***International Journal of Engineering & Technology**, 4 (2) (2015) 408-413*
www.sciencepubco.com/index.php/IJET
©Science Publishing Corporation
doi: 10.14419/ijet.v4i2.4221
Research Paper

Design of modified ripper algorithm to predict customer churn

M. Rajeswari *, T. Devi

School of Computer Science and Engineering, Bharathiar University, Coimbatore, India
**Corresponding author E-mail: mraji2231@gmail.com*

Abstract

Technologies such as data warehousing, data mining, and campaign management software have made Customer Relationship Management (CRM) a new area where firms can gain a competitive advantage. It is becoming common knowledge in business that retaining existing customers is an important strategy to survive in industry. Once identified, these customers can be targeted with proactive retention campaigns in a bid to retain them. These proactive marketing campaigns usually involve the offering of incentives to attract the customer into carrying on their service with the supplier. These incentives can be costly, so offering them to customers who have no intention to defect results in lost revenue. Also many predictive techniques do not provide significant time to make customer contact. This time restriction does not allow sufficient time for capturing those customers who are intending to leave. This research aims to develop methodologies for predicting customer churn in advance, while keeping misclassification rates to a minimum.

***Keywords**: Churn; Class Imbalance; Customer Relationship Management; Data Mining.*

1. Introduction

This chapter discusses on various algorithms towards customer churn prediction. The Banks usually make a distinction between voluntary churn and involuntary churn. Voluntary churn occurs due to a decision by the customer to switch to another bank, involuntary churn occurs due to circumstances such as a customer's relocation to a long-term care facility, death, or the relocation to a distant location. In most applications, involuntary reasons for churn are excluded from the analytical models. Analysts tend to concentrate on voluntary churn, because it typically occurs through bank-customer relationship which is controlled by bank, such as transaction interactions, services offered etc. The existing algorithms such as Decision Tree, Weighted Random Forest and Gradient Boosting Algorithm are depicted in this chapter. The proposed mechanism for customer churn prediction makes use of Genetic Algorithm, k-Nearest Neighbor and Ripper. A new algorithm has been proposed namely Modified Ripper Algorithm. The next section depicts the framework of the research design.

2. Framework

The proposed research work aims in design and development of customer churn prediction in banking sector. The framework is developed to extract and classify the churners among the customers through class imbalance. The proposed work uses mechanisms namely Decision Tree (DT), Gradient Boosting (GB), Weighted Random Forest (WRF), Genetic Algorithm (GA), k-Nearest Neighbor (k-NN) and Ripper Algorithm (RA). It also demonstrates three sampling methods namely Random over Sampling, Random under Sampling and Advanced Random under Sampling with experimental results and discussions. A new algorithm has been proposed as Modified Ripper Algorithm (MRA). Evaluation techniques such as Area under Curve (AUC), Error and Accuracy are used. By using these entire techniques class imbalance is predicted. All these concepts are clearly defined in the following Fig.1.

