Data Min Knowl Disc (2008) 17:225–252
DOI 10.1007/s10618-008-0087-0

Automatically countering imbalance and its empirical relationship to cost

Nitesh V. Chawla · David A. Cieslak · Lawrence O. Hall · Ajay Joshi

Received: 11 November 2006 / Accepted: 8 January 2008 / Published online: 17 February 2008
Springer Science+Business Media, LLC 2008

Abstract Learning from imbalanced data sets presents a convoluted problem both from the modeling and cost standpoints. In particular, when a class is of great interest but occurs relatively rarely such as in cases of fraud, instances of disease, and regions of interest in large-scale simulations, there is a correspondingly high cost for the misclassification of rare events. Under such circumstances, the data set is often re-sampled to generate models with high minority class accuracy. However, the sampling methods face a common, but important, criticism: *how to automatically discover the proper amount and type of sampling?* To address this problem, we propose a wrapper paradigm that discovers the amount of re-sampling for a data set based on optimizing evaluation functions like the f-measure, Area Under the ROC Curve (AUROC), cost, cost-curves, and the cost dependent f-measure. Our analysis of the wrapper is twofold. First, we report the interaction between different evaluation and wrapper optimization functions. Second, we present a set of results in a cost-sensitive environment, including scenarios of unknown or changing cost matrices. We

Responsible editor: Gary M. Weiss.

N. V. Chawla (✉) · D. A. Cieslak
Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA
e-mail: nchawla@cse.nd.edu

D. A. Cieslak
e-mail: dcieslak@cse.nd.edu

L. O. Hall · A. Joshi
Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620-5399, USA

L. O. Hall
e-mail: hall@cse.usf.edu

A. Joshi
e-mail: ajoshi@cse.usf.edu

also compared the performance of the wrapper approach versus cost-sensitive learning methods—MetaCost and the Cost-Sensitive Classifiers—and found the wrapper to outperform the cost-sensitive classifiers in a cost-sensitive environment. Lastly, we obtained the lowest cost per test example compared to any result we are aware of for the KDD-99 Cup intrusion detection data set.

Keywords Classification · Unbalanced data · Cost-sensitive learning

1 Introduction

Imbalance in class distribution is pervasive in a variety of real-world applications, including but not limited to telecommunications, WWW, finance, biology, and medicine. The minority or positive class is often of interest and also accompanied with a higher cost of making errors. A typical example of this problem is fraud detection. The instances of fraud in the population are generally a very small proportion (often in the neighborhood of 2%). However, it is quite important to be able to detect a fraudulent transaction. At the same time, it is also important to minimize false positives because these result in investigation costs and can also result in losing a customer. Thus, there is a distribution of costs associated with both false positives and false negatives. Another example is large-scale simulation. While, there are not "dollar" costs attached with errors like there may be with fraud detection, there is a cost attached to time spent in poring over "uninteresting regions" in a simulation. It is known that the regions of interest in such simulations are typically rare (Bowyer et al. 2000; Banfield et al. 2005). Hence, an intelligent tool should be accurate in identifying interesting regions, without too many false alarms.

Weiss and Provost (2003) observed that the naturally occurring distribution is not always optimal. Thus, one needs to modify the data distribution, conditioned on an evaluation function. Re-sampling, by adding to the minority (positive) class or removing the majority (negative) class of a given data set, has become a de facto standard to counter the curse of imbalance in various domains. There have been numerous papers and case studies exemplifying their advantages (Chawla et al. 2003, 2004; Kubat and Matwin 1997; Ling and Li 1998; Weiss and Provost 2003; Ferri et al. 2004; Dietterich et al. 2000; Kubat et al. 1998; Zadrozny and Elkan 2001; Batista et al. 2004; Maloof 2003; Drummond and Holte 2003). However, the one common critique of various works is: *how does one effectively identify the potentially optimal sampling technique and parameters for a given data set?* An accompanying question is: *can the techniques for imbalance generalize across cost-sensitive scenarios?*

Driven by these important and pertinent questions and appreciative of the problem that sampling will be highly data dependent, we propose a wrapper framework for empirically discovering the sampling parameters. We fill an important hole in the application of sampling methods to counter the problem of class imbalance by comprehensively demonstrating the efficacy of the wrapper paradigm. Our goal was to empirically evaluate the usefulness of popular techniques for countering class imbalance by tuning the parameters on a validation set before reporting the performance on the testing set. All the work, thus far, has reported the best performance on the testing set by applying different sampling strategies. Thus, there is no empirical estimate of generalization capacity. In addition, we evaluate the generalization performance of the classifiers, learned on such re-sampled data sets, in a large variety of experimental scenarios. We want to analyze the flexibility of learned models in a dynamic environment since misclassification costs are often unknown (Weiss et al. 2007) or may change. We empirically investigated the relationship between choosing the "best" sampling strategy via a wrapper method and its relationship to classification in a cost-sensitive environment, especially when the costs are unknown or can change. Thus, the over-arching question is: *Can we utilize measures and methods that counter class imbalance and do well in a cost-sensitive framework?*

Contributions. Our main contributions in this paper are centered around the following research questions.

- Can we effectively discover sampling strategies for a given classifier and domain?
- Does the proposed wrapper paradigm for countering imbalance, guided by the cost-insensitive criteria such as the Area Under the Receiver Operator Characteristic Curve *AUROC* and the *f-measure*, perform well in a cost-sensitive testing environment? How does it compare to using cost directly in the wrapper mode to guide the search? How do changing costs or unknown costs effect the performance of the wrapper?
- In a cost-sensitive environment, are there potential advantages in accounting for the imbalance first by applying sampling strategies as compared to MetaCost (Domingos 1999) and Cost-Sensitive Classifier (Zadrozny et al. 2003)?

The remainder of this article is organized as follows. Section 2 introduces the wrapper framework to discover the coupling of a sampling method and corresponding parameters with a classifier and domain. Section 3 presents the experimental framework. Our empirical analyses first illustrate the efficacy of the wrapper framework across data sets using cost-transparent evaluation measures in Sect. 4. Then in Sect. 5, the impact of introducing costs during testing is demonstrated using a variety of cost-ratios. Finally, Sect. 6 discusses the results.

2 Wrapper paradigm

Re-sampling is a popular solution to the class imbalance problem. However, one persistent limitation of the sampling methods has been automatically discovering the amount and type of sampling to apply to a given data set. To address this limitation, we propose a comprehensive wrapper infrastructure that applies cross-validation to first discover the best amounts of undersampling and oversampling. For oversampling, we use the Synthetic Minority Over-Sampling TEchnique (SMOTE) (Chawla et al. 2002), which oversamples the minority class by introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors. Depending upon the amount of over-sampling required, neighbors from the k nearest neighbors are randomly chosen. Synthetic samples are generated in the following way: take the difference between the feature vector (sample) under consideration and its nearest neighbor. Multiply this difference by a random number between 0 and 1, and add it to the feature vector under consideration. This causes the selection of a random point

```
1: Input: List of minority classes $MinorList$
2: List of majority classes $MajorList$
3: Number of cross-validation folds $NumFolds$
4: Output: Wrapper selected UnderSampling percentages for majority classes; Wrapper
   selected SMOTE percentages for minority classes
5: for all $MClass$ in $MajorList$ do
6:    $UnderSampleList[MClass] = 100$
7: end for
8: for all $MClass$ in $MinorList$ do
9:    $SmoteList[MClass] = 0$
10: end for
11: for $Fold \leftarrow 1$ to $NumFolds$ do
12:    Build classifier on training fold $Fold$ and evaluate on validation set
13:    Update $BestMinorMetricValues$ and $BestMajorMetricValues$
14: end for
15: if $MajorList$ is not empty then
16:    $WRAPPER\_UNDERSAMPLE(UnderSampleList)$
17: end if
18: if $MinorList$ is not empty then
19:    $WRAPPER\_SMOTE(SmoteList, UnderSampleList)$
20: end if
21: Output $UnderSampleList$ and $(SmoteSampleList)$
```

Fig. 1 Wrapper Undersample SMOTE Algorithm

along the line segment between two specific features. This approach effectively forces the decision region of the minority class to become more general.

Obviously, searching the entire space of undersampling and SMOTE combinations can quickly become intractable, so we proceed in a step-wise fashion. This strategy removes the "excess" examples of the majority classes, which reduces the size of the training data set. This also makes learning time more tractable. Then SMOTE is used to add synthetic examples of the minority classes and increase the generalization performance of the classifier over the minority classes. Figure 1 shows the Wrapper_UnderSample_SMOTE algorithm, which can extend to multiple majority and minority classes. The metric values in the pseudo-code indicate the performance criterion, which will be discussed in subsequent sections. We will analyze the cross-validation approach used in Sect. 2.1, outline the wrapper approach to selecting sampling levels in Sects. 2.2 and 2.3, and survey the wrapper search evaluation metrics in Sect. 2.4.

