Learning Rules from Highly Unbalanced Data Sets

Jianping Zhang [1]
AOL, Inc.
44900 Prentice Drive
Dulles, VA 20166
jianpingz032@aol.com

Eric Bloedorn Lowell Rosen Daniel Venese
MITRE Corporation
7515 Colshire Drive
McLean, Virginia 22102-750
{bloedorn, lrosen, venese}@mitre.org

Abstract

This paper presents a simple and effective rule learning algorithm for highly unbalanced data sets. By using the small size of the minority class to its advantage this algorithm can conduct an almost exhaustive search for patterns within the known fraudulent cases. This algorithm was designed for and successfully applied to a law enforcement problem, which involves discovering common patterns of fraudulent transactions.

1. Introduction

Most of the standard classification algorithms usually assume that training examples are evenly distributed among different classes. However, as indicated in [1], unbalanced data sets often appear in many practical applications. Studies show that unbalanced class distributions in many applications cause poor performances from standard classification algorithms.

There have been many attempts at dealing with classification of unbalanced data sets. Methods include resizing training sets, adjusting misclassification costs, and recognition-based learning. Resizing training sets is a simple strategy that includes over-sampling minority class examples [2] and down-sizing the majority class [3]. Cost-sensitive classifiers [4] have been developed to handle the problems with different misclassification error costs, but may also be used for unbalanced data sets. Recognition-based learning approaches learn rules from the minority class examples with or without using the examples of the majority class [5].

In this paper, we describe a simple and effective recognition-based rule learning algorithm, RLSD (Rule Learning for Skewed Data), for highly unbalanced data sets. RLSD was developed for a law enforcement problem and achieved encouraging results in a field test. It is currently being transferred for a routine use. The goal of this project is neither to replace human inspectors nor to automate the fraudulent transaction detection process. Instead, it is intended to assists inspectors in identifying potential fraudulent transactions in an operational environment.

2. The Data Mining Task

Application of this algorithm is presented in terms of a hypothetical application for fraud detection. Each transaction in the database has a time stamp and is associated with a person. A person is often involved in more than one transaction. In addition to time and person, each transaction has six other features. One of the six features indicates if the transaction is fraudulent.

While fraud is defined over individual transactions, the aggregation of a person's transactions plays a vital role in identifying him. To prepare the data for mining, we create an example for each transaction. Each example is a vector of 70 attribute values that are mostly numeric. Most attributes are computed from the person's historical transactions during a prior period.

An example is marked as *bad*, if the corresponding transaction was found to be fraudulent. Otherwise, it is marked as *good*. The distribution of examples among these two classes, *bad* and *good*, is highly unbalanced with about 0.01% to 0.05% of all examples belonging to class *bad*.

Unlike many other classification tasks, the discovery of highly accurate rules is improbable due to the insufficient information. Human inspectors may have to pull out thousands of transactions before actually finding a fraudulent one. Human inspectors and the fraudulent detection system complement each other. Rules with 1% classification accuracy can help human inspectors focus on a small set of highly suspicious transactions.

Because of resource limits, the number of transactions that could be pulled out for detailed inspection is restricted by a maximum inspection rate, called inspection budget, for example 2% of all transactions. Under the constraint of the maximum inspection rate, the rules generated cannot be overly general.

3. RLSD Rule Learning Algorithm for Skewed Datasets

We assume that the learning task involves two classes: the minority class P and the majority class N. The goal of RLSD is to learn rules for the minority class. RLSD consists of three

[1] The work was done when the author was with MITRE Coporation

phases: feature discretization, rule generation, and rule evaluation and selection.

3.1 Feature Discretization

Most of the attributes in our application are numeric. Therefore it is critical to develop an effective and efficient feature discretization algorithm in RLSD. We modified the ChiMerge algorithm [6] for unbalanced data sets. For each numeric attribute, ChiMerge first sorts all training examples according to their values of the attribute. Each pair of neighboring values forms an initial interval. Then, ChiMerge repeatedly merges adjacent intervals with the most similar class distributions until no adjacent intervals have similar class distributions.

