ClusterOSS: a new undersampling method for imbalanced learning

Victor H Barella, Eduardo P Costa, and André C P L F Carvalho,

Abstract—A dataset is said to be imbalanced when its classes are disproportionately represented in terms of the number of instances they contain. This problem is common in applications such as medical diagnosis of rare diseases, detection of fraudulent calls, signature recognition. In this paper we propose an alternative method for imbalanced learning, which balances the dataset using an undersampling strategy. We show that *ClusterOSS* outperforms *OSS*, which is the method *ClusterOSS* is based on. Moreover, we show that the results can be further improved by combining *ClusterOSS* with random oversampling.

Keywords—Imbalanced data, classification, sampling, clustering.

1 Introduction

A dataset is said to be imbalanced when its classes are disproportionately represented in terms of the number of instances they contain. For example, in a problem about a rare disease, the number of cases of infected people is usually much lower than that of healthy people. Other examples of problems containing imbalanced data are: detection of fraudulent calls [1]; detection of fraudulent credid cards transactions [2]; and signature recognition [3].

Traditional machine learning methods usually yield unsatisfactory results in problems of this nature. While they present good results for the majority classes, they perform very poorly w.r.t. the minority classes. The main problem with this is that, in imbalanced learning, the minority classes are usually the classes we are interested in. For this reason, many methods have been proposed, as we discuss in Section 2. One popular method in this context is *OSS* (*One-sided Selection*) [4], which artificially balance the dataset by disregarding instances of the majority class which look to be redundant.

In this paper we propose an alternative method for imbalanced learning in the context of binary classification; the proposed method is based on *OSS*. We show that our method yields better results than *OSS*. As an additional contribution of this paper, we present an empirical comparison among six methods, including *OSS* and our proposed method.

The remaining of this paper is organized as follows. In section 2 we discuss related work. In section 3 we present our proposed method. We present our experimental results in Section 4. We conclude in Section 5.

2 Related work

There are two main general approaches for classification problems involving imbalanced data: (1) pre-processing the data in order to make it more balanced; and (2) development of algorithms able to handle imbalanced data. In this paper we focus on the former.

Pre-processing methods can be categorized into two groups: undersampling and oversampling methods. Undersampling methods make the data more balanced by removing instances of the majority class, while oversampling methods do that by inserting instances in the minority class. Both undersampling and oversampling can be done randomly or according to an informative strategy. Next we discuss the main pre-processing strategies.

- *V.H. Barella, E.P. Costa and A.C.P.L.F. Carvalho are with Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 13560-970 São Carlos, SP, Brazil.*
  E-mail: victorhb@icmc.usp.br

Manuscript received June 2, 2014.

2.1 Random undersampling

In the random undersampling, instances of the majority class are removed at random until a more balanced class distribution is reached.

2.2 Random oversampling

In the random oversampling, instances of the minority class are replicated at random until a more balanced class distribution is reached.

2.3 SMOTE

SMOTE (*Synthetic Minority Oversampling Technique*) [5] is an oversampling technique that creates artificial data by interpolation, as follows. At each iteration, SMOTE selects an instance x at random in the minority class and then it looks for the k nearest neighbors of x. SMOTE then selects one of the neighbors z at random and creates a new instance which is a combination of x and z. This step is repeated until a more balanced distribution of instances is reached.

2.4 CBO

CBO (*Cluster-Based Oversampling*) [6] is an oversampling technique that takes into account both inter- and intra-class imbalance. Intra-class imbalance occurs when there is a disproportion w.r.t. the instances which form subsets inside a class.

This technique starts by performing a clustering procedure in the instances belonging to the majority class and the instances belonging to the minority class, separately; this step will generate two sets of clusters - one for each class. Next, *CBO* applies random oversampling to all clusters belonging to the majority class with exception of the largest one. In the end, each cluster of the majority class should have the same number of instances as the largest one. Finally, oversampling is applied to all clusters belonging to the minority class such that in the end (1) the total number of instances in the minority class equals the total number of instances in the majority class after the oversampling, and (2) each cluster in the minority class has the same number of instances.

2.5 OSS

OSS (*One-sided Selection*) [4] is an undersampling technique that keeps only the most representative instances of the majority class. To select those instances, *OSS* first chooses one instance x of the majority class at random. Then, using the instances of the minority class and x as training data, *OSS* uses the k-Nearest Neighbors (*KNN*) algorithm with k$= 1$ to classify the remaining instances of the majority class. The correctly classified instances are then excluded of the majority class because they are considered redundant. Thus, after the undersampling, the majority class will contain only the instances which were incorrectly classified and x. Finally, *OSS* uses a data cleaning technique to remove borderline and noisy instances, which is, originally, *Tomek Links* [7].

