Team 58

CLASSIFICATION WITH CLASS IMBALANCE PROBLEM

Swetha Vipparla
Ayush Agrawal
Shubh Agarwal
Rohan Madineni

I. INTRODUCTION

Any real-world dataset may come along with several problems. The problem of the imbalanced class is one of them. The problem of imbalanced classes arises when one set of classes dominates over another set of classes. The former is called the majority class while the latter is called the minority class. It causes the machine learning model to be more biased towards the majority class. It causes poor classification of minority classes. Hence, this problem throws the question of "accuracy" out of the question. This is a very common problem in machine learning where we have datasets with a disproportionate ratio of observations in each class.

Imbalanced classes problem is one of the major problems in the field of data science and machine learning. It is very important that we should properly deal with this problem and develop our machine learning model accordingly. If this is not done, then we may end up with higher accuracy. But this higher accuracy is meaningless because it comes from a meaningless metric which is not suitable for the dataset in question. Hence, this higher accuracy no longer reliably measures model performance.

II. CHALLENGES WITH CLASS IMBALANCE CLASSIFICATION

A. CHALLENGES

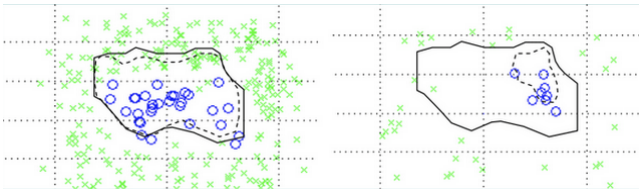
1. IMBALANCED CLASS DISTRIBUTION

Considering the ratio of the number of instances of the minority class to that of the majority class as a measure of imbalance, results have shown a relatively balanced distribution between classes gives better results.

In certain domain problems, datasets can have imbalance ratios as extreme as 1:10000. However, the degree of imbalance past which it starts to hinder performance is not explicitly known i.e a 50-50 distribution is not always the best distribution. The imbalanced Classification problem was illustrated in our dataset by showing that every minority class instance (fraud) was misclassified.

2. LACK OF INFORMATION CAUSED BY SMALL SAMPLE SIZE

Lack of training data causes difficulties in discerning patterns and patterning uniformity. This problem is especially severe for the minority class.
The decision boundary plotted is unsatisfactory and not sufficient to capture the true class regions.
The data size problem was illustrated in our dataset by showing that the error rate of minority classification generally decreased with an increase in the training data set.



3. CLASS OVERLAPPING OR CLASS COMPLEXITY

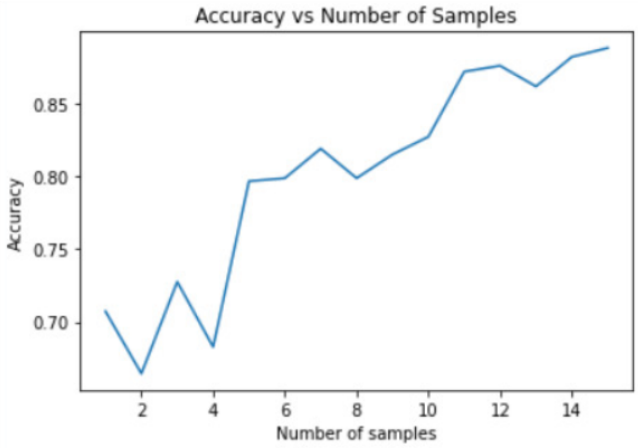
Class overlapping refers to the degree of separability between classes present in the data. When two classes possess overlapping patterns for some feature space it is difficult to separate them. This leads to certain features being rendered redundant as they are no longer useful in classification. Standard classifiers which aim to maximise accuracy in classification incorrectly classify overlapping regions in imbalance class problems as belonging to the majority class leading to minority class instances in that region being misclassified.

B. IMPLEMENTATION

We first tried to locate the issue that normal classifiers (classifiers built for balanced data sets) face with imbalanced data sets. We ran our test on three classifiers: KNN, Decision Trees, and Random Forest.

We found out that KNN is the worst-performing classifier and Random Forest is the best classifier.

Moreover, we tried to see how the recall changes with the number of samples.



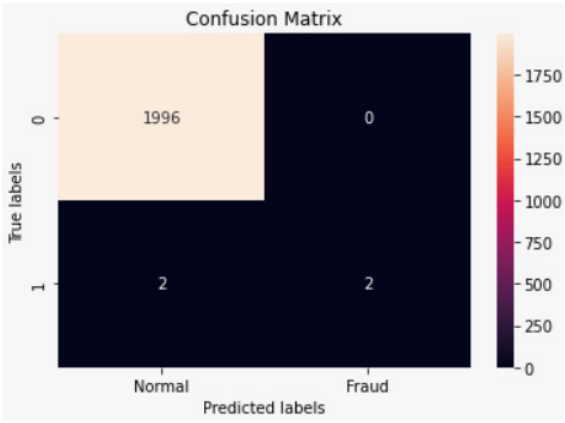
Here, the Accuracy is the recall. This is because we tested the classifier with only the positive samples.

$$\text{Recall} = \frac{TP}{TP + FN} \quad , \text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

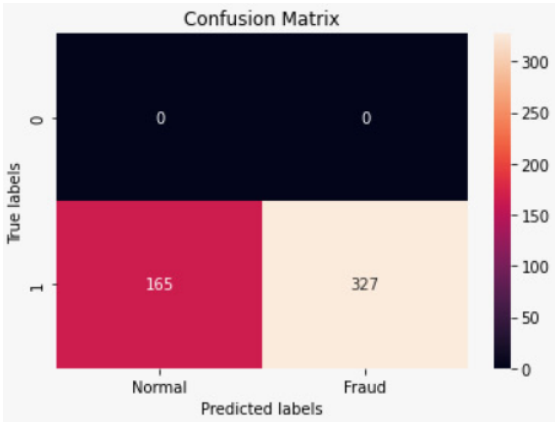
Here, TP and TP don't exist as there are no negative classes.

Therefore, the formula becomes $\frac{TP}{TP + FN}$, which is the same as that of Recall.

By the plot, we can see that, generally, just by increasing the sample size, we are able to better classify the results. But this still isn't the best. Because having a recall < 0.90 means that there is a 10% error in classifying a positive sample.



