Data mining with imbalanced class distributions: concepts and methods

Ronaldo C. Prati[1], Gustavo E. A. P. A. Batista[2], and Maria Carolina Monard[2]

[1] Centro de Matemática, Computação e Cognição — CMCC
Universidade Federal do ABC — UFABC
ronaldo.prati@ufabc.edu.br
[2] Departamento de Ciência da Computação — SCC
Instituto de Ciências Matemáticas e de Computação — ICMC
Universidade de São Paulo, Campus de São Carlos — USP/São Carlos
{mcmonard,gbatista}@icmc.usp.br

Abstract. Some real world data mining applications present imbalanced or skewed class distributions. In these domains, the underrepresented classes are often the ones we are more interested in. However, most learning algorithms are not able to induce meaningful classifiers in some imbalanced domains. One reason for this poor performance is that learning algorithms tend to focus in abundant classes to maximize classification accuracy. This paper reviews recent work in this subject, focusing in concepts and methods to deal with imbalanced data sets.

1 Introduction

Inducing classifiers from data sets having skewed class distributions is a problem frequently encountered in the data mining process. In numerous applications, the relative and/or absolute number of some classes might be heavily outnumbered by the frequency of others. Some examples are credit card fraud detection, where the number of fraudulent operations is much lower than the number of non-fraudulent ones [1]; rare disease medical findings, where the number of patients having the disease is very low in the population [2]; and continuous fault-monitoring tasks where non-faulty cases heavily outnumber faulty cases, to name but a few.

This problem is frequently referred to in the literature as the "class imbalance" problem, as numerous studies point out degradation in performance of the models extracted from skewed domains, especially when predicting the low represented (minority) classes. This poor performance regarding the minority classes is very undesirable, as they are often the classes we are more interested in. Even though class imbalance is a problem of great importance in data mining, a complete understanding of how this problem affects the classifiers' performance is not clear yet.

The objective of this paper is to review some of the most important concepts and methods related to the problem of learning in the presence of class imbalance. We start by providing an overview about the problem of learning with imbalanced classes (Section 2); and we review some of the most used strategies to treat imbalanced classes, such as sampling (Section 3), cost sensitive learning (Section 4) and other approaches such as ensembles and one-class learning (Section 5). We also discuss another important related topic, the evaluation of classifiers with imbalanced data (Section 6), and present our concluding remarks (Section 7).

2 The class imbalance problem

Learning algorithms are widely used during the pattern extraction phase of the data mining process. As this process deals with “real-world” data, several problems of applying existing and well-established learning algorithms to real data have emerged. Among them, a relevant practical problem is learning in the presence of class imbalances. Many learning algorithms were designed assuming well-balanced class distributions, *i.e.* no significant differences in class prior probabilities. However, this is not always the case in real world data where one class might be represented by a large number of examples, while the others are represented by only a few.

Generally, the problem of imbalanced data sets occurs whenever one class represents a circumscribed concept, while the other represents the counterpart of that concept, so that examples from the counterpart class heavily outnumber examples from the positive concept class. In this case, the inductive bias of learning algorithms which are not specially designed to deal with class imbalances, tends to focus in the class which is represented by the largest number of examples.

Several research papers have reported class imbalances of 1% in the minority class and 99% in the majority one, and upwards. In such scenarios, learning algorithms tend to induce classifiers having very small overall error rates, by simply classifying every new example as belonging to the majority class. It is clear that such classifiers are useless, since the minority class with rare cases is the one we are usually interested in predicting well.

For example, in a medical diagnosis problem of a certain rare cancer, diagnosing a cancer patient as healthy is much more serious than diagnosing a healthy patient as cancerous. In the former case, the patient could lose his/her life due to the lack of treatment, while in the latter case posterior exams could correctly classify the patient as healthy. In intrusion detection, not identifying a fraudulent connection in order to deny connection is more serious than denying access to a legitimate connection. Similarly, in spam detection, we rather prefer to receive some span messages than to lose a valid one. In general, class imbalance is intrinsic to domains where some facilities are offered, such as bank loans and credit cards usage, where the majority of the people make an honest use of the offered facilities. Furthermore, class imbalance could also be present in non intrinsic problems, in case the data collection process is limited due to privacy or economic reasons.

In the last years, a considerable number of papers have been published in the machine learning literature aiming at dealing with class imbalance in order to overcome this problem. Workshops related to this matter were initially sponsored by the American Association for Artificial Intelligence (AAAI) [3], followed by another workshop held together with the Twentieth International Conference on Machine Learning (ICML'03) [4], and a more recent one held together with the Thirteenth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-09) [5]. The SIGKDD Explorations also dedicated an special issue to the subject [6]. Furthermore, in the last years papers related to class imbalance are usually present in the major conferences.

In general, research on this topic has mainly focused on solutions at the data and the algorithmic levels. Data level methods consist of balancing classes by resampling the original data set, such that under-represented classes are over-sampled, and over-represented classes are under-sampled. Two widely used sampling methods are random over-sampling and random under-sampling. Random over-sampling balances a data set by making exact copies of randomly chosen minority cases, and random under-sampling randomly discards majority class cases. One critic to over-sampling is that, as this method makes exact copies of examples from the minority class, it can increase the likelihood of overfitting. On the other hand, under-sampling can eventually discard data potentially important for learning. Both sampling strategies, as well as other methods which combine different sampling strategies to achieve further improvements, are discussed in Section 3.