3 Proposed method

This section introduces our proposed method, which is based on the *OSS* strategy. We first motivate our method, by pointing out situations in which *OSS* might not work well. Then, we introduce our method, called *ClusterOSS*.

3.1 Motivation

OSS assumes that is enough to choose only one majority instance at random to start the undersampling process. However, the final result of the undersampling method will depend on that random choice. More importantly, *OSS* does not explicitly take into account the fact that there might exist subsets inside the majority class (as *CBO* does, for example), and that the undersampling might not work equally well in all those subsets, given the random start.

Consider, for example, the dataset displayed in Fig. 1. In the figure, the instances of the majority class are represented by circles, while the instances of the minority class are represented by triangles. Note that the majority class is divided into two subsets - one to each side of the minority class. Fig. 2.a shows the randomly selected majority instance together with the minority instances. Fig. 2.b shows the resulting dataset given by *OSS*.

Note that the effect of the undersampling process is limited by the fact the majority class contains two subsets and by the choice of the

instance to start the process (which is far from the minority class in the feature space). Most of the instances of the majority class are kept after the undersampling process, resulting in a dataset which is still very imbalanced.

Suppose now the instance would have been chosen in the center of the cluster at the right of the figure. In this case, the undersampling would have worked well for that cluster, but would have had little (or no) effect on the other subset of the majority class.

Next, we present an alternative method that avoid these situations.

3.2 ClusterOSS

Our proposed method (*ClusterOSS*) is an adaptation of the strategy used by *OSS*. Before describing the *ClusterOSS* algorithm, we point out the two main differences of *ClusterOSS* w.r.t. OSS.

The first difference is that *ClusterOSS* can start the undersampling process from more than one instance. This, in itself, already tackles the drawback of OSS that the quality results is strongly dependent on the choice of that one instance chosen to start the undersampling.

The second difference is that we do not start the undersampling process at random. Instead, we define how many and which instances will be chosen to start that process. More specifically, we look for subsets in the majority class, by applying a clustering procedure. Then we choose the instance at the center of each subset to be one of the instances which will start the undersampling. By doing this, we enhance the effectiveness of undersampling, since the undersampling will start from points in distinct regions in the feature space.

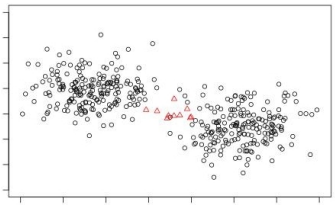


Fig. 1: Original dataset

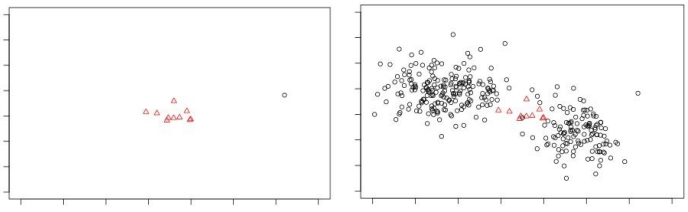


Fig. 2: OSS. Left: Majority instances selected and minority instances. Right: Pre-processed dataset

Algorithm

The *ClusterOSS* algorithm is showed in Fig. 3.a in the form of pseudocode. In the beginning of the algorithm we use a clustering procedure (e.g., *k-means*) to cluster the instances belonging to the majority class. Then, for each cluster, we use the closest instances to the center. These instances are used to start the undersampling process, which is identical to *OSS*. Finally, as in *OSS*, we use the data cleaning technique *Tomek Links* (Fig. 3.b) to remove borderline and noisy instances. Basically, this technique removes every instance z from the majority class for which (1) its closest neighbor z' is an instance of the minority class, and (2) the closest neighbor of z' is also z.

Example

We illustrate how *ClusterOSS* works using the same dataset we showed in Fig. 1. Fig. 4.a shows the selected majority instances(each of them being in the center of the subsets identified by *k-means*) together with all minority instances. Fig. 4.b shows the resulting dataset given by *ClusterOSS*.

Note that *ClusterOSS* is able to obtain a more balanced dataset (Fig. 4.b) than that obtained by *OSS* (Fig. 2.b). The original dataset has a proportion of 1:40 (minority class:majority class), while the proportions of the resulting datasets are approximately 1:30 and 1:5 for *OSS* e *ClusterOSS*, respectively. It is important to mention that both strategies reduce the majority class in distant regions from the minority class. It is the *Tomek Links* step that acts on the closer region to the minority class.