ChiMerge does not scale well for a large number of training examples, because it must sort all training examples for every numeric attribute and the number of initial intervals is often very large. We developed a new version of ChiMerge, ChiMerge-RLSD, for unbalanced data sets. In ChiMerge-RLSD, initial intervals are formed based on values of all minority class examples, so it sorts only minority class examples and the number of initial intervals is significantly smaller. Majority class examples are then assigned to one of the initial intervals according to their values. ChiMerge-RLSD scales very well and is thousands of times faster than ChiMerge in our application. ChiMerge-RLSD serves also as a feature selection algorithm, because it ignores the attributes in which class distributions are not significantly different.

3.2 Rule Generation

Rule generation in RLSD is a frequent pattern discovery task and it discovers all frequent patterns of the minority class examples. Each frequent pattern constitutes a minority class rule that covers some minority class examples. The algorithm of rule generation is given in Table 1.

This algorithm conducts a specific-to-general search. For each positive example, an initial rule is created as the example itself. This rule is the most specific rule covering the example. If this rule has already been generated, then it is discarded. Otherwise, the algorithm tries to merge this rule with each of the existing rules to generate more general rules covering this example. This process repeats for each of the positive examples. After all rules are generated, rules with recall less than the minimum recall are removed.

Two rules may be merged to form a new rule if they share some common conditions. The merged rule, which includes all shared conditions, is more general than both rules and covers the examples covered by either of them. It can be shown that the merged rule is the most specific rule covering these examples.

When the number of rules exceeds the maximum number of rules allowed, an existing rule is removed. In the current implementation, a randomly selected rule is removed. The random method works well because many similar rules are generated.

Table 1. Rule Generation Algorithm

```
P: the set of all positive examples
M: the maximum number of rules allowed
RuleGeneration(P, M)
  Rules = empty
  For each positive example p in P
    CurrentRule = p and mark CurrentRule as initial rule
    If CurrentRule is in Rules
      For each rule r in Rules
        If r is more general than CurrentRule
          increase #positive-examples covered by r by 1
    Else
      MergeRules(CurrentRule, Rules, M)
      AddRule(CurrentRule, Rules, M)
  For each rule r in Rules
    If recall(r) < minimum_recall   Remove r

MergeRules(cr, Rules, M)
  For each rule r in Rules
    If r is more general than cr
      increase # of positive examples covered by r by 1
    Else If r is more specific than cr
      If r is marked as initial rule
        increase # of positive examples covered by cr by 1
      Else if r and cr share common conditions
        generate a new rule nr with all common conditions
        AddRule(nr, Rules, M)

AddRule(r, Rules, M)
  If r is not in Rules
    Add r to Rules
  If # of rules in Rules is larger than M
    Randomly remove a non-initial rule from Rules
```

3.3 Rule Evaluation And Selection

Rules generated in the rule generation phase are matched against all negative examples in this phase. This algorithm consists of two steps: rule evaluation and rule selection. The rule evaluation step repeatedly matches each rule with each negative example. The rule is removed, should its precision fall below minimum precision. By the end of the process, a subset of rules whose precision are larger than or equal to the minimum precision survives.

The rule selection algorithm is simple and selects rules with the largest F-measure scores. F-measure of a rule r is defined below.

$$F-measure(r) = \frac{\beta^2 + 1}{\frac{\beta^2}{recall(r)} + \frac{1}{precision(r)}}$$

It selects the rule with the largest F-measure score and then removes the positive examples covered by the selected rule. F-measure scores of the remaining rules are recomputed with the remaining positive examples. It repeats this process until there are no remaining positive examples or no rule that satisfies both minimum precision and recall covers any remaining positive examples.