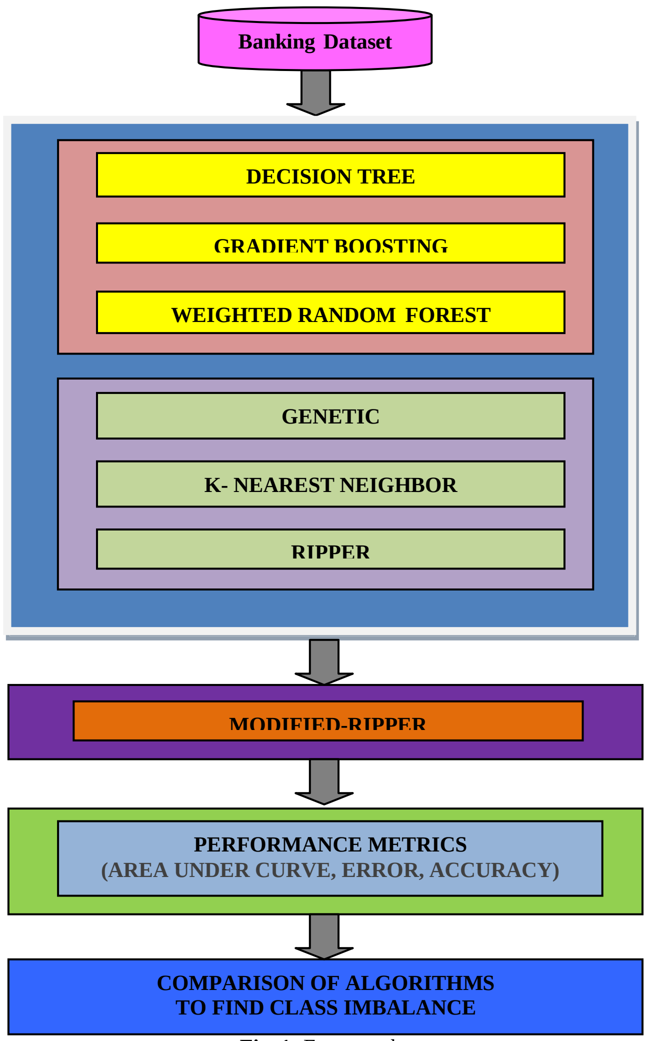

Fig. 1: Framework.

3. Evaluation metrics

The six categories of problems are associated while mining imbalanced classes. These classes are mapped with their relevant problems. Depending upon the problem the necessary results are chosen using the algorithms. Such cases are used ti handle relevant real time problems in various places which are shown below (Table.1).

4. Classification algorithms

4.1. Decision tree

A decision tree is a classifier expressed as a recursive partition of the instance space. The decision tree consists of nodes that form a rooted tree, meaning it is a directed tree with a node called “root” that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called an internal or test node. All other nodes are called leaves (also known as terminal or decision nodes). In a decision tree, each internal node splits the instance space into two or more sub-spaces according to a certain discrete function of the input attributes values. In the simplest and most frequent case, each test considers a single attribute, such that the instance space is partitioned according to the attribute’s value. In the case of numeric attributes, the condition refers to a range. Each leaf is assigned to one class representing the most appropriate target value. Alternatively, the leaf may hold a probability vector indicating the probability of the target attribute having a certain value. Instances are classified by navigating them from the root of the tree down to a leaf, according to the outcome of the tests along the path. Fig 2 describes a decision tree that reasons whether or not a potential customer will respond to a direct mailing. Given this classifier, the analyst can predict the response of a potential customer (by sorting it down the tree), and understand the behavioral characteristics of the entire potential customers population regarding direct mailing [8].

Table 1: Data Mining Problems Mapped to its Solutions to Address these Problems [8]

Data Mining ProblemData Mining	Methods Mapped to Address the problem Method Mapped to Address the problem
Improper Evaluation Metrics	More appropriate evaluation Metrics
	Cost-sensitive learning
Absolute Rarity	Learn only the rare class
	Sampling (over- and under)
	Cost-sensitive learning
Relative Rarity	More appropriate evaluation metrics
	More appropriate inductive bias
	Boosting
	Non-greedy search techniques
Data fragmentation	Learn only the rare class
	Sampling (over- and under)
	More appropriate inductive bias
Inappropriate bias	Appropriate evaluation metrics
	Cost-sensitive learning
Noise	Advanced sampling
	More appropriate inductive bias

4.2. Weighted random forest

Weighted Random Forests are an ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees. The algorithm for inducing a random forest was developed and "Random Forests" is their trademark [1]. The term came from random decision forests that were first proposed [3]. The method combines Breiman's "bagging" idea and the random selection of features, introduced independently by them in order to construct a collection of decision trees with controlled variance. The selection of a random subset of features is an example of the random subspace method, which, in Ho's formulation, is a way to implement classification proposed [5]. The early development of random forests introduced the idea of searching over a random subset of the available decisions when splitting a node, in the context of growing a single tree as explained in fig 4. The idea of random subspace selection from influential in the design of random forests [3]. In this method a forest of trees is grown, and variation among the trees is introduced by projecting the training data into a randomly chosen subspace before fitting each tree. Finally, the idea of randomized node optimization is explained where the decision at each node is selected by a randomized procedure, rather than a deterministic optimization.

