Scalable Feature Mining for Sequential Data

Neal Lesh, Mitsubishi Electric Research Lab
Mohammed J. Zaki, Rensselaer Polytechnic Institute
Mitsunori Ogihara, University of Rochester

FEATUREMINE COMBINES SEQUENCE MINING AND CLASSIFICATION ALGORITHMS TO EFFICIENTLY SELECT FEATURES FROM LARGE DATA SETS.

MANY REAL-WORLD DATA SETS contain irrelevant or redundant attributes. This might be because the data was collected without data mining in mind or without a priori knowledge of the attribute dependences. Many data mining methods such as classification and clustering degrade prediction accuracy when trained on data sets containing redundant or irrelevant attributes or features. Selecting the right feature set not only can improve accuracy but also can reduce the running time of the predictive algorithms and lead to simpler, more understandable models. Good feature selection is thus a fundamental data-preprocessing step in data mining.

To provide good feature selection for sequential domains, we developed FeatureMine, a scalable feature-mining algorithm that combines two powerful data mining paradigms: sequence mining and classification algorithms. Tests on three practical domains demonstrate that FeatureMine can efficiently handle very large data sets with thousands of items and millions of records.

Working in sequential domains

Most feature-selection research has focused on nonsequential domains. Here the problem is to select an optimal feature subset of size m from the full d-dimensional feature space, where ideally $m \ll d$. The selected subset should maximize some optimization criterion such as classification accuracy, or it should faithfully capture the original data distribution. The subset search space has an exponentially large number of features.

Instead of traditional nonsequential data, we focus on sequential data, where a sequence of "events" represents each example. Each event might be described by a set of predicates; that is, we are dealing with categorical sequential domains. Examples of sequential data include text, DNA sequences, Web-use data, multiplayer games, and plan-execution traces. In sequential domains, features are ordered sets of partial event descriptions. For example, a sequential feature that describes a chess game is "Black moves a knight, and then white moves a bishop to square d6." This feature holds in some chess games but not in others, and thus might be used to classify chess games into, for example, ones played by experts versus ones played by novices.

Selecting the right features in sequential or temporal domains is even more challenging than in nonsequential data. The original feature set is itself undefined; potentially, an infinite number of sequences of arbitrary length exist over d categorical attributes or dimensions. Even if we restrict ourselves to some maximum sequence length k, we have potentially $O(d^k)$ subsequences over d dimensions. The complexity is d^d if we consider the maximum subsequence length to be d, as opposed to 2^d in the nonsequential case.

Feature selection in sequential domains aims to select the best subset of sequential features out of the d^k possible sequential features (that is, subsequences) that can be composed out of the d attributes for describing individual events. We chose data mining techniques because of the exponentially large set of possible features. Alternatively, you

1094-7167/00/$10.00 © 2000 IEEE

could say that we are constructing new features out of the primitives for describing events. These new features augment the original space's dimensionality by effectively pulling apart examples of the same class, making them more easily distinguishable by classification algorithms. Of course, constructing features out of primitives is equivalent to selecting features from the space of all combinations of those primitives.

The input to our system is a set of labeled training sequences, and the output is a function that maps from a new sequence to a label. In other words, we are interested in selecting (or constructing) features for sequence classification. To generate this function, our algorithm first uses sequence mining on a portion of the training data to discover frequent and distinctive sequences. It then uses these sequences as features to feed into a classification algorithm to generate a classifier from the remainder of the data.

Previous research has used the rules produced by data mining algorithms to construct classifiers primarily by ordering the rules into decision lists[1,2] or by merging them into more general rules that occur in the training data.[3] In this article, we convert the patterns discovered by the mining algorithm into a set of Boolean features to feed into standard classification algorithms. The classification algorithms, in turn, assign weights to the features, which lets evidence for different features be combined to classify a new example.