4. Experiments

We ran experiments on four data sets from the law enforcement application discussed in Section 2. Each of the four data sets consists of the data of three consecutive months for training and the fourth month data for testing. Each of the four training data set contains more than 6 millions transactions which involve more than 600,000 people and each test data set contains more than 2 million transactions with more than 200,000 people. To run experiments in a controlled environment, all training and test examples were generated from transactions that were inspected. This made the distributions of the data sets less skewed than deployment environments. Table 2 shows the summary of the four data sets.

RLSD has three important parameters: β, *minimum recall*, and *minimum precision*. With different values of β, rules are ranked differently. With a high minimum recall, specific rules are pruned to avoid generating rules that overfit the training data. With a high minimum precision, only accurate rules survive. We ran RLSD with different values of minimum recall and minimum precision. β was set to 2. For comparison, we conducted experiments using a popular decision tree learning system C5.0 with two different strategies for unbalanced data distributions, resizing training data sets and adjusting misclassification costs.

Table 2. Summary of the four experimental data sets

		#bad Exas	#good Exas
Data Set 1	Training	273	68988
	Test	59	24317
Data Set 2	Training	220	73421
	Test	84	26169
Data Set 3	Training	237	67350
	Test	74	25924
Data Set 4	Training	232	68333
	Test	266	38630

Table 3 shows the results with varying values of the minimum recall and precision. The purpose of this experiment is to understand the role of the minimum recall and minimum precision and how recall may be traded for precision. It provides us a base line performance. Results reported in Table 3 are the average on the four data sets. Results in the last four columns are the results on the test sets.

Table 3. RLSD's experimental results with varying minimum recall and precision values

Minimum Recall	Minimum Precision	# of Rules Selected	Recall on Training Set	All Selected Rules		Best Rule	
				Recall	Precision	Recall	Precision
0.01	0.005	48.75	1	0.65	0.0076	0.062	0.048
	0.01	49.75	0.97	0.59	0.0087	0.095	0.041
	0.02	50	0.85	0.38	0.0115	0.057	0.042
	0.03	56.25	0.72	0.26	0.0122	0.050	0.040
	0.04	51.25	0.65	0.19	0.0131	0.035	0.046
0.05	0.005	41.75	1	0.72	0.0072	0.077	0.029
	0.01	43	0.96	0.62	0.0093	0.092	0.037
	0.02	39.75	0.79	0.40	0.0102	0.074	0.042
	0.03	29.25	0.57	0.27	0.0126	0.062	0.041
	0.04	17	0.37	0.15	0.0180	0.048	0.043
0.1	0.005	25.5	1	0.80	0.0068	0.128	0.029
	0.01	28	0.94	0.72	0.0077	0.089	0.025
	0.02	17.75	0.57	0.35	0.0124	0.135	0.030
	0.03	5.75	0.27	0.12	0.0148	0.058	0.024
0.2	0.005	14.5	1	0.91	0.0062	0.236	0.018
	0.01	19.25	0.82	0.70	0.0076	0.238	0.018

When the minimum precision was low (0.005), RLSD was able to find some rule for every fraudulent transaction. With the increase of the minimum precision, the recall on training set decreases quickly. The recall on training set also decreases as the minimum recall increases. No rule was found when the minimum recall and precision were respectively set to 0.1 and 0.04 or 0.2 recall and 0.02. The number of rules selected decreases with the increase of the minimum recall. RLSD may generate rules that overfit the training data with low minimum recalls and precisions.

When the minimum recall increases, the number of selected rules decreases. This is because when the minimum recall is high, rules with low recalls and high precisions were pruned and RLSD had to select more general rules with a lower precision. The number of selected rules increases with the increase of the minimum precision until a point and then decreases with the increase of the minimum precision. This is due to the fact that when the minimum precision increases, the selected rules become more and more specific and therefore more rules have to be selected to cover fraudulent transactions. When the minimum precision gets too large, only a small number of rules satisfy it so that the number of selected rules starts to drop.

