imbalanced classification problems.

3.5.2 Random Forest With Class Weighting

A simple technique for modifying a decision tree for imbalanced classification is to change the weight that each class has when calculating the "impurity" score of a chosen split point. Impurity measures how mixed the groups of samples are for a given split in the training dataset and is typically measured with Gini or entropy. The calculation can be biased so that a mixture in favor of the minority class is favored, allowing

some false positives for the majority class. This modification of random forest is referred to as Weighted Random Forest.

3.5.3 Random Forest With Class Weighting in Bootstrap samples

Given that each decision tree is constructed from a bootstrap sample (e.g. random selection with replacement), the class distribution in the data sample will be different for each tree. Class weighting can be changed on the basis of class distribution in each bootstrap sample, instead of the entire training dataset. This can be achieved by setting the class_weight argument to the value 'balanced_subsample'.

3.5.4 Random Forest With Random Undersampling

It was proposed by by Chen (2004). Another useful modification to the random forest is to perform data resampling on the bootstrap sample in order to explicitly change the class distribution. The BalancedRandomForestClassifier class from the imbalanced-learn library implements this and performs random undersampling of the majority class in each bootstrap sample.

3.6 SMOTEBoost

SMOTEBoost algorithm combines SMOTE and the standard boosting procedure. It lets us utilize SMOTE for improving the prediction of the minority classes and utilizes boosting to not sacrifice accuracy over the entire data set.

The goal is to improve the number of true positives. The standard boosting procedure gives equal weights to all misclassified examples and since boosting algorithm samples from a pool of data that predominantly consists of the majority class, samplings of the training set may be skewed towards the majority class for imbalanced datasets.

Boosting has a very strong learning bias towards the majority class cases in a skewed data set, and subsequent iterations of boosting can lead to a broader sampling from the majority class. To reduce this inherent bias in the learning procedure due to the class imbalance, and **increase the sampling weights for the minority** class, we introduce **SMOTE in each round of boosting**. This enables each learner to be able to sample more minority class cases, and also learn better and broader decision regions for the minority class.

3.7 RUSBoost

3.7.1 Problems with SMOTEBoost

SMOTEBoost has two major drawbacks, complexity and increased model training time. RUSBoost provides a faster and simpler alternative for learning from imbalanced data with performance that is usually as good (and often better) than that of SMOTEBoost.

3.7.2 RUSBoost Algorithm

RUSBoost Seiffert et al. (2010) is an algorithm that combines data-sampling with boosting, used to learn from skewed training data. RUSBoost is made with a data-sampling technique called **RUS**, which is basically random undersampling of the majority class, and it is combined with **AdaBoost** boosting algorithm. We give a brief overview of the RuS algorithm Below.

1. Load dataset and identify the minority and majority class;
2. Calculate the number of instances to be removed based on the percentage of under-sampling;
3. Identify a random instance from majority class and remove it from the majority class;
4. Repeat step 3 till the number of instances removed as per the given percentage.

The RuS algorithm when combined with Boosting via the AdaBoost algorithm gives us the RUSBoost algorithm, which looks as follows.

1 We have our data-set $S = \{(x_1, y_1), (x_2, y_2), ...(x_m, y_m)\}$, and the minority class as $y^r \in Y, |Y| = 2$. Further, we are using *Classifier* and have percentage oversample as N.

2 Initialize weight as $D_t(i) = \frac{1}{m}$

3 Repeat for $t = 1, 2, 3, ...T$

   a Create $S_t^{'}, D_t^{'}$ from (**RUS**) algorithm (described above).

   b get hypothesis $h_t()$ from *Classifier* by passing it $S_t^{'}, D_t^{'}$

   c Calculate **pseudo-loss** as $\epsilon_t = \sum_{(i,y):y_i \neq y} D_t(i)(1 - ht(x_i, yi) + h_t(x_i, y)$

   d Calculate weight-update parameter $\alpha_t = \frac{\epsilon_t}{1-\epsilon_t}$

   e Update $D_{t+1}(i) = D_t(i)\alpha^{\frac{1}{2}(1+h_t(x_i,y_i)-h_t(x_i,y:y\neq_i}$ and Normalize $D_{t+1}(i)$

4 Final Hypothesis is $H(x) = \arg\max y \in Y \sum_{t=1}^{T} h_t(x, y) log(\frac{1}{\alpha_t})$

4 Implementation & Results

4.1 Undersampling Algorithms

Dataset	Number of samples	Number of features	Class Ratio
UCI SatImage	6435	36	9.3:1
UCI US Crime	1994	100	12:1
UCI Oil	937	49	22:1
Synthetic	10000	2	7:3

For Near Miss algorithm, number of neighbours is defined as 3 for all three versions. For condensed nearest neighbour and one-sided selection algorithm, number of neighbours is taken as 1. Decision tree and logistic regression are used as classification algorithms, for all the datasets and undersampling methods. Two classification algorithms are used so that the independent effect of undersampling algorithms can be studied.

For decision tree, geometric mean of specificity and sensitivity is used as evaluation metric. For logistic regression, AUC score is calculated.