4.3. Gradient boosting

Gradient Boosting is a machine learning technique for regression problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically Decision Trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. The Gradient Boosting method can also be used for classification problems by reducing them to regression with a suitable loss function. The method was invented and published in a series of two papers, the first of which introduced the method, and the second one described an important tweak to the algorithm, which improves its accuracy and performance. Gradient Boosted Trees (GBT) is a generalized boosting algorithm introduced by Jerome Friedman. In contrast to the AdaBoost.M1 algorithm, GBT can deal with both multiclass classification and regression problems. Moreover, it can use any differential loss function, some popular ones are implemented. Decision trees usage as base learners allows processing ordered and categorical variables. Gradient Boosted Trees model represents an ensemble of single regression trees built in a greedy fashion. Training procedure is an iterative process similar to the numerical optimization via the gradient descent method. Summary loss on the training set depends only on the current model predictions for the training samples. At every training step, a single regression tree is built to predict antigradient vector components. Step length is computed corresponding to the loss function and separately for every region determined by the tree leaf. It can be eliminated by changing values of the leaves directly [4].

4.4. Genetic algorithm

A Genetic Algorithm is a probabilistic search technique that computationally simulates the process of biological evolution. It mimics evolution in nature by repeatedly altering a population of candidate solutions until an optimal solution is found. The GA evolutionary cycle starts with a randomly selected initial population. The changes to the population occur through the processes of selection based on fitness, and alteration using crossover and mutation. The application of selection and alteration leads to a population with a higher proportion of better solutions [6]. The evolutionary cycle continues until an acceptable solution is found in the current generation of population, or some control parameter such as the number of generations is exceeded. The smallest unit of a genetic algorithm is called a gene, which represents a unit of information in the problem domain. A series of genes, known as a chromosome, represents one possible solution to the problem. Each gene in the chromosome represents one component of the solution pattern. The most common form of representing a solution as a chromosome is a string of binary digits. Each bit in this string is a gene. The process of converting the solution from its original form into the bit string is known as coding. The specific coding scheme used is application dependent. The solution bit strings are decoded to enable their evaluation using a fitness measure [7].

4.5. K- nearest neighbor algorithm

This method for classifying data based on closest training examples in the feature space. k-NN is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification. It can also be used for regression. The k-NN is amongst the simplest of all machine learning algorithms. An object is classified by a majority vote of its neighbors, with the object being assigned the class most common amongst its k nearest neighbors. k is a positive integer, typically small. If k = 1, then the object is simply assigned the class of its nearest neighbor. In binary (two class) classification problems, it is helpful to choose k to be an odd number as this avoids difficulties with tied votes [9]. The same method can be used for regression, by simply assigning the property value for the object to be the average of the values of its k-Nearest Neighbors. It can be useful to weight the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. The neighbors are taken from a set of objects for which the correct classification is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. In order to identify neighbors, the objects are represented by position vectors in a multidimensional feature space. It is usual to use the Euclidean distance, though other distance measures, such as the Manhattan distance could in principle be used instead. The k-Nearest Neighbor algorithm is sensitive to the local structure of the data. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If k = 3 it is classified to the second class because there are 2 triangles and only 1 square inside the inner circle. If k = 5 it is classified to first class (3 squares vs. 2 triangles inside the outer circle).The training examples are vectors in a multidimensional feature space. The space is partitioned into regions by locations and labels of the training samples. A point in the space is assigned to the class c if it is the most frequent class label among the k nearest training samples. Usually Euclidean distance is used. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.In the actual classification phase, the test sample (whose class is not known) is represented as a vector in the feature space [2].