Poker and feature mining

A simple poker game will illustrate the basic problem. Suppose we observe three players with betting sequences and outcomes such as

Example: P_1 Bets \$3, P_2 Calls, P_3 Raises \$2, P_1 Raises \$1, P_2 Folds, P_3 Calls $\Rightarrow$ P_1 wins

Example: P_2 Passes, P_3 Bets \$1, P_1 Folds, P_2 Raises \$2, P_3 Raises \$2, P_2 Calls $\Rightarrow$ P_3 wins

We want to learn a function that predicts who is most likely to win given a betting sequence. This task resembles standard classification: we are given labeled training examples and must produce a function that classifies new, unlabeled examples. Many classifiers require, however, that examples be represented as vectors of feature–value pairs. This article concerns selecting features to represent the betting sequences.

First, consider an obvious feature set. Let N be the length of the longest betting sequence. We can represent betting sequences with $3N$ features by generating a distinct feature for the person who made the ith bet, the type of the ith bet, and the amount of the ith bet, $0 \le i \le N$. However, this feature set leads to poor classification, as we show later.

One problem with these features is that an individual feature can express only that a particular, complete bidding sequence took place, not that an interesting subsequence occurred, such as

Feature: P_1 Raises twice

Feature: P_2 Folds and then P_1 Raises \$2

WE CONVERT THE PATTERNS DISCOVERED BY THE MINING ALGORITHM INTO A SET OF BOOLEAN FEATURES TO FEED INTO STANDARD CLASSIFICATION ALGORITHMS.

The first feature would be important if P_1 tends to win when she raises twice. A classifier could construct a Boolean expression out of these features to capture the notion "P_1 Raises twice." However, the expression would have N^2 disjuncts, because we need a disjunct for "P_1 raises in the ith bet and in the jth" for all $i, j < N$ where $i \neq j$. Considering partial specifications is important because of the difficulty of knowing in advance whether "P_1 raises twice" or "P_1 raises by 2 and then raises by 3" will be the more useful feature.

An alternative is to use a much larger feature set. If there are three players, four bids, and five different amounts, then there are $4 \times 5 \times 6 = 120$ partial specifications of a bet, such as "Someone bets 3." We can chain partial specifications together with an "and then" relation, as in "P_1 raises and then someone bets 3." The number of such features of length K is 120^K. However, this feature set is too large; sets of 10,000 features are considered large for classification algorithms. Furthermore, as we mentioned previously, irrelevant or redundant features can reduce classification accuracy.[4]

We adopt a middle ground between these two extremes. FeatureMine searches through the second, huge feature set and selects a subset for classification.

Data mining for features

The formulation of our FeatureMine algorithm involved specifying a language for features that can express sequences of partial descriptions of events (with gaps), such as "P_1 raises and then in some later bid folds." We also had to determine the criteria for selecting a subset of the features from the entire set that can be expressed in our language.

A language for expressing sequences. We adopted terminology that closely resembles sequence-mining terminology.[5] Let $\mathcal{F}$ be a set of distinct features, each with some finite set of possible values. Let $\mathcal{I}$ contain a unique element for every possible feature–value pair. A *sequence* is an ordered list of subsets of $\mathcal{I}$. For example, if $\mathcal{I} = \{A, B, C, \ldots\}$, one example sequence is $AB \rightarrow A \rightarrow BC$. We denote a sequence α as $(\alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \rightarrow \alpha_n)$, where each sequence element α_i is a subset of $\mathcal{I}$. A sequence's *length* is n, and its *width* is the maximum size of any α_i for $1 \le i \le n$. We say that α is a *subsequence* of β, denoted as $\alpha < \beta$, if integers $i_1 < i_2 < \ldots < i_n$ exist such that $\alpha_j \subseteq B_{i_j}$ for all α_j. For example, $AB \rightarrow C$ is a subsequence of $AB \rightarrow A \rightarrow BC$. Let $\mathcal{H}$ be a set of class labels; $c \in \mathcal{H}$ is a label. An *example* is a pair $<\alpha, c>$. Each example has a unique identifier *eid*, and each α_i has a time stamp at which it occurred. An example $<\alpha, c>$ *contains* sequence β if $\beta < \alpha$.