This is an example of the confusion matrix of a classifier on 10k datasets.
8,000 were used for training and the rest 2,000 for testing it.
We can see that all of the normal transactions are predicted truly. The issues come with fraudulent transactions. In this case, there is basically a 50% chance that we will be able to catch a fraudulent transaction and this is poor.



This is another case where we just ran the classifier on fraudulent transactions. We get an error rate of 33% which is again poor.

We are therefore able to locate the issue. The accuracy of these training models is good, but the recall scores are bad. For class-imbalanced datasets, it's important to have a good recall as positive classes are more important to us mostly.

III. APPROACHES IN CLASS IMBALANCE CLASSIFICATION

A. DATA LEVEL APPROACH FOR HANDLING CLASS IMBALANCE PROBLEM

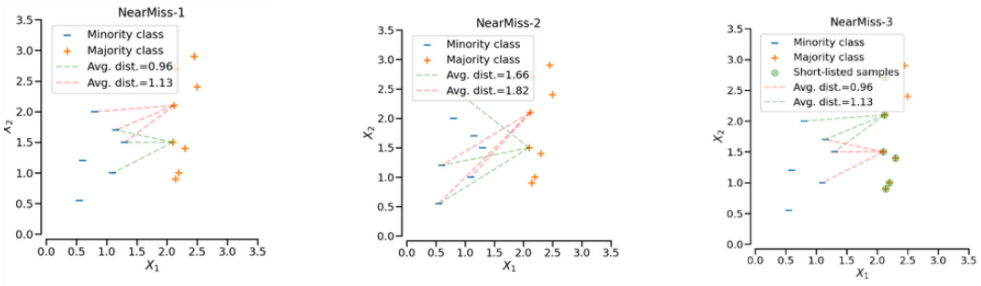
Pre-processing is used in the data-level approach, often referred to as external procedures, to rebalance the class distribution. In order to do this, the imbalance ratio in the training data is reduced by either using under-sampling or over-sampling. We can also perform feature selection which involves removing irrelevant and redundant features to improve the classifier's performance.

1. SAMPLING

1.1) Undersampling

Undersampling is a technique used to remove instances of the majority class so that the two classes have a balanced distribution. The under-sampling methods work with the majority class. In these methods, we randomly eliminate instances of the majority class. It reduces the number of observations from the majority class to balance the dataset. This method is applicable when the dataset is huge and reducing the number of training samples makes the dataset balanced.
Two kinds of undersampling techniques:

a. Random Undersampling involves randomly choosing N majority class samples where N is the number of minority class data points.
b. Near Miss Undersampling: In Near Miss Undersampling, we only sample the data points from the majority class which are necessary to distinguish the majority class from other classes. There are 3 kinds of Near-Miss Sampling:
    i. In NearMiss-1 sampling technique, we select samples from the majority class for which the average distance of the N closest samples of a minority class is smallest.
    ii. In NearMiss-2 sampling technique, we select samples from the majority class for which the average distance of the N farthest samples of a minority class is smallest.
    iii. In NearMiss-3 sampling technique, we first select all M nearest neighbours for every point of the minority class. Then, we select those positive class instances whose distance to N closest samples is the largest.



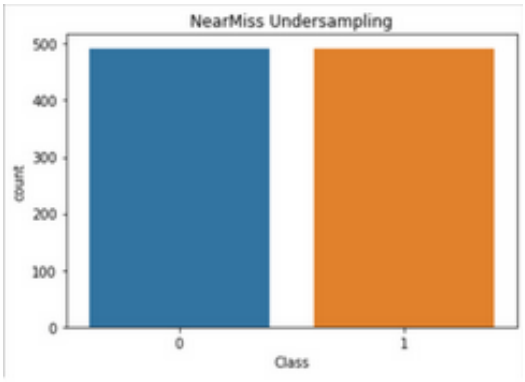
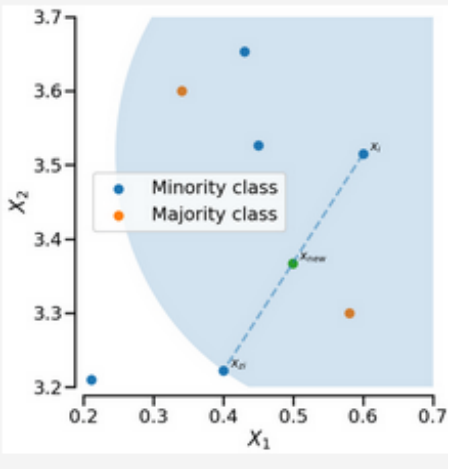


Figure: Through undersampling, recall improved from 60% to 93%

1.2) Oversampling

Oversampling is a technique used to create instances of the minority class so that the two classes have a balanced distribution. This method is useful when we have fewer data available and undersampling cannot be used.
Two kinds of oversampling techniques:

a. Random oversampling involves randomly selecting examples from the minority class, with replacement, and adding them to the training dataset.
b. Informed oversampling methods: SMOTE & ADASYN are improved oversampling methods which can be used to oversample data.
    i. The Synthetic Minority Oversampling Technique (SMOTE) involves creating artificial data based on feature space. New samples are generated using k-nearest neighbours. One of the nearest neighbours is selected at random and interpolated to form the new sample.
    ii. Adaptive Synthetic Oversampling (ADASYN) works in a similar way but the number of samples generated is proportional to the number of nearby samples which do not belong to the same class. Therefore, more samples will be generated in the area where the nearest neighbour rule is not respected.



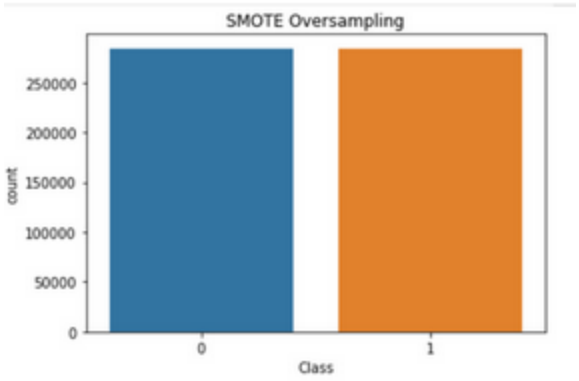


Figure: Through oversampling, recall improves from 60% to 96%

2. FEATURE SELECTION

2.1) Correlation Matrix

A table displaying the correlation coefficients between groups of variables is called a correlation matrix. There is a correlation between each random variable (X_i) and each value in the table (X_j). This allows you to see which pairs have the highest correlation.

