Brute-Force Mining of High-Confidence Classification Rules

Roberto J. Bayardo Jr.
The University of Texas at Austin
Department of Computer Sciences and Microelectronics and Computer Corporation
Austin, TX 78712 USA
bayardo@cs.utexas.edu
http://www.cs.utexas.edu/users/bayardo

Abstract

This paper investigates a brute-force technique for mining classification rules from large data sets. We employ an association rule miner enhanced with new pruning strategies to control combinatorial explosion in the number of candidates counted with each database pass. The approach effectively and efficiently extracts high confidence classification rules that apply to most if not all of the data in several classification benchmarks.

Introduction

Several data mining tasks require dividing up the entities of a database into various classes. Junk-mailers are well-known users of classification technology, using it to avoid sending out flyers to persons unlikely to be interested in the product being promoted. The task requires a *classifier* that is usually automatically generated from a "training database" of pre-classified entities. Several approaches have appeared in the AI, statistics, and data-mining literature, and some methods made to scale to large data sets [Shafer et al. 96]. Because of the exponential complexity of constructing an optimal classifier through various means, existing techniques employ heuristics for controlling resource consumption. Naive-Bayes classifiers make independence assumptions to avoid computing an exponential number of probabilities from the underlying data. The tree-induction class of classifiers effectively grow in a greedy manner classification rules that recursively partition the database, thereby limiting the space of rules that are considered.

This paper investigates brute-force identification of classification rules by only resorting to heuristics when necessary to guarantee that the rules will be produced in reasonable time. We demonstrate that an association rule miner enhanced with additional pruning strategies is remarkably effective at identifying rules which classify most data in several training database benchmarks with high confidence. On several data sets, this is accomplished without any heuristic pruning whatsoever.

There are several uses for the rules produced by a brute-force technique. They are often immediately useful to a knowledge discovery end-user for understanding the relationships and dependencies between instance features and their class. Decision trees induced from data are also easy for an end user to understand, but because each "rule" implied by a decision tree must be composed entirely of a limited number of conditions, these rules will not usually be as concise as those produced by brute-force, and may also contain irrelevant conditions. This suggests that rules identified by brute-force could be exploited by tree-inducers in order to select better split points. Existing systems typically use local metrics sometimes prone to error (e.g. see the "Corral" data set in the Irvine repository). Fukuda et al. [96] have demonstrated a related result by showing that associations discovered between two continuous valued attributes can be used during tree induction to minimize tree height (thereby resulting in more concise classification criteria).

Background

The problem we are interested in is to produce classification rules that apply to the given training database. More formally, a *training database* D is a collection of *instances* $\langle v_1, v_2, \ldots, v_m \rangle$ where the v_i range over the domain of *feature* A_i, and the value of feature A_1 represents the *class* of the instance. We assume the feature domains are nominal (discrete). As a pre-processing step, any continuously valued attributes can be discretized, usually without hurting classification accuracy [Kohavi & Sahami 96]. A *classification rule* states the confidence with which particular features determine some class according to the instances within the training database. For instance,

$A_i v_1, A_j v_2 \rightarrow A_1 v_c$, 95%

is how we denote a classification rule stating that an instance with value v_1 for feature A_i and value v_2 for feature A_j has class v_c in 95% of all cases (or with 95% *confidence*). Since the right hand side of a rule must contain only a class feature for it to be a classification rule, we will use the shorthand format $A_i v_1, A_j v_2 \rightarrow v_c$.

The *support* of a rule or a set of feature values is the number of training database instances (or percent of training database instances) that contain the values mentioned. Because we wish classification rules to apply to data outside the training database, we are only interested in rules which have a reasonably high level of support. We are also only interested in rules which have good classification power (high confidence). Following Agrawal et al. [96], we assume there is a minimum-specified level of support (*minsup*) and confidence (*minconf*) which a rule must meet for it to be produced. A rule or set of feature values with minimum support is said to be *frequent*.

Association rule miners have already been developed to mine arbitrary rules from transactional (e.g. shopping transaction) data [Agrawal et al 96]. Thus, the problem of mining classification rules in a brute-force manner is a special case of this particular problem, though applied to more-structured data. Unfortunately, as we later demonstrate, association rule miners when applied without modification to classification data lead to combinatorial explosions which prevents them from being useful in all but the simplest of data sets. We enhance an association rule miner with several additional pruning strategies to make it more useful for mining classification rules from classification data. First, we review the ideas behind efficient association rule mining of transactional data.