There are different proposals addressing the class imbalance problem from an algorithmic point of view, mainly by adapting existing algorithms and techniques to the especial characteristics of imbalanced data. These proposals include cost-sensitive learning, one-class classifiers, and ensembles of classifiers, among others.

The goal of cost-sensitive learning, discussed in Section 4, is to minimize the cost of misclassification. Class imbalance can be handled in a similar manner by assigning higher classification costs to the classes represented by only a few examples. However, although misclassification costs might be transformed into a class distribution by adjusting the expected class ratio, a complicating factor is that the costs associated to each misclassification are not generally know in advance.

Instead of having more than one class, as in the standard classification problem, one-class classification leads with one class of objects, known as the target class, which should be distinguished from all other possible objects, known as the outlier class. It is assumed that examples from the target class are well represented while examples from the outlier class are scarce or can be totally absent. This general idea has been applied to the class imbalance problem.

Ensemble of classifiers is already a well-established research line. An ensemble consists of a set of individually trained classifiers whose decisions are combined to classify new examples. In class imbalance, ensembles have mainly been used to combine the results of several classifiers, where each one of the classifiers has been induced after over-sampling or under-sampling the data with different over/under-sampling rates. Approaches to deal with class imbalance based on ensembles or one-class learning are discussed in Section 5

As a final remark, it is worth noting that although class imbalance is often reported as an obstacle to the induction of good classifiers by learning algorithms, it has been observed that learning algorithms are able to achieve meaningful results in some domains, even in the presence of highly imbalanced data sets. Therefore, it does not seem fair to directly correlate class imbalance to the loss of performance of learning algorithms. The argument against a direct relationship between class imbalance and classifiers performance degradation is treated in [7], where a broad empirical study is carried out to question whether class imbalances are a problem by themselves and are truly to blame for the loss of performance of learning algorithms. The experimental results suggest that the problem is not directly caused by class imbalances, but is also related to the degree of overlapping among the classes. Furthermore, a good understanding of this correlation would be useful in the analysis and development of tools to treat imbalanced data or in the (re)design of learning algorithms for practical applications.

3 Sampling

Sampling methods are a direct approach to tackle the problem of class imbalance. These methods sample a data set in order to alter the class distributions, aiming to obtain a more balanced distribution. Two well-known sampling methods to deal with the problem of class imbalance are random over-sampling and random under-sampling, where:

Random over-sampling is a non-heuristic method that aims to balance class distribution through the random replication of minority class examples.

Random under-sampling is also a non-heuristic method that aims to balance class distribution through the random elimination of majority class examples.

As previously mentioned, as random over-sampling makes exact copies of examples from the minority class, it can increase the likelihood of overfitting. On the other hand, random under-sampling can eventually discard data potentially important for learning.

Numerous methods have been proposed to overcome the shortcomings of both random under and over-sampling. An approach frequently used to overcome the limitations of random under-sampling is the use of heuristics to identify and remove training examples that might be less important for learning. For instance, majority class examples that are noisy or redundant might be removed from the training set, contributing to balance the classes. In the case of random over-sampling, a frequently used approach is to interpolate minority class cases that lie together. As these new interpolated examples are not exact copies of the original examples, the likelihood of overfitting is reduced. These methods are discussed in more details in the next sections.

3.1 Heuristic under-sampling methods

Heuristic under-sampling methods use two basic approaches: one class of methods aims to identify and remove examples that are considered less important for learning, such as examples that lie far away from the decision border. A second class of methods acts more as data cleaning methods, aiming to identify noisy examples such that their removal would be beneficial for learning. Usually, these data cleaning methods are applied as under-sampling methods, *i.e.*, only majority class examples are removed, even when a minority class example is suspicious of being noisy. An argument in favor of removing only majority class cases is that minority class cases are rare, and the removal of some of these cases would increase the imbalance degree between the classes.

A short description of some of the most used heuristic under-sampling methods follows:

NearMiss NearMiss is a class of methods proposed in [8]. The main idea behind the NearMiss methods is to select a set of majority class cases that are close to the minority class cases in order to better represent the decision border. NearMiss-1 selects majority class examples which have the smallest average distance to the three nearest minority class examples; NearMiss-2 selects majority class examples which have the smallest average distance to the three farthest minority class examples; and, NearMiss-3 selects a predefined number of majority cases that are the nearest neighbors of the minority class cases. Some experimental evaluation performed in [8] shows that NearMiss-2 provides good results.

Condensed Nearest Neighbor Rule Hart's Condensed Nearest Neighbor Rule (CNN) [9] is used to find a consistent subset of examples. A subset $\hat{E} \subseteq E$ is consistent with E if using a 1-nearest neighbor, $\hat{E}$ correctly classifies the examples in E. An algorithm to create a subset $\hat{E}$ from E as an under-sampling method is the following [10]: first, randomly draw one majority class example and all examples from the minority class and put these examples in $\hat{E}$. Afterwards, use a 1-NN over the examples in $\hat{E}$ to classify the examples in E. Every misclassified example from E is moved to $\hat{E}$. It is important to note that this procedure does not find the smallest consistent subset from E. The idea behind this implementation of a consistent subset is to eliminate the examples from the majority class that are distant from the decision border, since those examples might be considered less relevant for learning.