2.1 Using cross-validation to guide sampling

The wrapper approaches are optimized on an independent validation set to avoid any bias in performance on the testing data. To that end, we construct a careful cross-validation framework. We first split the data set into ten partitions. Each partition is used once as a testing fold, with the remaining 90% as the training fold. This results in ten pairs of training and testing folds (10-fold cross-validation), forming the basis for our comparison of the different methods. We further split each of the training folds, independently, into five partitions for an internal fivefold cross-validation. The wrapper applies this independent validation stage to each fold to discover the appropriate percentages of sampling for a given method and classifier combination. Once these percentages are discovered, the classifier is re-learned on the original training fold using the discovered percentages and tested on the corresponding testing fold. Thus, the purpose of internal fivefold cross-validation is only to guide an independent wrapper stage, keeping the original testing fold separate for an unbiased assessment of the performance. Note that this procedure is repeated for each of the 10-folds. Furthermore, due to the inherent random nature of undersampling and SMOTE, the process of training and testing with wrapper selected undersampling and SMOTE percentages was done a total of five times to get an averaged (more stable) performance measure.

To summarize, for five different runs, the 10-folds were constructed and on each fold a fivefold cross-validation was applied to discover the amounts of SMOTE and undersampling. The final reported performances are the averages over the 50 sets (5 runs $\times$ 10 folds). We will now explore the heuristic functions used to optimize the sampling levels.

2.2 Wrapper-based algorithm to select undersampling percentages

This algorithm uses a wrapper approach to perform the search through the parameter space of undersampling percentages for the majority class(es), using the chosen learning algorithm as a part of the evaluation function for a fivefold cross-validation on the training data. The purpose is to search for the sampling level in the parameter space with the highest evaluation score guided by some heuristic function. This search in the parameter space has to be restricted to the training data to avoid any estimation biases. Thus, we utilized a fivefold cross-validation on the training set to discover the amount of undersampling.

The wrapper starts with no undersampling for all majority classes and obtains baseline results on the training data. Then it traverses through the search space of undersampling percentages in decrements of *Sample Decrement* (in this case 10%), in a greedy iterative fashion, to increase performance over the minority classes without sacrificing performance on the majority class. We start with 100% undersampling, implying that all of the majority class examples are retained. And then we remove the majority class examples 10% at a time. This search process continues as long as it does not hamper the performance of any minority class, *Minor Metric Value*, or drop the performance for any majority class, *Major Metric Value*, by more than some amount specified by (1-*Increment Min*). We used *Increment Min* = 5% for our experiments. The algorithm terminates when the performance threshold is violated for any class specified in *Major List* or *Minor List*. The details are presented in Fig. 2.

Note that *Minor Metric Value* and *Major Metric Value* are dependent on the objective evaluation function guiding the wrapper.

2.3 Wrapper-based algorithm to select SMOTE percentages

The data set is undersampled by the amounts discovered in the previous step. We choose to undersample first, as we want to first eliminate any majority class examples that did not add any improvement in the evaluation metric. Moreover, it also

```
1:  $SampleDecrement = 10\%$
2:  $IncrementMin = 5\%$
3:  for all $MClass$ in $MajorList$ do
4:    $StopUnderSamplingFlag[MClass] = false$
5:  end for
6:  repeat
7:    for all $MClass$ in $MajorList$ do
8:      if $StopUnderSamplingFlag[MClass] = false$ then
9:        $UnderSampleList[MClass] = UnderSampleList[MClass] - SampleDecrement$
10:       for $Fold \leftarrow 1$ to $NumFolds$ do
11:         $OutputTrainingData(Fold, UnderSampleList)$
12:         $OutputTestingData(Fold)$
13:         Build classifier on the undersampled training set and evaluate on validation set
14:       end for
15:       Update $MinorMetricValues$ & $MajorMetricValues$
16:       if $(Average(MinorMetricValues) < Average(BestMinorMetricValues))$ OR $(Average(MajorMetricValues) < (1.0 - IncrementMin) \times Average(BestMajorMetricValues))$ then
17:         $UnderSampleList[MClass] = UnderSampleList[MClass] - SampleDecrement$
18:         $StopUnderSamplingFlag[MClass] = true$
19:       else
20:         Update $BestMinorMetricValues$ & $BestMajorMetricValues$
21:       end if
22:     end if
23:   end for
24: until $StopUnderSamplingFlag$ for at least one $MClass = true$
```

Fig. 2 Wrapper Undersample Algorithm

reduces the size of the training set, speeding up learning. The performance estimate on the minority or positive class (*Minor Metric Value*) obtained by undersampling becomes the new benchmark or baseline to guide the wrapper. We only decide to SMOTE if it improves performance over this baseline.

The Wrapper SMOTE algorithm evaluates different amounts of SMOTE at steps of 100%.[1] This is a greedy search, and at each step the new performance estimates become the new baseline. That is, the initial baseline is the performance obtained via the Wrapper Undersample. If SMOTE = 100% improves the performance over that baseline by some margin *Increment Min* (set at 5% for our experiments; see line 17 in the algorithm), then the performance achieved at SMOTE = 100% becomes the new baseline. The amount of SMOTE is then incremented by *Sample Increment*, and another evaluation is performed to check if the performance increase at new SMOTE amount is at least greater than *Increment Min*. This process repeats, greedily, until no performance gains are observed. The amount of SMOTE is added at increments of

[1] These steps are denoted by *Sample Increment* in the algorithm.

100%. The fivefold cross-validation, as previously described, is used for guiding the wrapper.

However, there is an important caveat to the search to avoid being trapped in a local maximum. If the average does not improve by 5% then to verify that we have not settled on a local maximum, we look ahead two more steps at increasing amounts of SMOTE. If the look-ahead does not result in an improvement in performance, then the amount of SMOTE is reset to the value discovered prior to the look-aheads. This is done to allow SMOTE to introduce additional examples with the aim of improving performance. However, if the addition of examples does not help, then we go back to using the lesser amount of SMOTE discovered prior to the look-ahead. The pseudo code for the wrapper SMOTE algorithm is presented in Fig. 3.

We do not check for the *Major Metric Value*, as we are primarily interested in optimizing the performance on the minority class in this stage, with the premise that the true positives are more important than false positives.

```
$SampleIncrement = 100$
$LookupAheadValue = 3$
for all $MClass$ in $MinorList$ do
  $StopSmoteFlag[MClass] = false$
  $LookupAhead[MClass] = 1$
end for
repeat
  for all $MClass$ in $MinorList$ do
    if $StopSmoteFlag[MClass] = false$ then
      $SmoteList[MClass] = SmoteList[MClass] + SampleIncrement$
      for $Fold \leftarrow 1$ to $NumFolds$ do
        $OutputTrainingData(Fold, UnderSampleList, SmoteList)$
        $OutputTestingData(Fold)$
        Build classifier on the sampled training set and evaluate on validation set
      end for
      Update $MinorMetricValues$
      if $Average(MinorMetricValues) < (1.0 + IncrementMin) \times Average(BestMinorMetricValues)$ then
        if $LookupAhead[MClass] < LookupAheadValue$ then
          $LookupAhead[MClass] = LookupAhead[MClass] + 1$
        else
          $SmoteList[MClass] = SmoteList[MClass] - (LookupAhead[MClass] \times SampleIncrement)$
          $StopSmoteFlag[MClass] = true$
        end if
      else
        Update $BestMinorMetricValues$
        $LookupAhead[MClass] = 1$
      end if
    end if
  end for
until $StopSmoteFlag$ for at least one $MClass = true$
```

Fig. 3 Wrapper SMOTE Algorithm

2.4 Evaluation metrics

The only way to determine optimal sampling levels is through empirical analysis. Cross-validation is required because optimality of the sampling methods must be determined on the training data only. One question remains: *given our sampling space, how does one determine the "best" sampling levels?* Thus, we will explore four classifier evaluation criteria which were deployed and empirically tested.

2.4.1 Cost

When optimizing sampling levels to improve overall cost, a logical evaluation criteria is the cost itself. Cost is calculated as shown in Eq. 1 when we assume $C(+|+) = C(-|-) = 0$.

$$cost = FN\ rate \times C(-|+) + FP\ rate \times C(+|-) \tag{1}$$

Therefore, once classification occurs in the cross-validation phase, the wrapper calculates the validation cost and uses this information to select optimal sampling levels. This approach is dependent on a priori knowledge of the cost relationship between classes. Different cost ratios may cause selection of the different optimal sampling levels and thus generate very different classifiers.

In the subsequent discussion, we will refer to the wrapper approach using *cost* as *wrapper-cost*.

2.4.2 F-measure

The *f-measure* tends to be used in information retrieval. It is a composite metric based on *precision* and *recall* (Dumais et al. 1998; Mladenic and Grobelnik 1999; Lewis and Ringuette 1994; Cohen 1995b), where *precision* and *recall* are defined as follows:

$$precision = \frac{TP}{TP + FP} \tag{2}$$

$$recall = \frac{TP}{TP + FN}, \tag{3}$$

where TP = True Positives, FP = False Positives, TN = True Negatives, and FN = False Negatives.

As a composite statistic, *f-measure* is then calculated from precision and recall to summarize the effects of the two types of errors.

$$f\ measure = \frac{2 \times precision \times recall}{recall + precision} \tag{4}$$

It is desirable to increase recall without a sacrifice in precision. Therefore, an effective wrapper strategy is to determine the sampling levels which maximize the classifier's *f-measure*. This reduces error on the relatively expensive minority class while maintaining accuracy on the majority class. This wrapper evaluation function has an advantage over the cost-based wrapper in that a priori knowledge of the cost matrices is not required. The generated classifier can be consistently applicable throughout alternative unseen class distributions and cost relationships. In the subsequent discussion, we refer to the wrapper approach using the *f-measure* as *wrapper-f measure*.