4.6. Ripper algorithm

The Repeated Incremental Pruning to Produce Error The Repeated Incremental Pruning to Produce Error Reduction (Ripper) is a classification algorithm designed to generate rules set directly from the training dataset. The name is drawn from the fact that the rules are learned incrementally. A new rule associated with a class value will cover various attributes of that class .The algorithm was designed to be fast and effective when dealing with large and noisy datasets compared to decision trees. During the growing phase of the algorithm, a greedy approach of learning is applied, i.e. each rule is learned one at a time. In datasets with very large dimensions, this causes over-fitting of the data. This in turn increases the classification error rate significantly if the algorithm is tested with data with missing values. The Ripper model is not as popular as the decision trees in the insurance domain, but it has been applied in financial risk analysis. It has been used in financial institutes to help find the best policy for credit products, increase revenue as well as decreasing losses.Building Stage repeats 2 and 3 until the Description Length (DL) of the ruleset and examples is 64 bits greater than the smallest DL met so far, or there are no positive examples, or the error rate is greater equal than 50 percent. Grow Phase grows one rule by greedily adding antecedents (or conditions) to the rule until the rule is perfect (i.e. 100 percent accurate). The procedure checks every possible value of each attribute and selects the condition with high information gain. Prune Phase incrementally prune each rule and allow the pruning of any final sequences of the antecedents. Optimization Stage after generating the initial ruleset, generate and prune two variants of each rule from randomized data using procedure 2 and 3. But one variant is generated from an empty rule while the other one is generated by greedily adding antecedents to the original rule. Moreover, the pruning metric used is. Then the smallest possible DL for each variant and the original rule is computed. The variant with the minimal DL is selected as the final representative of in the ruleset. After all the rules in have been examined and if there are still residual positives, more rules are generated based on the residual positives using Building Stage again. Delete the rules from the ruleset that would increase the DL of the entire ruleset and add resultant ruleset [10].

4.7. Modified ripper algorithm

RIPPER was introduced as a successor of the Incremental Reduced Error Pruning (IREP) algorithm for rule induction. Even though the key principles remain the same, MRIPPER improves IREP in many details and is also able to cope with multiclass problems. A single MRIPPER rule consists of an antecedent part and a consequent part. The antecedent part is a conjunction of predicates (selectors) and the consequent part is a class assignment. MRIPPER learns such rules in a greedy manner, following a separate-and-conquer strategy. Prior to the learning process, the training data are sorted by class labels in ascending order according to the corresponding class frequencies.Rules are then learned for the first m − 1 classes, starting with the smallest one. Once a rule has been created, the instances covered by that rule is removed from the training data, and this is repeated until no instances from the target classes are left. The algorithm then proceeds with the next class. Finally MRIPPER finds no more rules to learn, a default rule (with empty antecedent) is added for the last (and hence, most frequent) class. Rules for single classes are learned until either all positive instances are covered or the last rule has been added was "too complicated." The latter property is implemented in terms of the total description length: the stopping condition is fulfilled if the description length of r is at least d bits longer than the shortest description length encountered so far are shown in Fig.12.

5. Sampling methods used

Oversampling and undersampling in data analysis are techniques used to adjust the class distribution of a data set (i.e. the ratio between the different classes/categories represented). Oversampling and undersampling are opposite and roughly equivalent techniques. They both involve using a bias to select more samples from one class than from another. The usual reason for oversampling is to correct for a bias in the original dataset. One scenario where it is useful is when training a classifier using labeled training data from a biased source, since labeled training data is valuable but often comes from un-representative sources. A number of resampling methods have been proposed and studied in the past [9]. Resampling methods can be divided into two categories: oversampling methods and undersampling methods. Oversampling methods balance training class priors by increasing the number of minority class data points, while undersampling methods balance training class priors by decreasing the number of majority class data points. Some widely used approaches are random oversampling, random undersampling, and cost-proportionate rejection sampling. Random oversampling increases the number of minority class data points in the training set by randomly replicating existing minority class members. While simplistic, random oversampling has performed well in empirical studies even when compared to other, more complicated oversampling methods. Unfortunately, since random oversampling only replicates existing data points, it has been argued that random oversampling does not add any actual data to the training set. Instead of replicating existing data points, "synthetic" minority class members are added to the training set by creating new data points. Empirically, SMOTE has shown to perform well against random oversampling [8].

6. Conclusion

Customer churn is one among the major research topics in customer relationship management. Several approaches were proposed for customer churn prediction. This chapter discussed the various techniques for predicting customer churn, comparison with other techniques that deals with the concerned research problem. The next chapter deals with the development of prototype.