A correlation matrix is a typical "square," with the same variables displayed in both the rows and the columns. This demonstrates relationships between people's expressed values for various items. The primary diagonal, which runs from the top left to the bottom right, illustrates how each variable is always perfectly correlated with itself.
The correlations presented above and below the main diagonal of this symmetrical matrix are mirror images of one another.
Most correlation matrices use Pearson's Product-Moment Correlation (r). It is also common to use Spearman's Correlation and Kendall's Tau-b. Both of these are non-parametric correlations and are less susceptible to outliers than r.

Values in the correlation matrix fall between -1 and +1. Positive correlations are understood as values close to +1 and negative correlations as values close to -1. When two variables have a positive correlation, their values rise or fall together. One of the two variables that have a negative connection in value rises while the other one declines. There is no relationship between these two variables if the value is close to 0. Our objective is to identify and remove any properties that have a quality property correlation value that is close to 0.

$$\operatorname{cov}(X,Y) = \frac{\sum_{i=1}^{N}(X_i - \overline{X})(Y_i - \overline{Y})}{N-1}$$

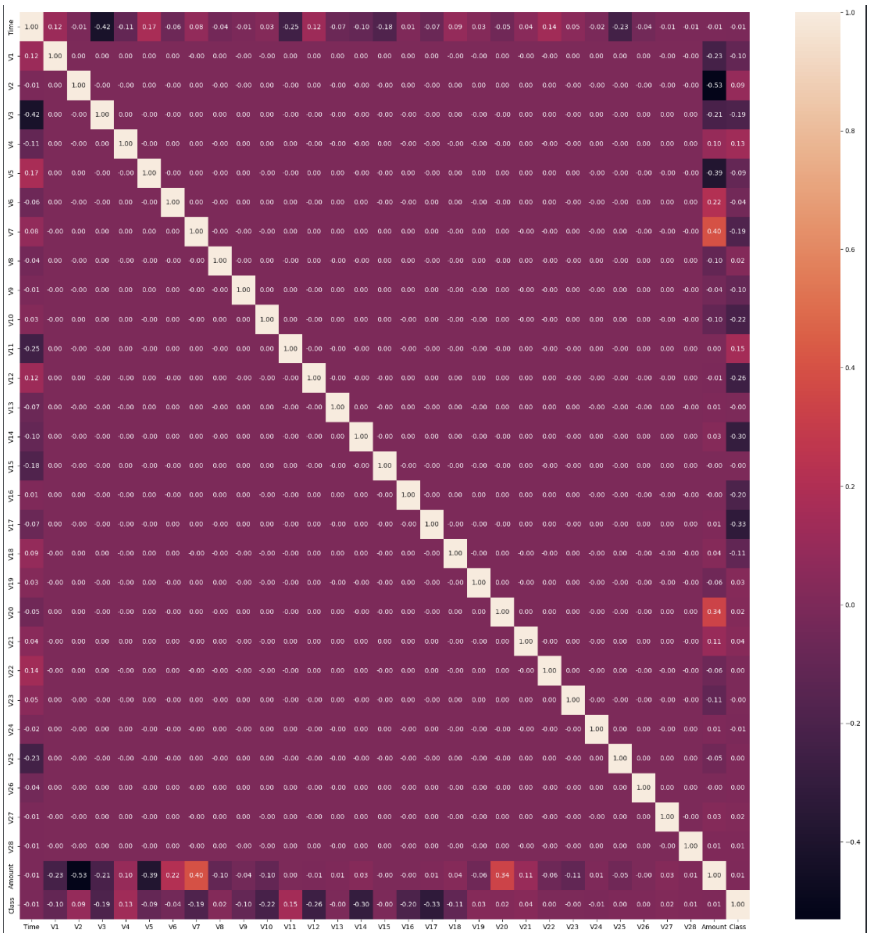


Figure: Here, we see that all features are not greatly correlated with each other, so the inference we made was that all features are important in this area.

1.2) Chi-Square Test

The chi-square is typically used to assess how well or poorly a given predictive categorization model performs at making accurate predictions. The test specifically compares the actual distribution of the data points to the expected distribution of those data points into different labels. There are simply actual numbers involved, not percentages. The chi-square test is primarily used for categorical data, not continuous data unless the continuous data is binned.

As with the majority of analytical methods used in hypothesis testing, the initial input is a degree of significance, which we'll name alpha. But first, let's speak about the other inputs before we talk about alpha. A list of quotients makes up the following inputs. The numerator is made up of the expected result deducted from the actual result for each quotient. After that, the numerator is squared and divided by the anticipated result. The sum of these quotients is calculated. The inputs, as opposed to the actual number of observations per label, refer to the number of observations per label based on expected

distributions. Degrees of freedom, which are equal to 1 less than the number of labels, make up the final input.

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

After employing the Chi-squared test on our dataset, we found the best 10 features to consider. On applying fit_transform to the data to remove all other features, we noticed that the recall improved to some extent.

B. ALGORITHM-LEVEL APPROACH FOR HANDLING CLASS IMBALANCE PROBLEM

1. IMPROVED ALGORITHMS

1.1) z-SVM

1.1.a) Intuition

Support Vector Machine (SVM) is widely used in binary classification scenarios due to its improved accuracy and performance. Given a dataset, it finds a discriminatory hyperplane that maintains an optimal margin from the boundary examples called support vectors. However, a number of recent studies have shown that the decision border orientation for an SVM trained with imbalanced data is skewed toward the minority class and as such, the prediction accuracy of the minority class is low compared to that of the majority ones. z-SVM concentrates on the post-adjustment of an SVM model's learnt decision boundary so that it maintains a good margin from the data of both classes and enhances classification performance in the unbalanced data domain.