Dataset	No algo	NearMiss-1	NearMiss-2	NearMiss-3	Tomek Links	Condensed Nearest Neighbour	One Sided selection
UCI SatImage	0.770	0.778	0.800	0.613	0.677	0.773	0.684
UCI US Crime	0.925	0.790	0.888	0.697	0.768	0.932	0.858
UCI Oil	0.877	0.873	0.884	0.947	0.908	0.923	0.892
Synthetic	0.958	0.914	0.919	0.826	0.836	0.969	0.905

Table : Logistic regression used, AUC score

Dataset	No algo	NearMiss-1	NearMiss-2	NearMiss-3	Tomek Links	Condensed Nearest Neighbour	One Sided selection
UCI SatImage	0.69	0.78	0.79	0.65	0.71	0.74	0.71
UCI US Crime	0.61	0.71	0.85	0.55	0.66	0.68	0.64
UCI Oil	0.54	0.85	0.81	0.76	0.89	0.62	0.73
Synthetic	0.86	0.78	0.83	0.60	0.56	0.91	0.80

Table : Decision tree used, Geometric mean score

For multiclass classification, iris dataset is loaded. It has 4 features, 3 classes and 150 samples. It is then sampled in the ratio of 2:2:1. Geometric mean score is used as evaluation metric.

No algo	NearMiss-1	NearMiss-2	NearMiss-3	Tomek Links	Condensed Nearest Neighbour	One Sided selection
0.93	0.89	0.89	0.80	0.76	0.93	0.93

Table : Multiclass classification

4.2 Balanced Random Forest - Ensemble

These 4 algorithms are tested on 3 real-world and 1 synthetic dataset. The number of estimators for each of these models is kept as 10. Repeated stratified K-folds is used with 10 splits and 3 repeats. The evaluation metric used is the mean AUC score.

On all 4 datasets, balanced random forest performs best and there is considerable jump in performance from standard random forest to balanced random forest.

Dataset	Number of samples	Number of features	Class Ratio
UCI SatImage	6435	36	9.3:1
UCI US Crime	1994	100	12:1
UCI Oil	937	49	22:1
Synthetic	10000	2	99:1

Dataset	Standard Random Forest	Random Forest With Class Weighting	Random Forest With Class Weighting in Bootstrap samples	Random Forest With Random Undersampling
UCI SatImage	0.930	0.931	0.932	0.940
UCI US Crime	0.875	0.856	0.858	0.899
UCI Oil	0.864	0.866	0.873	0.899
Synthetic	0.871	0.876	0.878	0.964

4.3 SMOTE with KNN classifier

We have considered the real-world datasets of PIMA and the Credit Fraud for analysis.

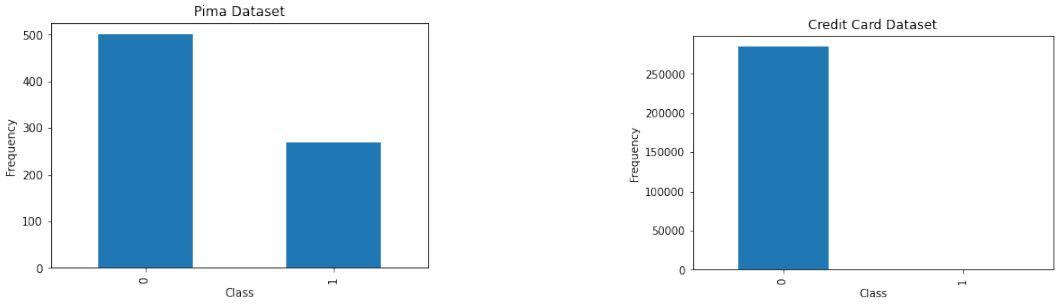
4.3.0.1 PIMA Dataset In the medical domain, most diseases occur rarely. This makes disease classification datasets imbalanced with the minority class being of interest. We explored one such problem using the PIMA Diabetes dataset, to predict if a patient is diabetec or not.

The PIMA dataset consists of several medical predictor variables and one target variable, Outcome. Predictor variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on. We can see there is an imbalance in the dataset, which would lead to erroneous models.

Diabetic	Non-Diabetic
268	500

4.3.0.2 Credit Fraud Another place where smote is applicable is credit fraud detection. In this case also the dataset is skewed, but the degree of skewness is extreme. 99.83% of the dataset belongs to the Majority (Negative or non-fraudulent class) and only 0.17% belongs to the Minority (positive or fraudulent class).

Fraudulent	Non-Fraudulent
492	284,807

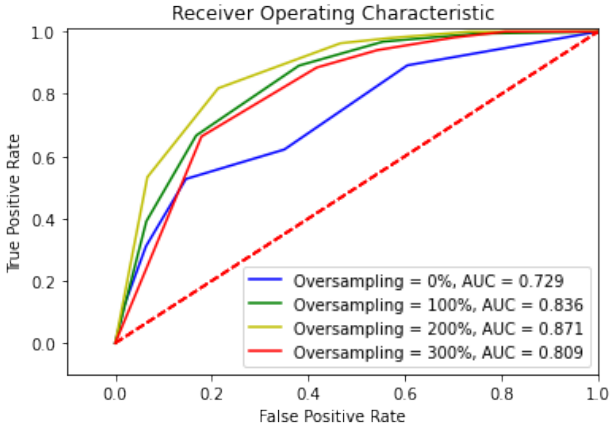


(a) PIMA Imbalance (b) Credit Fraud Imbalance

Figure 3: Imbalances on Real-World Datasets

4.3.0.3 Synthetic Datasets Synthetic datasets have also been considered to analyse the performance of the approaches. These datasets vary by the number of samples, number of features, and number of samples in the majority and minority classes (imbalance ratio).

