3

IMBALANCED DATASETS: FROM SAMPLING TO CLASSIFIERS

T. Ryan Hoens and Nitesh V. Chawla

Department of Computer Science and Engineering, The University of Notre Dame, Notre Dame, IN, USA

Abstract: Classification is one of the most fundamental tasks in the machine learning and data-mining communities. One of the most common challenges faced when trying to perform classification is the class imbalance problem. A dataset is considered imbalanced if the class of interest (positive or minority class) is relatively rare as compared to the other classes (negative or majority classes). As a result, the classifier can be heavily biased toward the majority class. A number of sampling approaches, ranging from under-sampling to over-sampling, have been developed to solve the problem of class imbalance. One challenge with sampling strategies is deciding how much to sample, which is obviously conditioned on the sampling strategy that is deployed. While a wrapper approach may be used to discover the sampling strategy and amounts, it can quickly become computationally prohibitive. To that end, recent research has also focused on developing novel classification algorithms that are class imbalance (skew) insensitive. In this chapter, we provide an overview of the sampling strategies as well as classification algorithms developed for countering class imbalance. In addition, we consider the issues of correctly evaluating the performance of a classifier on imbalanced datasets and present a discussion on various metrics.

3.1 INTRODUCTION

A common problem faced in data mining is dealing *class imbalance*. A dataset is said to be *imbalanced* if one class (called the *majority*, or *negative* class) vastly

Imbalanced Learning: Foundations, Algorithms, and Applications, First Edition.
Edited by Haibo He and Yunqian Ma.
© 2013 The Institute of Electrical and Electronics Engineers, Inc. Published 2013 by John Wiley & Sons, Inc.

outnumbers the other (called the *minority*, or *positive* class). The class imbalance problem is when the positive class is the class of interest.

One obvious complication that arises in the class imbalance problem is the effectiveness of accuracy (and error rate) in determining the performance of a classifiers. Consider, for example, a dataset for which the majority class represents 99% of the data, and the minority class represents 1% of the data (this dataset is said to have an imbalance ratio of 99 : 1). In such cases, the naïve classifier, which always predicts the majority class, will have an accuracy of 99%. Similarly, if a dataset has an imbalance ratio of 9999 : 1, the majority classifier will have an accuracy of 99.99%.

One consequence of this limitation can be seen when considering the performance of most traditional classifiers when applied in the class imbalance problem. This is due to the fact that the majority of traditional classifiers optimize the accuracy, and therefore generate a model that is equivalent to the naïve model described previously. Obviously such a classifier, in spite of its high accuracies, is useless in most practical applications as the minority class is often the class of interest (otherwise a classifier would not be necessary, as the class of interest almost always happens). As a result, numerous methods have been developed, which overcome the class imbalance problem. Such methods fall into two general categories, namely, sampling methods and skew-insensitive classifiers.

Sampling methods (e.g., random over-sampling and random under-sampling) have become standard approaches for improving classification performance [1]. In sampling methods, the training set is altered in such a way as to create a more balanced class distribution. The resulting sampled dataset is then more amenable to traditional data-mining algorithms, which can then be used to classify the data.

Alternatively, methods have been developed to combat the class imbalance problem directly. These methods are often specifically designed to overcome the class imbalance problem by optimizing a metric other than accuracy. By optimizing a metric other than accuracy that is more suitable for the class imbalance problem, skew-insensitive classifiers are able to generate more informative models.

In this chapter, we discuss the various approaches to overcome class imbalance, as well as various metrics, which can be used to evaluate them.

3.2 SAMPLING METHODS

Sampling is a popular methodology to counter the problem of class imbalance. The goal of sampling methods is to create a dataset that has a relatively balanced class distribution, so that traditional classifiers are better able to capture the decision boundary between the majority and the minority classes. Since the sampling methods are used to make the classification of the minority class instances easier, the resulting (sampled) dataset should represent a "reasonable" approximation of the original dataset. Specifically, the resulting dataset should contain only instances that are, in some sense, similar to those in the original dataset, that is, all instances in the modified dataset should be drawn from the same (or

similar) distribution to those originally in the dataset. Note that sampling methods need not create an exactly balanced distribution, merely a distribution that the traditional classifiers are better able to handle.

Two of the first sampling methods developed were random under-sampling and random over-sampling. In the random under-sampling, the majority class instances are discarded at random until a more balanced distribution is reached. Consider, for example, a dataset consisting of 10 minority class instances and 100 majority class instances. In random under-sampling, one might attempt to create a balanced class distribution by selecting 90 majority class instances at random to be removed. The resulting dataset will then consist of 20 instances: 10 (randomly remaining) majority class instances and (the original) 10 minority class instances.