4 Experiments

We present an empirical evaluation of our method. The main goal of it is to verify whether the alternative undersampling strategy used by *ClusterOSS* yields better results than those

```
procedure ClusterOSS(D)
    Train := {}
    Test := {}
    MajInstances := GetMajInstances(D)
    C := Clustering(MajInstances)
    for all clusters C_c ∈ C:
        x := ChooseClosestInstanceCenter(C_c)
        Train := Train ∪ {x}
        Test := Test ∪ (C_c-{x})
    Result := KNN(Train,Test)
    Misclass := GetMisclassifications(Result)
    NewD := Train ∪ Misclass
    TLinks := TomekLinks(NewD)
    for all instances z ∈ TLinks:
        if z ∈ MajInstances
                NewD := NewD - {z}
    return NewD

procedure TomekLinks(D)
    TLinks := {}
    for each pair E_i and E_j:
        if (class(E_i) != class(E_j)) and
                (∄ E_l | d(E_i,E_l)<d(E_i,E_j) or
                    d(E_j,E_l)<d(E_i,E_j))
        TomekLinks := TLinks ∪ {E_i,E_j}
    return TLinks
```

Fig. 3: Pseudocode of *ClusterOSS*. *GetMajInstances()* returns the majority instances from a dataset. *Clustering()* returns a set of identified clusters. *ChooseClosestInstanceCenter()* returns the closest instance of the center of a cluster. *KNN()* uses a train set to classify test instances, with $k = 1$. *GetMisclassifications()* returns the misclassified instances by *KNN*. *d()* is the distance between two instances.

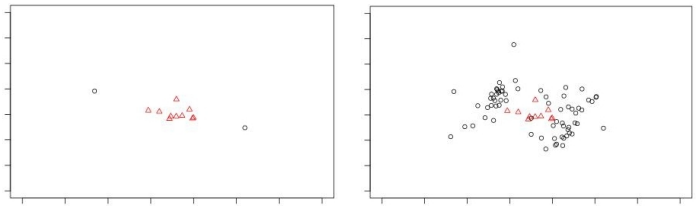


Fig. 4: ClusterOSS. Left: Majority instances selected and minority instances. Right: Pre-processed dataset

given by *OSS*. Additionally, we verify how *ClusterOSS* compare to other existing methods in the literature, namely random undersampling, random oversampling, *SMOTE*, *CBO* and *OSS*. We also evaluate how *ClusterOSS* performs when combined to random oversampling. The motivation for the latter being that we want to combine the strengths of both undersampling and oversampling.

4.1 Experimental Settings

We implemented *ClusterOSS* with the clustering method *k-means*, and we determine the number of clusters by the average silhouette of the training set. For *k-means* we consider squared Euclidean distance as proximity measure, 10 as maximum number of iteration and 1 initial configuration. To obtain the average silhouette we consider the Euclidean distance as proximity measure. For *CBO* we use the same strategy. We combined *SMOTE* with random undersampling as suggested by its creators. We use random oversampling and random undersampling with final proportion of 1:1. We use SMOTE with parameters such it increases the minority class in 200% and decreases the majority class such that the final number of instances in the minority class is 75% of the final number of instances in the majority class. To test the quality of the pre-processing of the data, we apply three different classification algorithms - *KNN* (k= 3), *C5.0* and *SVM* - to the resulting dataset given by each method.

The evaluation was performed on 10 datasets, which are showed in Table 1; the table contains the name, number of attributes (including the class attribute), number of instances and proportion ratio of the datasets.

We obtained the datasets Vowel, Haberman, Pima Diabetes and Yeast from the UCI repository[8], and Cleveland, Poker and Vehicle from the Keel repository[9]. Vowel, Yeast, Cleveland, Poker and Vehicle are originally multiclass problems and were turned into binary problems by choosing a specific class as the positive one and making the following relation of positive classes x negative classes: 0 x rest, 4 x rest, 0 x 4, 8 x 6 and 2 x rest, respectively.