Tomek links Tomek links [11] can be defined as follows: given two examples E_i and E_j belonging to different classes, and the distance $d(E_i, E_j)$ between E_i and E_j. A (E_i, E_j) pair is called a Tomek link if there is not an example E_l, such that $d(E_i, E_l) < d(E_i, E_j)$ or $d(E_j, E_l) < d(E_i, E_j)$. If two examples form a Tomek link, then either one of these examples is noise or both examples are in the borderline. Tomek links can be used as an under-sampling method or as a data cleaning method. As an under-sampling method, only examples belonging to the majority class are eliminated, and as a data cleaning method, examples of both classes are removed.

One-sided selection One-sided selection (OSS) [10] is an under-sampling method resulting from the application of Tomek links followed by the application of CNN. Tomek links are used as an under-sampling method and removes noisy and borderline majority class examples. Borderline examples can be considered "unsafe" since a small amount of noise can make them fall on the wrong side of the decision border. CNN aims to remove examples from the majority class that are distant from the decision border. The remainder examples, *i.e.* "safe" majority class examples and all minority class examples, are used for learning.

Neighborhood Cleaning Rule Neighborhood Cleaning Rule (NCL) [12] uses the *Wilson's Edited Nearest Neighbor Rule (ENN)* [13] to remove majority class examples. ENN removes any example whose class label differs from the class of at least two of its three nearest neighbors. NCL modifies the ENN in order to increase the data cleaning. For a two-class problem the algorithm can be described in the following way: for each example E_i in the training set, its three nearest neighbors are found. If E_i belongs to the majority class and the classification given by its three nearest neighbors contradicts the original class of E_i, then E_i is removed. If E_i belongs to the minority class and its three nearest neighbors misclassify E_i, then the nearest neighbors that belong to the majority class are removed.

3.2 Heuristic over-sampling methods

As previously mentioned, random over-sampling might increase the likelihood of overfitting, since this method creates exact copies of minority class examples. Heuristic over-sampling methods use interpolation of minority class examples that lie together to avoid the creation of exact copies. In addition, over-sampling methods can be allied to data cleaning methods in order to remove noisy examples from both classes. A short description of some of the most used heuristic over-sampling methods follows.

SMOTE Synthetic Minority Over-sampling Technique (SMOTE) [14] is an over-sampling method with synthetic data generation. Its main idea is to form new minority class examples by interpolating between several examples from the minority class that lie together. More specifically, SMOTE randomly selects a minority example, E_i, and its k nearest neighbors. An example E_j is selected from the set of nearest neighbors and a new example is created using the following equation:

$$E_{new} = E_i + (E_j - E_i)\delta$$

where δ is a randomly chosen constant in the interval $[0, 1]$. It is important to note that as the selected nearest neighbor might be from a class different than the class of E_i, SMOTE can increase the minority class space, allowing the creation of synthetic examples that spread further into the majority class space.

SMOTE + Tomek links Although over-sampling minority class examples can balance class distributions, some other problems usually present in data sets with skewed class distributions are not solved. Frequently, class clusters are not well defined since some majority class examples might be invading the minority class space. The opposite can also be true, since interpolating minority class examples can expand the minority class clusters, introducing artificial minority class examples too deeply into the majority class space. Inducing a classifier under such a situation can lead to overfitting. In order to create better-defined class clusters, Tomek links can be applied to the over-sampled training set as a data cleaning method. Thus, instead of removing only the majority class examples that form Tomek links, examples from both classes are removed. The SMOTE + Tomek links method was first used to improve the classification of examples for the problem of annotation of proteins in Bioinformatics [15].

SMOTE + ENN The motivation behind this method is similar to SMOTE + Tomek links. ENN tends to remove more examples than Tomek links does, so it is expected that it will provide a more in depth data cleaning. Differently from NCL which is an under-sampling method, ENN is used to remove examples from both classes. Thus, any example that is misclassified by its three nearest neighbors is removed from the training set. SMOTE + ENN is empirically evaluated and compared with other sampling methods in [16].

Borderline SMOTE Borderline SMOTE [17] is a variation of SMOTE that gives priority data generation for minority borderline examples. This method first uses the k-nearest neighbor algorithm to identify the k nearest neighbors of each minority class example. If a minority class example E_i has more than $\frac{k}{2}$ nearest neighbors from other classes, then E_i is considered a borderline example that might be misclassified, and E_i is fed to SMOTE in order to create synthetic examples. An interesting detail about Borderline SMOTE is that if E_i has exactly k nearest neighbors from other classes, then E_i is considered noise and no synthetic examples are generated for E_i.

4 Cost sensitive learning

An alternative approach to deal with skewed class distributions is to take misclassification costs into account. Most learning algorithms assume that different misclassification errors are equally costly. However, in many real-world applications this assumption does not hold. This is especially true in imbalanced domains, where incorrectly classifying an instance of the minority class as belonging to the majority class is generally more expensive than incorrectly classifying an instance of the majority class as belonging to the minority class.