Alternatively, in random over-sampling, minority class instances are copied and repeated in the dataset until a more balanced distribution is reached. Thus, if there are two minority class instances and 100 majority class instances, traditional over-sampling would copy the two minority class instances 49 times each. The resulting dataset would then consists of 200 instances: the 100 majority class instances and 100 minority class instances (i.e., 50 copies each of the two minority class instances).

While random under-sampling and random over-sampling create more balanced distributions, they both suffer from serious drawbacks. For example, in random under-sampling (potentially), vast quantities of data are discarded. In the random under-sampling example mentioned above, for instance, roughly 82% of the data (the 90 majority class instances) was discarded. This can be highly problematic, as the loss of such data can make the decision boundary between minority and majority instances harder to learn, resulting in a loss in classification performance.

Alternatively, in random over-sampling, instances are repeated (sometimes to very high degrees). Consider the random over-sampling example mentioned above, where each instance had to be replicated 49 times in order to balance out the class distribution. By copying instances in this way, one can cause drastic overfitting to occur in the classifier, making the generalization performance of the classifier exceptionally poor. The potential for overfitting is especially true as the class imbalance ratio becomes worse, and each instance must be replicated more and more often.

In order to overcome these limitations, more sophisticated sampling techniques have been developed. We now describe some of these techniques.

3.2.1 Under-Sampling Techniques

The major drawback of random under-sampling is that potentially useful information can be discarded when samples are chosen randomly. In order to combat this, various techniques have been developed, which aim to retain all useful information present in the majority class by removing *redundant noisy*, and/or *borderline* instances from the dataset. Redundant instances are considered safe to remove as they, by definition, do not add any information about the majority

class. Similarly, noisy instances are the majority class instances, which are the product of randomness in the dataset, rather than being a true representation of the underlying concept. Removing borderline instances is valuable as small perturbations to a borderline instance's features may cause the decision boundary to shift incorrectly.

One of the earliest approaches is due to Kubat and Matwin [2]. In their approach, they combine Tomek Links [3] and a modified version of the condensed nearest neighbor (CNN) rule [4] to create a directed under-sampling method. The choice to combine Tomek Links and CNN is natural, as Tomek Links can be said to remove borderline and noisy instances, while CNN removes redundant instances. To see how this works in practice, let us consider how Tomek Links and CNN are defined.

Given two instances a and b, let $\delta(a, b)$ define the distance (e.g., Euclidean) between a and b. A pair of instances a and b, which belong to different classes, is said to be a Tomek Link if there is no instance c such that $\delta(a, c) < \delta(a, b)$ or $\delta(b, c) < \delta(a, c)$. In words, instances a and b define a Tomek Link if: (i) instance a's nearest neighbor is b, (ii) instance b's nearest neighbor is a, and (iii) instances a and b belong to different classes. From this definition, we see that instances that are in Tomek Links are either boundary instances or noisy instances. This is due to the fact that only boundary instances and noisy instances will have nearest neighbors, which are from the opposite class.

By considering the entire dataset, one can remove Tomek Links by searching through each instance and removing those which fit the criteria. When using CNN, on the other hand, one builds up the dataset from a small, seed group of instances. To do this, let D be the original training set of instances, and C be a set of instances containing all of the minority class examples and a randomly selected majority instance. CNN then classifies all instances in D by finding its nearest neighbor in C and adding it to C if it is misclassified. Thus, we see that redundant instances are effectively eliminated in the resulting dataset C as any instance correctly classified by its nearest neighbors is not added to the dataset.

As an alternative to Tomek Links and CNN, another method for under-sampling is called the neighborhood cleaning rule (NCR) due to Laurikkala [5]. NCR uses Wilson's edited nearest neighbor rule (ENN) to select majority class instances to remove from the dataset. In NCR, for each instance a in the dataset, its three nearest neighbors are computed. If a is a majority class instance and is misclassified by its three nearest neighbors, then a is removed from the dataset. Alternatively, if a is a minority class instance and is misclassified by its three nearest neighbors, then the majority class instances among a's neighbors are removed.

3.2.2 Over-Sampling Techniques

While random under-sampling suffered from the loss of potentially useful information, random over-sampling suffers from the problem of overfitting. Specifically, by randomly replicating instances in the dataset, the learned model

might fit the training data too closely and, as a result, not generalize well to unseen instances.

In order to overcome this issue, Chawla et al. developed a method of creating synthetic instances instead of merely copying existing instances in the dataset. This techniques is known as the synthetic minority over-sampling technique (SMOTE) [6]. As mentioned, in SMOTE, the training set is altered by adding synthetically generated minority class instances, causing the class distribution to become more balanced. We say that the instances created are *synthetic*, as they are, in general, new minority instances that have been extrapolated and created out of existing minority class instances.

