Imbalanced Dataset Techniques to Improve Classification of Uneven Class Distributions

Miguel Jiménez Aparicio
Atlanta, USA
maparicio6@gatech.edu

Belén Martín Urcelay
San Sebastián, Spain
burcelay3@gatech.edu

Cristian Gómez Peces
Atlanta, USA
cpeces3@gatech.edu

***Abstract*—This document analyzes the impact of imbalanced datasets on machine learning classifiers. It discusses several techniques to deal with the imbalance. The results are applied to train a Twitter fake account detection algorithm where the performance of these techniques is compared. It has been shown how state-of-the-art ensemble classifiers lead to a substantial increase in F1-score compared to traditional classifiers while both true and empirical risks are lower.**

***Index Terms*—imbalance, undersampling, oversampling, boosting, bagging, cost-sensitive.**

I. Introduction

A. Motivation

Many canonical machine learning algorithms used for classification assume that the number of objects in the respective classes is roughly the same. However, in reality, classes are rarely represented equally. In fact, class distribution skews are not only common, but many times expected [1], especially in decision systems aiming to detect rare but important cases. For instance, Covid-19 testing at Georgia Institute of Technology showed that less than 1% of the samples contained the virus. This means that a naive classifier could achieve a 99% accuracy just by labeling all samples as negative for Covid-19.

Imbalanced datasets significantly compromise the performance of many traditional learning algorithms. The disparity of classes in the training dataset may lead the algorithm to bias the classification towards the class with more instances, or even to ignore the minority class altogether. Therefore, it is vital to find efficient ways of dealing with data imbalances.

The overall goal of our project is to provide an overview of the state-of-the-art approaches to solve the issues introduced by imbalanced datasets. Including, a performance comparison of the various techniques. We also aim to implement an efficient scheme that is able to deal with highly complex and imbalanced datasets.

This study covers the impact of resampling and cost-weighting techniques to improve the classification. The work on [5], suggests that when combining these techniques with ensemble methods the performance of the classification is highly improved. Hence, we also covered the combination of upsampling and downsampling with ensemble learning algorithms such as bagging and boosting. These algorithms have been implemented from scratch, facilitating the embedding of cost-sensitive frameworks as well in the ensemble learning process.

B. Methodology

Firstly, we study a synthetic dataset characterized by its simplicity. It is made up of two classes following $N(\boldsymbol{\mu}_0, \Sigma_0)$ and $N(\boldsymbol{\mu}_1, \Sigma_1)$), where

$$\boldsymbol{\mu}_0 = \begin{pmatrix} -1 \\ -0.5 \end{pmatrix},\ \boldsymbol{\mu}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix},\ \Sigma_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ \Sigma_1 = \begin{pmatrix} 4 & 0 \\ 0 & 2 \end{pmatrix}$$

and where the minority class only accounts for 15% of the samples. This simple dataset is especially useful to analyze the imbalance-compensating techniques from a mathematical perspective. Not only do we study the concepts learnt in class at a theoretical level, but we also use plugin machine learning models to illustrate how they affect density distributions.

Secondly, we target a more complex dataset, which gather data of more than thirty-seven thousand Twitter accounts to determine wheter they are a bot or human accounts. The number of bot accounts represents a 30% of the total accounts. The dataset contains thirteen features, such as the existence of a profile image, location or favorites and followers count. Some variables are categorical, while others are binary or integer variables.

The performance of the classification will be evaluated using the F_1 score $\in [0,1]$, where the best possible score is 1. This metric is computed as

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

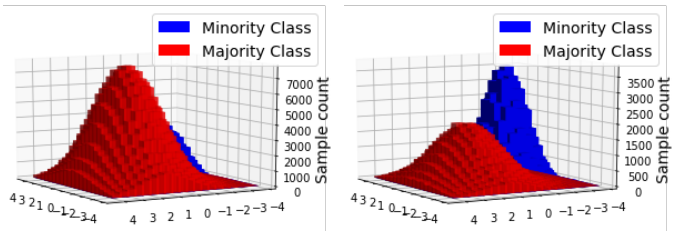
where the precision is the ratio between correctly identified minority samples and the total number of minority samples, while the recall is given by the fraction of correctly identified minority samples over all samples. This metric clearly explains if the classes (including the minoritary) are correctly handled by the model or not.