The three artificial datasets were created using a normal distribution for each class (or for each subset of the class, when the class is divided in more than one subset). The artificial datasets are used to assay the performance of the techniques in different situations. They are all binary problems and have three attributes(two are numeric and one is the class). They are plotted in the Figure 5, where the 'X' are instances from the majority class and the

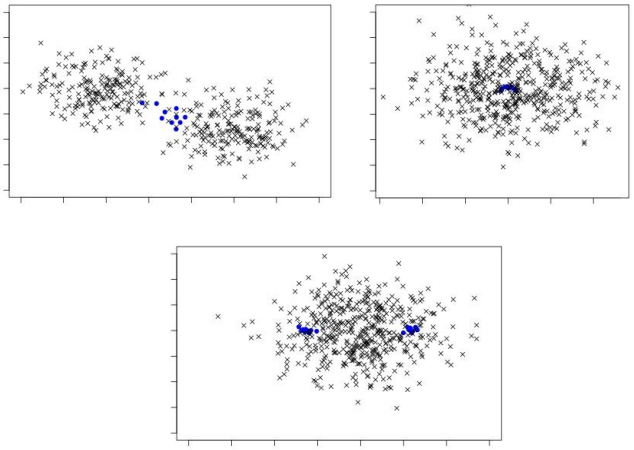


Fig. 5: Artificial Datasets. Top-left: Artificial dataset (a). Top-right: Artificial dataset (b). Bottom: Artificial dataset (c).

Dataset	# Attributes	# Examples	Proportion
Artificial (a)	3	410	1 : 40
Artificial (b)	3	510	1 : 50
Artificial (c)	3	520	1 : 25
Vowel0	11	990	1 : 10
Haberman	4	306	1 : 3
Yeast4	8	1479	1 : 28
Pima Diabetes	9	768	1 : 1.86
Cleveland0x4	14	173	1 : 12.31
Poker8x6	11	1477	1 : 85.88
Vehicle2	19	846	1 : 2.88

TABLE 1: Dataset Information

circles are from the minority class. The dataset (a) has two majority regions and one minority region between them. The dataset (b) has one majority and one minority region completely overlapped. The dataset (c) has a majority region rounded by two minority regions.

We perform the experiments with stratified 5-fold cross validation; we do it 100 times. We choose the stratified variant to keep the class distribution in each fold, and we choose 5 folds to avoid situations in which there is too few examples in the minority class to be able to apply the classification methods.

We used the following evaluation measures: Positive Accuracy ($\frac{TP}{FN+TP}$), Negative Accuracy ($\frac{TN}{TN+FP}$), Geometric Mean of Accuracies ($\sqrt{Pos.Accuracy * Neg.Accuracy}$) and the Area Under the Roc Curve (AUC).

Measure	# OSS victories	# ClusterOSS victories	Ties
Positive Accuracy	4	19	7
Negative Accuracy	13	12	5
Geometric Mean	5	20	5
AUC	9	21	0

TABLE 2: OSS x ClusterOSS

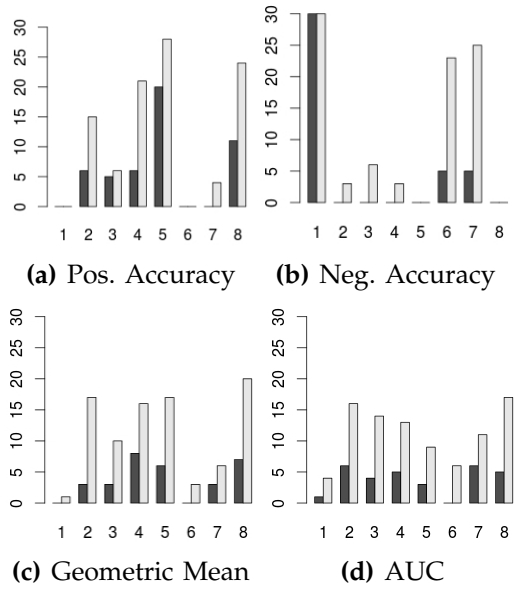
4.2 Results and Discussion

First, we compare *OSS* and *ClusterOSS*. Table 2 shows the number of wins and ties for them. For each line of the table, we show the results for one of the evaluation measures, considering the results for the combination of the 10 datasets and the 3 classification methods. The results show that *ClusterOSS* yields better results for the positive class (minority class), which is the class of interest. The results for the negative class are comparable, with a slight advantage for *OSS*. This shows a trade-off between the performance in the positive and negative classes. However, when we consider measures that evaluate the performance on both positive and negative classes - geometric mean and AUC - *ClusterOSS* outperforms *OSS*.