4.3.0.4 Results on PIMA dataset By varying the oversampling percentage on the PIMA dataset, we got the results as shown in 4.



(a)

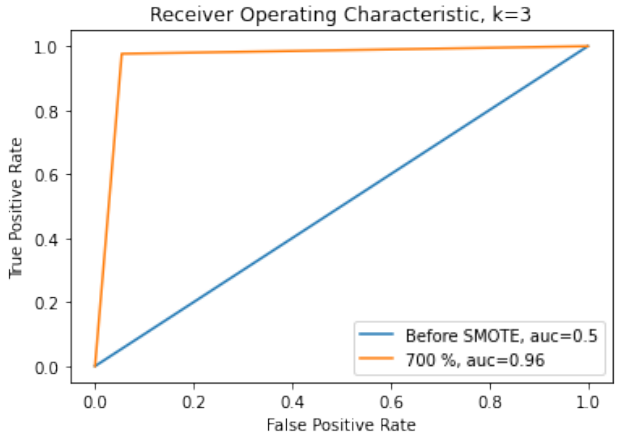
```
Over-sampling %:  0
Precision:  0.6290322580645161
Recall:  0.527027027027027

Over-sampling %:  100
Precision:  0.6945812807881774
Recall:  0.9038461538461539

Over-sampling %:  200
Precision:  0.7751677852348994
Recall:  0.9545454545454546

Over-sampling %:  300
Precision:  0.7933673469387755
Recall:  0.9688473520249221
```

(b)

Figure 4: Precision, Recall, ROC curves for different % of over-sampling using SMOTE on PIMA

We see that as the over-sampling % increases, the AUC value increases and then slightly decreases. Without any over-sampling, due to class imbalance, we had AUC of 0.729 and it rose to 0.836 already by just doubling the number of samples (100% over-sampling) using SMOTE technique. The precision and recall values show continuous improvement with increase in over-sampling.

4.3.0.5 Results on Credit Fraud dataset By varying the oversampling percentage on the Credit Fraud dataset, we got the results as shown in 5.

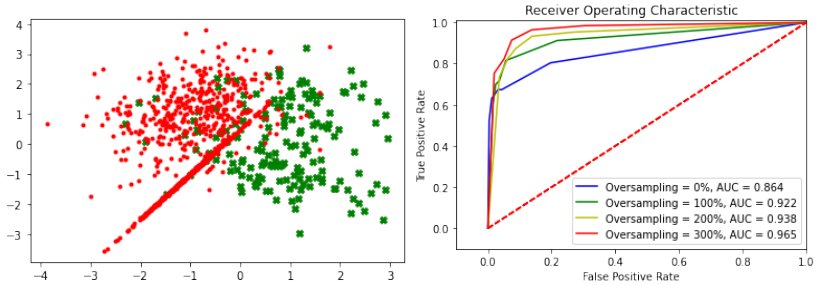



```
Precision :  0.9480461370189918
Recall :  0.9763452972198045
```

Figure 5: Precision, Recall, ROC curves for 700% over-sampling using SMOTE on Credit Card Fraud Dataset

Due to the high class imbalance in the credit card fraud dataset, the AUC value was just 0.5 before over-sampling and the recall and precision values were 0.0. On over-sampling by 700% using SMOTE technique, we see a massive increase in AUC score to 0.96 and the precision and recall values rise to 0.95 and 0.98 respectively.

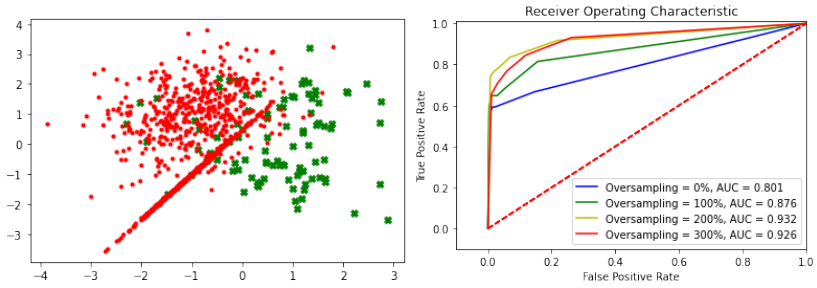
4.3.0.6 Results on Synthetic Datasets By varying imabance ratio and oversampling percentage of synthetic dataset with 1000 samples and 4 features, we got the results as shown below.




```
Over-sampling %:  0
Precision:  0.7948717948717948
Recall:  0.6739130434782609

Over-sampling %:  100
Precision:  0.875
Recall:  0.7475728155339806

Over-sampling %:  200
Precision:  0.86
Recall:  0.8543046357615894

Over-sampling %:  300
Precision:  0.9030612244897959
Recall:  0.8894472361809045
```

(a) (b) (c)

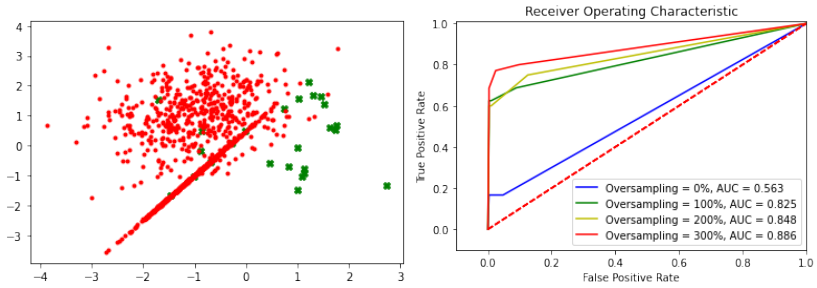
Figure 6: SMOTE on Synthetic Dataset with IR = 5




```
Over-sampling %:  0
Precision:  0.8
Recall:  0.5925925925925926

Over-sampling %:  100
Precision:  0.8181818181818182
Recall:  0.6666666666666666

Over-sampling %:  200
Precision:  0.8607594936708861
Recall:  0.8

Over-sampling %:  300
Precision:  0.875
Recall:  0.7844827586206896
```