II. Overview of the techniques

A. Undersampling

Undersampling is frequently employed to balance datasets before any machine learning algorithm is applied. Undersampling involves randomly removing entries from

the majority class. Figure 1 shows the effects of undersampling on the Gaussian training dataset. The class imbalance is somewhat countered. However, the algorithm learnt from this undersampled dataset will be affected. Namely, its ability to generalize and its posterior distribution.



(a) Original. (b) Undersampled. $\beta = 0.25$.

Fig. 1: Gaussian dataset.

1) Generalization Ability: Induction algorithms require a sufficient amount of data to learn a model that generalizes well. If the training set is not large, a classifier may just memorize the characteristics of the training data. Moreover, undersampling has the potential of eliminating valuable samples from consideration of the classifier entirely [2], so it may exacerbate this problem of lack of data. The obtained training set may vary greatly from one undersampling to another, this leads to a high variance of the learned model. Hence, the achievable complexity of the hypothesis set must be reduced to ensure a good generalization.

2) Posterior Bias: One goal of undersampling is to change the priori probabilities of the classes to make them more balanced. The classifier assumes that the features it encounters at testing follow the same distribution as the training set. This mismatch introduced by design is known as sampling selection bias [3] on posterior distribution.

Let $(\mathscr{X}, \mathscr{Y})$ denote the pairs of feature vectors, $\mathbf{x} \in \mathbb{R}^n$, and binary labels, $y \in \{0,1\}$, contained in our original dataset. We assume that the number of samples labeled as zero is small compared with the number of samples in class one. Undersampling randomly removes points from the majority class, we describe this sampling with the binary random variable S, which takes the value 1 if a sample is selected.

It is reasonable to assume that the selection is independent of the features given the class. Then, applying Bayes rule, the law of total probability and noting that the samples from the minority class are always selected we obtain

$$p' = P(y=0|\mathbf{x}, s=1) = \frac{P(s=1|y=0,\mathbf{x})P(y=0|\mathbf{x})}{P(s=1|\mathbf{x})} = \\ = \frac{P(y=0|\mathbf{x})}{P(y=0|\mathbf{x}) + P(s=1|y=1)P(y=1|\mathbf{x})} = \\ = \frac{p}{p + \beta(1-p)},$$

where p and p' denote the posterior probability of encountering a sample from the minority class when employing the original and the undersampled dataset respectively. Whereas β denotes the probability of keeping a sample from the majority class.

The posterior is highly affected by the rate of the sampling. As more samples are removed, the classification is more biased towards the minority class. Figure 2 shows the decision region of a naive Bayes classifier. As the training set is undersampled the region of points that are labeled as the minority class grows. The rate of undersampling not only influences the posterior bias, but also the algorithm's ability to generalize. Thus, β should be chosen with care. Figure 3 presents the average F1-score over different training sets. We observe that the score is concave and in this case the optimum occurs with $\beta = 0.82$. I don't think so

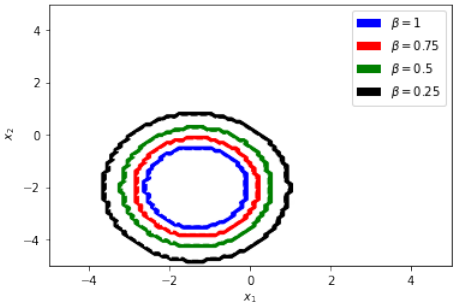


Fig. 2: Influence of undersampling on the classification region of a naive Bayes classifier trained with the Gaussian dataset. The area within each circle corresponds to the cluster of points that are classified as the minority class for a given undersampling rate.