2.4.3 β varied f-measure

Another wrapper evaluation criterion is based on an alternate derivation of the previously outlined *f-measure*. Here, a separate β factor is incorporated to introduce an element of relative importance between recall and precision:

$$f\ measure_{\beta} = \frac{(1+\beta^2) \times precision \times recall}{\beta^2 \times recall + precision} \tag{5}$$

As β decreases, the importance of precision diminishes. To adapt this behavior to a cost-based framework the β value must depend on the cost ratio. Therefore, β will be set as:

$$\beta = \frac{C(+|-)}{C(-|+)} \tag{6}$$

As misclassifying minority examples becomes more costly relative to majority examples, improving recall affects the *f-measure* more heavily than precision. Therefore, the minority class accuracy becomes more important as the costs become more divergent. Like the cost-based wrapper, this metric is dependent on a predefined cost matrix. In the subsequent discussion we refer to the wrapper approach using *β varied f-measure* as *wrapper-β*.

2.4.4 Area Under the ROC Curve (AUROC)

A Receiver Operating Characteristic curve (or ROC curve) is obtained by modifying the parameters of the learning algorithm to get different true positive/false positive percentages for two class problems. For classifiers which produce class probabilities, a varying threshold on the probability of belonging to a class can be applied to get the different "operating" points (e.g., a threshold of 0.3 would mean all examples classified with probability greater than 0.3 or class A are considered to have label A). After obtaining a set of points, they can be graphed. The points can be connected by lines to approximate a curve. The area under this curve provides a method for comparing classifiers (AUROC). An AUROC of one indicates a classifier, which gets all of the true positives correct and has no false positives. An AUROC of 0.5 would indicate that for every true positive a classifier generates a false positive. The greater the AUROC, the better the classifier is at obtaining true positives with fewer false positives. Like the *f-measure wrapper*, this metric is cost independent and a classifier constructed in these terms should be flexible through a wide range of costs and distributions. In the subsequent discussion, we will refer to the wrapper approach using *AUROC* as *wrapper-AUROC*.

2.5 A scalable implementation

The Achilles heel of any wrapper system is the computation time due to the exploration of different parameters. However, the proposed wrapper framework for undersampling and SMOTE lends itself very well to a distributed computing paradigm. This also offers an optimal utilization of idle machines in a department or institution. We used the *Condor* framework (Thain et al. 2005) to enable distributed computation on the University of Notre Dame machines resulting in significantly reduced time to obtain a final classifier.[2] Figure 4 shows the general implementation framework and is discussed below.

1. First, partition the data into 10-folds. Distribute each of the folds to a different computer.
2. Further partition each training fold into fivefolds for the wrapper. Subsequently, each of the fivefolds can also be distributed to five different computers.
3. The sampling parameters were used in each of the fivefolds simultaneously, and an average performance estimate was computed. First, the search for undersampling parameters was undertaken. Once the "best" undersampling parameters were found the SMOTE phase was started. Again, the SMOTE parameters were applied to each of the fivefolds simultaneously. The performance estimates were aggregated from each of the fivefolds, and once the best combination of undersampling and SMOTE parameters were found, the search was stopped.

We exploited this inherent parallelism in the parameter search, and were able to achieve on the order of (average) 30 times speedups as discussed in Sect. 5.1. It is worth noting that our largest data set contains just over 11,000 examples. Using larger data sets may affect the parallel speedup, either negatively or positively.

2.6 Classifiers

We used two different machine learning algorithms for our experiments as base classifiers—a decision tree learner, C4.5 release 8 (Quinlan 1993) and a rule learner, Ripper (Cohen 1995a). We converted the leaf frequencies produced by C4.5 decision trees to probability estimates after applying Laplace smoothing (Provost and Domingos 2003). For Ripper, the probabilities were generated based on rule confidence: the proportion of a single class versus the total examples covered. As with C4.5, this was modified with Laplace smoothing by introduction of a $1/C$ class prior probability.

We also considered class prediction by setting a probability threshold. Elkan (2001) describes a system for generating optimal predictions in a two class problem based on a known cost matrix. Here, we are interested in the probability threshold for

[2] Condor can be downloaded from the University of Wisconsin Condor website.

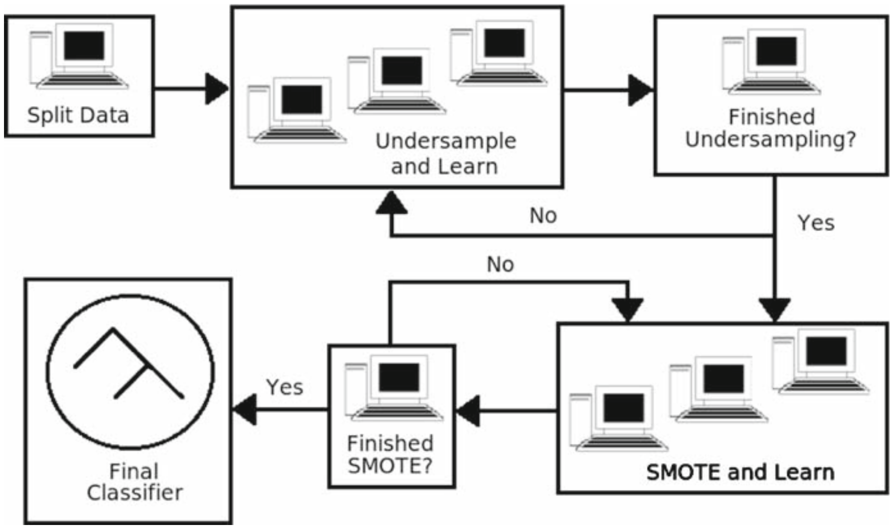

Fig. 4 Distributed wrapper implementation

predicting one class over another. Therefore, we consider the conditions under which the expected costs for such a decision are equivalent:

$$P(y=-|x)C(+|-)+P(y=+|x)C(+|+)=P(y=+|x)C(-|+) + P(y=-|x)C(-|-) \tag{7}$$

Given that $p^* = P(y=+|x)$ and that $P(y=+|x) = 1 - P(y=-|x)$ under the two class assumption:

$$(1-p^*)C(+|-)+p^*C(+|+)=p^*C(-|+)+(1-p^*)C(-|-) \tag{8}$$

Rearranging this equation yields:

$$p^* = \frac{C(+|-)-C(-|-)}{C(+|-)-C(+|+)+C(-|+)-C(-|-)} \tag{9}$$

Under our assumptions, $C(+|+) = C(-|-) = 0$ and we may therefore drop these terms from consideration.

$$p^* = \frac{C(+|-)}{C(+|-)+C(-|+)} \tag{10}$$

Therefore, when we observe a probability estimate p on a testing example, an optimal decision is generated by comparing p to the threshold p^*. When $p > p^*$, the example is labeled as the positive class, otherwise it is predicted as belonging to the negative class.

3 Experimental framework

We used a variety of data sets for our study, including the KDD-99 cup data set. Eight data sets have no associated costs and we call them unfixed cost data sets. These are summarized in Table 1. Four of the data sets are from the UCI repository, and the other four are real-world examples that we acquired from different sources. To establish a context, we will briefly introduce the eight unfixed cost data sets. The fixed cost KDD-99 cup data set will be discussed separately in Sect. 5.

1. The Phoneme data set is used to distinguish between nasal and oral sounds based on five features, and is from the ELENA project. The distribution is 3818 nasal and 1586 oral samples.
2. The Segment data set (Blake et al. 1998) originally consisted of six equally distributed classes from an image segmentation problem. We transformed it into a two class problem by considering one class of size 330 examples as the minority class, and the rest of classes as a single majority class with 1980 examples.
3. The E-state data set (Chawla et al. 2002) consists of electro-topological state descriptors for a series of compounds from the National Cancer Institute's Yeast AntiCancer drug screen. The activity classes are either active—at least one single yeast strain was inhibited more than 70%, or inactive—no yeast strain was inhibited more than 70%. The data set has 5,322 samples with 635 samples of active compounds.
4. The Page data set (Esposito et al. 1994) consists of classifying all the blocks of a page layout of a document that has been detected by a segmentation process. This data set originally had five classes, and we have converted it to two classes: 4,913 text examples and 560 non-text examples.
5. The Satimage data set (Blake et al. 1998) originally contained six classes. For our purposes, we used the two-class conversion outlined in (Provost et al. 1998), where the smallest class receives one label and the others are part of the larger class. This produced a data set with 5,809 majority class examples and 626 minority class examples.

Table 1 Data set distributions, in an increasing order of class imbalance

Data set	Examples	Features	Class balance
Phoneme	5,404	5	79:21
Segment	2,310	19	86:14
E-state	5,322	12	88:12
Page	5,473	10	90:10
Satimage	6,435	36	90:10
Compustat	Training = 10,358; Testing = 3,258	20	96:4
Oil	937	49	96:4
Mammography	11,183	6	98:2

6. Compustat North America is a database of U.S. and Canadian fundamental and market information on various active and inactive publicly held companies. It provides more than 300 annual Income Statement, Balance Sheet, Statement of Cash Flows, and supplemental data items. We extracted various financial ratios from the database, and after filtering companies based on the rating we had a total of 10,358 in the training set and 3,258 in the testing set. To make the problem resemble a real-world scenario, the training set comprised data from 1995 to 2001 and the testing set comprised data from 2002 to 2004. The interested reader can acquire the Compustat data set from the authors.
7. The Oil data set was provided by Kubat et al. (1998). This data set has 41 oil slick samples and 896 non-oil slick samples.
8. The Mammography data set (Woods et al. 1993) has 11,183 total examples with 260 calcifications. Using a trivial classifier would yield 97.68% default accuracy. However, such a classifier would most likely yield a poor total cost, particularly when the cost ratio is very skewed.