To create the new synthetic minority class instances, SMOTE first selects a minority class instance a at random and finds its k nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b.

Given SMOTE's effectiveness as an over-sampling method, it has been extended multiple times [7–9]. In Borderline-SMOTE, for instance, only borderline instances are considered to be SMOTEd, where borderline instances are defined as instances that are misclassified by a nearest neighbor classifier. Safe-Level-SMOTE, on the other hand, defines a "safe-level" for each instance, and the instances that are deemed "safe" are considered to be SMOTEd.

In addition to SMOTE, Jo and Japkowicz [10] defined an over-sampling method based on clustering. That is, instead of randomly choosing the instances to oversample, they instead first cluster all of the minority class instances using k-means clustering. They then oversample each of the clusters to have the same number of instances, and the overall dataset to be balanced. The purpose of this method is to identify the disparate regions in the feature space where minority class instances are found and to ensure that each region is equally represented with minority class instances.

In addition to cluster-based over-sampling, Japkowicz et al. [11] also developed a method called *focused resampling*. In focused resampling, only minority class instances that occur on the boundary between minority and majority class instances are over-sampled. In this way, redundant instances are reduced, and better performance can be achieved.

3.2.3 Hybrid Techniques

In addition to merely over-sampling or under-sampling the dataset, techniques have been developed, which perform a combination of both. By combining over-sampling and under-sampling, the dataset can be balanced by neither losing too much information (i.e., under-sampling too many majority class instances), nor suffering from overfitting (i.e., over-sampling too heavily).

Two examples of hybrid techniques that have been developed include SMOTE+Tomek and SMOTE+ENN [12], wherein SMOTE is used to

oversample the minority class, while Tomek and ENN, respectively, are used to under-sample the majority class.

3.2.4 Ensemble-Based Methods

One popular approach toward improving performance for classification problems is to use ensembles. Ensemble methods aim to leverage the classification power of multiple base learners (learned on different subsets of the training data) to improve on the classification performance over traditional classification algorithms. Dietterich [13] provides a broad overview as to why ensemble methods often outperform a single classifier. In fact, Hansen and Salamon [14] prove that under certain constraints (the average error rate is less than 50% and the probability of misprediction of each classifier is independent of the others), the expected error rate of an instance goes to zero as the number of classifiers goes to infinity. Thus, when seeking to build multiple classifiers, it is better to ensure that the classifiers are diverse rather than highly accurate.

There are many popular methods for building diverse ensembles, including bagging [15], AdaBoost [16], Random Subspaces [17], and Random Forests [18]. While each of these ensemble methods can be applied to datasets that have undergone sampling, in general, this is not optimal as it ignores the power of combining the ensemble generation method and sampling to create a more structured approach. As a result, many ensemble methods have been combined with sampling strategies to create ensemble methods that are more suitable for dealing with class imbalance.

AdaBoost is one of the most popular ensemble methods in the machine learning community due, in part, to its attractive theoretical guarantees [16]. As a result of its popularity, AdaBoost has undergone extensive empirical research [13, 19]. Recall that in AdaBoost, base classifier L is learned on a subset S_L of the training data D, where each instance in S_L is probabilistically selected on the basis of its weight in D. After training each classifier, each instance's weight is adaptively updated on the basis of the performance of the ensemble on the instance. By giving more weight to misclassified instances, the ensemble is able to focus on instances that are difficult to learn.

SMOTEBoost is one example of combining sampling methods with AdaBoost to create an ensemble that explicitly aims to overcome the class imbalance [20]. In SMOTEBoost, in addition to updating instance weights during each boosting iteration, SMOTE is also applied to misclassified minority class examples. Thus, in addition to emphasizing minority instances by giving higher weights, misclassified minority instances are also emphasized by the addition of (similar) synthetic examples. Similar to SMOTEBoost, Guo and Viktor [21] develop another extension for boosting called *DataBoost-IM*, which identifies hard instances (both minority and majority) in order to generate similar synthetic examples and then reweights the instances to prevent a bias toward the majority class.

An alternative to AdaBoost is Bagging, another ensemble method that has been adapted to use sampling. Radivojac et al. [22] combine bagging with

over-sampling techniques in the bioinformatics domain. Liu et al. propose two methods, EasyEnsemble and BalanceCascade [23], that generate training sets by choosing an equal number of majority and minority class instances from the training set. Hido and Kashima [24] introduce a variant of bagging, "Roughly Balanced Bagging" (RB bagging), that alters bagging to emphasize the minority class.