The learning task in SVM comprises finding the solution to the following expression:

$$max_\alpha L(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \alpha_i \alpha_j y_i y_j K(\mathbf{x_i}, \mathbf{x_j})$$
$$s.t.$$
$$\sum \alpha_i y_i = 0$$

The optimal decisic

$$\mathbf{w} = \sum \alpha_i y_i \phi(\mathbf{x}_i)$$

For the classification of a given test instance x, SVM uses the following decision function:

$$f(\mathbf{x}) = \sum_{\mathbf{x_i} \varepsilon SV} \alpha_i y_i K(\mathbf{x}, \mathbf{x_i}) + b$$

It has been found that heavily skewed class distribution causes the solution to the above equation to be dominated by the majority class.

For imbalanced training data, the penalty introduced in the objective function for the relatively small number of positive samples is outweighed by that introduced by a large number of negative samples. As a consequence, it focuses more on maximizing the margin

from the majority samples, resulting in a decision hyper-plane more skewed towards the minority class.

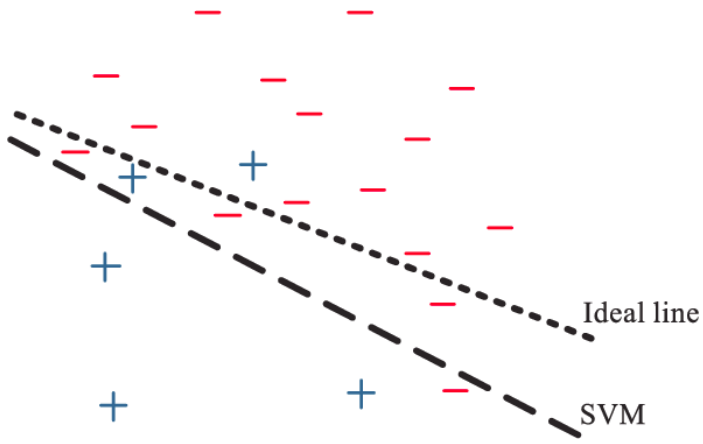


Figure: SVM decision boundary tends to shift towards the minority class in case of class imbalance data

1.1.b) z-SVM Approach

The learned weight vector equation can be re-written as:

$$\mathbf{w} = \sum \alpha_p y_p \phi(\mathbf{x}_p) + \alpha_n y_n \phi(\mathbf{x}_n)$$

The classification decision equation can be re-written as:

$$f(\mathbf{x}) = \sum_{\mathbf{x_p} \varepsilon SV; y_p > 0} \alpha_p y_p K(\mathbf{x}, \mathbf{x_p}) + \sum_{\mathbf{x_n} \varepsilon SV; y_n < 0} \alpha_n y_n K(\mathbf{x}, \mathbf{x_n}) + b$$

As the above equation illustrates, classification depends on a number of factors: the Lagrange multiplication constants associated with positive and negative support vectors, the kernel function and also the number of positive and negative support vectors. For imbalanced training data, these factors are more biased towards the negative (majority) class, causing the tendency to classify an unknown instance as negative.

To reduce the bias of a trained SVM to the majority class for imbalanced data, we introduce a multiplicative weight, z, associated with each of the positive class support vectors.

$$f(\mathbf{x}, z) = z \sum_{\mathbf{x_p} \varepsilon SV; y_p > 0} \alpha_p y_p K(\mathbf{x}, \mathbf{x_p}) + \sum_{\mathbf{x_n} \varepsilon SV; y_n < 0} \alpha_n y_n K(\mathbf{x}, \mathbf{x_n}) + b$$

Weighting the αp-s shifts the weight vector from the learned position to another position that is further away from the positive class, thereby reducing the skew towards the minority class. Thus the introduced multiplicative weight z is, in effect, a correction of the originally learned boundary of SVM to cope with class imbalance.

1.1.c) Determination of z

We use the G-mean measure to adjust the SVM learning which is defined as:

$$gmean = \sqrt{acc_+ * acc_-}$$

As g-mean penalises the misclassification of small classes (eg., failing to recognize a few minority examples reduces acc+ drastically) heavily, adjusting the SVM learning according to this measure, in effect, auto-incorporates the class imbalance information for training an SVM.

If we gradually increase the value of z from 0 to some positive value, M in the modified classification equation, and use the new model to classify data, the new model will classify everything as negative for z = 0 and classify everything as positive for z = M . Since g-mean is a function of the accuracy of both classes, it's value will increase from 0 (at some point $z = z^l$) to some maximum value (at $z = z^*$) and drop to 0 again (at $z = z^h$). The problem is an unconstrained optimisation problem of single variable z and is stated as

$$max_z J(z) = \sqrt{\frac{\sum_{\mathbf{x_u} \epsilon X; y_u > 0} I(y_u f(\mathbf{x_u}, z))}{P} . \frac{\sum_{\mathbf{x_v} \epsilon X; y_v < 0} I(y_v f(\mathbf{x_v}, z))}{N}}$$

A univariate unconstrained optimization technique is used to determine the optimal $z = z^*$. For our strategy, we have applied the Golden section search algorithm for the optimization process. The derived z^* value is used to make classification decisions according to the classification equation described earlier.

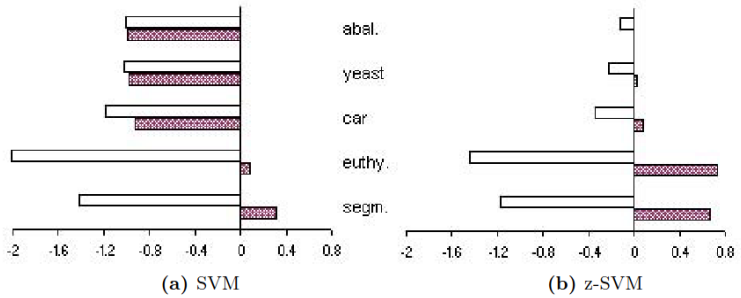
1.1.d) Time Complexity

The runtime complexity of an SVM algorithm depends on the number of training points and the kernel computation involved in quadratic optimization. Hence for a training set of size N, the complexity of SVM learning is $O(N^3)$, with some practical software approximating it close to $O(N^2)$. In this approach, the extra time required is only due to the calculation of z. To calculate z, we need to do a number of g-mean evaluations each of which takes $O(N^2)$ due to the kernel evaluation of each of the data points. The total number of g-mean evaluations is O(1) with respect to the dataset. Therefore, the total time complexity is $O(N^2)$.