(a) (b) (c)

Figure 7: SMOTE on Synthetic Dataset with IR = 10




```
Over-sampling %:  0
Precision:  0.5
Recall:  0.16666666666666666

Over-sampling %:  100
Precision:  0.8333333333333334
Recall:  0.625

Over-sampling %:  200
Precision:  0.7894736842105263
Recall:  0.75

Over-sampling %:  300
Precision:  0.9230769230769231
Recall:  0.6857142857142857
```

(a) (b) (c)

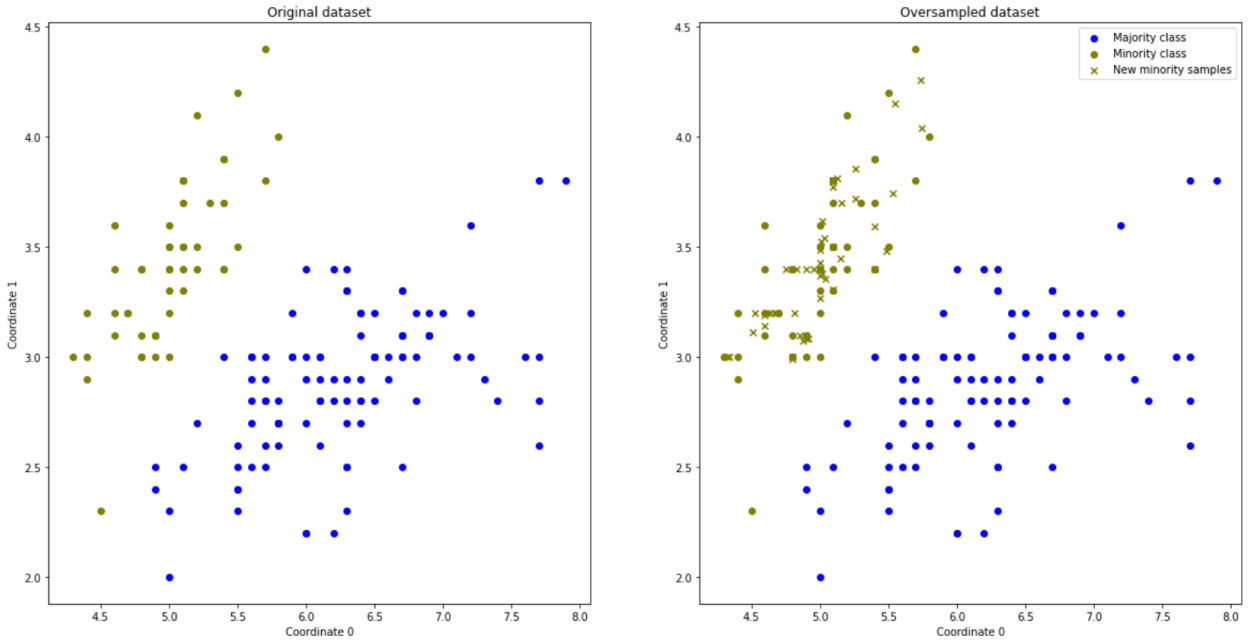
Figure 8: SMOTE on Synthetic Dataset with IR = 50

Before applying SMOTE, as expected, we see lower AUC scores, precision and recall with increasing Imbalance Ratio (IR). For a fixed IR, we see that as we increase the over-sampling percentage, there is a general increase in the values of AUC, precision and recall.

The above results show that SMOTE technique on imbalanced datasets can improve the classification performance. More over-sampling leads to an increase in performance till a point. Too much over-sampling can lead to over-fitting due to too many similar copies of the minority samples.