We used five random runs of 10-fold cross-validation framework for all but the Compustat data set since it was already associated with a testing set. We wanted to retain the complexity of the original Compustat testing set, as it was derived from a different time period.

While these data sets are devoid of any known testing costs, we injected a range of cost ratios to carefully establish the generality of the wrapper framework and compare to the cost-sensitive learning methods. As a convention, we will use the notation of

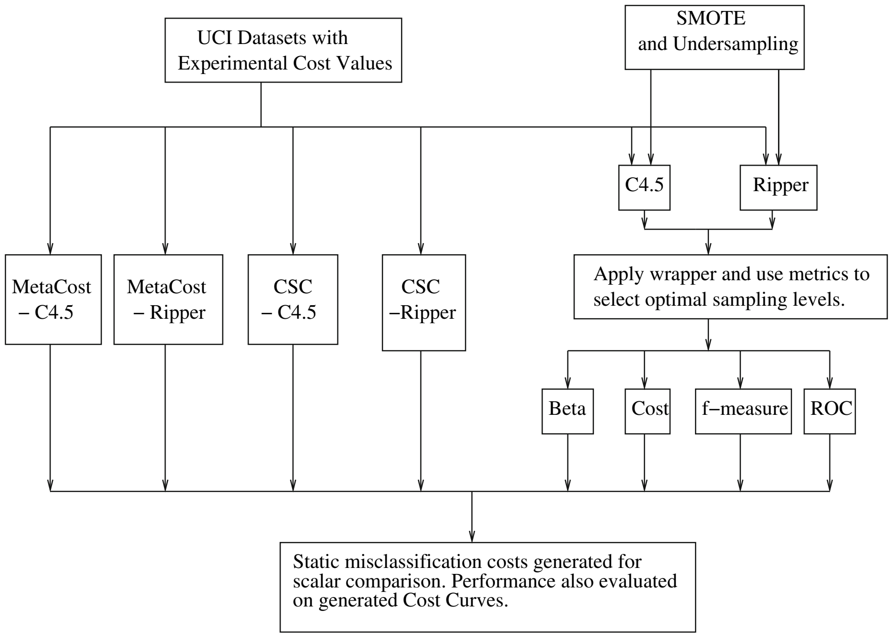
UCI Datasets with Experimental Cost Values
SMOTE and Undersampling
C4.5
Ripper
MetaCost – C4.5
MetaCost – Ripper
CSC – C4.5
CSC –Ripper
Apply wrapper and use metrics to select optimal sampling levels.
Beta
Cost
f–measure
ROC
Static misclassification costs generated for scalar comparison. Performance also evaluated on generated Cost Curves.

Fig. 5 Experiments overview

(*FN*:*FP*) to indicate the cost ratio for making a false negative error to a false positive error. For instance a ratio of (2:1) indicates that it is twice as costly to make a false negative than to make a false positive. This aligns with the premise that it is more costly to call a positive (minority) class negative than to call a negative (majority) class positive.

We compare the wrapper and cost-based learners under several cost scenarios. First, we consider a case where costs are known at training and testing times. Then, we investigate a situation where costs will remain unknown at both training and testing times. Finally, we study the generalization of classifiers through the cost space. We also compare the impact of different evaluation criteria in the wrapper mode. The C4.5 and Ripper classifiers learned on the resampled data sets (after Wraper) are compared with MetaCost (Domingos 1999) and Cost-Sensitive Classifier (CSC) (Zadrozny et al. 2003) using the same base C4.5 and Ripper classifiers. We used the Weka implementations of MetaCost and Cost-Sensitive Classifier (Witten and Frank 2005). Figure 5 provides an overview of our experiments.

4 Results: unfixed cost data sets

We will first discuss the analysis using *AUROC* and *f-measure* and then present the cost-based results.

4.1 AUROC and f-measure analysis

Our first experiment considered the effect of using *AUROC* and *f-measure* for guiding the wrapper search under a (1:1) cost environment. This experiment allowed us to observe the interaction between an objective function for the wrapper, sampling level, and an evaluation metric on the testing set. Tables 2 and 3 summarize the results with C4.5 and Ripper as base classifiers, respectively. The results are averaged over the 50 runs (five different random trials and 10-fold CV during each trial). The standard deviations are also shown beside the averaged performance measures. The cost column represents the cost of making errors at (1:1); thus, it is essentially the total number of errors. For instance, for the Mammography data set if *AUROC* is used with the wrapper, the amount of SMOTE is 720%, 14% of the majority class is removed, and the testing set *f-measure* is 0.619 and *AUROC* is 0.907.

We found that SMOTE and undersampling generally help the base classifiers learned on the different data sets. The results carry more significance for Ripper (see Table 3). Irrespective of the wrapper evaluation function, the sampling methods always result in an improved AUROC over the base classifier. In all but two cases, the *f-measure* is also improved. The two deviations from the norm are the Page and Mammography data sets. While Page had marginal deterioration over the base classifier, the *wrapper-AUROC* on the Mammography data set results in a 2.5% drop over the base classifier for the *f-measure*. However, *AUROC* is always improved. We believe this to be an artifact of a higher FPrate as we focus on the minority class being less detrimental.

Table 2 Average sampling levels and 10-fold performance measures for a (1:1) cost matrix (equal costs of errors) with the C4.5 classifier, including the standard deviations

	Wrapper-evaluation	S	US	Cost	f-measure	AUROC
Phoneme	Base	–	–	364 ± 50	0.772 ± 0.032	0.905 ± 0.018
	f-meas	212	6	415 ± 53	**0.778 ± 0.025**	**0.922 ± 0.017**
	AUROC	562	2	436 ± 51	0.772 ± 0.024	0.921 ± 0.019
Segment	Base	–	–	9 ± 6	0.973 ± 0.018	0.957 ± 0.011
	f-meas	460	8	5 ± 3	**0.985 ± 0.012**	**0.966 ± 0.008**
	AUROC	420	13	7 ± 4	0.977 ± 0.012	**0.966 ± 0.008**
E-State	Base	–	–	319 ± 5	0.003 ± 0.009	0.495 ± 0.006
	f-meas	640	20	1176 ± 297	0.227 ± 0.018	0.580 ± 0.030
	AUROC	1030	21	1298 ± 276	**0.242 ± 0.021**	**0.597 ± 0.027**
Page	Base	–	–	78 ± 19	**0.857 ± 0.039**	0.959 ± 0.013
	f-meas	200	3	86 ± 22	0.856 ± 0.033	**0.972 ± 0.004**
	AUROC	690	3	94 ± 22	0.847 ± 0.033	**0.972 ± 0.006**
Satimage	Base	–	–	259 ± 19	0.569 ± 0.029	0.900 ± 0.019
	f-meas	622	6	275 ± 27	0.609 ± 0.036	0.919 ± 0.010
	AUROC	644	4	258 ± 16	**0.628 ± 0.026**	**0.926 ± 0.014**
Oil	Base	–	–	25 ± 6	0.314 ± 0.256	0.631 ± 0.119
	f-meas	260	2	21 ± 7	0.467 ± 0.119	0.722 ± 0.110
	AUROC	300	15	22 ± 8	**0.518 ± 0.133**	**0.736 ± 0.101**
Mammo.	Base	–	–	76 ± 15	0.644 ± 0.101	0.863 ± 0.031
	f-meas	200	2	92 ± 19	**0.672 ± 0.063**	0.896 ± 0.035
	AUROC	720	14	120 ± 29	0.619 ± 0.058	**0.907 ± 0.038**

The results are averaged over the five different random runs. Wrapper evaluation indicates the evaluation function used for guiding the wrapper mode. The SMOTE and Undersample columns indicate the amounts of SMOTE and undersampling discovered by the wrapper. The subsequent columns indicate the evaluation function used on the testing set. Base indicates the base classifier without any sampling. The best performance is shown in bold

We note that there is not a distinct positive correlation between the *f-measure* in the wrapper-mode and corresponding improvements in the final evaluation. For instance, C4.5 improved the *f-measure* on four data sets when using *AUROC* as the wrapper evaluation metric rather than the *f-measure*. The question then becomes, why does this happen with the *f-measure* and not with *AUROC*? We attribute this to the nature of the two metrics. The *f-measure* is tuned to a fixed threshold of 0.5. This means that the fixed point threshold of 0.5 can become unstable. We believe this results in more generalized performances when using *AUROC* as the wrapper evaluation function. Essentially, our goal was to identify which one was less prone to reaching a local maximum in the wrapper scheme.