Finally, Hoens and Chawla [25] propose a method called *RSM+SMOTE* that involves combining random subspaces with SMOTE. Specifically, they note that SMOTE depends on the nearest neighbors of an instance to generate synthetic instances. Therefore, by choosing different sets of features to apply SMOTE in (and thereby altering the nearest neighbor calculation used by SMOTE to create the synthetic instance), the resulting training data for each base learner will have different biases, promoting a more diverse—and therefore effective—ensemble.

3.2.5 Drawbacks of Sampling Techniques

One major drawback of sampling techniques is that one needs to determine how much sampling to apply. An over-sampling level must be chosen so as to promote the minority class, while avoiding overfitting to the given data. Similarly, an under-sampling level must be chosen so as to retain as much information about the majority class as possible, while promoting a balanced class distribution.

In general, wrapper methods are used to solve this problem. In wrapper methods, the training data is split into a training set and a validation set. For a variety of sampling levels, classifiers are learned on the training set. The performance of each of the learned models is then tested against the validation set. The sampling method that provides the best performance is then used to sample the entire dataset.

For hybrid techniques, such wrappers become very complicated, as instead of having to optimize a single over- (or under-) sampling level, one has to optimize a combination of over- and under-sampling levels. As demonstrated by Cieslak et al. [26], such wrapper techniques are often less effective at combating class imbalance than ensembles built of skew- insensitive classifiers. As a result, we now turn our focus to skew-insensitive classifiers.

3.3 SKEW-INSENSITIVE CLASSIFIERS FOR CLASS IMBALANCE

While sampling methods—and ensemble methods based on sampling methods—have become the de facto standard for learning in datasets that exhibit class imbalance, methods have also been developed that aim to directly combat class imbalance without the need for sampling. These methods come mainly from the cost-sensitive learning community; however, classifiers that deal with imbalance are not necessarily cost-sensitive learners.

3.3.1 Cost-Sensitive Learning

In cost-sensitive learning instead of each instance being either correctly or incorrectly classified, each class (or instance) is given a *misclassification cost*. Thus, instead of trying to optimize the accuracy, the problem is then to minimize the total misclassification cost. Many traditional methods are easily extensible with this goal in mind. Support vector machines (SVMs), for instance, can be easily modified to follow the cost-sensitive framework.

To see how this is done, consider the problem of learning an SVM. SVMs attempt to learn a weight vector **w**that satisfies the following optimization problem:

$$\min_{w,\xi,b} \left\{ \frac{1}{2}||w||^2 + C \sum_{i=1}^{n} \xi_i \right\} \tag{3.1}$$

subject to (for all $i \in \{1, \ldots, n\}$):

$$y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 - \xi_i \tag{3.2}$$

$$\xi_i \geq 0 \tag{3.3}$$

where $\mathbf{x}_i$ denotes the instance i, y_i denotes the class of instance i, ξ_i denotes the "slack," for instance, i (i.e., how badly misclassified, if at all, instance i is by **w**), and C determines the trade-off between training error and model complexity.

In order to make SVMs cost sensitive, the objective function is modified resulting in:

$$\min_{w,\xi,b} \left\{ \frac{1}{2}||w||^2 + C \sum_{i=1}^{n} c_i \xi_i \right\}, \tag{3.4}$$

where c_i is the misclassification cost, for instance i.

In addition to modifying traditional learning algorithms, cost-sensitive ensemble methods have also been developed. AdaCost [27] and cost-sensitive boosting (CSB) [28] are two extensions of AdaBoost, which incorporate the misclassification cost of an instance in order to provide more accurate instance weights and not just weights derived from misclassification error as done by AdaBoost.

3.3.2 Skew-Insensitive Learners

While cost-sensitive learning is a popular way of extending classifiers for the use on the class imbalance problem, some classifiers allow for easier extensions that directly combat the problem of class imbalance.

Naïve Bayes, for instance, is trivially skew insensitive. This is due to the fact that it makes predictions $p(y|\mathbf{x}_i)$ by first computing $p(\mathbf{x}_i|y)$ and $p(y)$ from the training data. Bayes rule is then applied to obtain a classification, for instance, $\mathbf{x}_i$ as: $p(y|\mathbf{x}_i) = p(\mathbf{x}_i|y) \cdot p(y)$. Since $p(\mathbf{x}_i|y)$ is difficult to calculate in general, a simplifying assumption is made (the "naïve" in "naïve Bayes"), namely, each

of the features is assumed independent. With this assumption, we can compute $p(\mathbf{x}_i|y) = \prod_{\forall j,k} p(\mathbf{x}_{ij} = \mathbf{x}_{ijk}|y)$, where $\mathbf{x}_{ij}$ denotes feature j, for instance, i, and $\mathbf{x}_{ijk}$ denotes the kth possible feature value for feature j. Therefore, naïve Bayes is simply skew insensitive as predictions are calibrated by $p(y)$ or the prior probability of class y.