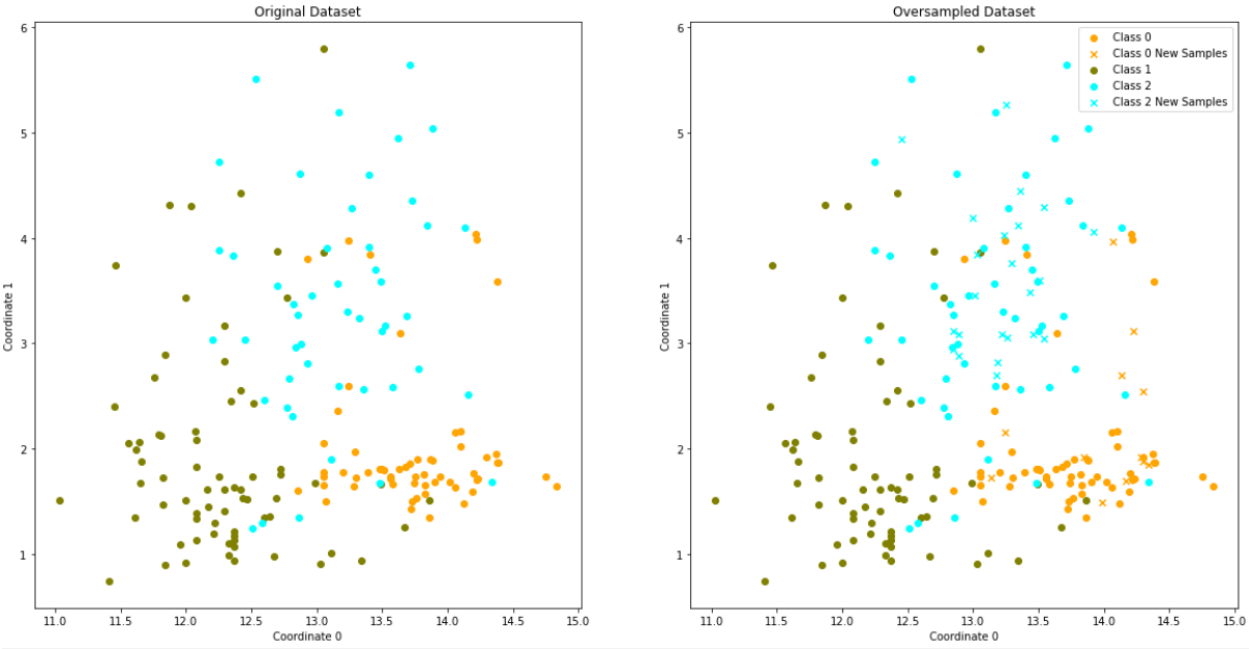
4.4 SMOTE-IPF

For the mid-evaluation, we have visualized the performance of SMOTE-IPF on a smaller dataset in binary class and multi-class scenarios. Code is available in the repo. The following figures show the improvement by SMOTE-IPF. For binary classification, we used the iris dataset. For multiclass scenario, we used the wine dataset.



(a)

Figure 9: Binary dataset iris with SMOTE-IPF



(a)

Figure 10: Multiclass dataset wine with SMOTE-IPF

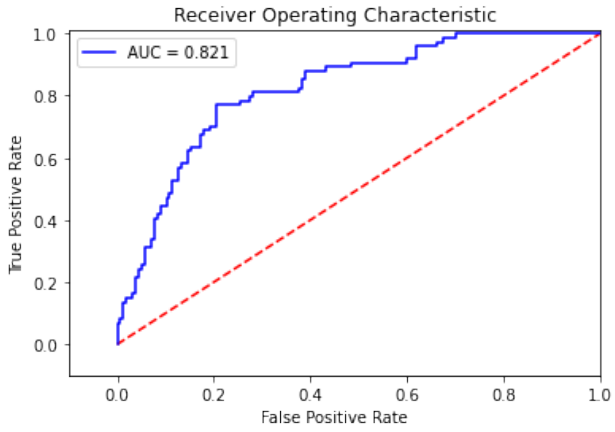
4.5 Comparative studye of Borderline-SMOTE and SMOTE-IPF

4.6 SMOTEBoost

Datasets used are same as those taken in SMOTE analysis.

4.6.0.1 Results on PIMA dataset

On the PIMA dataset, we got the results as shown in 11.



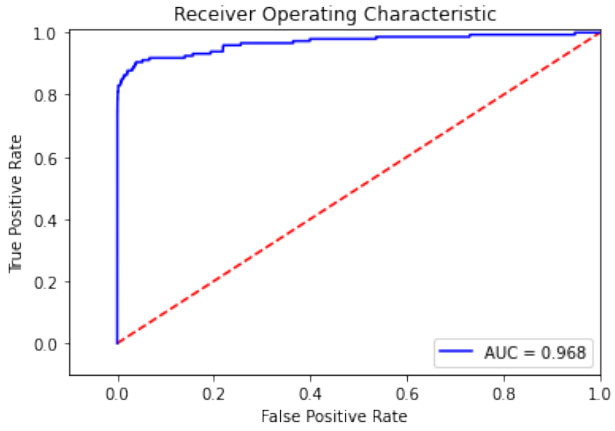

```
Precision:  0.6619718309859155
Recall:  0.6351351351351351
```

Figure 11: Precision, Recall, ROC curve on PIMA for SMOTEBoost

For the PIMA dataset, we see improvement in precision, recall and AUC values from 0.629, 0.527 and 0.729 respectively for KNN with no over-sampling to precision, recall and AUC values of 0.661, 0.635 and 0.821 respectively for SMOTEBoost.

4.6.0.2 Results on Credit Fraud dataset

On the Credit Fraud dataset, we got the results as shown in 5.