With transactional data, each instance in the database does not have a fixed number of features. Each instance is instead a collection of one or more *items*. Let I denote the set of all items mentioned in the database. An *itemset* is any subset of I. Association rule miners such as Apriori [Agrawal et al. 96] operate by making several passes over the database, with the i th pass used for computing supports of itemsets of size i. Rather than counting the support of every potential itemset of size i during database pass i (there can be as many as $\binom{|I|}{i}$), Apriori utilizes a pruning strategy we call *subset-support-based pruning*. Before each pass i, it uses the set of frequent itemsets found from the previous pass $i-1$ (denoted as F_{i-1}) to compute a set of candidate itemsets C_i for which support will be counted. The candidate set C_i is computed as the set consisting of any i-itemset such that all of its $i-1$ item subsets are in F_{i-1}. For the first pass, $C_1 = I$. Though this pruning strategy does not change the worst case complexity of each pass, in practice it is effective because the irregularity of transaction data ensures that only few itemsets, once i becomes sufficiently large, are frequent. Efficient data-structures for implementing these techniques are detailed in [Agrawal & Srikant 94].

There are other strategies for improving the performance of association rule miners such as applying hash-based reducers for minimizing candidates to count [Park et al. 95] and restricting the set of tuples examined to those that are (potentially) relevant to the current pass (e.g. AprioriTID [Agrawal et al. 96], and [Park et al. 95]). These techniques are complementary to the pruning strategies we propose next.

Additional Pruning Strategies

Given a classification training database, we can treat the potential feature values $A_i v$ as the set of items I and apply an association rule miner such as Apriori to mine association rules. Given unlimited resources, it would find all rules (including classification rules) that meet the minimum support and confidence requirements. The problem with this approach is that every instance in a classification database has exactly m features, so assuming the domain of each feature is reasonably restricted, even large itemsets are likely to have minimum support. We often found the subset-support-based pruning strategy ineffective on classification data until after several passes of the algorithm, leading to uncontrollable growth in the number of candidates. Additional strategies for controlling this explosion are described here. The first techniques are non-heuristic in the sense that they do not reduce the potential rule space considered. The last techniques involve heuristics that can either increase the granularity of the rule space searched, or reduce it by eliminating some items from consideration.

Pruning Strategy 1 - Value Exclusion

This strategy exploits the fact that in classification data, items from the same feature (e.g. $A_i v_1$ and $A_i v_2$) can never be contained by the same instance. To implement it, the candidate generator should avoid producing candidates with more than one value for the same attribute. After the second database pass, this pruning strategy is effectively accomplished by the subset-support strategy already present in Apriori since any 2-itemset with values from the same feature will have support zero. Nevertheless, this strategy speeds up the second pass, and it also speeds candidate generation on subsequent passes since subsets for candidates pruned by this technique do not have to be explicitly checked by subset-support-based pruning.

Pruning Strategy 2 - (Near) Equivalence Exploitation

Occasionally an itemset will have support that is (nearly) equivalent to the support of one of its subsets. For example, suppose we have an itemset $S = \{A_i v_1, A_j v_2\}$ whose support is equal to the support of $\{A_i v_1\}$. Given this fact, we know that any rule containing S as a subset is equivalent to the rule formed by removing $A_j v_2$ since it will have equivalent support, equivalent confidence, and applies to an equivalent set of database instances. To prevent these equivalent rules from being generated or incurring any overhead, the exact-equivalence strategy removes from the set of frequent itemsets any set S having a subset with equivalent support before forming the next set of candidates.

A more common case arises when the support of an itemset is very near the support of one of its subsets. For example, suppose the support of $\{X, Y\}$ is equal to the support of $\{X\}$ minus a small value t. Now consider any other feature Z and a class feature C. We can estimate the confidence of rule $X, Y, Z \rightarrow C$ using the following inequality:

$$\frac{\sup(XZC) - t}{\sup(XZ)} \le \text{confidence} \le \frac{\sup(XZC)}{\sup(XZ) - t}$$

If this confidence interval is entirely below *minconf*, then there is no need to compute the support of itemset $\{X, Y, Z, C\}$ during the fourth pass.

Unlike exact-equivalence, the near-equivalence rule must be carefully applied because in subsequent passes the support of a superset of an itemset pruned by the technique may be required. Extending the previous example, even if we know $\{X, Y, Z, C\}$ does not imply a high confidence rule, itemset $\{W, X, Y, Z, C\}$ may imply a high confidence rule (estimated using the same inequality technique) and be potentially frequent. For this case even though $\{X, Y, Z, C\}$ is not contained in the set of computed frequent itemsets, we must ensure that $\{W, X, Y, Z, C\}$ gets generated as a candidate in order to preserve completeness. Unfortunately, not having the support of $\{X, Y, Z, C\}$ hinders the ability of subset-support based pruning to determine whether $\{W, X, Y, Z, C\}$ is potentially frequent, so there is a trade-off between strength of this "near equivalence" strategy and the strength of subset-support based pruning.