Another classifier that has recently been made skew insensitive are decision trees. Hellinger distance decision trees (HDDTs) [29] are strongly skew insensitive, using an adaptation of the Hellinger distance as a decision tree splitting criterion. They mitigate the need for sampling.

For the sake of clarity, we present the basic decision tree-building algorithm. The algorithm (Algorithm $Build\_Tree$) differs from the traditional C4.5 [30] algorithm in two important facets, both motivated by the research of Provost and Domingos [31]. First, when building the decision tree, $Build\_Tree$ does not consider pruning or collapsing. Second, when classifying an instance, Laplace smoothing is applied. These choices are because of empirical results, demonstrating that a full tree with Laplace smoothing outperforms all other configurations [31], which are particularly relevant for imbalanced datasets. When C4.5 decision trees are built in this way (i.e., without pruning, without collapsing, and with Laplace smoothing), they are called *C4.4 decision trees* [31].

Algorithm $Build\_Tree$

Require: Training set T, Cut-off size C
if $|T| < C$ **then**
  return
end if
for each feature f of T **do**
  $H_f \leftarrow Calc\_Criterion\_Value(T, f)$
end for
$b \leftarrow \max(H)$
for each value **v** of b **do**
  $Build\_Tree(T_{x_b=v}, C)$
end for

An important thing to note is that $Build\_Tree$ is only defined for nominal features. For continuous features, a slight variation to $Build\_Tree$ is used, where $Calc\_Criterion\_Value$ sorts the instances by the feature value, finds all meaningful splits, calculates the binary criterion value at each split, and returns the highest distance; this is identical to the procedure used in C4.5.

The important function to consider when building a decision tree is $Calc\_Criterion\_Value$. In C4.5, this function is gain ratio, which is a measure of purity based on entropy [30], while in HDDT, this function is Hellinger distance. We now describe the Hellinger distance as a splitting criterion.

Hellinger distance is a distance metric between probability distributions used by Cieslak and Chawla [29] to create HDDTs. It was chosen as a splitting criterion for the binary class imbalance problem because of its property of skew insensitivity. Hellinger distance is defined as a splitting criterion as [29]:

$$d_{\mathrm{H}}(X_+, X_-) = \sqrt{\sum_{j=1}^{p} \left(\sqrt{\frac{|X_{+j}|}{|X_+|}} - \sqrt{\frac{|X_{-j}|}{|X_-|}} \right)^2} \tag{3.5}$$

where X_+ is the set of all positive examples, X_- is the set of all negative examples, and X_{+j} (X_{-j}) is the set of positive (negative) examples with the jth value of the relevant feature.

3.4 EVALUATION METRICS

One common method for determining the performance of a classifier is through the use of a confusion matrix (Fig. 3.1). In a confusion matrix, TN is the number of negative instances correctly classified (True Negatives), FP is the number of negative instances incorrectly classified as positive (False Positive), FN is the number of positive instances incorrectly classified as negative (False Negatives), and TP is the number of positive instances correctly classified as positive (True Positives).

From the confusion matrix, many standard evaluation metrics can be defined. Traditionally, the most often used metric is accuracy (or its complement, the error rate):

$$\text{accuracy} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN}} \tag{3.6}$$

As mentioned previously, however, accuracy is inappropriate when data is imbalanced. This is seen in our previous example, where the majority class may

	Predicted negative	Predicted positive
Actual negative	TN	FP
Actual positive	FN	TP

Figure 3.1 Confusion matrix.

make 99% of the dataset, while only 1% is made up of minority class instances. In such a case, it is trivial to obtain an accuracy of 99% if one merely predicts the majority class. This accuracy is obtained by simply predicting all instances as majority class. While the 99% accuracy seems high, it is obviously completely discounting the performance of the classifier on the class that matters (positive class). Thus, accuracy can be misleading in imbalanced datasets as there are two different types of errors (false positives and false negatives), and each of them carries different costs.

We now present a discussion on a number of alternate metrics used for evaluating the performance of classifiers on imbalanced datasets.