Our input database $\mathcal{D}$ consists of a set of examples. This means that the data we look at has multiple sequences of sets of items. The *frequency* of sequence β in $\mathcal{D}$, denoted $fr(\beta, \mathcal{D})$, is the fraction of examples in $\mathcal{D}$ that contain β. The *confidence* of the rule $\beta \Rightarrow c$, denoted $conf(\beta, c, \mathcal{D})$, is the conditional probability that c is the label of an example in $\mathcal{D}$ given that it contains sequence β. That is,

$$conf(\beta, c, \mathcal{D}) = \frac{fr(\beta, \mathcal{D}_c)}{fr(\beta, \mathcal{D})}$$

where $\mathcal{D}_c$ is the subset of examples in $\mathcal{D}$ with class label c. A sequence is *frequent* if its frequency is more than a user-specified *min_freq* threshold. A rule is *strong* if its confidence is more than a user-specified *min_conf* threshold. Our goal is to mine for frequent and strong patterns.

(a)

EID	Event time	Items	Class
1	10	*A B*	c_1
	20	*B*	
	30	*A B*	
2	20	*A C*	c_1
	30	*A B C*	
	50	*B*	
3	10	*A*	c_1
	30	*B*	
	40	*A*	
4	30	*A B*	c_1
	40	*A*	
	50	*B*	
5	10	*A B*	c_2
	50	*A C*	
6	30	*A*	c_2
	40	*C*	
7	20	*C*	c_2

Frequent sequences
Class = c_1
min_freq (c_1) = 75%

Item	Percent
A	100
B	100
A→A	100
AB	75
A→B	100
B→A	75
B→B	75
AB→B	75

Class = c_2
min_freq (c_2) = 67%

Item	Percent
A	67
C	100
A→C	67

(b)

New Boolean features

EID	*A*	*A→A*	*B→A*	*B*	*AB*	*A→B*	*B→B*	*AB→B*	*C*	*A→C*	Class
Examples 1	1	1	1	1	1	1	1	1	0	0	c_1
2	1	1	0	1	1	1	1	1	1	0	c_1
3	1	0	1	1	0	1	0	0	0	0	c_1
4	1	1	1	1	1	1	1	1	0	0	c_1
5	1	1	1	1	1	0	0	0	1	1	c_2
6	1	0	0	0	0	0	0	0	1	1	c_2
7	0	0	0	0	0	0	0	0	1	0	c_2

Figure 1. Two databases: (a) the initial version with seven examples in two classes; (b) a new version with Boolean features.

Figure 1a shows a database of seven examples, four in class c_1 and three in class c_2. Of course, more than two classes are possible. We are looking for different *min_freq* sequences on each class. For example, although C is frequent for class c_2, it's not frequent for class c_1. The rule $C \Rightarrow c_2$ has a confidence of 3/4 (0.75), while the rule $C \Rightarrow c_1$ has a confidence of 1/4 (0.25).

A *sequence classifier* is a function from sequences to class labels. To evaluate a classifier, we can use standard metrics such as accuracy and coverage.

Here's how frequent sequences $\beta_1, \ldots, \beta_n$ can serve as features for classification. Recall that the input to many standard classifiers is an example represented as vector of feature–value pairs. We represent an example sequence α as a vector of feature–value pairs by treating each sequence β_i as a Boolean feature that is true if and only if $\beta_i \preccurlyeq \alpha$. For example, suppose the features are $f_1 = A \rightarrow D$, $f_2 = A \rightarrow BC$, and $f_3 = CD$. So, the sequence $AB \rightarrow BD \rightarrow BC$ would be <*f*1, *true*>, <*f*2, *true*>, <*f*3, *false*>, and the sequence $ABCD \rightarrow B \rightarrow DE$ would be <*f*1, *true*>, <*f*2, *false*>, <*f*3, *true*>. Features can "skip" steps: the feature $A \rightarrow BC$ holds in $AB \rightarrow BD \rightarrow BC$. Figure 1b shows the new data set created from the frequent sequences of our example database. Although we use all frequent sequences as features in this example, generally we use only a "good" subset of all frequent sequences as features, as we describe next.