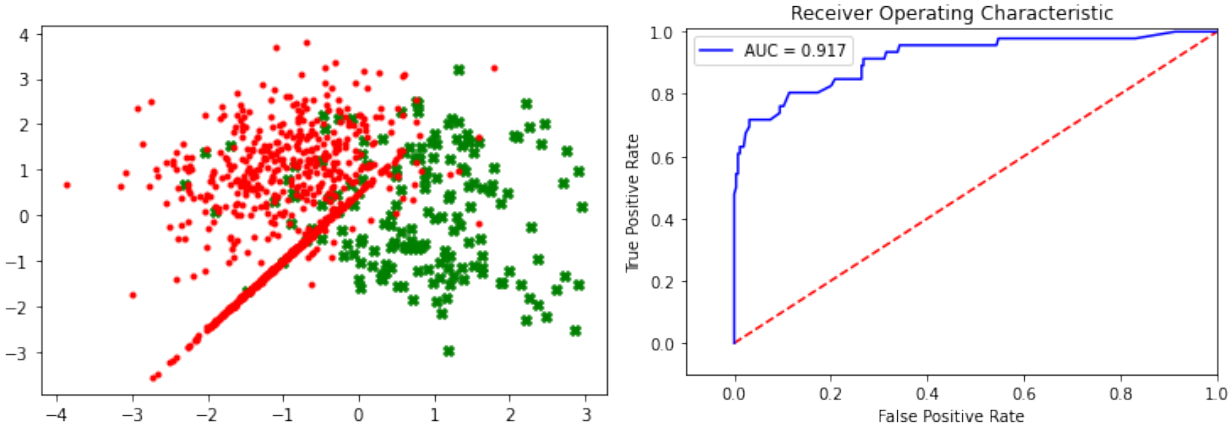



```
Precision:  0.0711484593837535
Recall:  0.8639455782312925
```

Figure 12: Precision, Recall, ROC curve on Credit Card Fraud Dataset for SMOTEBoost

For the Credit Card Fraud dataset, we see massive improvement in precision, recall and AUC values from 0.0, 0.0 and 0.5 respectively for KNN with no over-sampling to precision, recall and AUC values of 0.0711, 0.863 and 0.968 respectively for SMOTEBoost.

4.6.0.3 Results on Synthetic Datasets By varying imabance ratio and oversampling percentage of synthetic dataset with 1000 samples and 4 features, we got the results as shown below.

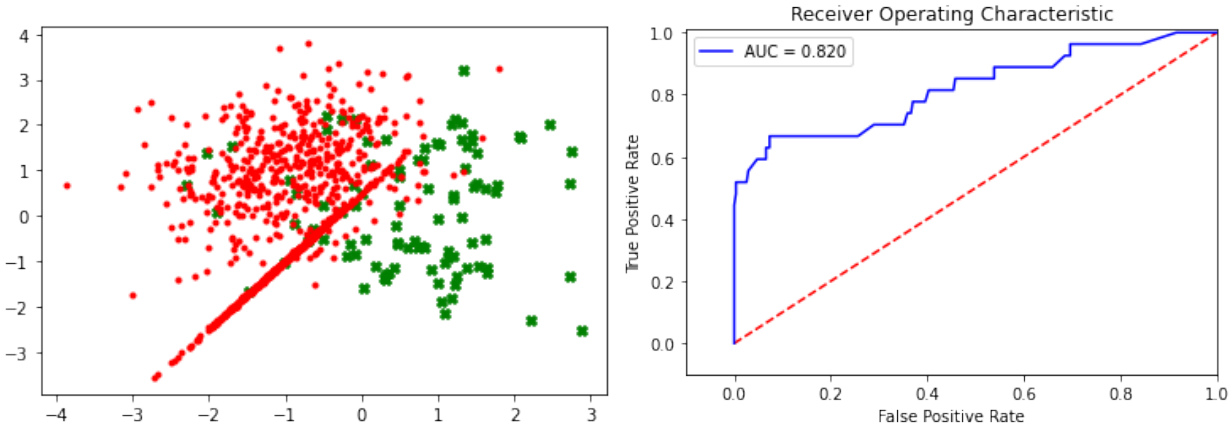


(a) (b)

```
Precision:  0.6875
Recall:  0.717391304347826
```

(c)

Figure 13: SMOTEBoost on Synthetic Dataset with IR = 5

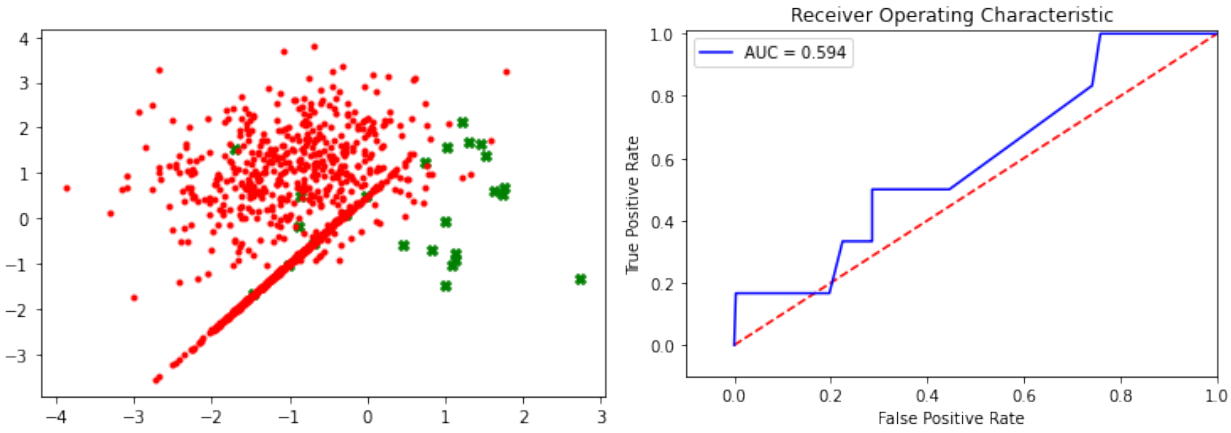


(a) (b)

```
Precision:  0.4857142857142857
Recall:  0.6296296296296297
```

(c)

Figure 14: SMOTEBoost on Synthetic Dataset with IR = 10



(a) (b)

```
Precision:  0.1
Recall:  0.16666666666666666
```

(c)

Figure 15: SMOTEBoost on Synthetic Dataset with IR = 50

Imbalance Ratio	Recall	Precision	AUC
5:1	0.673	0.794	0.864
10:1	0.592	0.800	0.801
50:1	0.166	0.500	0.563

Table 1: Performance with KNN and no over-sampling

Imbalance Ratio	Recall	Precision	AUC
5:1	0.717	0.687	0.917
10:1	0.629	0.485	0.820
50:1	0.166	0.100	0.594

Table 2: SMOTEBoost Performance

We see that in general SMOTEBoost worked better for lower values of imbalance than high values of imbalance ratios on the synthetic datasets.