Cost sensitive learning takes misclassification costs into account. Instead of minimizing classification error rate (a common goal for most machine learning algorithms), algorithms based on cost sensitive learning aim at minimizing misclassification costs. As domains with class imbalance are often accompanied by different misclassification costs, cost sensitive learning might be applied to overcome the class imbalance problem [18].

The theory beneath cost sensitive learning has been mostly formalized in [19]. Cost sensitive learning works as follows: without loss of generality, consider a binary classification problem, where the two classes are given the general labels positive (+) and negative (-). Costs are not necessarily monetary; they can also refer to time wasted or the severity of an illness, for instance. Let c_{ij} (where i and j, $i \neq j$, designate either the positive or the negative class) be the associated cost of mistakenly classifying an instance of class i as belonging to class j. Notice that c_{ii} (where i can be either the positive or the negative class) designate the "benefit" when an instance is correctly classified. Usually, in imbalanced domains, the positive class is generally associated to the minority class and, as discussed early, mistakenly classifying an instance of the minority class as belonging to the negative class is more expensive than the other case. In other words, $c_{+-} >> c_{-+}$. Costs and benefits can be tabulated into a cost matrix, as shown in Table 1.

Table 1. Cost matrix for a binary classification problem

	actual negative	actual positive
predicted negative	c_{--}	c_{+-}
predicted positive	c_{-+}	c_{++}

Given this cost matrix, the optimal decision for classifying a given example x is to assign it to the class i that minimizes the expected cost $C(x, i)$, as shown in Equation 1

$$C(x,i) = \sum_{j} p(j|x)c_{ij} \tag{1}$$

There are three different ways in which a non cost sensitive learning algorithm can be adapted into a cost sensitive version:

– Directly introducing the cost information into the algorithm, so that the classifier is induced taking this cost information into account.
– Weight or sample the instances taking the cost information into account.
– Construct a score classifier without taking the costs into account, but thresholding the score using the cost information.

In the first case, the most common approach is to, instead of trying to minimize the overall error rate (as most of the learning systems do), minimize the overall expected cost [20], as shown in Equation 2.

$$EC = \sum_{i=0}^{n} C(x,i) \tag{2}$$

where n is the number of examples.

Other approaches try to use the cost information for building the model, as in [21], which uses the cost information to construct a splitting criteria for decision tree building which minimizes the cost; or to construct algorithms which are no or little affected by costs, as the cost insensitive decision tree splitting criteria [22, 23].

Sampling first modifies the class distribution of the training data set by removing instances of the negative class or by duplicating instances of the positive class as discussed in Section 3. However, the cost information is used to determine the final class proportion in the training data set. For instance, let the class ratio c of the instances be the number of positive instances (pos) divided by the number of negative instances (neg), *i.e.*, $c = pos/neg$. The sampling is performed so that the class ratio in the training data is adjusted to $c' = \frac{pos * c_{+-}}{neg * c_{-+}}$. On the other hand, if weights are used, sampling is not carried out. Instead, a weight is associated to each instance so that the weighed class proportion matches c'. Theoretically, both approaches are equivalent, although sampling does not require a modification in the learning algorithm, while weighting does. Non naïve approaches use elaborated sampling techniques [24] or a combination of multiple classifiers induced with different class proportions [25], although the general idea is to induce a cost sensitive classifier by directly or indirectly (using weights) changing the class proportion.

Finally, the thresholding involves the use of classifiers which not only produces a "crisp" classification, but also gives a score or probability information on how confident they are in classifying an instance. In this approach, no modification is performed in the learning algorithm or in the training set. However, the cost information is used to "adjust" the classifier. If the classifier is very conservative in assigning a positive classification, the threshold is very restrictive and can be adjusted to better reflect the misclassification costs. The threshold is then set to be adjusted to the cost weighted class distribution of instances c'.

Although cost information can be used to alleviate the class imbalance problem, in many real world applications costs are very difficult to estimate. Furthermore, most of the cost sensitive approaches consider that cost is uniform for all instances. On the other hand, some of the sampling methods described in Section 3 are able to treat individual instances in a different way (for example, instances near the borderline might be more important for classification than instances near the class centre).

5 Other methods

Other methods have been proposed in the literature to deal with the class imbalance problem. Some of these methods use hybrid approaches such as combining different classifiers induced using differently sampled training sets into an ensemble. In [26], several samples are generated from the training set. Generally, these samples contain all minority instances and a subset of majority instances. Majority instances were sampled so that each instance belonging to this class occurs in a different training set at least once. A learning algorithm is then applied to each sample to generate a classifier, and the classifiers are combined using meta-learning.

MetaCost [25] uses bagging to estimate class probabilities and then re-labels the training instances with the class that minimizes the expected cost. A classifier can then be induced by any learning algorithm using this modified training set. Boosting uses weights to focus on "difficult" instances when inducing a classifier. It is an iterative process that, at each iteration, puts more weight into instances that are mistakenly classified. In [27], a well known boosting algorithm, AdaBoost, was modified to incorporate costs in the algorithm AdaCost. In [28], SMOTE and Boosting are combined into the SMOTEBoost algorithm. Instead of weighting, SMOTEBoost alters the distribution by adding new minority-class examples using the SMOTE algorithm.