Selection criteria for mining. We want to find sequences such that representing examples with them will yield a highly accurate sequence classifier. However, we do not want to search over the space of all subsets of features;[4] we want instead to evaluate each new sequential feature in isolation or by pairwise comparison to other candidate features.

Heuristics. Certainly, the criteria for selecting features might depend on the domain and the classifier being used. We believe, however, that these domain- and classifier-independent heuristics are useful for selecting sequences to serve as features:

- Features should be frequent.
- Features should be distinctive of at least one class.
- Feature sets should not contain redundant features.

The intuition behind the first heuristic is simply that rare features are, by definition, only rarely useful for classifying examples. In our problem formulation, this heuristic translates into a requirement that all features have some minimum frequency in the training set. Because we use a different *min_freq* for each class, patterns that are rare in the entire database can still be frequent for a specific class. We ignore only those patterns that are rare for any class.

The intuition behind the second heuristic is that features that are equally likely in all classes do not help determine the class to which an example belongs. Of course, a conjunction of multiple nondistinctive features can be distinctive. In this case, our algorithm prefers to use a distinctive conjunction as a feature rather than a nondistinctive conjunction. We encode this heuristic by requiring that each selected feature correlate significantly with at least one class in which it is frequent.

The basis for our third heuristic is that if two features are closely correlated, either of them is as useful for classification as both are together. As we show later, we can reduce the number of features and the time needed to mine for features by pruning redundant rules. Besides wanting to prune features that provide the same information, we want to prune a feature if another feature is available that provides strictly more information. Let $M(f, \mathcal{D})$ be the set of examples in $\mathcal{D}$ that contain feature f. Feature f_1 *subsumes* feature f_2 with respect to predicting