We now perform a comparative analysis of all 8 methods considered in our experiments. To summarise the results, we count the number of victories for each pre-processing technique in each dataset and in each classification algorithm for different measure performances. The results are shown in Fig. 6 which represents this rank. The dark bar represents the number of victories a technique perform compared to the others, and the white bar represents how many times a technique is in the top 3 performances.

First of all, we can see that the dataset without pre-processing yields the best results for the negative class, but very poor results for the positive class. This is expected, since the imbalance of the datasets leads to a bias w.r.t. the negative class.

Random undersampling presents an opposite behaviour. While, it gives the best performance w.r.t. the positive accuracy, it presents a poor performance w.r.t. the negative accuracy when compared to the other methods. Because of this difference in the results, random undersampling is outperformed by other methods (e.g., *SMOTE* and *ClusterOSS followed by ran-*



(a) Pos. Accuracy **(b)** Neg. Accuracy

(c) Geometric Mean **(d)** AUC

Fig. 6: Best and top 3 best performances. 1:Non pre-processed dataset; 2:oversampling; 3:*CBO*; 4:*SMOTE*; 5:Random undersampling; 6:*OSS*; 7:*ClusterOSS*; 8:*ClusterOSS followed by random oversampling*

Measure	# *SMOTE* victories	# *ClusterOSS* + random oversamplig victories	Ties
Positive Accuracy	18	10	2
Negative Accuracy	9	21	0
Geometric Mean	16	14	0
AUC	15	15	0

TABLE 3: *SMOTE* x *ClusterOSS followed by random oversamplig*

dom oversampling) w.r.t. AUC and the geometric mean.

Even though we saw that *ClusterOSS* outperforms *OSS*, the comparative analysis with all methods shows that *ClusterOSS* does not rank among the best methods. However, when we combine *ClusterOSS* and random oversampling, the results are compared to those o *SMOTE*; these 2 methods being the best ones in a general analysis of the measures.

Now, we have a closer look at the results of *ClusterOSS followed by random oversampling*, and *SMOTE*. We show the comparative analysis of these 2 methods in Table 3. We can see that while *SMOTE* performs better w.r.t. the positive accuracy, *ClusterOSS followed by random oversampling* performs better w.r.t the negative accuracy. For the other 2 measures the results are comparable.

5 Conclusions and Future work

In this paper, we introduced a new undersampling method to pre-process imbalanced datasets. Our method, which we call *ClusterOSS*, outperforms *OSS*, which is the method *ClusterOSS* is based on. Moreover, we showed that when we combine *ClusterOSS* with random oversampling, the results are comparable with those of the state-of-the-art *SMOTE*.

As future work, we will investigate why *SMOTE* performs better than *ClusterOSS* with random oversampling regarding the positive class, in order to see possible directions in which we can improve our results.

Acknowledgments

The authors would like to thank the Brazilian research agencies FAPESP, CAPES and CNPq for financially supporting this work.

References

[1] T. Fawcett, Foster, and F. Provost, "Adaptive Fraud Detection," *Data Mining and Knowledge Discovery*, vol. 1, pp. 291–316, 1997.

[2] S. Stolfo, A. L. Prodromidis, S. Tselepis, W. Lee, W. Fan, and P. K. Chan, "JAM: Java Agents for Meta-Learning over Distributed Databases," in *In Proc. 3rd Intl. Conf. Knowledge Discovery and Data Mining*. AAAI Press, 1997, pp. 74–81.

[3] M. Souza, G. D. C. Cavalcanti, and T. I. Ren, "Off-line Signature Verification: An Approach Based on Combining Distances and One-class Classifiers," in *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, vol. 1, Oct 2010, pp. 7–11.

[4] M. Kubat and S. Matwin, "Addressing the Curse of Imbalanced Training Sets: One-Sided Selection," in *In Proceedings of the Fourteenth International Conference on Machine Learning*. Morgan Kaufmann, 1997, pp. 179–186.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *J. Artif. Int. Res.*, vol. 16, no. 1, pp. 321–357, Jun. 2002.

[6] T. Jo and N. Japkowicz, "Class Imbalances Versus Small Disjuncts," *SIGKDD Explor. Newsl.*, vol. 6, no. 1, pp. 40–49, Jun. 2004.

[7] I. Tomek, "Two Modifications of CNN," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-6, no. 11, pp. 769–772, 1976.

[8] K. Bache and M. Lichman, "UCI machine learning repository," 2014. [Online]. Available: http://archive.ics.uci.edu/ml

[9] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework," 2011. [Online]. Available: http://sci2s.ugr.es/keel/datasets.php