Pruning Strategy 3 - Rule Structure Exploitation

Recall that we are only interested in classification rules. Computing the support of a classification rule requires we know the support of both the entire rule, and the support of the left-hand side of the rule. If we can determine that for every class item v_c the rule $A_iv_1, A_jv_2 \rightarrow v_c$ is not of interest, then there is no need to count the support of $\{A_iv_1, A_jv_2\}$. This pruning strategy exploits this fact. An itemset S of C_i is said to be *useless* if, for some $i-1$ item subset S' of S, there exists no class item v_c such that $S' \cup \{A_1v_c\}$ is in C_i. To implement this strategy, useless candidates are pruned before the database pass commences.

There are other potential strategies for exploiting rule structure, but several of them may conflict with the other strategies we propose. For instance, the miner could avoid counting the support of an itemset S that does not contain a class item until an itemset consisting of S and some class item v is known to be frequent. This strategy unfortunately delays the point at which we can apply a strategy that requires the support of S (such as, potentially, strategies 2 and 4). The strategy we suggest above allows rule structure to be exploited to a lesser degree, though it provides the guarantee that after pass i we know all frequent itemsets of size i, and not just those containing class items. Inevitably some data sets will benefit from stronger rule-structure exploitation at the expense of other strategies affected by it, though this trade-off remains to be fully investigated.

Pruning Strategy 4 - Redundancy Exploitation

The previous pruning strategies do not result in any information loss in the sense that an association rule miner ignoring them will produce a set of rules that applies to the same set of tuples with the same confidence and support. Strategy 2 can result in fewer rules being returned, but only because rules that are equivalent to some other rule are removed. The next pruning strategy is different than those previous because it can result in some information loss, though the information lost can be regarded as irrelevant with respect to the given level of confidence desired.

The idea is to prevent continued effort at classifying instances already classified by existing rules with high confidence. To implement such a technique, after each pass i, we identify the frequent itemsets that imply rules with confidence exceeding *minconf.* During the next database pass, tuples can be marked if they are covered by this set while counting candidate supports. During counting, also determine whether a candidate is supported by at least one uncovered instance. At the end of the pass, if the candidate is not supported by at least one covered instance, it can be excluded from the set of frequent itemsets before forming the next set of candidates.

Marking tuples is unfortunately expensive or impractical in some situations. Luckily it is possible to approximate this pruning technique without any tuple marking. The idea is to exploit the fact that if a rule R meets minimum confidence, then any rule containing R will apply to only covered instances. Itemsets representing such rules can therefore be pruned from the candidate set before counting commences.

Pruning Strategy 5 - Granularity Modification

Though an explosion in candidates to be counted with a database pass is often substantially ameliorated by the above strategies, in several cases they are not enough and further information-loss must be accepted. In the worst case, given m features and l potential valuations of each feature, there are up to $l^i\binom{m}{i}$ candidates during pass i. This pruning strategy reduces the l^i term of this bound. The idea is to combine particular valuations of some feature into a single, larger granularity feature. For instance, if a feature A_i, $i \neq 1$ can be labelled with values v_1, v_2, v_3, the rule considers treating any pair of those values as a single *composite item.* The rule can be applied recursively in that a composite item A_iV can be further combined with another feature A_iv or even A_iV_2 to form another composite item. The rule should not be applied to form a composite item consisting of all potential valuations of the involved feature, since every instance will, by definition, contain such an item. Once a composite item is determined, the next set of candidates is formed from the set of frequent itemsets modified by replacing each occurrence of the items used to form the composite item with the composite item itself. Since this results in duplicate itemsets, the implementation must be made to remove them.

Various strategies can be used to determine which feature valuations to combine in order to produce a composite valuation. The strategy we use is to seek out pairs of rules from the last produced set of frequent itemsets F_i that differ by exactly one valuation of a particular feature. The pair of rules which comes nearest in confidence will be used to obtain the composite item. Given a feature whose values were obtained by discretizing a continuous attribute, another technique would be to combine adjacent portions of the discretized space.

Pruning Strategy 6 - Feature Selection

This strategy is a feature selection step applied when necessary instead of as a preprocessing phase. It contends with the potential explosion in the number of candidates due to the $\binom{m}{i}$ term in the worst-case bound on candidates. The idea is to dynamically select a feature A_i mentioned in the most recently obtained F_i for removal. Any itemset in F_i to contain the selected feature is removed from F_i before forming the set of candidates for the next pass. The feature selected for removal should be, by some measure, the least useful in predicting the class. For the evaluation below we compute the mutual information $I(A_i, A_1)$ for every feature appearing in F_i and select the feature with the lowest value for removal. A better strategy could consider the information provided by the largest rules derived by the miner at the point the strategy is applied.