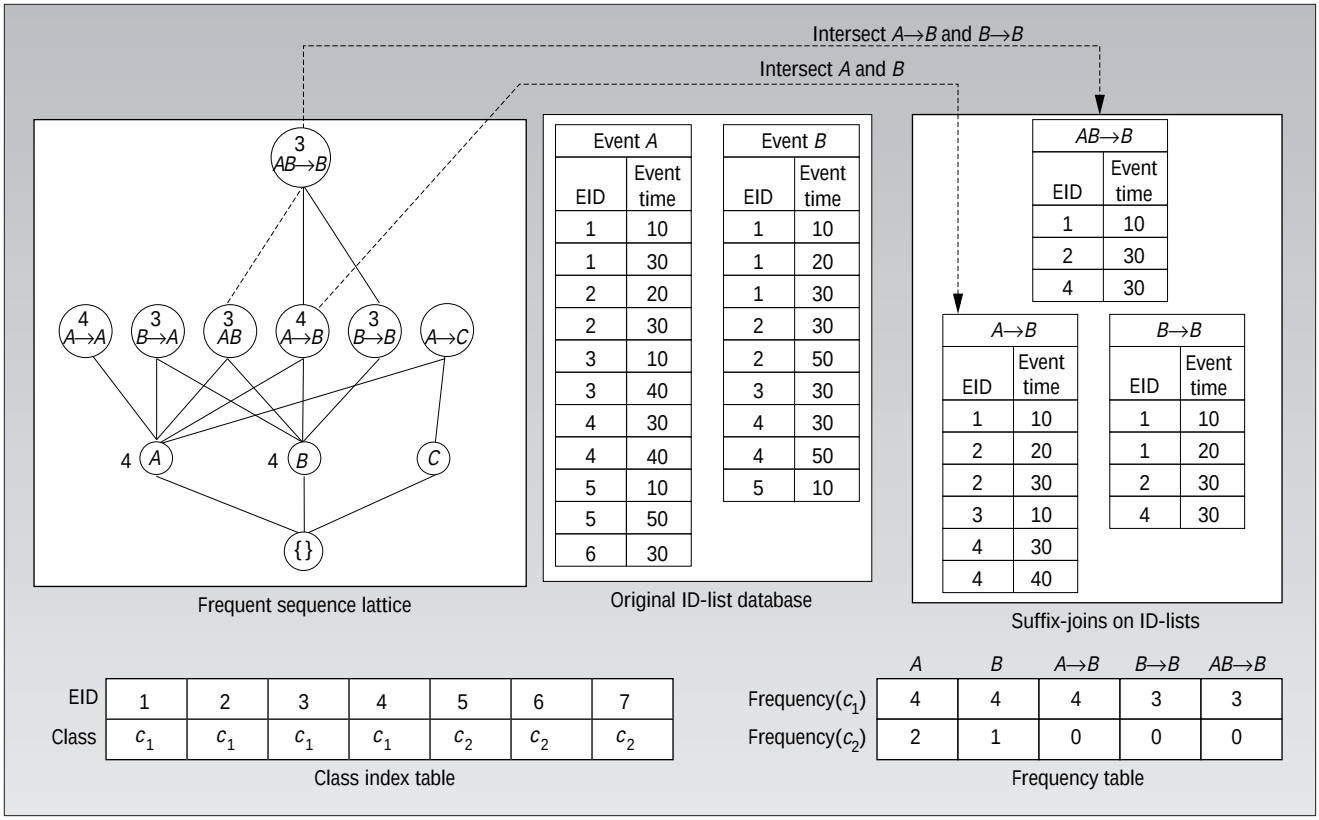

Figure 2. A frequent-sequence lattice and frequency computation for the database in Figure 1a.

class c in data set $\mathcal{D}$ if and only if $M(f_2, \mathcal{D}_c) \subseteq M(f_1, \mathcal{D}_c)$ and $M(f_1, \mathcal{D}_{\neg c}) \subseteq M(f_2, \mathcal{D}_{\neg c})$. Intuitively, if f_1 subsumes f_2 for class c, then f_1 is superior to f_2 for predicting c because f_1 covers every example of c in the training data that f_2 covers and f_1 covers only a subset of the non-c examples that f_2 covers. A feature f_2 can be a better predictor of class c than f_1 even if f_1 covers more examples of c if, for example, every example that f_2 covers is in c but only one-half the examples that f_1 covers are in c. In this case, neither feature subsumes the other.

Pruning rules. The third heuristic leads to two pruning rules. The first is that we do not extend (that is, specialize) any feature with 100% accuracy. Let f_1 be a feature contained by examples of only one class. Specializations of f_1 might pass the frequency and confidence tests in the definition of feature mining but will be subsumed by f_1. The following lemma, which follows from the definition of subsume, justifies this pruning rule:

Lemma 1: If $f_i \prec f_j$ and $conf(f_i, c, \mathcal{D}) = 1.0$, then f_i subsumes f_j with respect to class c.

Our next pruning rule concerns correlations between individual items. Recall that the examples in $\mathcal{D}$ are represented as a sequence of sets. We say that $A \rightsquigarrow B$ in examples $\mathcal{D}$ if B occurs in every set in every sequence in $\mathcal{D}$ in which A occurs. The following lemma states that if $A \rightsquigarrow B$, any feature containing a set with both A and B will be subsumed by one of its generalizations:

Lemma 2: Let $\alpha = \alpha_1 \rightarrow \alpha_2 \rightarrow ... \rightarrow \alpha_n$ where $A, B \in \alpha_i$ for some $1 \le i \le n$. If $A \rightsquigarrow B$, then α will be subsumed by $\alpha_1 \rightarrow ... \alpha_{i-1} \rightarrow (\alpha_i - B) \rightarrow \alpha_{i+1} \ldots \rightarrow \alpha_n$.

We precompute the set of all $A \rightsquigarrow B$ relations; during the search, FeatureMine immediately prunes any feature that contains a set with both A and B. In "Empirical evaluation," we discuss why $A \rightsquigarrow B$ relations arise and show they are crucial for our approach's success for some problems.