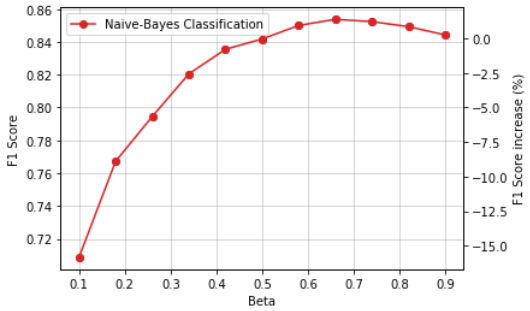


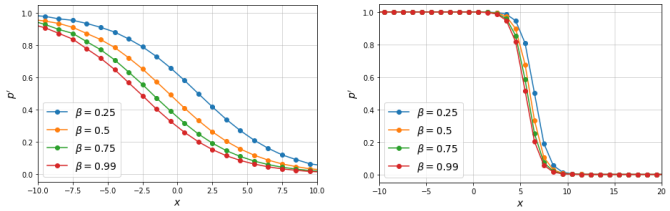
Fig. 3: F1-score vs. undersampling rate.

Another factor that strongly influences the posterior bias is class separability. The bias is higher when conditional distributions are similar across the classes [4]. To analyze this behavior we reduced the problem to a one-dimensional setting, the results are depicted in Figure 4.

We confirm that undersampling shifts the posterior distribution in favor of the minority class. Nevertheless, the shift caused by β is lower under the configuration with lower overlap.

B. Oversampling

Another technique to balance datasets is oversampling, which is artificially creating new intances of the minority

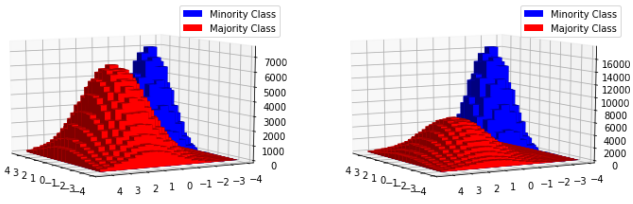


(a) $|\mu_0 - \mu_1| = 3$. (b) $|\mu_0 - \mu_1| = 13$.

Fig. 4: Influence of undersampling on posterior probability of the minority class.

class. This can be done just by cloning existing samples, but this is not really adding new information to the model. More elaborated methods create samples by interpolation. The most widely used is Synthetic Minority Oversampling Technique (SMOTE). It works by selecting a random sample, taking one near sample and creating points in between. This procedure is repeated multiple times until the desired amount of new samples is created.

In Figure 5, it is possible to observe the effect of oversampling on the 2D dataset. For the chosen ratio of 1, the number of samples of both classes is the same.



(a) Original. (b) Oversampled. Ratio=1.

Fig. 5: Gaussian dataset.

A sensitivity analysis for several ratios has been conducted and the optimum ratio is determined to be around 0.55, which means that the minority class is oversampled until the number of minority samples is the 55% of the samples of the majority class. The increase in F1 score for this ratio is almost 0.6%. This can be appreciated in Figure 6.

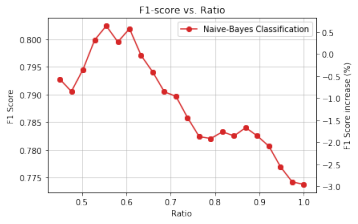


Fig. 6: F1-score vs. oversampling rate.

C. Cost-Sensitive Techniques

Blabla

III. Classification Impact on Real Data

For the rest of the paper, the Twitter dataset will be used. The first attempt was to use the same techniques that have been previously explained to the bi-dimensional dataset. The presence of categorical variables makes unsuitable the oversampling and custom cost-sensitive approaches. Therefore, an approach that relays on undersampling and Naive Bayes is employed. F1-score is 67.83% and the undersampling doesn't make any improvements (even trying different betas). The only solution is to use more complex classifiers that can cope with a dataset like this.

Ensemble Classifiers are known to improve the performance of single classifiers in imbalanced datasets [5] by combining separate weak learners into a composite whole. Both bagging and boosting algorithms have been implemented from scratch to explore their advantages and compare them with highly-effective classifiers such as XGBoost.

A. Bagging

Baggings algortihms are composed by two parts: bootstrapping and aggregation. The first part consist of selecting a subsample of the dataset. In bagging algorithms, weak learners are not trained over the whole dataset, but only over a part. For this task, decission tree classifiers are employed. Once all the classifiers are trained, the predictions of all models are then aggregated to provide the final prediction. Important to mention that these algorithms are purely 'variance' reducing procedures intended to mitigate instability (especially associated to decision trees).