1.1.e) Results

In the figure, the z-SVM algorithm has been evaluated for 5 skewed datasets with the first 3 being extremely skewed. We can observe that in 1 (a), the distance of the hyperplane from the positive class is negative, which means it misclassified many positive instances as negative. It starts to correctly classify as the dataset becomes less skewed (euthy and segm) but the distance from the majority class is significantly higher than the distance from the minority class.

1(b) shows the distances of the hyperplane found by z-SVM which is considerably much better than in the case of normal SVM. z-SVM has been successful in overcoming the error in training by shifting the decision boundary away from positive class and thus increasing the margin of hyperplane from positive class data.

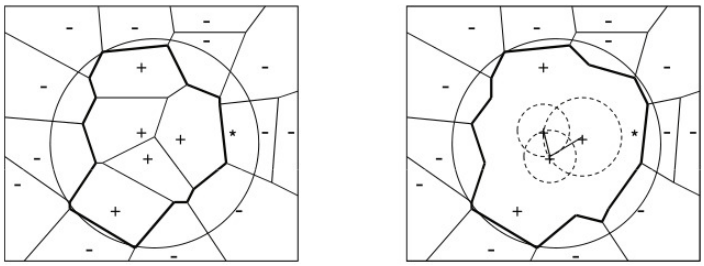


(a) SVM (b) z-SVM

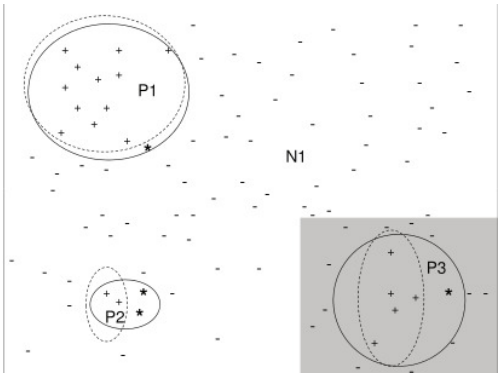
Fig. 1. Average distance of examples from the trained hyperplane for (a) SVM and (b) z-SVM derived from SVM. White bar and shaded bar indicate the distance for negative and positive classes respectively, with positive class being the minority.

1.2) KNN EXEMPLAR GENERALIZATION

1.2.a) Issues with KNN



In this image, it is evident that in a class-imbalanced dataset such as this, where the subconcepts P1, P2, and P3 belong to the positive class, the decision boundaries made by the normal KNN are far from ideal. The decision boundary is the dotted line while the ideal boundary is shown as the solid line. The '*', as a test point, gets incorrectly classified by KNN. We try to solve this issue by using Exemplar KNN or ENN.



If we look at this subconcept P3, if we run the standard 1NN, we are left with the solid bold line as the decision boundary. This leaves out a lot of the points which are of the positive class but not lying within this smaller boundary. But, after implementing the Exemplar 1NN, on the right side, we can clearly see how the decision boundary is expanded to include more points. It is closer to ideal, but still not ideal. To reach this decision boundary, we define something called the Positive Pivot Instances (PPI).

We define distance r as the distance between the point in question to the nearest positive neighbour e. We define False Positive Rate FP as the ratio of the number of negative points to all points within distance r from the point in question. Therefore, a PPI is a point whose FP rate is below a threshold. I.e. FP rate <= delta.

If we look at the right figure above, the middle 3 points have an FP rate below the threshold. But, the other two points have their, FP rate higher than the threshold.

1.2.b) Algorithm

1. Go through each of your positive points.
2. Find the nearest positive neighbor.
3. Calculate the FP ratio.
4. If lesser than the threshold, add to the PPI list, otherwise go to step 2 for the next point.

When we calculate the distance between two points such as (t, x), where X is PPI, the distance between them both is the distance(t, x) - x.radius, where the radius is the distance to the nearest positive point. By doing this, more points will be classified in the positive class than before. Moreover, as we are only doing this for some points rather than all points, we are making sure that those points which have high positive points in their neighborhood are only increasing their boundary.

1.2.c) Performance Based on Other Classifiers

Dataset	3ENN	Naive	3NN	3NNSmt+	3NNMeta	C4.5	C4.5Smt+	C4.5Meta
Oil	**0.811**	0.788	0.796	0.797	0.772	0.685	0.771	0.764
Hypo-thyroid	0.846	0.831	0.849	0.901	0.846	0.924	**0.948**	0.937
PC1	**0.806**	0.786	0.756	0.755	0.796	0.789	0.728	0.76
Glass	0.749	0.623	0.645	0.707	0.659	0.696	0.69	**0.754**
Satimage	**0.925**	0.839	0.918	0.902	**0.928**	0.767	0.796	0.765
CM1	**0.681**	0.606	0.637	0.666	0.625	0.607	0.666	0.668
New-thyroid	**0.99**	0.945	0.939	0.972	0.962	0.927	0.935	0.931
KC1	**0.794**	0.732	0.759	0.756	0.779	0.64	0.709	0.695
SPECT_F	**0.767**	0.728	0.72	0.725	0.735	0.626	0.724	0.643
Hepatitis	**0.783**	0.71	0.758	0.772	0.744	0.753	0.713	0.745
Vehicle	0.952	0.945	**0.969**	0.942	0.956	0.921	0.926	0.929
German	**0.714**	0.677	0.69	0.686	0.705	0.608	0.649	0.606
Average	**0.818**	0.768	0.786	0.798	0.792	0.745	0.771	0.766

Here, the best results are in bold. 3ENN significantly outperforms 3NN, 3NNMeta, C4.5Smt+ and C4.5Meta. C4.5Smt+ and C4.5Meta are already strategies to improve on C4.5 for class imbalanced problems. But they have other issues plaguing them such as the proper distribution of the positive class. Therefore, they don't show significant

improvement on 3NN. But, by employing the exemplar-based method, we are able to increase the decision boundary in a selective measure which provides optimum classification.

1.2.d) Impact of threshold on kENN

On highly imbalanced data, it is beneficial to have more thresholds. This makes the data sensitive to positive class and even a few positive points can make the point as a PPI, which increases the decision boundary. On other hand, if we look at a balanced dataset, a high threshold will give more wrong answers as there is already a huge number of positive points. Increasing the threshold will induce more negative values which we do not want in that case.