One class learning or novelty detections algorithms were also applied to handle the class imbalance problem. [29] notice that, in their experiments, one class SVMs outperforms traditional SVM in imbalanced domains. A hybrid approach which first uses one class learning and then binary class learning was developed by [30]. A novelty detection approach was used in [31] with severe imbalance data sets. Their results show that, in these conditions, novelty detection outperforms traditional SVMs.

6 Evaluation

Classification accuracy or misclassification error rate are the most common metrics used to evaluate learning system. These metrics calculate the fraction of test instances which the classifier predicts correctly or incorrectly. Although widely adopted, this approach has some drawbacks when applied to imbalanced domains. This is because accuracy or error rate place more weight on the most common classes, and errors in the minority classes have little impact in these measures. For instance, it is very easy to achieve an accuracy of 99.9% (or equivalently, 0.1% error rate) in a domain where the majority class has 99.9% prevalence, by always predicting the majority class. Although at a first glance a 99.9% accuracy is quite impressive, such a naïve classifier is worthless, since in domains having imbalanced class distributions we are often interested in correctly classifying instances belonging to the minority classes.

This problem occurs in many imbalanced domains. Although the overall accuracy is high or the error rate is low, the predicted positive value — PPV — (a.k.a. precision of the positive class prediction, the percentage of instances predicted as positive that in fact belongs to the positive class) or its true positive rate — TPR — (a.k.a. recall, the percentage of instances that belong to the positive which are classified as positive) are low. Therefore, it is desirable to use different metrics to evaluated classifiers in imbalanced domains.

For a binary problem, the performance of a classifier can be fully characterized using a two-by-two contingency table (or confusion matrix), as shown in Table 2, alongside some metrics that can be derived from this table which are commonly used to evaluate learning systems. In this table TP, FP, TN

and FN represent true/false positive/negative example counts, respectively, pos, neg, $ppos$ and $pneg$ represent the number of examples actual/predicted as positives/negatives, respectively, and n is the data set size. A true positive is an instance that belongs to the positive class which is classified as positive. Similarly, a false positive is an instance that belongs to the negative class, but it is mistakenly classified as positive. True/false negatives can be defined in a similar fashion.

Table 2. A two-by-two contingency table for a binary classification problem

	actual negative	**actual positive**	
predicted negative	true negative (TN)	false negative (FN)	$pneg$
predicted positive	false positive (FP)	true positive (TP)	$ppos$
	pos	neg	n

$$Accuracy = \frac{TP+TN}{n} \quad Error\ rate = \frac{FP+FN}{n}$$

$$PPV = \frac{TP}{ppos} \qquad PNV = \frac{TN}{pneg}$$

$$TPR = \frac{TP}{pos} \qquad FPR = \frac{FP}{pos}$$

Instead of using accuracy, which can be seen as a weighted mean between PPV and PNV, where weights are proportional to the prevalence of each class, a common approach is to calculate the geometric [32], arithmetic [26] or harmonic means of PPV or PNV. When misclassification costs are known, the minimum expected cost can be used as a performance measure. This minimum expected cost can be seen as a weighed mean of the error rate, using costs as weights. Although these approaches might alleviate the problem of the weight that prevalent classes have into accuracy, they impose an arbitrary commitment between the two kinds of errors. This is because performance evaluation is inherently a multifaceted task, and any attempt to reduce the performance evaluation to a single scalar number might lose some information. Although attractive from a practical standpoint, this reduction will necessarily give incomplete pictures of prediction performance.

A complete and reliable performance analysis must consider all the various components of performance quality. A way to accomplish this is to use a graph, such as the ROC graph. A ROC graph is a plot of the false positive rate (FPR) *versus* the true positive rate (TPR). A binary classifier produces a single point with coordinates (FPR, TPR) in the ROC space. Some points in the ROC space are worth noticing. The lower left corner $(0, 0)$ represents the classifier which always predicts negative; such a classifier does not produce any false positive errors, although it is not able to classify any positive ones as well. The upper right corner $(1, 1)$ represents the opposite strategy of unconditionally classifying every case as positive. The upper left corner $(0, 1)$ represents perfect classification, while the lower right corner $(1, 0)$ represents the always wrong classifier. Any classifier which performs no better or worse than chance falls in a point in the ascending diagonal. Classifiers which perform worse than random appear in the lower right triangle formed by the points $(0,0), (1,1)$ and $(1,0)$. For this reason, this area is usually empty. A point in the ROC space is better than another point if it is to the northwest (TPR is higher and FPR is lower) than the first one.

One advantage of ROC graphs is that they allow visualizing and organizing classifiers' performance without considering class distributions or misclassification costs. This ability is very important when investigating learning algorithms in skewed class distributions or cost-sensitive learning situations. The performance of a set of classifiers can be graphed, and as long as the class conditional likelihoods do not change, the graph will remain invariant with respect to the class skew and misclassification costs (operational conditions). As these operational conditions change, the region of interest may change, but the graph itself does not change.