4.7 RUSBoost

4.7.0.1 Synthetic Dataset We varied the imbalance ratio of the dataset and tried to observe the AUC score, precision, recall and the F1-score for the synthetic datasets. The expected trend is that accuracy and the other metrics like F1-Score, AUC, precision recall etc. are negatively correlated to the imbalance ratio (ratio between majority and minority classes).

Imbalance Ratio	Recall	Precision	F1-Score	AUC-Score
98:2	0.813	0.087	0.877	0.763
96:4	0.856	0.176	0.892	0.830
92:8	0.867	0.404	0.889	0.865
84:16	0.896	0.620	0.901	0.889
68:32	0.909	0.829	0.899	0.893

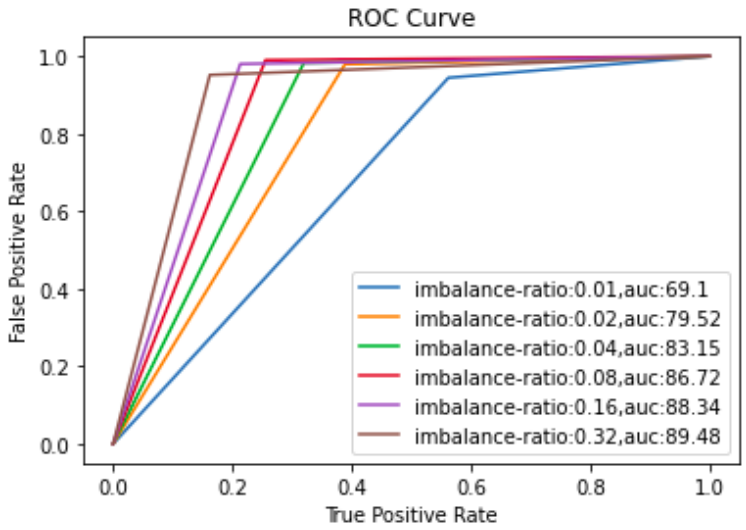
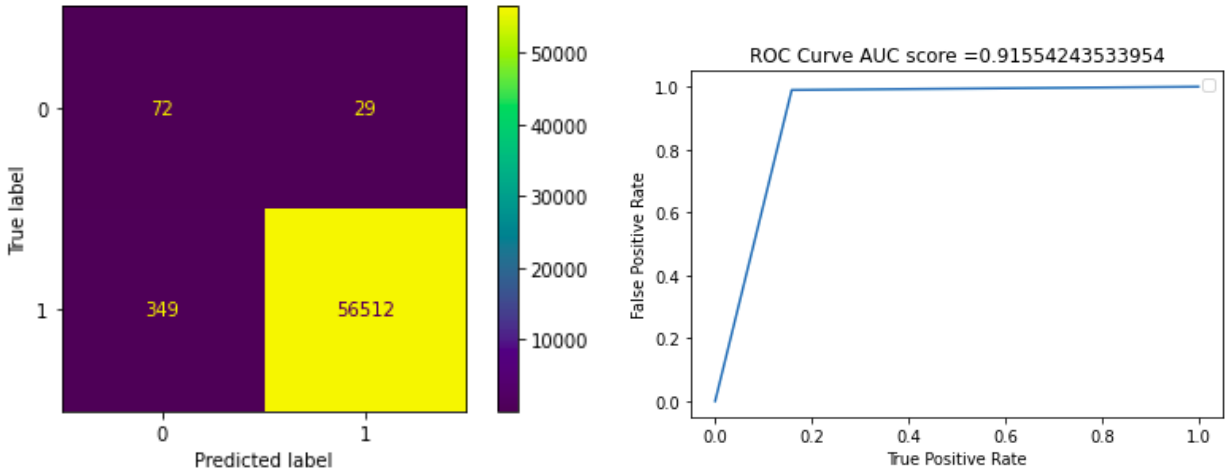