Defining feature mining. We can now define the feature-mining task. The inputs to the FeatureMine algorithm are a set of examples $\mathcal{D}$ and the parameters *min_freq*, max_w, and max_l. The output is a nonredundant set of the frequent and distinctive features of width max_w and length max_l. Here's the formal definition of feature mining:

Given examples $\mathcal{D}$ and parameters *min_freq*, max_w, and max_l, return feature set $\mathcal{F}$ such that for every feature f_i and every class $c_j \in \mathcal{H}$, if $length(f_i) \le max_l$ and $width(f_i) \le max_w$ and $fr(\beta, \mathcal{D}_{c_j}) \ge min\_freq(c_j)$ and $conf(\beta, c_j, \mathcal{D})$ is significantly greater than $|\mathcal{D}_{c_j}|/|\mathcal{D}|$, then $\mathcal{F}$ contains f_i or contains a feature that subsumes f_i with respect to class c_j in data set $\mathcal{D}$ (we use a chi-squared test to determine significance).

Efficient feature mining. FeatureMine is based on the recently proposed Spade algorithm for fast discovery of sequential patterns.[5] Spade is a scalable, disk-based algorithm that can handle millions of example sequences and thousands of items. To construct FeatureMine, we adapted Spade to search databases of labeled examples. FeatureMine simultaneously mines the patterns predictive of all the classes in the database. Unlike previous approaches that first mine millions of patterns and then apply pruning as a postprocessing step, FeatureMine integrates pruning techniques in the mining algorithm. This lets it search a large space where previous methods would fail.

FeatureMine uses the observation that the subsequence relation $\preccurlyeq$ defines a partial order on sequences. If $\alpha \prec \beta$, we say that α is more general than β or that β is more specific than α. The relation $\preccurlyeq$ is a *monotone specialization relation* with respect to the frequency $fr(\alpha, D)$; that is, if β is a frequent sequence, all subsequences $\alpha \preccurlyeq \beta$ are frequent. The algorithm systematically searches

```
FeatureMine (𝒟, min_freq(c_i)):
    𝒫 = { parent partitions, 𝒫_i}
    for each parent partition 𝒫_i do Enumerate-Features(𝒫_i)

Enumerate-Features(S):
    for all elements A_i ∈ S do
        for all elements A_j ∈ S, with j > i do
            R = A_i ∪ A_j; ℒ(R) = ℒ(A_i) ∩ ℒ(A_j);
            if Rule-Prune(R, max_w, max_l) == FALSE and
                frequency(R, c_i) ≥ min_freq(c_i) for any c_i then
                    T = T ∪ {R}; ℱ = ℱ ∪ {R};
        Enumerate-Features(T);

Rule-Prune(R, max_w, max_l):
    if width(R) > max_w or length(R) > max_l) return TRUE;
    if accuracy(R) == 100% return TRUE;
    return FALSE;
```

Figure 3. The FeatureMine algorithm.

the sequence lattice spanned by the subsequence relation, from general to specific sequences, in a depth-first manner. Figure 2 shows the frequent sequences for our example database.

Frequency computation. FeatureMine uses a vertical database layout that associates with each item X in the sequence lattice its *idlist*, denoted $\mathcal{L}(X)$. The idlist is a list of all example eids and event-time pairs containing the item. Figure 2 shows the idlists for all items A and B. Given the sequence idlists, we can determine the support of any k-sequence by simply intersecting the idlists of its lexicographically last two $(k-1)$-length subsequences. A check on the resulting idlist's cardinality tells us whether the new sequence is frequent.