2. ONE-CLASS LEARNING

Outlier or anomaly detection is the process of finding outliers in data, and one-class classification is a branch of machine learning that focuses on this issue. These algorithms for unsupervised learning make an effort to simulate "normal" instances in order to categorise fresh examples as either normal or abnormal (e.g. outliers).
Binary classification tasks with a significantly skewed class distribution can be handled using one-class classification methods. These methods can be tested on a holdout test dataset after being fitted to the input instances from the majority class in the training dataset.

One-class classification algorithms can be successful for imbalanced classification datasets where there are no or very few examples of the minority class, or datasets where there is no coherent structure to separate the classes that could be learned by a supervised algorithm, despite not being designed for these kinds of problems.

3. COST-SENSITIVE LEARNING

3.1) INTUITION

The studies in cost-sensitive learning are motivated by the various forms of domain applications with class imbalance datasets and misclassification cost being viewed as equal by many classic learning algorithms.
Penalty-sensitive learning algorithms are created with the premise that when a classifier makes a mistake, it incurs a high cost. For instance, a classifier may give false negatives a higher cost than false positives, emphasising any right classification or mistake relating to the positive class.
A PSO-based cost-sensitive neural network was explored, and an optimised cost-sensitive SVM was proposed in a number of works on cost-sensitive learning for unbalanced class distribution.

The hyperplane decision boundary that divides the instances into two groups most effectively is found by the SVM method.

The adoption of a buffer that permits some points to be incorrectly categorised softens the split.
On unbalanced datasets, this margin favours the majority class by default; however, it can be modified to take into consideration the significance of each class and significantly enhance the algorithm's performance on certain datasets.

Weighted SVM, also known as cost-sensitive SVM, is a variant of SVM that counts the margin proportionally to the relevance of the class.
SVMs are efficient, but they struggle when the class distribution has a significant skew. As a result, the method has been extended in numerous ways to improve its performance on datasets with imbalances. When the model is fitted, specifically when the decision boundary is determined, the C parameter is employed as a penalty. The default weighting for each class is the same, so the softness of the margin is symmetrical. The majority class will benefit from the soft margin and decision boundary because there are significantly more examples in the majority class than in the minority class.

To weight the C value in accordance with the importance of each class is perhaps the simplest and most frequent addition to SVM for imbalanced classification.
Particularly, each sample in the training dataset contains a unique penalty term (C value) that is employed in the SVM model's margin computation.
A weighting of the global C-value that is defined proportionally to the class distribution can be used to determine the C-value of an example.
The margin can be made softer for the minority class by using a larger weighting, while the margin can be made harder and misclassified examples are avoided by using a smaller weighting for the majority class.

Small Weight: Smaller C value, larger penalty for misclassified examples.
Larger Weight: Larger C value, smaller penalty for misclassified examples.

As a result, the margin is encouraged to limit the majority class's flexibility while allowing the minority class to be flexible by misclassifying samples from the majority class onto the minority class side as necessary. Weighted Support Vector Machine (SVM), Class-Weighted SVM, Instance-Weighted SVM, or Cost-Sensitive SVM are other names for this variation of SVM.

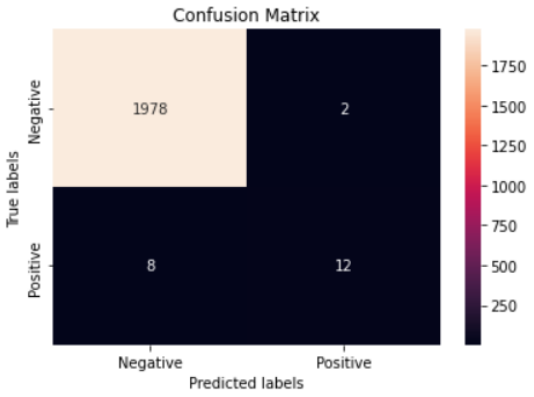
3.2) IMPLEMENTATION & OBSERVATIONS

We used the *make_classification()* function to define a synthetic imbalanced two-class classification dataset. We generated 10,000 examples with an approximate 1:100 minority-to-majority class ratio.
The LinearSVC and SVC classes provide the *class_weight* argument that can be specified as a model hyperparameter. *class_weight* is a dictionary that defines each class label (e.g. 0 and 1) and the weighting to apply to the C value in the calculation of the soft margin.

A best practice for using class weighting is to use the inverse of the class distribution present in the training dataset. The class distribution of the test dataset is a 1:100 ratio for the minority class to the majority class. The invert of this ratio could be used with 1 for the majority class and 100 for the minority class.
Without Cost-Sensitive SVM:
ROC AUC: 0.829, Recall: 0.657, Precision: 1.000, Specificity: 1.000, G-mean: 0.811
With Cost-Sensitive SVM:
ROC AUC:0.912, Recall: 0.829, Precision: 0.674, Specificity: 0.996, G-mean: 0.908

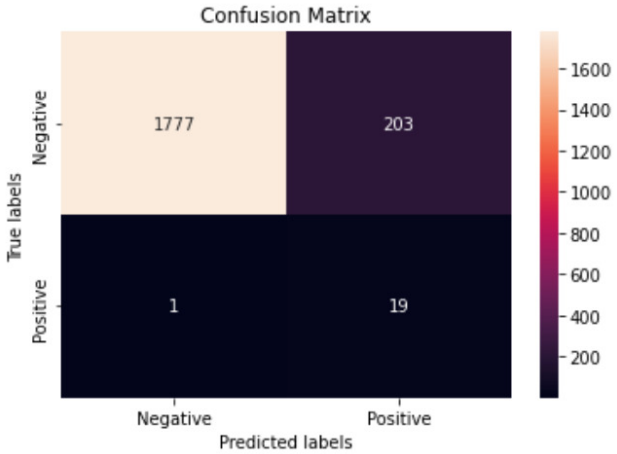
4. ENSEMBLE METHODS

a) AdaBoost



Adaboost or Adaptive boosting is a sampling technique used in training ensemble classifiers. It's a variation of the bootstrap sampling algorithm. The difference is that it assigns higher weights to samples that are misclassified by a model, which dictates the probability of that sample being used in the training data for the subsequent classifier model. A higher weight corresponds to a higher probability of being sampled. Thus, each subsequent classifier model in the ensemble is tuned to work where the preceding classifier failed. Thus we obtain an ensemble of weak classifiers that results in a strong classifier.