3.4.1 Balanced Accuracy

While accuracy (and error rate) is not an effective method of evaluating the performance of classifiers, one common alternative is *balanced accuracy*. Balanced accuracy differs from accuracy in that instead of computing *accuracy*, one computes

$$\text{BalancedAccuracy} = \frac{\text{TP}}{2(\text{TP} + \text{FN}))} + \frac{\text{TN}}{2(\text{TN} + \text{FP}))} \tag{3.7}$$

That is, one computes the average of the percentage of positive class instances correctly classified and the percentage of negative class instances correctly classified. By giving equal weight to these relative proportions, we see that the previous problem of the naïve classifier obtaining very good performance has been eliminated.

To see this, consider the balanced accuracy of the naïve classifier on a dataset consisting of 99% majority class instances and 1% minority class instances. We know that the accuracy of the naïve classifier is 99%. The balanced accuracy, on the other hand, is: $99/(2(99 + 0)) + 0/(2(1 + 0)) = 0.5 + 0 = 0.5$. A performance estimate of 0.5 is a much more valid assessment of the naïve classifier.

3.4.2 ROC Curves

The receiver operating characteristic (ROC) curve is a standard technique for evaluating classifiers on datasets that exhibit class imbalance. ROC curves achieve this skew insensitivity by summarizing the performance of classifiers over a range of true positive rates (TPRs) and false positive rates (FPRs) [32]. By evaluating the models at various error rates, ROC curves are able to determine what proportion of instances will be correctly classified for a given FPR.

In Figure 3.2, we see an example of an ROC curve. In Figure 3.2, the X-axis represents the FPR ($\text{FPR} = \text{FP}/(\text{TN} + \text{FP})$), and the Y-axis represents the TPR ($\text{TPR} = \text{TP}/(\text{TP} + \text{FN})$). For any given problem, the ideal classifier would have

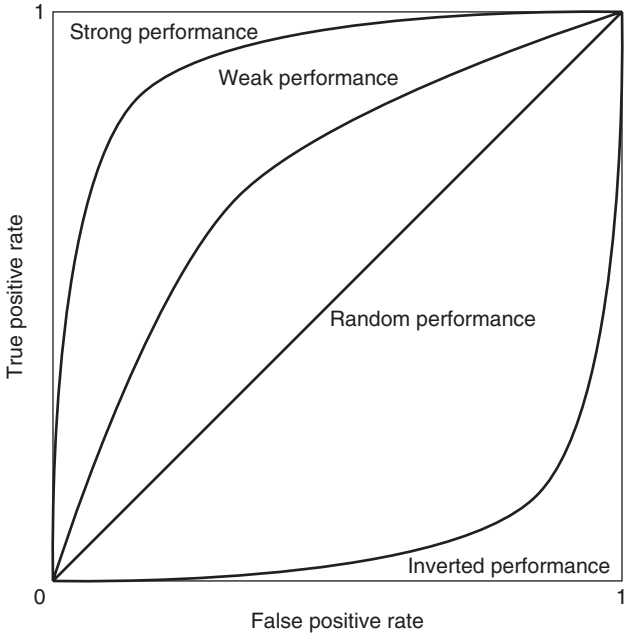

Figure 3.2 Examples of ROC curves. Each curve represents the performance of a different classifier on a dataset.

a point in the ROC space at $(0, 1)$, that is, all positive instances are correctly classified, and no negative instances are misclassified. Alternatively, the classifier that misclassifies all instances would have a single point at $(1, 0)$.

While $(0, 1)$ represents the ideal classifier and $(1, 0)$ represents its complement, in ROC space, the line $y = x$ represents a random classifier, that is, a classifier that applies a random prediction to each instance. This gives a trivial lower bound in ROC space for any classifier. An ROC curve is said to "dominate" another ROC curve if, for each FPR, it offers a higher TPR. By analyzing ROC curves, one can determine the best classifier for a specific FPR by selecting the classifier with the best corresponding TPR.

In order to generate an ROC curve, each point is generated by moving the decision boundary for classification. That is, points nearer to the left in ROC space are the result of requiring a higher threshold for classifying an instance as positive instance. This property of ROC curves allows practitioners to choose the decision threshold that gives the best TPR for an acceptable FPR (Neyman–Pearson method) [33].

The ROC convex hull can also provide a robust method for identifying potentially optimal classifiers [34]. Given a set of ROC curves, the ROC convex hull is generated by selecting only the best point for a given FPR. This is advantageous, since, if a line passes through a point on the convex hull, then there is

no other line with the same slope passing through another point with a larger TPR intercept. Thus, the classifier at that point is optimal under any distribution assumptions along with the slope [34].

While ROC curves provide a visual method for determining the effectiveness of a classifier, the area under the ROC curve (AUROC) has become the de facto standard metric for evaluating classifiers under imbalance [35]. This is due to the fact that it is both independent of the selected threshold and prior probabilities, as well as offering a single number to compare classifiers. One of the main benefits of AUROC is that it can be considered as measuring how often a random positive class instance is ranked above a random negative class instance when sorted by their classification probabilities.