B. AdaBoost

Unlike bagging, boosting fits the weak learners in an adaptive way: a different significance is assigned to each of the base learners based on the samples that were misclassified in the previous iteration. It was first introduced by Schapire in 1990, who proved in [**?**] that a weak learner can be turned into a strong learner in the sense of probably approximately correct (PAC) learning framework. In particular, the AdaBoost algorithm stands out in the field of ensemble learning as one of the top ten data mining algorithms [**?**]. One its main advantages is the versatility to incorporate cost-sensitive techniques. We implemented a custom AdaBoost classifier that enables us to gain control over the algorithm at a lower level, make adjustments when necessary and create other Adaboost-based classifiers such as AdaCost or Boosted SVM. The algorithm is fully detailed below.

This custom implementation uses single decision trees with 2 leaf nodes. It is given the number of maximum iterations (weak learners) that will be fitted, and a dataset D formed by N training samples, each constituted by a feature vector $\boldsymbol{x}_i$ and an associated binary label y_i. For each iteration, a weak classifier is created using the weighted training samples. For the first iteration, we allocate the same weight for all the samples $(1/N)$. Once the decision tree has

Algorithm 1: AdaBoost Algorithm

Input: Training set $D = \{x_i, y_i\}, i = 1,\dots,N$; and $y_i \in \{-1,+1\}$; T: Number of iterations; I: Weak learner

Output: Boosted Classifier: $H(x) = sign\left(\sum_{t=1}^{T} \alpha_i h_t(x)\right)$ where h_t, α_t are the induced classifiers and their significance, respectively

$W_1(i) \leftarrow 1/N$ for $i = 1,\dots,N$

/* Create a weak learner in each iteration */

for *t=1 to T* **do**

  $h_t \leftarrow I(D, W)$

  $\epsilon_t \leftarrow \sum_{i=1}^{N} W_t(i)[h_t(x_i) \neq y_i]$

  if $\epsilon_t > 0.5$ **then**

    $T \leftarrow t - 1$

    return

  $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

  /* Update Weights */

  $W_{t+1}(i) = W_t(i) e^{(-\alpha_t h_t(x_i) y_i)}$ for $i = 1,\dots,N$

  Normalize W_{t+1} such that $\sum_{i=1}^{N} W_{t+1}(i) = 1$

been created, we predict the class of the training samples and compute the weighted misclassification error ϵ_t. Then, the significance α_t is calculated based on that error. Finally, the weights of each training samples are updated depending on the significance and whether the sample was correctly classified or not. Note that the weights are increased by e^{α} if the sample is misclassified or decreased by $e^{-\alpha}$ if correctly classified, with $\alpha_t \in [0,1]$ because $\epsilon_t \in [0, 0.5]$.

AdaBoost uses the whole training samples to create each weak learner serially –in contrast to *Bagging*, where *bootstrap aggregating* is used to construct ensembles– giving more focus to challenging instances and turn the incorrect classifications into good predictions in the next iteration. Hence, AdaBoost can be considered as a "bias" reducing procedure, intended to increase the flexibility of stable (highly biased) weak learners when incorporated properly in a serial additive expansion. That is the reason behind why weak learners with low variance but high bias –such as decision trees– are well adapted for boosting. Lastly the overall literature suggest AdaBoost algorithms are generally resistant to overfitting regardless the number of weak estimators we use, and there are very few cases reported to overfit the training data [**?**]. This fascinating issue is explained due to 1) as the iterations proceed the weak learners tend to have less significance and 2) similarly to SVM, ensemble classifiers maximize the margin which allows promoting good generalization [**?**]. The impact of its complexity on overfitting is covered in the result section.

C. AdaCost

The AdaBoost algorithm is an accuracy-oriented algorithm. In other words, it assumes each class has an even distribution. In case the class distribution is uneven, the algorithm may incur in systematic biases toward the majority class. Therefore, several methods have been proposed to incorporate an asymmetric weight update and eliminate these biases. The literature suggest two different strategies to implement these changes [**?**] 1) modify the model learnt from data or 2) modifying how the model is used to make a decision. AdaCost falls into the former category, and it aims at modifying the update of the weights based on a slight modification:

$$W_{t+1}(i) = W_t(i) \exp\left(-\alpha_t y_i h_t(x_i) \boxed{\phi(i)}\right) \text{ with } W_1(i) = \frac{c_i}{\sum_{i=1}^{N} c_i}$$