b) RUSBoost

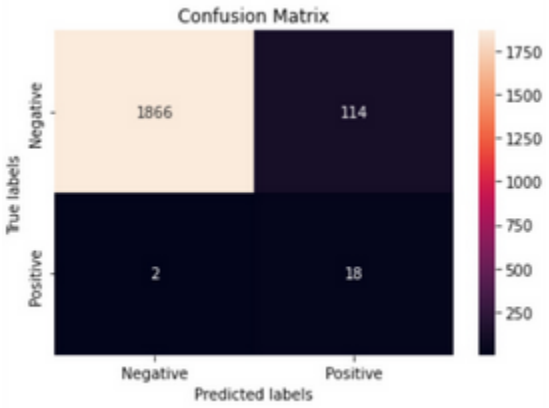


RUSBoost is a boosting-based ensemble technique similar to SMOTEBoost. It makes use of the RUS (Random Undersampling) algorithm. The RUS algorithm carries out undersampling by randomly removing instances of the data we wish to undersample. The RUSBoost performs the Random UnderSampling on the majority class before each boosting step to obtain a balanced dataset. It's advantageous as it is faster than the SMOTEBoost oversampling method and gives the highest recall, by far, of the minority class.

We used the RUS sampling algorithm in combination with AdaBoost to carry out RUSBoosting and obtained this confusion matrix.

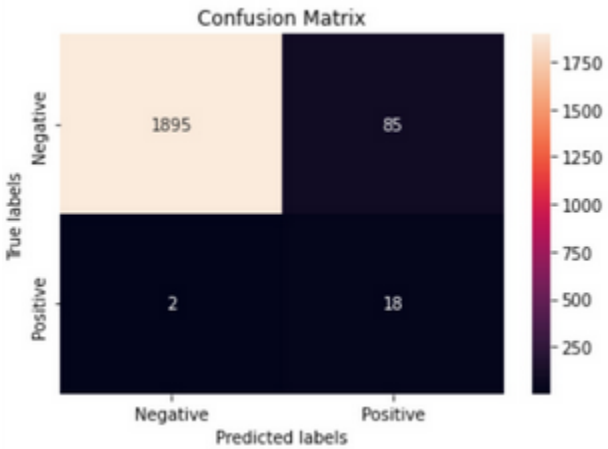
c) SMOTEBoost

SMOTEBoost is a boosting-based ensemble technique. It makes use of the SMOTE(Synthetic Minority Oversampling Technique) algorithm. The SMOTE algorithm is an oversampling technique. The SMOTE algorithm oversamples a class by creating synthetic data points of that class. It does so by randomly selecting a point in the dataset of that particular class, and finding its k-nearest neighbours. It then creates synthetic data points at random locations on the lines joining the selected point and its k-nearest neighbours in the feature space.



The SMOTEBoost algorithm injects this SMOTE algorithm at each step of the boosting iteration of a standard boosting algorithm like AdaBoost. This means that while a standard boosting algorithm places importance on all misclassified samples, the SMOTEBoost places an additional emphasis on the minority class by increasing instances of it.
We used the SMOTE sampling algorithm in combination with AdaBoost to carry out SMOTEBoosting and obtained this confusion matrix.

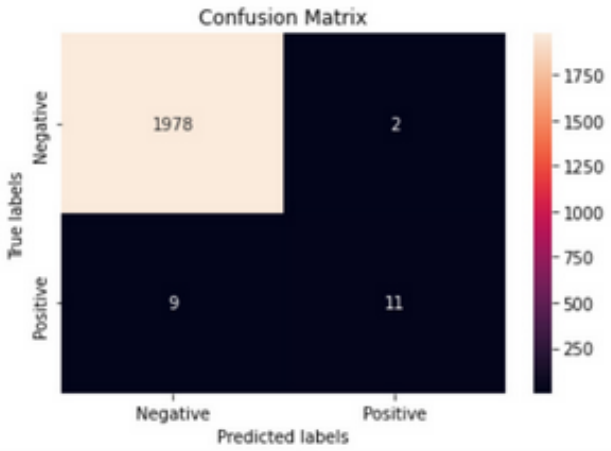
d) UnderBagging



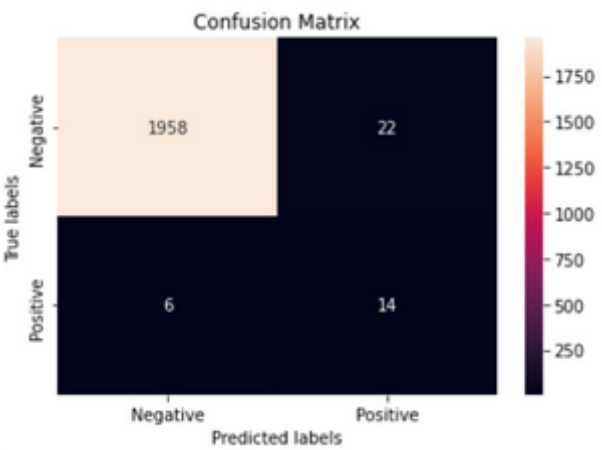
Underbagging is a bagging-based ensemble method. It involves a combination of undersampling and bagging techniques. In this algorithm, each bag created in the bagging process contains all the minority class instances along with an appropriate number of majority class instances which are randomly sampled.
We obtained this confusion matrix for our classifier model.

e) OverBagging

Overbagging is a bagging-based ensemble method. It involves a combination between oversampling and bagging techniques. Overbagging increases or adds minority class instances(oversamples) in each bag created during the bagging process. The purpose of this is to obtain a more balanced dataset.
We obtained this confusion matrix for our classifier model.