Both *wrapper-AUROC* and *wrapper-f measure* generally result in different amounts of SMOTE and undersampling, as they optimize different properties of the classifier. As expected using different classifiers also results in different levels of sampling. We

Table 3 Average sampling levels and 10-fold performance measures for a (1:1) cost matrix (equal costs of errors) with the Ripper classifier

	Wrapper-evaluation	SM	US	Cost	f-measure	AUROC
Phoneme	Base	–	–	385 ± 45	0.748 ± 0.032	0.823 ± 0.022
	f-meas	150	9	491 ± 39	0.753 ± 0.017	0.899 ± 0.015
	AUROC	162	6	481 ± 59	**0.759** ± **0.025**	**0.902** ± **0.020**
Segment	Base	–	–	9 ± 4	0.971 ± 0.014	0.956 ± 0.007
	f-meas	300	5	6 ± 4	**0.982** ± **0.015**	**0.968** ± **0.006**
	AUROC	460	9	7 ± 5	0.979 ± 0.016	0.967 ± 0.007
E-State	Base	–	–	318 ± 4	0.000 ± 0.000	0.493 ± 0.000
	f-meas	520	24	1501 ± 372	**0.236** ± **0.018**	**0.579** ± **0.029**
	AUROC	660	18	1480 ± 160	0.235 ± 0.012	0.574 ± 0.018
Page	Base	–	–	78 ± 16	0.859 ± 0.026	0.906 ± 0.017
	f-meas	190	5	84 ± 16	**0.862** ± **0.024**	0.966 ± 0.011
	AUROC	680	14	99 ± 25	0.845 ± 0.036	**0.969** ± **0.006**
Satimage	Base	–	–	226 ± 23	0.571 ± 0.055	0.720 ± 0.029
	f-meas	400	6	263 ± 47	**0.645** ± **0.047**	0.918 ± 0.017
	AUROC	488	7	287 ± 48	0.633 ± 0.037	**0.925** ± **0.013**
Oil	Base	–	–	24 ± 8	0.270 ± 0.249	0.529 ± 0.122
	f-meas	310	6	21 ± 9	0.521 ± 0.180	0.695 ± 0.092
	AUROC	570	11	21 ± 10	**0.572** ± **0.155**	**0.723** ± **0.110**
Mammo.	Base	–	–	78 ± 16	0.648 ± 0.096	0.756 ± 0.052
	f-meas	180	5	81 ± 14	**0.699** ± **0.047**	0.869 ± 0.023
	AUROC	240	9	92 ± 22	0.675 ± 0.061	**0.874** ± **0.026**

Same convention as Table 2 holds

posit that the amount of SMOTE or undersampling is not dependent on the class skew, but on the properties of the feature space. Consider the E-state data set, which has the largest amount of SMOTE and undersampling and yet it is certainly not the most unbalanced data set. If we look at the statistics, both C4.5 and Ripper have a very poor performance on E-state. However, there is a significant improvement offered by the sampling methods. We believe that a wrapper method can allow one to empirically discover the relevant amounts of sampling, as it is certainly intrinsically tied in with the data properties. Lastly, as expected, the base classifier achieves the lowest cost at (1:1) when both FP and FN are equally costly.

4.2 Timing

Table 4 shows the average time (in seconds) taken on a data set for a single fold. We show the time taken by both the sequential and distributed versions of the wrapper framework. As a representative of all the wrappers, we show the timings of the *wrapper-AUROC* method, especially given that this results in larger amounts of

Table 4 Time in seconds by different approaches

Data set	*Wrapper-AUROC* SEQ (C4.5)	*Wrapper-AUROC* DIST (C4.5)	*Wrapper-AUROC* SEQ (Ripper)	*Wrapper-AUROC* DIST (Ripper)
Compustat	1,061	22	3,353	89
Estate	3,028	71	727	22
Mammography	181	7	308	14
Oil	30	2	45	2
Page	75	9	276	16
Phoneme	1,816	87	673	33
Satimage	2,797	105	3,010	81
Segment	726	33	120	5
Average	1,214.25	42	1,064	32

SEQ: Sequential version; DIST: Distributed version

sampling than the *wrapper-f measure*. Exactly the same underlying code-base is part of the sequential and distributed runs. The distributed implementations provide a significant improvement in timing, making the wrapper a very competitive approach. On an average, it takes only 42 s with C4.5 and 32 s with Ripper to complete the wrapper search, which is very encouraging especially in the light of performance improvements.

We note that the wrapper with Ripper is faster than the wrapper with C4.5 because of the relatively lower amounts of undersampling and SMOTE with Ripper as the base classifier. The time taken for the entire wrapper paradigm is largely a function of the number of minority class examples and the associated learning complexity, which drives the search for the SMOTE amount.

4.3 Comparison to cost-sensitive classifiers

We wanted to examine an empirical relationship between learning from imbalanced data sets and cost-sensitive classification. Can we effectively handle the problem of imbalance and achieve lower costs during testing than cost-sensitive classifiers?

We compared the wrapper-induced classifiers with MetaCost (Domingos 1999) and CostSensitiveClassifier (CSC) (Zadrozny et al. 2003). MetaCost wraps a "meta-learning" stage around an error-based classifier to effectively minimize cost. It estimates the conditional probability distributions of a data set via bagging (Breiman 1996) Combining probability estimates with a simple expected-cost framework, MetaCost selects the class as $argmin_i \sum_j P(y = j|x)C(i|j)$. This essentially relabels a class in the training data such that the expected cost is minimized. A classifier is then learned on the relabeled data and evaluated on the testing set.

The CSC method draws a new training set in accordance with each example's misclassification cost. However, when costs are highly skewed, overfitting tends to occur as a limited number of examples are resampled heavily. To counteract this, rejection sampling is used. The newly drawn training set keeps a selected example with a probability of $\frac{c}{Z}$ where c is the examples' misclassification cost and Z is a selected constant greater than the maximum cost of the original training set.

We show comparisons using two illustrations. First, we use cost trends where the *y-axis* indicates the total cost as the cost ratio varies on the *x-axis*. Second, we use the cost curves (Drummond and Holte 2006).

4.3.1 Cost trends

Here, we compared the wrapper framework for countering class imbalance with the MetaCost and Cost-Sensitive Classifier (CSC). We considered various cost ratios of (*FN*:*FP*)—(2:1), (5:1), (10:1), and (20:1). We used these cost ratios, as these have been previously considered in other studies (Domingos 1999; Drummond and Holte 2006).

If the classifiers were built without utilization of a cost-matrix, then the generated posterior probabilities were thresholded at the corresponding cost ratios for the testing set. This scenario can be used when the costs are completely unknown during training. The classifiers generate a posterior probability distribution on the testing set. Depending on the cost environment during the testing stage, the probabilities can effectively be thresholded to reflect the cost distributions. That is the threshold for decision making can be derived from the operating cost ratio (see Eq. 10). However, the classifiers are not re-learned. The base classifier, *wrapper-f measure*, and *wrapper-AUROC* fall under this umbrella. This allows us to directly compare the effectiveness of dealing with imbalance without incorporating costs during the training phase. We would like to point the reader to Appendix A, which shows the benefits from thresholding to reflect the costs after applying SMOTE and undersampling. On the other hand, if the classifiers incorporated costs during learning, such as *wrapper-cost*, *wrapper-β*, MetaCost and CSC, then no thresholding was performed. This is under the assumption that the classifiers have already been optimized on an objective function attuned to a particular cost. Thus, we generated multiple such classifiers for the different cost ratios.

Figures 6 and 7 show the range of costs obtained across the eight data sets, for different cost ratios. The x-axis in the figures show the different data sets and the y-axis shows the normalized costs. We normalized the costs between 0 and 1 to enable a better display of results.

The results show an interesting trend. At (2:1), the cost-sensitive classifiers generally resulted in lower costs than the wrapper-based methods. However, as we increased the cost ratios, particularly at (10:1) and (20:1), the wrapper based methods begin to result in lower costs than the cost-sensitive classifiers that are directly trained on the corresponding costs. The *wrapper-AUROC* and *wrapper-f measure* make a very compelling case at higher costs; without using costs during training, they are able to often outperform the cost-sensitive classifiers. The results hold with both C4.5 and Ripper. Thus, a combination of the sampling approaches for countering class imbalance and thresholding is effective in overcoming the cost distributions. In fact we found that the

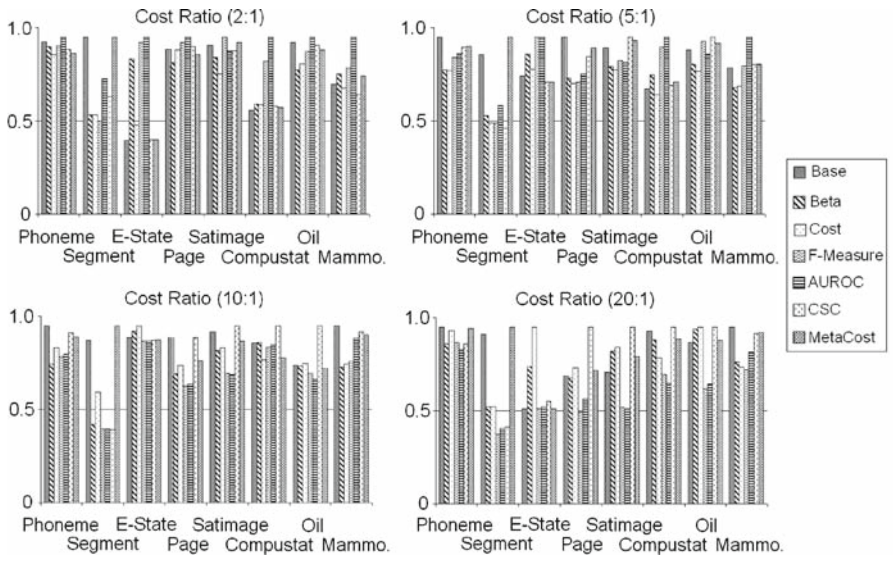

Fig. 6 Cost trends with C4.5 as base classifier. Beta is wrapper-β and Cost is wrapper-cost

wrapper-AUROC and *wrapper-f measure* are quite effective across the cost ranges. We conjecture this to be an outcome of an improvement in the quality of estimates, thus resulting in the reduced overall costs (Cieslak and Chawla 2006). The effective treatment of imbalance without using costs during training can result in robust posterior probability estimates applicable to a variety of cost scenarios during testing. This is certainly desirable, especially when the costs are unknown or can change during testing. Between C4.5 and Ripper, the former is more effective as a classifier in reducing the costs during testing. Lastly, all the methods result in lower costs over the base classifier at cost ratios > (2:1). Thus, treating for imbalance or changing the objective function to be cost-sensitive is always preferred over the thresholded estimates from a base classifier.