Intersections come in two types: *temporal* and *equality*. For example, Figure 2 shows the idlist for $A \rightarrow B$ obtained by performing a temporal intersection on the idlists of A and B—that is, $\mathcal{L}(A \rightarrow B) = \mathcal{L}(A) \cap_t \mathcal{L}(B)$. FeatureMine does this by determining if, within the same eid, A occurs before B, and listing all such occurrences. On the other hand, we obtain the idlist for $AB \rightarrow B$ by an equality intersection—that is, $\mathcal{L}(AB \rightarrow B) = \mathcal{L}(A \rightarrow B) \cap_e \mathcal{L}(B \rightarrow B)$. Here we check to see if the two subsequences occur in the same eid at the same time. (For additional details, see the article by Mohammed Zaki.[5])

We also maintain the *class index table* (see Figure 2), which indicates the classes for each example. Using this table, we can determine a sequence's frequency in all the classes at the same time. For example, A occurs in eids {1, 2, 3, 4, 5, 6}. However, eids {1, 2, 3, 4} have label c_1 and {5, 6} have label c_2. So, the frequency of A is four for c_1 and two for c_2. The *frequency table* (see Figure 2) shows the class frequencies for each pattern.

To use only a limited amount of main memory, FeatureMine breaks up the sequence search space into small, independent, manageable chunks that can be processed in memory. The algorithm accomplishes this through suffix-based partitioning. Two k-length sequences are in the same equivalence class or *partition* if they share a common $(k-1)$-length suffix. The partitions, such as {[A], [B], [C]}, based on length-one suffixes are *parent partitions*. Each parent partition is independent in that it has complete information for generating all frequent sequences that share the same suffix. For example, if a class [X] has the elements $Y \rightarrow X$ and $Z \rightarrow X$, the only possible frequent sequences at the next step are $Y \rightarrow Z \rightarrow X$, $Z \rightarrow Y \rightarrow X$, and $(YZ) \rightarrow X$. No other item Q can lead to a frequent sequence with the suffix X, unless (QX) or $Q \rightarrow X$ is also in [X].

Feature enumeration. FeatureMine processes each parent partition in a depth-first manner, as Figure 3 shows. The input to the procedure is a partition, along with the idlist for each of its elements. The algorithm generates frequent sequences by intersecting the idlists of all pairs of sequences in each partition and checking the cardinality of the resulting idlist against $min\_sup(c_i)$. The sequences found to be frequent for some class c_i at the current level form partitions for the next level. This process repeats until FeatureMine has enumerated all frequent sequences.

Integrated constraints. The *Rule-Prune* procedure eliminates features based on our two pruning rules and based on length and width constraints. While we must test the first pruning rule each time we extend a sequence with a new item, FeatureMine uses a very efficient one-time method for applying the $A \rightsquigarrow B$ rule. We first compute the frequency of all length-two sequences. If $P(B|A) = fr(AB)/fr(A) = 1.0$, then $A \rightsquigarrow B$, and we can remove AB from the suffix partition [B]. This guarantees that AB will never appear in any set of any sequence.

Empirical evaluation

We tested FeatureMine to see whether the features it produces would improve the performance of the Winnow[6] and Naive Bayes[7] classification algorithms.

Winnow is a multiplicative weight-updating algorithm. We used a variant of Winnow that maintains a weight $w_{i,j}$ for each feature f_i and class c_j. For an example, the activation level for class c_j is

$$\sum_{i=0}^{n} w_{i,j} x_i$$

where x_i is 1 if feature f_i is true in the example, or 0 otherwise. Given an example, Winnow outputs the class with the highest activation level. During training, Winnow iterates through the training examples. If Winnow's classification of a training example does not agree with its label, Winnow updates the weights of each feature f_i that was true in the example: it multiplies the weights for the correct class by some constant $\alpha > 1$ and multiples the weights for the incorrect classes by some constant $\beta < 1$. In our experiments, $\alpha = 1.1$ and $\beta = .91$. Learning is often sensitive to the values of α and β; we chose our values based on what is common in the literature and a little experimentation.

Winnow can actually be used to prune irrelevant features. For example, we can run Winnow with large feature sets (say 10,000) and then throw away any features that are assigned weight 0 or near 0. However, this is not practical for sequence classification because the space of potential features is exponential.