One way of computing AUROC is, given n_0 points of class 0, n_1 points of class 1, and S_0 as the sum of ranks of class 0 examples [36]:

$$\text{AUROC} = \frac{2S_0 - n_0(n_0 + 1)}{2n_0n_1} \tag{3.8}$$

3.4.3 Precision and Recall

Alternatives to AUROC are precision and recall. Precision and recall can be computed from the confusion matrix (Fig. 3.1) as [37]:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.9}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.10}$$

From the equations, we see that precision measures how often an instance that was predicted as positive that is actually positive, while recall measures how often a positive class instance in the dataset was predicted as a positive class instance by the classifier.

In imbalanced datasets, the goal is to improve recall without hurting precision. These goals, however, are often conflicting, since in order to increase the TP for the minority class, the number of FP is also often increased, resulting in reduced precision.

In order to obtain a more accurate understanding of the trade-offs between precision and recall, one can use precision–recall (PR) curves. PR curves are similar to ROC curves in that they provide a graphical representation of the performance of classifiers. While the X-axis of ROC curves is FPR and the Y-axis is TPR, in PR curves, the X-axis is recall and the Y-axis is precision. Precision–recall curves are therefore similar to ROC curves as recall is the same as FPR; however, the Y-axes are different. While TPR measures the fraction of positive examples that are correctly classified, precision measures the fraction of examples that are classified as positive that are actually positive.

Similar to ROC curves are being compared on the basis of AUROC, PR curves are also compared on the basis of AUPR. This practice has become more common, as recent research suggests that PR curves (and AUPR) are a better discriminator of performance than their ROC (and AUROC) counterparts [38].

3.4.4 F_β-Measure

A final common metric is the F_β-measure. F_β-measure is a family of metrics that attempts to measure the trade-offs between precision and recall by outputting a single value that reflects the *goodness* of a classifier in the presence of rare classes. While ROC curves represent the trade-off between different TPRs and FPRs, F_β-measure represents the trade-off among different values of TP, FP, and FN [37].

The general equation for F_β-measure is:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}, \tag{3.11}$$

where β represents the relative importance of precision and recall. Traditionally, when β is not specified, the F_1-measure is assumed.

In spite of its (relatively) useful properties for imbalance, F_β-measure is not commonly used when comparing classifiers, as AUROC and AUPR provide more robust and better performance estimates.

3.5 DISCUSSION

In this chapter, we covered various strategies for learning in imbalanced environments. Specifically, we discussed various sampling strategies and skew- insensitive classifiers.

One key observation when attempting to choose between a sampling method and a skew-insensitive classifier is that while sampling methods are a widely applied standard, they require the tuning of parameters to select the proper sampling level for a given dataset. In general, this is a difficult optimization problem and may prove impractical in practice depending on the size of the dataset and level of imbalance. In such cases, skew-insensitive classifiers (and ensembles built of skew-insensitive classifiers) can provide a reasonable alternative that provides similar (and often better) performance as that of the sampling methods.

When attempting to evaluate the performance of the aforementioned models, we learned that accuracy is not a valuable evaluation metric when learning in imbalanced environments. The lack of utility of accuracy (and error rate) is due to the fact that they overemphasize the performance of the majority class at the detriment to the considerations of the performance of the minority class. In order to overcome this issue, we presented multiple alternative evaluation metrics. The most commonly used alternatives discussed were AUROC and AUPR.