Zhou and Liu (2006) found that there might be a disconnect between learning from imbalanced data sets and cost-sensitive learning, which is contrary to our observations. However, we believe the differences in their conclusions can be attributed to a lack of sufficient search in the sampling parameter space. They resort to "default" sampling parameters, but, as we point out, there are really no "default" sampling parameters for a data set and a classifier. We show that using wrapper-induced sampling methods on a multi-class data set, we were able to achieve lower costs than any other known method. Lastly, their cost functions were constrained to 1:10, while we extended our study to 1:100. However, our comprehensive experiments on real-world data sets, including the to be discussed, multi-class KDD-99 Cup data set, demonstrate that treating a data set for imbalance first is often beneficial when compared with cost-sensitive classifiers.

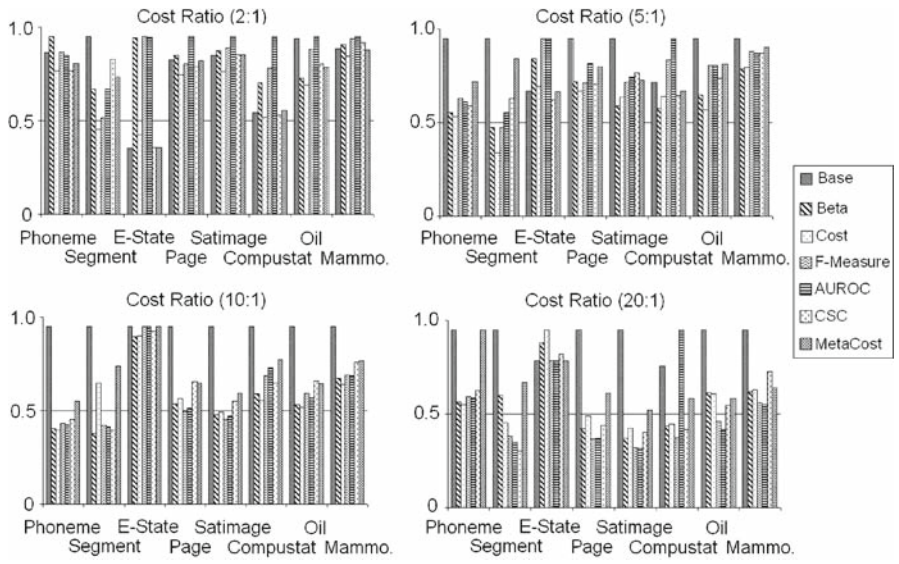

Fig. 7 Cost trends with Ripper as base classifier. Beta is wrapper-β and Cost is wrapper-cost

4.3.2 Cost curves

Conditioning on a particular *TP rate* or *FP rate* cut-off may not immediately establish the expected generalization capacity of a classifier. It may be more appropriate to gauge the performance of classifiers across a range of *TP rate* and *FP rate*. Thus, cost curves (Drummond and Holte 2006) founded on expected cost may yield a clearer understanding of the classifiers within that operating environment. These curves plot performance (expected cost normalized between 0 and 1) on the Y-axis. This function is given by:

$$NE[C] = (1 - TP\ rate - FP\ rate) \times PCF(+) + FP \tag{11}$$

The X-axis combines cost and class distribution in the following manner:

$$PCF(+) = \frac{p(+)C(-|+)}{p(+)C(-|+) + p(-)C(+|-)} \tag{12}$$

where $C(-|+)$ is the cost of misclassifying a positive example as negative, $C(+|-)$ is the cost of misclassifying a negative example as positive, $p(+)$ is the probability of a positive example and $p(-) = 1 - p(+)$ is the probability of a negative example. This is a multi-faceted tool which enables sophisticated analysis of classifier performance under fluid conditions. Overall, cost curves allow visualization of the interplay of classifiers as the weighted importance of two classes wax and wane. There is a correspondence of a point in ROC space to a line in cost space. A single (*TP rate*, *FP rate*) pair from the ROC space gets translated to a line segment in the cost space.

Thus, different operating points from the ROC curve will represent different lines in the cost space.

For clarity in presentation, we show the lower envelopes of the cost curves of the following four methods: *wrapper-f measure*, *wrapper-AUROC*, CSC, and MetaCost. We wanted to draw our comparisons among classifiers optimized on cost-insensitive measures and the cost-sensitive classifiers that incorporate costs during learning. We constructed the cost curves as follows. The cost-sensitive classifiers–MetaCost and CSC—were trained with the different cost ratios, drawn from the following *FN*:*FP* – 1:1, 2:1, 5:1, 10:1, 20:1, 30:1, 40:1, …100:1. This resulted in 13 pairs of (*TP rate*, *FP rate*) for each of the above methods or a total of 91 (13×7) cost curves. After plotting all the possible cost curves from different methods, we only retained their corresponding lower envelopes to reflect their best operating ranges.

Figures 8 and 9 show the cost curves for C4.5 and Ripper, respectively. The x-axis in the figures represents the Probability Cost Function (PCF) and the y-axis represents Normalized Expected Cost (NEC). As seen in the figures, the methods start-off generally overlapping with each other or the cost-sensitive classifiers resulting in lower envelopes, but as the PCF increases the classifiers begin to separate. At lower PCF, the cost-sensitive classifiers result in slightly lower costs than the wrapper classifiers for some data sets. As the PCF increases, the advantage of treating the data sets for class imbalance stands out. Based on the comparisons across all the data sets, we find that treating the data sets for imbalance using the wrapper mode results in the most effective sets of classifiers across the range of PCF. If a classifier produces the lowest envelope it is the most optimal across the entire range of PCF's. Our observations corroborate the findings in the previous subsection. The lower envelopes of the cost-insensitive measures generally dominate over the cost-sensitive classifiers, especially at higher costs. Thus, using wrappers to first counter the class imbalance problem was generally more cost-effective than the cost-sensitive classifiers. We also found that *wrapper-AUROC* was either comparable to or better than *wrapper-f measure*, and can be the recommended optimization criterion. The results are quite compelling as they imply that optimizing on class imbalance, without incorporating costs, is often a more optimal strategy. Between MetaCost and CSC, we found that MetaCost resulted in lower costs using C4.5 as the base classifier, but this trend got reversed with Ripper. We believe MetaCost relies heavily on the instability of the base classifier, given the bootstraps.

5 Fixed cost KDD cup intrusion data set

We considered the intrusion detection data set from 1999 KDD Cup. This data set not only provides actual costs for making different types of errors, but also demonstrates the proposed wrapper framework on a multi-class data set. A cost matrix was used in the scoring of the competition as shown in Table 5 (Elkan 1999).

We pre-processed the data set as follows. There were many duplicate examples in the original 5 million example training set. All duplicate examples were removed. We also undersampled both the normal and neptune (dos) class by removing examples