In [33] it is shown that the operating conditions may be easily transformed into the so-called expected cost iso-performance line in the ROC space. Two points in this space, (fpr_1, tpr_1) and (fpr_2, tpr_2), have the same expected cost if

$$\frac{tpr_2 - tpr_1}{fpr_2 - fpr_1} = \frac{c_{10} \times pos}{c_{01} \times neg} = m \tag{3}$$

This equation defines the slope m of a total cost iso-performance line. All classifiers lying at points on a line of slope m have the same expected minimum cost. Each set of class priors and misclassification costs defines a family of iso-performance lines. The "more northwest" a line is (having a larger TP-intercept) the better, because it corresponds to classifiers with lower expected costs. This implies that, regardless of operational conditions, a classifier is potentially optimal if and only if it lies on the convex hull of the set of points in the ROC space. The convex hull of the set of points in the ROC space is called the ROC convex hull (ROCCH) of the corresponding set of classifiers.

An example of a ROC graph is shown in Figure 1. In this graph, five hypothetical classifiers are depicted: A, B, C, D and E. The graph also shows the ROC convex hull (ROCCH). The convex hull is bounded only by the trivial points $(0,0)$ and $(1,1)$ and by the points A, B and C. Points D and E are not in the ROCCH and therefore are sub-optimal. Thus, as we are looking for optimal classification performance, classifiers D and E may be entirely removed from consideration. This is to say that there are no combinations of class and cost distributions in which classifiers D and E present lower expected costs than A, B or C.

However, the choice among A, B and C depends on information regarding operational conditions. For example, the slope of the line segment connecting the origin to point A is 6.5. If committing a false positive error is at least 6.5 times more costly than committing a false negative, or the proportion of negative cases is at least 6.5 times greater than the positives, or any combination of these two factors which leads to a slope m (Equation 3) greater than 6.5, the trivial classifier of never issuing a positive classification (the point $(0,0)$ in the graph)

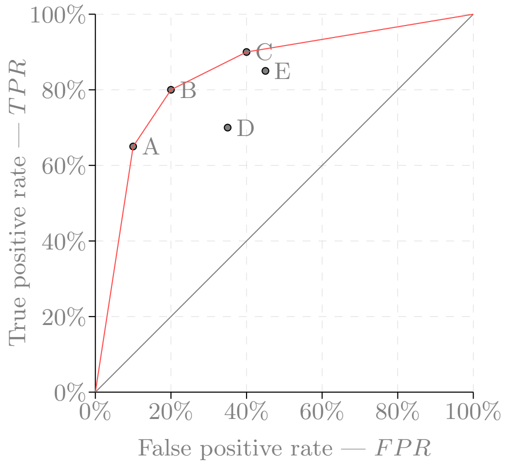

Fig. 1. ROC graph

would be preferred for A, B and C. If the slope is exactly 6.5, either the trivial classifier of always classifying everything as negative or the A classifier would have the same expected cost. The difference is how these two approaches perform in the different classes, but the expected cost (which is a weighted average of each class performance) is the same. In fact, we can achieve any point in between these two classifiers by randomly alternating between them. The points can be obtained by varying the probability in which we choose one of the classifiers.

A binary "crisp" classifier — one that predicts only the class label — produces a single point, represented by the pair (FPR, TPR) in the ROC space. On the other hand, if a classifier produces a score that can be mapped to a ranking function, this ranking could be thresholded to produce a binary classifier by predicting the top n-percent of the cases as positive. By varying this percentage from 0% to 100%, we can produce various points so that a curve in the ROC space can be traced. In other words, we can think of a ROC curve as a parameterized set of classifiers, where each parameter value produces a point in the ROC space. The curve is obtained by joining all these points.

The sharper the curve bends, the greater the ability in putting positive cases at the top of the rank. The crisper the curve flattens towards the (0,1)–(1,1) line, the greater the ability in leaving negative cases at the bottom part of the rank. If the curve hits the upper left corner (0,1) point, there is a perfect ranking. An example of two ROC curves is shown in Figure 2. For each curve, the two probabilities vary together from the lower left corner, where both true and false positive rates are near 0, as they would be for a very strict threshold, to the upper right corner, where both rates are near 1, as they would be for a very lenient threshold. In between, the curve would rise smoothly, with a decreasing slope, to represent all possible thresholds. Hence, the curve is independent of whatever threshold is chosen in a particular task.

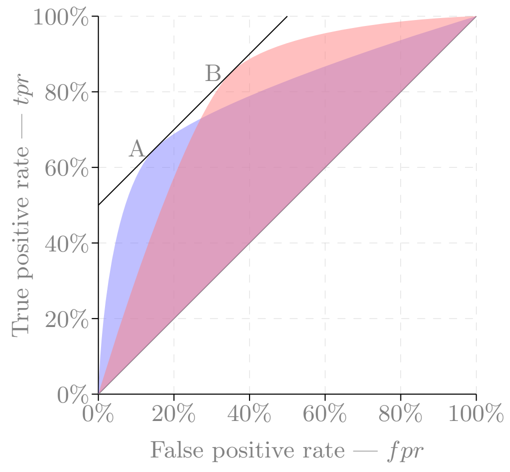

Fig. 2. ROC curves

Analyzing the curves, we can conclude that curve A is better at the top part of the rank (it is better in grouping positive cases at the top of the rank), and thus it might be appropriate in problems such as information retrieval, where we are interested in classifying the positive cases better. Curve B is better at the bottom of the rank (it is better in grouping negative cases at the bottom of the rank), and thus might be interesting in, for instance, some medical problems where a cheap test can be used to exclude people who do not have a certain disease and a more costly test can be further applied to the remaining people. The line tangential both curves is the line where the expected cost is the same for both models.