Where the new boxed term is called cost-adjustment function. We use this function to allocate different penalizations across different classes. This function is left for design since the literature suggest more than one alternatives. In this project we used the function suggested in [5]:

$$\phi(i) = 0.5 C_i \left(\mathbb{1}\{h_t(\boldsymbol{x}) = 1\} - \mathbb{1}\{h_t(\boldsymbol{x}) = -1\}\right) + 0.5$$

Where C_i is a hyper-parameter that establishes the misclassification cost of the sample i, which ultimately depends upon the class of that sample. This cost-adjustment function yields to an upper bound cumulative misclassification cost equal to $d\prod_{t=1}^{T} Z_t$, where $d = \sum c_i$ and Z_t is the sum of the costs calculated for W_{t+1}, i.e. the coefficient that we use to normalize the weights in AdaBoost [**?**]. Due to the fact that boosting minimizes Z_t, the significance ultimately needs to be updated in a different way [**?**]:

$$\alpha = \frac{1}{2}\ln\frac{1+r}{1-r} \quad \text{where } r = \sum_i D(i) u_i, \quad u_i = y_i h(x_i) \phi(i)$$

All in all, the AdaCost algorithm follows the same procedure as AdaBoost with changes in the significance and weight update rule in order to incorporate an asymmetric penalization cost.

D. AdaMEC

The AdaMEC algorithm falls into the second category introduced in the previous part. The model is trained with the original AdaBoost, but modifies the decision rule by exploiting the bayesian decision theory:

$$H_{AdaMEC}(\boldsymbol{x}) = sign\left(\sum_{y \in \{-1,+1\}} c(y) \sum_{\tau : h_\tau(\boldsymbol{x}) = y} \alpha_\tau h_\tau(\boldsymbol{x})\right)$$

This is called Minimum Expected Cost rule and it assumes that the wighted votes of each weak learners are proportional to the class probabilities. We have incorporated this classifier in this study as it has been reported as one of the best cost-sensitive AdaBoost classifiers [**?**].

E. Boosting SVM

One of the practical concerns about AdaBoost is the accuracy/diversity dilemma [**?**], which means that when the weak learners are actually strong classifiers and they do not disagree too much on their vote, the performance of boosting is weakened. Support vector machines (SVM) are known to be strong learners that minimize structural risk such that they should not work well when embedded. However, the work on ([**?**], [**?**]) suggests that if properly weakened, Boosted SVM can exceed the classification performance of traditional SVMs or AdaBoost on imbalanced datasets, and gain control over the accuracy/diversity issue. To do that, we have used radial basis function kernel (RBF), which uses the Gaussian width σ to map the features on a high dimensional feature space. The performance of RBFSVM can be conveniently controlled by adjusting σ. Large values of sigma will lead to very weak classifiers. The full algorithm description is detailed below:

Algorithm 2: Boosted SVM Algorithm

Input: Training set $D = \{x_i, y_i\}, i = 1,\ldots,N$; and $y_i \in \{-1,+1\}$; T: Maximum number of iterations; The initial $\sigma = \sigma_{ini}, \sigma_{min}, \sigma_{step}$

Output: Boosted Classifier: $H(x) = sign\left(\sum_{t=1}^{T} \alpha_i h_t(x)\right)$ where h_t, α_t are the induced classifiers and their significance, respectively

$W_1(i) \leftarrow 1/N$ for $i = 1,\ldots,N$

/* Create a weak learner in each iteration */

while $\sigma > \sigma_{min}$ *and* $t < T$ **do**

  $t \leftarrow t+1$

  $h_t \leftarrow RBFSVM(D, W, \sigma)$; // Train RBFSVM

  $\epsilon_t \leftarrow \sum_{i=1}^{N} W_t(i)[h_t(x_i) \neq y_i]$

  if $\epsilon_t > 0.5$ **then**

    $\sigma \leftarrow \sigma - \sigma_{step}$

  else

    $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

    /* Update Weights */

    $W_{t+1}(i) = W_t(i) e^{(-\alpha_t h_t(x_i) y_i)}$ for $i = 1,\ldots,N$

    Normalize W_{t+1} such that $\sum_{i=1}^{N} W_{t+1}(i) = 1$

The main disadvantage of SVM is that the running time to train the model does not scale well with the size of the training set [**?**]. Our training set has more than 30,000 samples so the training would take a significant amount of time. Therefore, a preprocessing step was applied to the training data, whose dimensionality is reduced using PCA and ultimately under-sampled to speed-up the training process.