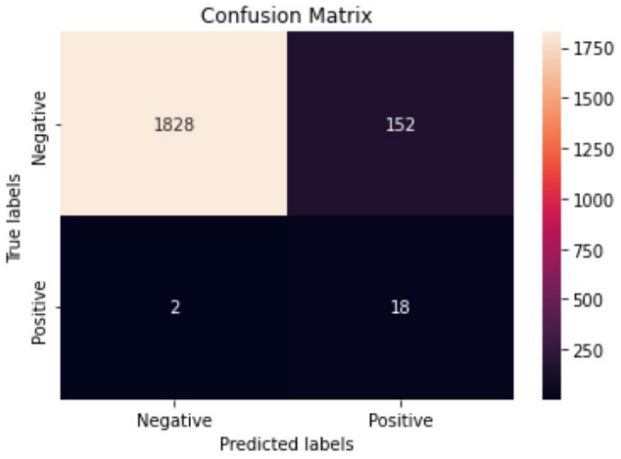
f) SMOTEBagging



SMOTEBagging is a bagging-based ensemble method. It involves a combination of the SMOTE and bagging algorithms. The minority class of each bag of data created during the bagging process is oversampled using SMOTE sampling algorithm.
We obtained this confusion matrix for our classifier model.

g) EasyEnsemble

EasyEnsemble is an undersampling algorithm that works in combination with AdaBoost. It samples several subsets from the majority class data with each subset being equal in size to the minority class dataset. The classifier models are trained on a union of each of



these subsets with the entire minority class dataset. The classification result is obtained through an aggregate of the results of each of these weak classifiers.

5. HYBRID APPROACH

Besides the one-class learning, cost-sensitive methods and ensemble approaches, a new breed of classification algorithms has been devised for handling class imbalance datasets in recent years. The hybridization is designed with the idea to alleviate the problem in sampling, feature subset selection, cost matrix optimization and fine-tuning the classical learning algorithms. There are also reported studies that hybridize classifiers in order to improve classification qualities with class imbalance problems. Hybrid approaches integrate the advantages of sampling, cost-sensitive, and ensembling methods. There are different blocks which perform dimensionality reduction, sampling, data construction and ensemble methods which give good performance overall.

IV. PERFORMANCE MEASURES

Accuracy is misleading for imbalanced classes. Most machine learning algorithms are designed to maximize overall accuracy by default. But this maximum accuracy is misleading. We consider a highly imbalanced dataset containing only 100 positive instances and 9900 negative instances. Even if the classifier classifies all the tuples as negative without any training or logic, its accuracy is 9900/10000 = 99% which is very high. However, this does not mean that our classifier is performing well.

There are certain situations in which the identification of positive class is very important. In such situations, it is desirable to classify some negative instances as positive if it ensures that no positive instances are missed out. This is expressed using the measure recall. For eg, consider a situation where we want to classify a person as having a rare medical condition based on some the attributes like height, weight, past history etc. We don't want to misclassify a person having a rare disease because it can turn out to be life-threatening for him. We want to identify as many positive cases as possible from the given sample.
Some of the measures relevant to class imbalance classification are:

1. Sensitivity or True Positive rate/recall:

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

   Sensitivity refers to the ability of a classifier to correctly identify a positive class as such. It ranges from 0 to 1 with 1 being the perfect score.

2. Specificity or True Negative rate:

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

   Specificity denotes the ability of a classifier to correctly identify negative classes as such.

The perfect score is 1 and 0 is the worst measure.

3. G-mean (geometric mean):

$$G - mean = \sqrt{Sensitivity \times Specificity}$$

G-mean or geometric mean is used for indicating the ability of a classifier in balancing the classification between positive class accuracy and negative class accuracy. By taking the G-mean of both sensitivity and specificity together, a low score for G-mean denotes a classifier that is highly biased towards one single class, and vice-versa.

4. Precision:

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

Precision is a measure of exactness, which is the proportion of observations from a positive class correctly classified as positive. That is the number of correct classifications for the positive class. It tells how well a classifier removes a negative class from being misclassified as a positive class.

5. ROC AUC Curve:

The graphical plot of a ROC curve is made up by plotting the false positive rate (FP) on the x-axis and the true positive rate (TP) on the y-axis.
Generally, the use of ROC curves and precision-recall curves are as follows:

1. ROC curves should be used when there are roughly equal numbers of observations for each class.
2. Precision-Recall curves should be used when there is a moderate to large class imbalance.

The main reason for this optimistic picture is because of the use of true negatives in the False Positive Rate in the ROC Curve and the careful avoidance of this rate in the Precision-Recall curve. The reason for this recommendation is that ROC curves present an optimistic picture of the model on datasets with a class imbalance.

V. CURRENT TRENDS & FUTURE DIRECTION IN CLASS IMBALANCE AND CLASSIFICATION

It is observed that there are more reported works in binary class imbalance problems but only a handful of studies are conducted to find solutions to multiple class imbalance problems. This is mainly due to the fact that class imbalance with binary class is more prevalent and handling multiple class imbalance data is more complicated. Many reported works have discovered that class overlapping severely hinders a classifier's performance more than class imbalance. To solve the issues of class overlapping, many studies have adopted the alternative solution by incorporating a sampling strategy in the pre-processing task.

Several studies have highlighted that class overlapping problems are caused by irrelevant or redundant features, and feature selection is one of the strategies used to address this issue.
The complexity of how the data is distributed in the multidimensional feature space coupled with a very small number of observations forming the minority class makes it very challenging to distinguish the minority class from the majority class. The task to learn the minority class from a very small training size using a very high number of features has become a burden to the classifier. Hence, there is a strong urgency in searching for a strong, dominant feature subset from all available features, especially in the classification of the multivariate data sets. The large volume of data set remains one of the major challenges in the classification domain. Data sets expand rapidly in number and also attribute/ feature-wise. It is foreseen that increasing demand for big data applications from the real world will most probably call for better advancement in machine learning algorithms for imbalanced big data management. In the future, the rapid development of big data computing most probably will shape the way classification tasks are performed and with anomaly patterns existing in most real-world problems, class imbalance problem is inevitable. There is a strong need to address such a problem and based on the current trends and development, we will most likely see new issues and innovative contributions open up to a new frontier in the area of class imbalance learning and classification.

VI. WORK DISTRIBUTION

Ayush Agrawal:
Data level approach (Undersampling & Oversampling), z-SVM, Measures of Classification

Swetha Vipparla:
Feature Selection, One-Class learning, Cost-sensitive learning

Shubh Agarwal:
Challenges with Class imbalance classification, KNN with exemplar-based generalisation, Measures of Classification

Rohan Madineni:
Challenges with Class imbalance classification, Ensemble Methods