Acknowledgements

The authors extended their sincere thanks to Department of Computer Applications for giving permission to do the research and all others directly or indirectly helped in pursuing this research.

```
procedure BUILDSET(P,N)
P = positive examples
N = negative examples
RuleSet = {}
DL = Deseriptiontength(RuleSet, P, N)
while P   {}
// Grow and prune a new rule
split (P, N) into (Grow Pas, Grow Neg) and (PrunePos, Prune Neg)
Rule := GrowRule(Grow Pos, Grow Neg)
Rule := PruneRule(Rule, Prune Pos, Prune Neg)
add Rule to RuleSet
if DescriptionLength(RuteSet, P, N) > DL + 69 then
// Prune the whole rule set and exit
for each rule R in RuleSet (considered in reverse order)
if DescriptionLength(RuleSet {R}, P, N) < DL then
delete R from RuleSet
DL := DescriptionLength(RuleSet, P, N)
end if
end for
return (RuleSet)
end if
DL := DescriptionLength(RuleSet, P, N)
delete from P and N all examples covered by Rule
end while
end BUILDRULESET
procedure OPTIM IZERULESET(RuleSet, P, N)
for each rule R in RuleSet
delete R from RuleSet
U Pos := examples in P not covered by RuleSet
U Neg := examples in N not covered by RuleSet
split (U Pos,U Neg) into (Grow Pos, Grow Neg) and (Prune Pos,PruneNeg)
Rep Rule := GrowRule(GrowPos,GrowNeg)
Rep Rule := PruneRule(Rep Rule, Prune Pos, Prune Neg)
Rev Rule := GrowRule(GrowPos,GrowNeg,R)
Rev Rule := PruneRule(Rev Rule, PrunePos,PruneNeg)
choose better of Rep Rule and Rev Rule and add to RuleSet
end for
end OPTIMIZERCIESET
procedure RIP PER( P, N, k)
RuleSet := BUILDRULESET(P, N)
repeat k times RuleSet :=
OPTIMIZERULESET(RuleS et, P, N)
return (RuleSet)
end RIPPER
```

Fig. 12: Modified Ripper Algorithm

References

[1] Breiman L., Friedman J., Olshen R., and Stone C. Classification and Regression Trees. Wadsworth Int. Group, 1984.

[2] Hancock T. R., Jiang T., Li M., Tromp J., Lower Bounds on Learning Decision Lists and Trees. Information and Computation 126(2): 114-122, 1996. http://dx.doi.org/10.1006/inco.1996.0040.

[3] Ho, Tin Kam (1995). Random Decision Forest. Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282.

[4] Hyafil L. and Rivest R.L., Constructing optimal binary decision trees is NPcomplete. Information Processing Letters, 5(1):15-17, 1976. http://dx.doi.org/10.1016/0020-0190(76)90095-8.

[5] Kleinberg, Eugene (1996). An Overtraining-Resistant Stochastic Modeling Method for Pattern Recognition. Annals of Statistics 24 (6): 2319–2349. http://dx.doi.org/10.1214/aos/1032181157.

[6] Naumov G.E., NP-completeness of problems of construction of optimal decision trees. Soviet Physics: Doklady, 36(4):270-271, 1991.

[7] Quinlan, J.R., Simplifying decision trees, International Journal of ManMachine Studies, 27, 221-234, 1987. http://dx.doi.org/10.1016/S0020-7373(87)80053-6.

[8] Weiss.G.M, Provost.F, The effect of class distribution on classifier learning: An empirical study, Dept. Comput. Sci., Rutgers Univ., Newark, NJ, Tech. Rep. TR-43, 2001.

[9] Weiss.G.M, Provost.F, Learning when training data are costly: The effect of class distribution on tree induction, J. Artif. Intell. Res., vol. 19, pp. 315–354, 2003.

[10] Zantema, H., and Bodlaender H. L., Finding Small Equivalent Decision Trees is Hard, International Journal of Foundations of Computer Science, 11(2):343-354, 2000. http://dx.doi.org/10.1142/S0129054100000193.