As expected, with the increase of the minimum precision, the recall on test data decreases and the precision increases. The precision increases with the minimum precision because rules with low precision were pruned when the minimum precision was high. When the minimum precision is small, the recall increases and precision decreases with the increase of the minimum recall. When the minimum precision becomes large, the recall starts decreasing and the precision starts increasing with the increase of the minimum recall. The highest precision (0.018) is achieved when the minimum recall is 0.05 and the minimum precision is 0.04 and is 4.5 times better than the random selection (0.0042). When both the minimum recall and precision were small, RLSD selected a large number of highly specific rules and many of these rules overfit the training data and did not cover any fraudulent transactions on test data.

The best rule is the rule with the highest F-measure score on test data. The recall of the best rule increases with the minimum recall with some exceptions and the precision decreases while the minimum recall increases. It seems that the minimum precision itself has little impact on the precision of the best rule. This is not surprising because when the minimum recall is fixed, the best rule is always the rule with the highest precision. The highest precision of the best rule is 0.048, which is about 12 times better than the random selection.

We conducted experiments with C5.0 on the same four data sets using two methods adjusting misclassification cost of minority class examples and downsizing majority class examples.

In our experiment, we set the misclassification cost of minority class examples to 100, 200, 300, and 400 respectively. When the cost was set to 100, no rule was generated for the minority class, so the recall is 0. When the cost was set to 400, all examples were classified as minority class examples, so the recall is 1. When the cost was set to 200 and 300, C5.0 generated a few highly specific rules for the minority class, each of which covers only one minority class example. A default rule was also generated for the minority class. When applying these rules to test data, only the default rule covered some minority class examples.

In downsizing majority class examples, we randomly selected 5%, 3%, 1%, 0.5%, 0.3%, and 0.1% majority class examples. The recall increases and the precision decreases when the percentage of randomly selected majority class examples decreases. With downsizing, C5.0 performed reasonably well. The curves shown in Figure 1 compare C5.0 results with RLSD results. RLSD-1 represents the results in Table 4 with the minimum recall = 0.01 while RLSD-2 is for the minimum recall = 0.05 in Table 4.

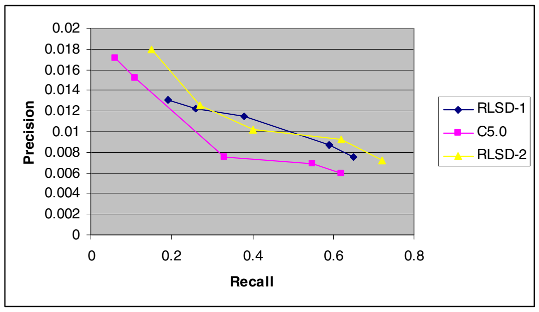

Figure 1. Comparison of RLSD and C5.0

5. Conclusion

We described a novel recognition-based rule learning algorithm, RLSD, for data sets with highly unbalanced class distributions and a law enforcement data mining application. The law enforcement application posed several challenges such as highly unbalanced class distributions, high uncertainty, and inspection budget for exiting data mining tools. RLSD was designed for these challenges and achieved reasonably well results in this application. Despite this application inspiration, RLSD is generic enough to be applied to other applications with similar challenges.

6. References

[1] Japkowicz N. and Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent Data Analysis*. 6(5).

[2] Ling, C.X., and Li, C. (1998). Data mining for direct marketing: Problems and solutions. *Proceedings of The Forth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 73-79, New York, NY, AAAI Press.

[3] Kubat, M. & Matwin, S. (1997). Addressing the curse of imbalanced data sets: One-sided sampleing. *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 179-186. Morgan Kaufmann.

[4] Domingos, P. (1999). MetaCost: A general method for making classifiers cost-sensitive. *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* pp. 155-164, San Diego, CA.

[5] Kubat, M., Holte, R., and Matwin, S. (1998). Machine learning for the detection of oil spills in satellite radar images. *Machine Learning* 30, pp.195-215.

[6] Kerber, R. (1992). ChiMerge: Discretization of Numeric Attributes. *Proceedings of the Tenth National Conference on Artificial Interlligence,* pp. 123-128, San Jose, CA.