A single measure of ranking performance can be derived by calculating the area under the ROC curve (AUC). This area can be interpreted as a randomly chosen positive case being ranked higher than a randomly chosen negative case. Furthermore, the AUC is numerically equivalent to the Wilcoxon signed rank test and correlated to the Gini index [34]. However, the AUC also has some drawbacks: for example, both curves in Figure 2 have the same AUC value, although one might be better than the other, depending on the circumstance. This is because the AUC integrates over all possible thresholds, and thus treats equally misrankings at the top or at the bottom of the ranking [35].

7 Concluding remarks

The class imbalance problem occurs in numerous real-world data mining applications where one of the classes is heavily outnumbered by the other classes. In some domains where this class imbalance occurs, learning algorithms tends to misclassify instances of the minority classes rather than instances of the majority classes. This is due to the fact that a common goal of these algorithms is to maximize classification accuracy. Under this goal, when in doubt about which class assign to a new instance, classifying it as belonging to a majority class is “the right thing to do”, as it majors the probability of correctly classifying an instance.

However, in domains where class imbalance occurs, it is often the case that incorrectly classifying an instance from the minority class is much more serious than incorrectly classifying an instance from the majority class. Therefore, in these domains, class imbalance might be a problem as even though a naïve classifier which always predicts the majority class will have a high accuracy, it is useless for identifying instances from the minority classes.

Numerous approaches were proposed to overcome this problem. These approaches include sampling methods and cost-sensitive learning. Sampling methods artificially modifies the class proportion of the instances in the training set. The objective is to compensate the class imbalance by either removing instances from the majority class or duplicating instances from the minority one. This can be done using an uninformed approach, where instances are removed or duplicated at random, or using a heurist approach. Some methods, like SMOTE, do not make an exact copy of minority instances, although use a kind of interpolation to increase the number of minority classes.

On the other hand, cost sensitive learning generally does not alter the proportion of instances in the training set but uses cost information to cope with class imbalance. This cost information can be incorporated into the learning algorithm; used to weight or sample the training data; or to fine tune a induced model into a non cost-sensitive approach.

Another common issue in domains with class imbalance is evaluation. One of the most used metrics to assess learning algorithms, accuracy, is meaningless in domains with severe class imbalance, as a naïve classifier would have 99.9% of accuracy in a domain where the most common class has 99.9% prevalence. Therefore, other approaches to evaluate classifiers in class imbalanced domains are needed, such as taking geometric, harmonic or arithmetic means of precision of each individual class, total expected cost or using a graphical approach, like the ROC graph.

A related problem is that sometimes the number of examples of the minority class is too small for classifiers to learn adequately. Although this problem may or may not be accompanied with class imbalance, it should be observed that this is the problem of rarity or insufficient training data is a problem in itself, and is different from that of the imbalanced data sets. As a final remark, it has been observed that learning algorithms are able to achieve meaningful results in some domains even in the presence of highly imbalanced data sets. Therefore, class imbalance cannot be directly correlated to the loss of performance of learning algorithms, but is complicating factor in non-linearly separable and complex domains.

Acknowledgements

We would like to thank the support of Brazilian Research Agencies CNPq and FAPESP.