Figure 16: Confusion Matrix, ROC Curve for RusBoost applied on Credit database

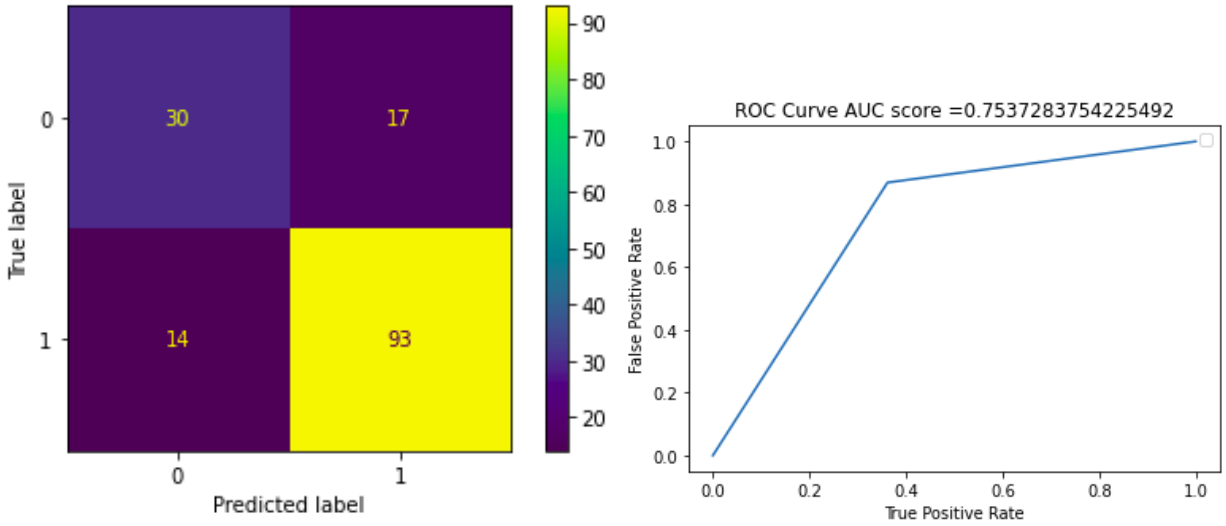
4.7.0.2 Real-World Datasets In the real-world datasets we used the PIMA-Dataset (Smith (1988)) and the Credit-fraud dataset(A. Dal Pozzolo and Bontempi (2017)). Details of the datasets has been explained under the 4.3.0.1 and 4.3.0.2 sections. We'd move to results and their analysis.



(a) (b)

Figure 17: Confusion Matrix, ROC Curve for RusBoost applied on Credit database

As we can see, there the AUC score is around 92%. Further, the F1-score is 99.78% For the PIMA Database, the AUC Score is around 76% and the 79.68%.



(a) Confusion Matrix PIMA

(b) ROC Curve PIMA

Figure 18: Confusion Matrix, ROC Curve for RusBoost applied on PIMA Database

5 Comparison

5.1 RusBoost vs SMOTEBoost

The RusBoost is an random undersampling pre-processing combined with AdaBoost classifier. SMOTEBoost is SMOTE oversampling pre-processing combined with AdaBoost. Beside being faster and simpler to implement than SmoteBoost, it also provides better generalization because of it's simplicity and thus often has better Accuracy on Datasets. However, one drawback of RusBoost against Smoteboost is that for datasets with higher entropy and high imbalance, the random under-sampling method often removes so many data points that we lose some properties and patterns.

Classifier	Imbalance Ratio	F1-Score	AUC-Score
RusBoost	98:2	0.877	0.763
SMOTEBoost	98:2	0.911	0.761

WORK DISTRIBUTION

This Project was a collaborative effort from Nirmal, Samyak, Megha and Varul. A mild division of the work is provided :

1 Report : Nirmal, Samyak, Megha, Varul

2 Smote : Megha, Varul
In this, the algorithm was implemented from scratch. Further, simulations are run on Synthetic dataset as well as PIMA and Credit-fraud dataset. A python script was written to test accuracy, f1-score etc. based on the hyper parameters which are tunable by passing command line arguments.

3 Smote-ipf : Nirmal

4 SmoteBoost : Megha, Varul
In this section, we explored SmoteBoost algorithm. We implemented it using AdaBoost ensemble method and SMOTE data-preprocessor. We experimented on Synthetic dataset, PIMA and Credit-fraud dataset. A python script was written to test accuracy, f1-score etc. based on the hyper parameters which are tunable by passing command line arguments.

5 Balanced Random Forest : Samyak

6 RusBoost : Varul, Megha
In this section, we explored RusBoost algorithm. We implemented it using AdaBoost ensemble method and RuS data-preprocessor. We experimented on Synthetic dataset, PIMA and Credit-fraud dataset. A python script was written to test accuracy, f1-score etc. based on the hyper parameters which are tunable by passing command line arguments.

References

Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002a.

Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing*, pages 878–887. Springer, 2005.

José A Sáez, Julián Luengo, Jerzy Stefanowski, and Francisco Herrera. Smote–ipf: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering. *Information Sciences*, 291:184–203, 2015.

N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, Jun 2002b. ISSN 1076-9757. doi:10.1613/jair.953. URL http://dx.doi.org/10.1613/jair.953.

Ivan Tomek. Two modifications of cnn. 1976.

Charles G. Hilborn and Demetrios G. Lainiotis. The condensed nearest neighbor rule. 1967.

Miroslav Kubát and Stan Matwin. Addressing the curse of imbalanced training sets: One-sided selection. In *ICML*, 1997.

C. Chen. Using random forest to learn imbalanced data. 2004.

Chris Seiffert, Taghi Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Rusboost: A hybrid approach to alleviating class imbalance. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 40:185 – 197, 02 2010. doi:10.1109/TSMCA.2009.2029559.

Jack W. et al. Smith. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, page 261—265, 1988.

O. Caelen C. Alippi A. Dal Pozzolo, G. Boracchi and G. Bontempi. Credit card fraud detection: a realistic modeling and a novel learning strategy. *IEEE transactions on neural networks and learning systems,*, 29: 3784–3797, 2017.