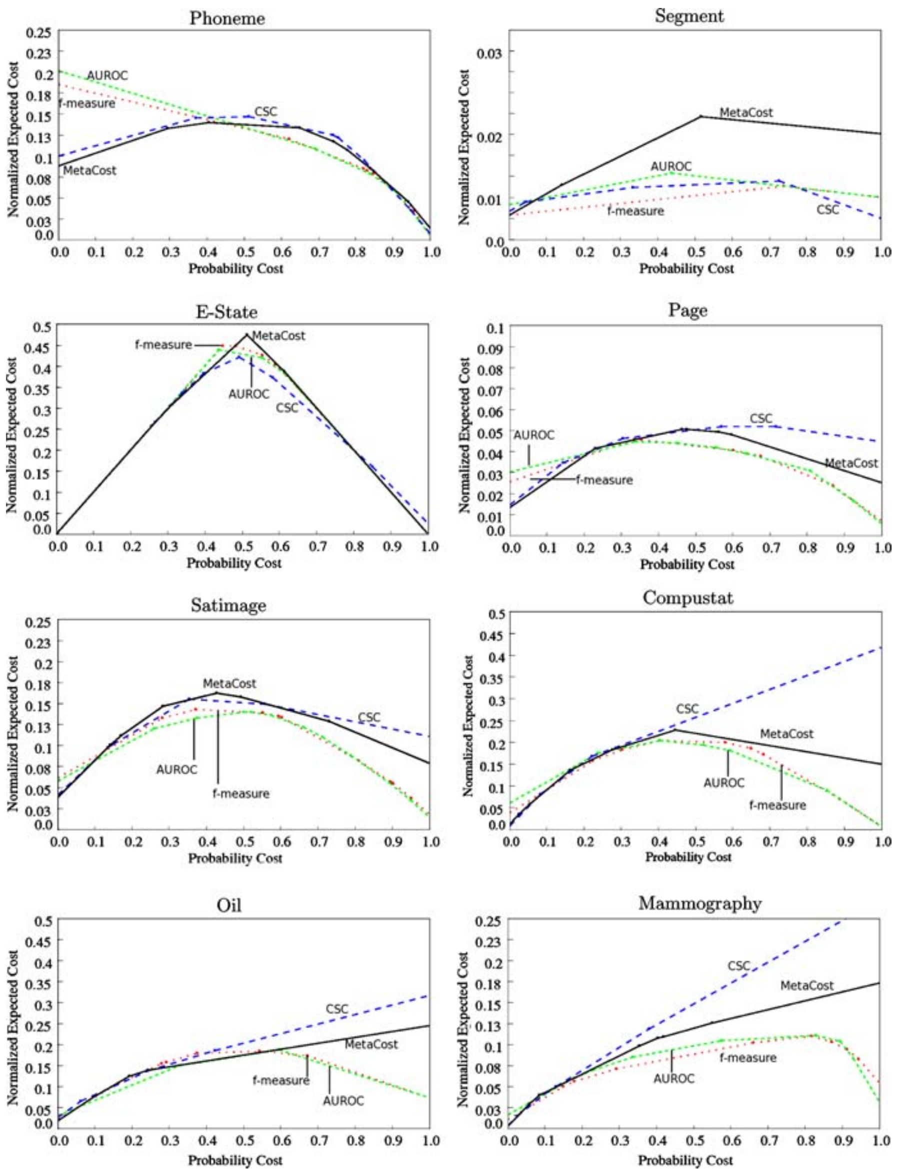

Fig. 8 C4.5 cost curves

which occurred only once. For Training Data 1 as in Table 5, we undersampled the normal class, and for the Training Data 2 we undersampled both the normal and neptune classes. Note that for both of these sets of experiments, the test set remained unchanged. Table 6 shows the results.

It can be seen that our approach with RIPPER as the classifier produced the lowest cost per example after undersampling for both the normal and neptune classes

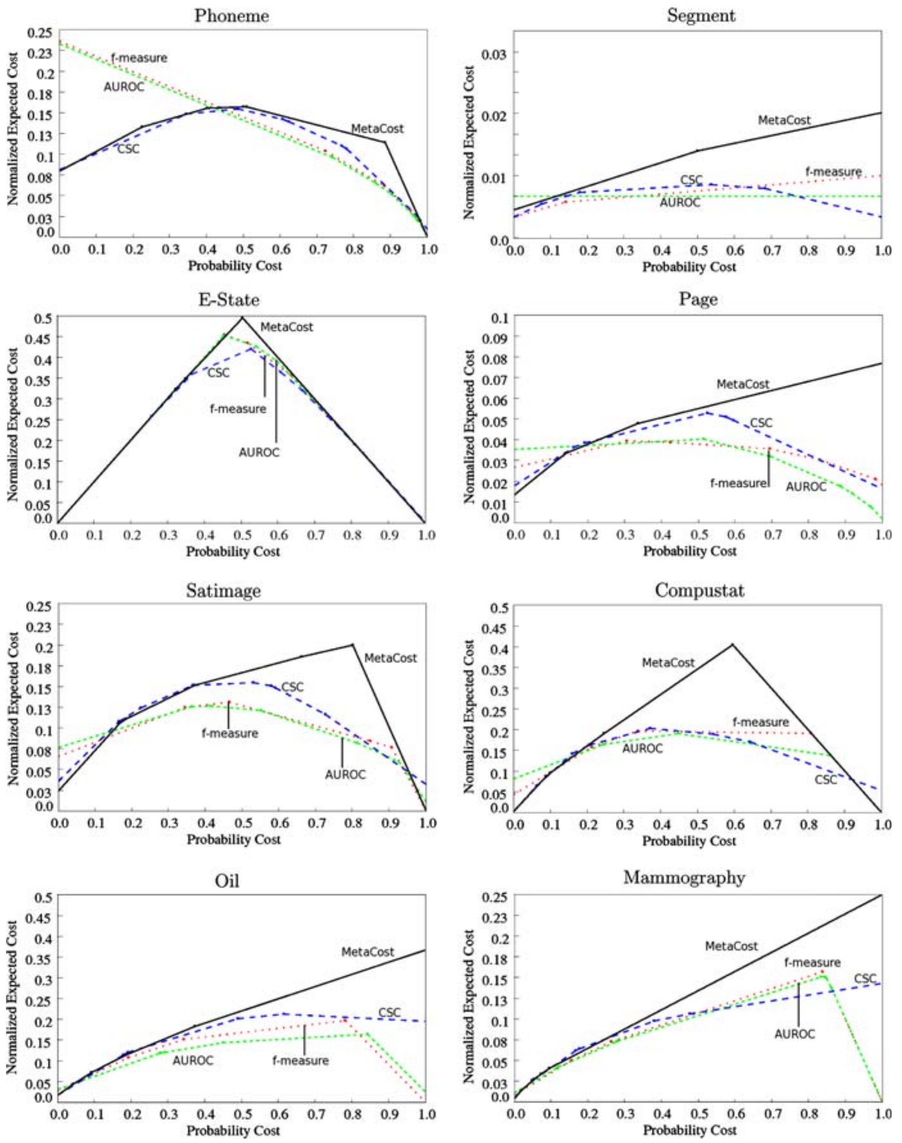

Fig. 9 Ripper cost curves

(Training Data 2), and applying 100% SMOTE to the u2r class, while keeping the r2l class unchanged. This was better than the winner of the contest and better than the subsequent results from the literature. Also using C4.5 as the base classifier with SMOTE (200% u2r and 300% for r2l) resulted in lower cost than the other published techniques. Note that the respective amounts of SMOTE and undersampling were discovered via the Wrapper.

Table 5 Cost matrix used for scoring entries in KDD-99 CUP competition

Actual/predicted	dos	u2r	r2l	Probe	Normal
dos	0	2	2	1	2
u2r	2	0	2	2	3
r2l	2	2	0	2	4
Probe	2	2	2	0	1
Normal	2	2	2	1	0

Table 6 Comparison of results obtained on the original KDD CUP 99 test data

	Cost per test example
Winning Strategy (Elkan 1999)	0.2331
Decision Tree (Amor et al. 2004)	0.2371
Naive Bayes (Amor et al. 2004)	0.2485
Multi-classifier (Sabhnani and Serpen 2003)	0.2285
Using C4.5 on Training Data 1 u2r (200)—r2l (0)	0.2478
Using RIPPER on Training Data 1 u2r (100)—r2l (0)	0.2444
Using C4.5 on Training Data 2 u2r (200)—r2l (300)	0.2051
Using RIPPER on Training Data 2 u2r (100)—r2l (0)	**0.2049** (Chawla et al. 2005)

The numbers beside u2r and r2l indicate the SMOTE percentage utilized for the experiments

6 Conclusions

In this work, a wrapper approach was utilized to determine the percentage of minority examples to add to the training set and the percentage for undersampling the majority class examples. We used a variety of data sets with different class distributions and characteristics, including a number of real world data sets. The wrapper approach works by performing a guided search of the parameter space. The evaluation function was applied with a fivefold cross-validation on the training set. Once the best percentages for undersampling and SMOTE are found they can be used to build a classifier on the updated training set and applied on the unseen testing set. We have demonstrated the ability to optimize sampling levels for an evaluation function, resulting in effective generalization performance.

We also wished to view the impact of using a threshold function on the probabilities produced by a classifier to generate optimal decisions in a cost-based problem. As we note in Appendix A, cost thresholding works particularly well at reducing the total cost of a classifier, particularly when operating at a higher cost ratio. Thus, we may assume that sampling has allowed classifiers to develop a calibration of probabilities better suited for higher cost ratios.

Our conclusions, connected to the questions posited in Sect. 1, can be summarized as follows.

- The proposed wrapper-based paradigm, when using different evaluation functions, was effective in automatically discovering the potentially optimal amounts of sampling for a data set. This was effective in countering the cost-imbalance during training and testing as well. We showed that the classifiers learned on sampling parameters discovered when using the cost-transparent metrics, such as *AUROC* and *f-measure*, retain their effectiveness when costs were introduced during the testing stage. Both the metrics were very effective compared to the base classifier as demonstrated by the *AUROC* and *f-measure* on the testing sets. Thus, changing the data distribution is important to learning the classifiers in an imbalanced domain.
- We also compared our approaches to cost-sensitive learners. In an analysis of raw total cost, we found a strong tendency for *wrapper-AUROC* and *wrapper-f measure* to generally outperform the other wrapper evaluation functions. Moreover, at higher cost ratios they also dominated over CSC and MetaCost.
  Another set of comparisons among the classifiers was done using cost curves. Using the lower envelope formed by multiple classifiers learned at different cost ratios, it was also possible to examine each evaluation metric and the cost-based learners throughout cost space. In this section, we have noted that *wrapper-AUROC* tends to produce classifiers that result in more effective generalization in the PCF space, even better than the cost-sensitive classifiers optimized at different cost ratios.
- We also showed that a combination of undersampling and SMOTE offers a significant advantage over several cost-based classifiers in a number of realistic cost environments, including champion level performance on a KDD Cup Challenge.