References

1. Phua, C., Alahakoon, D., Lee, V.: Minority report in fraud detection: Classification of skewed data. SIGKDD Explorations Newsletter **6**(1) (2004) 50–59
2. Cohena, G., Hilariob, M., Saxc, H., Hugonnetc, S., Geissbuhler, A.: Learning from imbalanced data in surveillance of nosocomial infection. Intelligent Data Analysis in Medicine **37**(1) (2006) 7–18
3. Japkowicz, N., ed.: AAAI Workshop on Learning from Imbalanced Data Sets, Menlo Park, CA, AAAI Press (2001) Techical report WS-00-05.
4. Chawla, N., Japkowicz, N., Kolcz, A., eds.: ICML'2003 Workshop on Learning from Imbalanced Data Sets (II), Proceedings available at http://www.site.uottawa.ca/ nat/Workshop2003/workshop2003.html (2003)
5. Chawla, N., Japkowicz, N., Zhou, Z.H., eds.: PAKDD'2009 Workshop on Data Mining When Classes are Imbalanced and Errors Have Costs. (2009)
6. Chawla, N., Japkowicz, N., Kolcz, A., eds.: SIGKDD Exploration. Volume 6. (2004) Special issue on Learning from Imbalanced Datasets.
7. Prati, R.C., Batista, G.E., Monard, M.C.: Class imbalance versus class overlaping: an analysis of a learning system behaviour. In: Mexican International Conference on Artificial Intelligence (MICAI'2004). Volume 2972 of LNAI., Mexico City (MX), Springer-Verlag (2004) 312–321
8. Zhang, J., Mani, I.: knn approach to unbalanced data distributions: A case study involving information extraction. In: Proceedings of the ICML'2003 workshop on learning from imbalanced datasets. (2003)
9. Hart, P.E.: The condensed nearest neighbor rule. IEEE Transactions on Information Theory **IT-14** (1968) 515–516
10. Kubat, M., Matwin, S.: Addressing the course of imbalanced training sets: One-sided selection. In: ICML. (1997) 179–186
11. Tomek, I.: Two modifications of cnn. IEEE Transactions on Systems Man and Communications **SMC-6** (1976) 769–772
12. Laurikkala, J.: Improving identification of difficult small classes by balancing class distribution. Technical Report A-2001-2, University of Tampere (2001)
13. Wilson, D.L.: Asymptotic properties of nearest neighbor rules using edited data. IEEE Transactions on Systems, Man, and Communications **2**(3) (1972) 408–421
14. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: Synthetic minority over-sampling technique. JAIR **16** (2002) 321–357
15. Batista, G.E.A.P.A., Bazan, A.L., Monard, M.C.: Balancing training data for automated annotation of keywords: a case study. In: WOB. (2003) 35–43
16. Batista, G.E.A.P.A., Prati, R.C., Monard, M.C.: A study of the behavior of several methods for balancing machine learning training data. SIGKDD Explorations **6**(1) (2004) 20–29
17. Han, H., Wang, W.Y., Mao, B.H.: Borderline-smote: A new over-sampling method in imbalanced data sets learning. (2005) 878–887
18. Ling, C.X., Sheng, V.S.: Cost-sensitive learning and the class imbalance problem. In Sammut, C., ed.: Encyclopedia of Machine Learning. Springer (2008)

19. Elkan, C.: The foundations of cost-sensitive learning. In: Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'2001), Morgan Kaufmann (2001) 973–978
20. Turney, P.D.: Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. J. Artif. Intell. Res. (JAIR) **2** (1995) 369–409
21. Ling, C., Yang, Q., Wang, J., Zhang, S.: Decision trees with minimal costs. In: Proceedings of 2004 International Conference on Machine Learning. (2004)
22. Kearns, M.J., Mansour, Y.: On the boosting ability of top-down decision tree learning algorithms. In: Proceedings of the Twenty-eighth ACM Symposium on Theory of Computing. (1996) 459–468
23. Drummond, C., Holte, R.C.: Exploiting the cost (in)sensitivity of decision tree splitting criteria. In Langley, P., ed.: Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Morgan Kaufmann (2000) 239–246
24. Zadrozny, B., Langford, J., Abe, N.: Cost-sensitive learning by cost-proportionate example weighting. In: IEEE International Conference on Data Mining (ICDM 2003), IEEE Computer Society (2003) 435–442
25. Domingos, P.: Metacost: A general method for making classifiers cost-sensitive. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'1999). (1999) 155–164
26. Chan, P.K., Stolfo, S.J.: Toward scalable learning with non-uniform class and cost distributions: a case study in credit card fraud detection. In: Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining. (2001) 164–168
27. Fan, W., Stolfo, S.J., Zhang, J., Chan, P.K.: Adacost: Misclassification cost-sensitive boosting. In Bratko, I., Dzeroski, S., eds.: Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Morgan Kaufmann (1999) 97–105
28. Chawla, N.V., Lazarevic, A., Hall, L.O., Bowyer, K.W.: Smoteboost: Improving prediction of the minority class in boosting. In Lavrac, N., Gamberger, D., Blockeel, H., Todorovski, L., eds.: 7th European Conference on Principles and Practice of Knowledge Discovery in Databases. Volume 2838 of Lecture Notes in Computer Science., Springer (2003) 107–119
29. Raskutti, B., Kowalczyk, A.: Extreme re-balancing for svms: a case study. SIGKDD Explor. Newsl. **6**(1) (2004) 60–69
30. Zhuang, L., Dai, H.: Parameter estimation of one-class svm on imbalance text classification. In Lamontagne, L., Marchand, M., eds.: 19th Conference of the Canadian Society for Computational Studies of Intelligence. Volume 4013 of Lecture Notes in Computer Science., Springer (2006) 538–549
31. Lee, H., Cho, S.: The novelty detection approach for different degrees of class imbalance. In: Proceedings of the 13th International Conference on Neural Information Processing. Volume 4233 of Lecture Notes in Computer Science., Springer (2006) 21–30
32. Kubat, M., Martin, S.: Addressing the curse of imbalanced training sets: One sided selection. In: Proceedings of the Fourteenth International Conference on Machine Learning, Morgan Kaufmann (1997) 179–186
33. Provost, F.J., Fawcett, T., Kohavi, R.: The case against accuracy estimation for comparing induction algorithms. In: Fifteenth International Conference on Machine Learning (ICML 1998), Morgan Kaufmann (1998) 445–453
34. Bradley, A.P.: The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognition **30**(7) (1997) 1145–1159

35. Adams, N.M., Hand, D.J.: Comparing classifiers when the misallocation costs are uncertain. Pattern Recognition **32**(7) (1999) 1139–1147