[TO-DO: Add comparison with SVM somehow

F. XGBoost

XGBoost is one of the most effective classification algorithms nowadays, and it has been the winner of many competitions in the last years. The working principle is additive tree learning, which means that what it is learned on an iteration is used to train the next tree of the model. This new tree has to minimize the objective function, that includes a training loss and a regularization term. The second order Taylor expansion is employed, as the actual function can be very complex. The structure of the new tree is going to be optimized to produce the maximum reduction of the objective function. The tree split that gives the maximum similarity between its prediction and the output is the one that it is going to be added.

In practice, it is the classification alogirthm that gives the highest F1-score and accuracy. The XGBoost library is used to train this algortihm for our dataset.

IV. Results and Discussion

This section discusses the performance of each classifier, addresses how these algorithms work for different imbalance rates and show how the number of learners affect the true and empirical risks.

A. Performance Results

The performance results of each methodology are shown in the table below. This table shows the metrics of F1-score, accuracy, precision, recall and AUC in order to compare the algorithms from multiple perspectives. All scores have been obtained with classifiers that had 100 weak learners.

TABLE I: Best Classification Performances

Type of Method	Classifier	Scores (%)				
		F-1	*Acc.*	*Prec.*	*Rec.*	*AUC*
Ensemble	AdaBoost*	84.14	84.55	84.05	84.55	77.91
	Bagging*	76.32	78.49	76.91	78.49	65.58
	R. Forest	70.73	76.96	78.39	76.96	57.71
	BoostSVM*	62.18	73.42	53.91	73.42	0.5
	XGBoost	**88.28**	**88.63**	**88.38**	**88.63**	**82.82**
Cost Sensitive	AdaCost*	77.98	79.86	78.65	79.86	67.71
	AdaMEC	85.15	85.61	85.14	85.61	78.83
Hybrid (using under-sampling)	AdaBoost*	83.30	83.18	83.46	83.18	79.23
	Bagging*	75.07	73.99	77.50	73.99	72.46
	R. Forest	80.44	80.98	80.23	80.98	73.17
	BoostSVM*					
	XGBoost	87.96	88.051	87.90	88.05	83.96
	AdaCost*	78.67	79.17	78.40	79.17	71.18
	AdaMEC	85.10	85.13	85.07	85.13	80.74

*Custom implementation.

In the category of ensemble algortihms, XGBoost outperforms every other algortihm in all metrics. Looking at F1-score, XGBoost is close to 90%. Next algortihm is AdaBoost, whose F1-score is above 84%. The rest of ensemble classifiers are clearly inferior. Regarding cost sensitive algortihms, for this particular dataset they doesn't give any advantage. For hybrid algortihms, which combine ensemble and cost-sensistive algortihms plus undersampling, the optimum beta (the one that gives the maximum F1-score) for each algortihm is selected. A few of the classifiers improve, as it is the case of Random Forest and AdaCost, while the others become a little bit more. However, XGBoost still has the largest F1-score.

B. Impact of Imbalance Data on Classification

This section study how each algortihm work if less samples of the majority class are considered (conceptually, this is similar to undersample the majority class). Figure 7 shows the perfomance of each algortihm for different rates. For example, the rate of 0.5 implies that only one half of the majority class samples are considered.

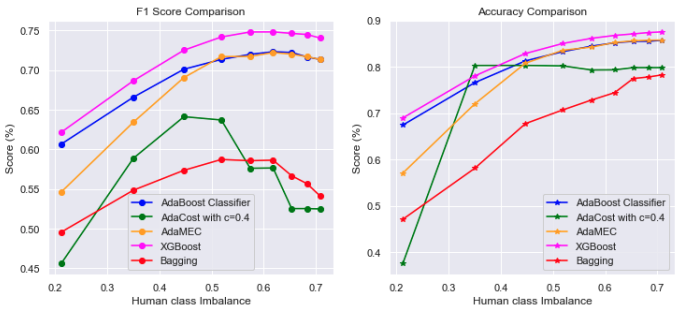


Fig. 7: Impact of imbalanced training set on classification.