Acknowledgements We are grateful to Robert Holte for providing the Cost Curves software and Oil data set. David Cieslak was partially supported by the the Arthur J. Schmitt Fellowship. We are very thankful to the reviewers and the guest editors for their helpful comments.

Appendix A: cost comparison between *thresholded* and *unthresholded* C4.5 leaf estimates

Table A1 contains cost comparison between C4.5 decision trees learned using the *wrapper-AUROC* and *wrapper-f measure*, called AUROC and f-measure in the table, respectively. AUROC and f-measure are followed by unthresholded and thresholded. Unthresholded is the decision at the leaf based on the default 0.5 threshold. Thresholded follows the technique outlined in Sect. 2.6 that uses the posterior probabilities and the known cost ratio to generate a threshold during testing. The subsequent columns in the table indicate the total misclassification costs at different cost-ratios. The classifiers are learned in the same manner; however, they differ in the method of evaluation on unseen examples. We note that thresholdeded predictions tend to be better at higher cost ratios, but weaker in some cases on lower cost ratios. Better performance on higher cost ratios is desirable, so we have elected to use the thresholded decision process throughout this article.

Table A1 Cost comparison between thresholded and unthresholded C4.5 classifiers

		(1:1)	(2:1)	(5:1)	(10:1)	(20:1)
Phoneme	AUROC thresholded	3495	4395	6005	7410	9125
	AUROC unthresholded	3758	4285	5866	8501	13771
	f-measure thresholded	3325	4165	5865	7280	9615
	f-measure unthresholded	3567	4169	5975	8985	15005
Segment	AUROC thresholded	75	105	195	220	405
	AUROC unthresholded	79	105	183	313	573
	f-measure thresholded	50	70	160	220	370
	f-measure unthresholded	61	82	145	250	460
E-State	AUROC thresholded	12985	16510	20590	22920	24085
	AUROC unthresholded	12985	14095	17425	22975	34075
	f-measure thresholded	11765	15930	20655	22960	23645
	f-measure unthresholded	11765	13215	17565	24815	39315
Page	AUROC thresholded	945	1305	1860	2315	3370
	AUROC unthresholded	1055	1259	1871	2891	4931
	f-measure thresholded	865	1260	1750	2265	2915
	f-measure unthresholded	976	1224	1968	3208	5688
Satimage	AUROC thresholded	2330	3505	5880	7580	9705
	AUROC unthresholded	2763	3679	6427	11007	20167
	f-measure thresholded	2480	3815	5970	7675	9805
	f-measure unthresholded	2902	3852	6702	11452	20952
Compu.	AUROC thresholded	1330	2285	4075	4770	6025
	AUROC unthresholded	1330	1690	2770	4570	8170
	f-measure thresholded	1200	1950	3815	4680	6505
	f-measure unthresholded	990	1375	2530	4455	8305
Oil	AUROC thresholded	220	405	710	1080	1630
	AUROC unthresholded	290	387	678	1163	2133
	f-measure thresholded	215	370	770	1140	1560
	f-measure unthresholded	277	383	701	1231	2291
Mammo.	AUROC thresholded	1200	1895	3465	4940	7840
	AUROC unthresholded	1451	1805	2867	4637	8177
	f-measure thresholded	925	1550	2860	4205	6835
	f-measure unthresholded	1165	1528	2617	4432	8062

References

Amor NB, Benferhat S, Elouedi Z (2004) Naive bayes vs. decision trees in intrusion detection systems. In: Proceedings of the ACM symposium on applied computing, pp 420–424

Banfield RE, Hall LO, Bowyer KW, Kegelmeyer WP (2005) Ensembles of classifiers from spatially disjoint data. In: Proceedings of the sixth international conference on multiple classifier systems, pp 196–205

Batista GEAPA, Prati RC, Monard MC (2004) A study of the behavior of several methods for balancing machine learning training data. SIGKDD Explorations 6(1):20–29

Blake CL, Newman DJ, Hettich S, Merz CJ (1998) UCI repository of machine learning databases. URL: http://www.ics.uci.edu/~mlearn/MLRepository.html

Bowyer KW, Hall LO, Chawla NV, Moore TE (2000) A parallel decision tree builder for mining very large visualization datasets. In: Proceedings of the IEEE International conference on systems, man and cybernetics

Breiman L (1996) Bagging predictors. Machine Learn 24(2):123–140

Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) SMOTE: synthetic minority over-sampling technique. J Artif Intel Res 16:321–357

Chawla NV, Hall LO, Joshi A (2005) Wrapper-based computation and evaluation of sampling methods for imbalanced datasets. In: KDD workshop: utility-based data mining

Chawla NV, Japkowicz N, Kołcz A (eds) (2003) Proceedings of the ICML'2003 workshop on learning from imbalanced data sets

Chawla NV, Japkowicz N, Kolcz A (2004) Editorial: learning from imbalanced datasets. SIGKDD Explorations 6(1):1–6

Cieslak D, Chawla NV (2006) Calibration and power of PETs on unbalanced datasets. TR 2006-12, Department of Computer Science and Engineering, University of Notre Dame

Cohen WW (1995a) Fast effective rule induction. In Prieditis A, Russell S (eds) 12th International conference on machine learning, Morgan Kaufmann, Tahoe City, CA, pp 115–123

Cohen WW (1995b) Learning to classify English text with ILP methods. In: 5th International workshop on inductive logic programming, pp 3–24

Dietterich T, Margineantu D, Provost F, Turney P (eds) (2000) Proceedings of the ICML'2000 workshop on cost-sensitive learning

Domingos P (1999) MetaCost: a general method for making classifiers cost-sensitive. In: Knowledge discovery and data mining, pp 155–164

Drummond C, Holte R (2003) C4.5, class imbalance, and cost sensitivity: Why under-sampling beats over-sampling. In: Proceedings of the ICML'03 workshop on learning from imbalanced data sets

Drummond C, Holte RC (2006) Cost curves: an improved method for visualizing classifier performance. Machine Learn 65(1):95–130

Dumais S, Platt J, Heckerman D, Sahami M (1998) Inductive learning algorithms and representations for text categorization. In: Seventh international conference on information and knowledge management, pp 148–155

Elkan C (1999) Results of the KDD'99 classifier learning contest. http://www.cse.ucsd.edu/~elkan/clresults.html

Elkan C (2001) The foundations of cost-sensitive learning. In: Proceedings of the seventeenth international joint conference on artificial intelligence, pp 973–978

Esposito F, Malerba D, Semeraro G (1994) Multistrategy learning for document recognition. Appl Artif Intel 8:33–84

Ferri C, Flach P, Orallo J, Lachice N (eds) (2004) First workshop on ROC analysis in AI. ECAI

Kubat M, Holte RC, Matwin S (1998) Machine learning for the detection of oil spills in satellite radar images. Machine Learn 30(2–3):195–215

Kubat M, Matwin S (1997) Addressing the curse of imbalanced training sets: one sided selection. In: Proceedings of the fourteenth international conference on machine learning. Morgan Kaufmann, Nashville, Tennesse, pp 179–186

Lewis D, Ringuette M (1994) A comparison of two learning algorithms for text categorization. In: 3rd Annual symposium on document analysis and information retrieval, pp 81–93

Ling C, Li C (1998) Data mining for direct marketing problems and solutions. In: Proceedings of the fourth international conference on knowledge discovery and data mining (KDD-98). AAAI Press, New York, NY, pp 73–79

Maloof M (2003) Learning when data sets are imbalanced and when costs are unequal and unknown. In: Proceedings of the ICML'03 workshop on learning from imbalanced data sets

Mladenic D, Grobelnik M (1999) Feature selection for unbalanced class distribution and naive bayes. In: ICML, pp 258–267

Provost FJ, Domingos P (2003) Tree induction for probability-based ranking. Machine Learn 52(3):199–215

Provost FJ, Fawcett T, Kohavi R (1998) The case against accuracy estimation for comparing induction algorithms. In: Fifteenth international conference on machine learning, pp 445–453

Quinlan JR (1993) Programs for machine learning. Morgan Kaufmann

Sabhnani MR, Serpen G (2003) Application of machine learning algorithms to KDD intrusion detection dataset with misuse detection context. In: Proceedings of the international conference on machine learning: models, technologies, and applications, pp 209–215

Thain D, Tannenbaum T, Livny M (2005) Distributed computing in practice: the condor experience. Concur Comput Pract Exp 17:323–356

Weiss G, McCarthy K, Zabar B (2007) Cost-sensitive learning vs. sampling: which is best for handling unbalanced classes with unequal error costs? In: DMIN, pp 35–41

Weiss G, Provost F (2003) Learning when training data are costly: the effect of class distribution on tree induction. J Artif Intel Res 19:315–354

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques. Morgan Kaufmann

Woods K, Doss C, Bowyer KW, Solka J, Priebe C, Kegelmeyer WP (1993) Comparative evaluation of pattern recognition techniques for detection of microcalcifications in mammography. Int J Pattern Recog Artif Intel 7(6):1417–1436

Zadrozny B, Elkan C (2001) Learning and making decisions when costs and probabilities are both unknown. In: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data Mining, pp 204–213

Zadrozny B, Langford J, Abe N (2003) Cost-sensitive learning by cost-proportionate example weighting. In: ICDM, pp 435–442

Zhou Z, Liu X (2006) Training cost-sensitive neural networks with methods addressing the class imbalance problem. IEEE Trans Knowledge Data Eng 18(1):63–77