REFERENCES

1. N. V. Chawla, D. A. Cieslak, L. O. Hall, and A. Joshi, "Automatically countering imbalance and its empirical relationship to cost," *Data Mining and Knowledge Discovery*, vol. 17, no. 2, pp. 225–252, 2008.
2. M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: One-sided selection," in *Machine Learning-International Workshop then Conference*, (Nashville, TN, USA), pp. 179–186. Morgan Kaufmann, 1997.
3. I. Tomek, "An experiment with the edited nearest-neighbor rule," *IEEE Transactions on Systems, Man, and Cybernetics Part C*, vol. 6, no. 6, pp. 448–452, 1976.
4. P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
5. J. Laurikkala, "Improving identification of difficult small classes by balancing class distribution," *Artificial Intelligence in Medicine*, (Cascais, Portugal), pp. 63–66, Springer-Verlag, vol. 2101, 2001.
6. N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.
7. H. Han, W. Y. Wang, and B. H. Mao, "Borderline-smote: A new over-sampling method in imbalanced data sets learning," in *Advances in Intelligent Computing*, (Hefei, China), vol. 3644, pp. 878–887, Springer-Verlag, 2005.
8. C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem," in *Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, (Bangkok, Thailand), pp. 475–482, Springer-Verlag, 2009.
9. C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "DBSMOTE: Density-based synthetic minority over-sampling technique," *Applied Intelligence*, vol. 36, pp. 1–21, 2011.
10. T. Jo and N. Japkowicz, "Class imbalances versus small disjuncts", *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 40–49, 2004.
11. N. Japkowicz "Learning from imbalanced data sets: A comparison of various strategies," in *AAAI Workshop on Learning from Imbalanced Data Sets*, (Austin, Texas), vol. 68, AAAI Press, 2000.
12. G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 20–29, 2004.
13. T. G. Dietterich, "Ensemble methods in machine learning," *Lecture Notes in Computer Science*, vol. 1857, pp. 1–15, 2000.
14. L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, 1990.
15. L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.
16. Y. Freund and R. Schapire. "Experiments with a new boosting algorithm," in *Thirteenth International Conference on Machine Learning*, (Bari, Italy), pp. 148–156, Morgan Kaufmann, 1996.

17. T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

18. L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

19. E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Machine Learning*, vol. 36, no. 1, pp. 105–139, 1999.

20. N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer. "Smoteboost: Improving prediction of the minority class in boosting," in *Proceedings of the Principles of Knowledge Discovery in Databases, PKDD-2003*, (Cavtat-Dubrovnik, Croatia), vol. 2838, pp.107–119, Springer-Verlag, 2003.

21. H. Guo and H. L. Viktor, "Learning from imbalanced data sets with boosting and data generation: The Databoost-IM approach," *SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 30–39, 2004.

22. P. Radivojac, N. V. Chawla, A. K. Dunker, and Z. Obradovic, "Classification and knowledge discovery in protein databases," *Journal of Biomedical Informatics*, vol. 37, no. 4, pp. 224–239, 2004.

23. X. Y. Liu, J. Wu, and Z. H. Zhou. "Exploratory under-sampling for class-imbalance learning", in *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pp. 965–969 Washington, DC: IEEE Computer Society, 2006.

24. S Hido and H. Kashima, "Roughly balanced bagging for imbalanced data," in *SDM*, pp. 143–152. SIAM, 2008.

25. T. Hoens and N. Chawla, "Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining," in *PAKDD*, (Hyderabad, India), vol. 6119, pp. 488–499, Springer-Verlag, 2010.

26. D. A. Cieslak, T. R. Hoens, N. V. Chawla, and W. P. Kegelmeyer, "Hellinger distance decision trees are robust and skew-insensitive," *Data Mining and Knowledge Discovery*, (Hingham, MA, USA), vol. 24, no. 1, pp. 136–158, Kluwer Academic Publishers, 2012.

27. W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. "Adacost: Misclassification cost-sensitive boosting," in *Machine Learning-International Workshop then Conference*, (Bled, Slovenia), pp. 97–105, Morgan Kaufmann, 1999.

28. K. M. Ting. "A comparative study of cost-sensitive boosting algorithms," in *Proceedings of the 17th International Conference on Machine Learning*, 2000.

29. D. A. Cieslak and N. V. Chawla. "Learning decision trees for unbalanced data," in *European Conference on Machine Learning (ECML)*, (Antwerp, Belgium), vol. 5211, pp. 241–256, Springer-Verlag, 2008.

30. J. R. Quinlan. *C4. 5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufman Publishers, Inc., 1993.

31. F. Provost and P. Domingos, "Tree induction for probability-based ranking," *Machine Learning*, vol. 52, no. 3, pp. 199–215, 2003.

32. J. A. Swets, "Measuring the accuracy of diagnostic systems," *Science*, vol. 240, no. 4857, pp. 1285–, 1988.

33. J. P. Egan. *Signal Detection Theory and ROC Analysis*. New York: Academic Press, 1975.

34. F. Provost and T. Fawcett, "Robust classification for imprecise environments," *Machine Learning*, vol. 42, no. 3, pp. 203–231, 2001.

35. A. P. Bradley, "The use of the area under the roc curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.

36. D. J. Hand and R. J. Till, "A simple generalisation of the area under the roc curve for multiple class classification problems," *Machine Learning*, vol. 45, no. 2, pp. 171–186, 2001.

37. M. Buckland and F. Gey, "The relationship between recall and precision," *Journal of the American Society for Information Science*, vol. 45, no. 1, pp. 12–19, 1994.

38. J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the Twentythird International Conference on Machine Learning*, pp. 233–240. ACM, 2006.