It is really interesting to see that most of the algortihms have a peak in the F1-score metric when rates around 0.6 are considered. Taking in account that the majority class is around a 70% of all the samples, using such rate implies that minority and majority class are much closer to be balanced.

C. Overfitting Analysis

Figure 8 shows the evolution of both empirical and true risk (training and test errors, respectively) for increasing algortihm complexity. It is supposed that complexity increases with the number of learners trained by the algortihm, which necessarily involve a larger number of hypotheses. As it can be appreciated, XGBoost is the algortihm that minimizes the true risk. Even with just a few estimators, it is better than any other algortihm. The true risk fastly decreases as the number of learners grow and it becomes estable below 12% around 50 learners.

Regarding other algortihms, AdaBoost and AdaMEC tend to be more and more similar when the number of weak learners increase. Not being comparable to XGBoost, both performances are satisfactory. However, AdaMEC seems to be always better. Then, AdaCost, Bagging and Random Forest doesn't have any great improvement when the number of learners increases and the true risk is quite high.

It is noticeable that the true risk remains stable after a certain number of learners, while the empirical risk always seems to be decrease (this is specially visible in XGBoost). This occurs because the algorithm learns patterns that are extremely specific to the training set, but which are not generalizable to the test set. Therefore, very little improvements are seen on the true risk. In the other algortihms, the different between empricial and true risk is not as significant, but the true risk always remain higher than the empirical risk.

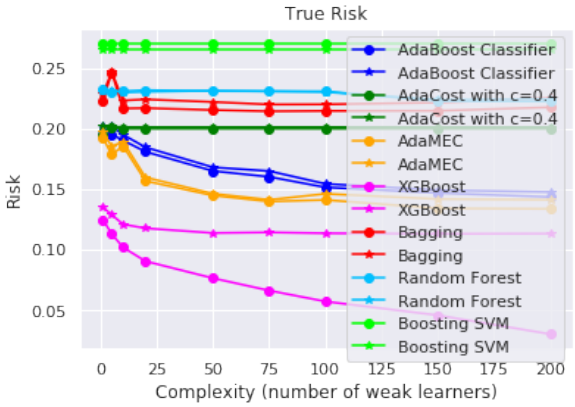


Fig. 8: True and empirical risks vs. complexity.

V. Conclusions

This paper has shown how classification on imbalanced datasets can be addressed. There are methods that change the dataset on a data level, such as oversampling or undersampling, while others try to develop more complex classifiers that are more suitable for this sort of problems. In the first dataset, which only had two features, the first type of methods were employed with satisfactory results. The F1-score, which is an excellent metric for imbalanced datasets, was higher. However, applying the same method to the second dataset didn't give a significant improvement. Consequently, several ensemble and hybrid classifiers have been tested (some of them have been customly implemented). It has been determined that a substantial improvement of 20% on F1-score can be achieved in relation to a basic Naive Bayes classifier. The ensemble classifiers that give the maximum F1-score are, first, XGBoost, and secondly, AdaMEC. It has been shown that boosting algortihms are superior than bagging algortihms, which are the other type of ensemble algorithms. Also, for this particular dataset, hybrid methods are less effective than ensemble methods. In addition, this paper has also explored the versatily of these algortihms against the number of weak-learners and the rate of imbalanced dataset. It was determined that XGBoost is the most resilient to both various imbalance rates and it has the lower empirical and true in a broad spectrum of weak learners.

References

[1] V. S. Spelmen and R. Porkodi, "A review on handling imbalanced data," in *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, pp. 1–11, 2018.

[2] M. Wasikowski and X. Chen, "Combating the small sample class imbalance problem using feature selection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1388–1400, 2010.

[3] Q.-C. Joaquin, *Dataset shift in machine learning*. MIT, 2009.

[4] A. D. Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi, "Calibrating probability with undersampling for unbalanced classification," in *2015 IEEE Symposium Series on Computational Intelligence*, pp. 159–166, 2015.

[5] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 4, pp. 463–484, 2012.