Evaluation

We implemented the Apriori algorithm and enhanced it with the above pruning strategies in order to evaluate their effectiveness. Pruning strategies 1 through 4 were uncondi-

Database	instances	features	values	confidence	runtime	speedup	passes	loss	rules	coverage
Chess	3,196	37	2	90%	176	?	11	33	279	92.5
				100%	189	?	11	33	237	74.5
Connect-4	67,557	43	3	90%	1382	?	8	35	140	57.6
				100%	1437	?	8	35	2	2.9
Corral	128	7	2	90%	.04	25%	4	0	10	100
				100%	.05	0%	4	0	6	100
DNA	3,174	61	4	90%	239	?	7	196	394	92.0
				100%	238	?	7	197	269	65.0
Flare	1,066	11	3	90%	.64	915%	7	0	19	75.8
				100%	.87	647%	7	0	5	9.3
Led7	3,200	8	2	90%	1.7	23%	8	0	6	23.6
				100%	1.7	32%	8	0	0	0
Letter	20,000	17	16	90%	23.4	182	7	0	4	4.0
				100%	23.4	182	7	0	1	1.1
Tic-Tac-Toe	958	10	3	90%	2.8	0%	5	0	136	100
				100%	2.8	0%	5	0	86	98.3
Vote	435	17	3	90%	3.7	?	10	0	152	99.5
				100%	22.0	?	12	0	378	99.5

tionally applied with each pass. For strategy 4, we used the approximation based on rule sub-setting instead of the complete tuple-marking implementation. For strategy 2, we only implemented the exact-equivalence case. When the candidate set being generated after a pass exceeds a particular threshold, pruning strategies 5 and 6 were invoked. On our machine (64MB Sparc Ultra 1), the threshold was set to 30,000 candidates since larger sizes caused thrashing. For applying strategies 5 or 6, our implementation first scores all items using the feature selection metric of pruning strategy 6. If the chosen feature has more than two potential valuations, then pruning strategy 5 is used to create a composite item on that feature. If the feature has only 2 potential valuations, then pruning strategy 6 is invoked and that feature is removed.

We ran the algorithm on several data sets from the Irvine repository that contained nominally valued features. The table above displays the performance in CPU seconds of our enhanced Apriori on the various data sets for 90% and 100% confidence levels, using a minimum support of 1% or 20, whichever is the maximum. It also presents the speedup over un-enhanced Apriori. In some cases un-enhanced Apriori caused our machine to thrash (we did not cache candidates to disk), at which point it was halted. For these cases speedup is not reported. After speedup, the table reports the number of passes required over the data.

On all but the DNA, Chess, and Connect-4 data sets, the miner did not have to ever invoke the lossy pruning strategies 5 and 6. DNA proved the most difficult in this regard because it consists of 62 features other than the class, each with four potential valuations. The table displays as "loss" the number of composite valuations formed summed with the number of features removed. Note that performance on the Vote data set increases substantially with lower confidence. This is the result of pruning strategy 4 which is very effective on this data set because after only two passes a large percentage of the database is covered by high confidence rules. The table also describes the rules and the number of rules found by the rule miner for each run on the various data sets. Except for the noisy data set Led7 and Letter, the rule miner was effective at attaining high coverage at 90% confidence. The poor coverage on Letter is not due to information loss and can be improved by lowering confidence and/or support.

References

Agrawal, R., Mannila, H., Srikant, R. Toivonen, H. and Verkamo, A. I. 1996. Fast Discovery of Association Rules. In *Advances in Knowledge Discovery and Data Mining*, AAAI Press, 307-328.

Agrawal, R., and Srikant, R. 1994. Fast Algorithms for Mining Association Rules. IBM Research Report RJ9839, June 1993, IBM Almaden Research Center, San Jose, CA.

Fukuda, T., Morimoto, Y. and Morishita, S. 1996. Constructing Efficient Decision Trees by Using Optimized Numeric Association Rules. In Proc. of the 22nd Very Large Data Bases Conference, 146-155.

Kohavi, R., and Sahami M. 1996. Error-Based and Entropy-Based Discretization of Continuous Features. In *Proc. of the Second Int'l Conf. on Data Mining and Knowledge Discovery,*

Park, J. S., Chen, M.-S., Yu, P. S. 1996. An Effective Hash Based Algorithm for Mining Association Rules. In *Proc. of the 1995 SIGMOD Conference on the Management of Data*, 175-186.

Shafer, J. C., Agrawal, R., and Mehta, M, 1996. SPRINT: A Scalable Parallel Classifier for Data Mining, In *Proc. of the 22th Int'l Conf. on Very Large Databases*, 544-555.

Srikant, R. and Agrawal, R. 1996. Mining Quantitative Association Rules in Large Relational Tables. In *Proc. of the ACM-SIGMOD Conference on Management of Data*, 1-12.