For each feature f_i and class c_j, Naive Bayes computes $P(f_i|c_j)$ as the fraction of training examples of c_j that contain f_i. Given a new example in which features $f_1, \ldots, f_n$ are true, Naive Bayes returns the class that maximizes $P(c_j) \times P(f_1|c_j) \times \ldots \times P(f_n|c_j)$. Even though the Naive Bayes algorithm appears to make the unjustified assumption that all features are independent, it performs surprisingly well, often doing as well as or better than C4.5.[8]

The test domains. We chose these domains:

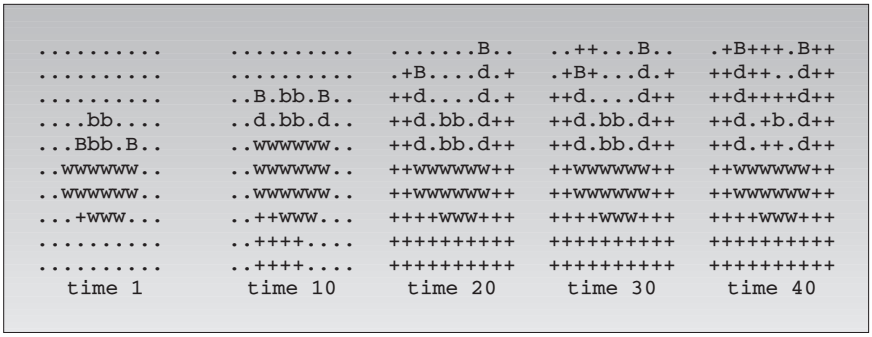

Figure 4. An ASCII representation of several time slices of a simulation of the forest-fire domain. A + indicates fire; b indicates a base; B indicates a bulldozer; d indicates where the bulldozer has dug a fire line; and w indicates water, an unburnable terrain.

random parity problems, forest fire planning, and context-sensitive spelling correction.

Random parity problems. We first describe a nonsequential problem on which standard classification algorithms perform very poorly. In this problem, every feature is true in exactly one-half the examples in each class. The only way to solve this problem is to discover which combinations of features are correlated with the different classes.

Intuitively, we construct a problem by generating N randomly weighted *metafeatures*, each of which consists of a set of M actual, or observable, features. The parity of the M observable features determines whether the corresponding metafeature is true or false, and an instance's class label is a function of the sum of the weights of the true metafeatures. So, to solve these problems, FeatureMine must determine which observable features correspond to the same metafeature. Discovering the metafeatures with higher weights is more important than discovering ones with lower weights. Additionally, to increase the problem's difficulty, we add irrelevant features—that is, ones that have no bearing on an instance's class.

More formally, the problem consists of N parity problems of size M with L distracting, or irrelevant, features. For every i, $0 \le i \le N$, and j, $0 \le j \le M$, a Boolean feature $F_{i,j}$ exists. Additionally, for every k, $0 \le k \le L$, an irrelevant Boolean feature I_k exists. To generate an instance, we randomly assign each relevant and irrelevant Boolean true or false with 50–50 probability. An example instance for $N = 3$, $M = 2$, and $L = 2$ is ($F_{1,1}$ = true, $F_{1,2}$ = false, $F_{2,1}$ = true, $F_{2,2}$ = true, $F_{3,1}$ = false, $F_{3,2}$ = false, I_1 = true, I_2 = false). There are $N \times M + L$ features and $2^{N \times M + L}$ distinct instances. All possible instances are equally likely.

We also choose N weights $w_1, ..., w_N$, from the uniform distribution between 0 and 1. We credit an instance with weight w_i if and only if the ith set of M features has an even parity. That is, an instance's "score" is the sum of the weights w_i for which the number of true features in $f_{i,1}, ..., f_{i,M}$ is even. If an instance's score is greater than one-half the sum of all the weights,

$$\sum_{i=1}^{N} w_i$$

we assign the class label On to that instance; otherwise, we assign Off. If $M > 1$, no feature by itself is indicative of On or Off, which is why parity problems are so hard for most classifiers.

FeatureMine essentially figures out which features should be grouped together. For example, features that FeatureMine produced for this domain include ($f_{1,1}$ = true, $f_{1,2}$ = true), and ($f_{4,1}$ = true, $f_{4,2}$ = false). We used a *min_freq* of .02 to .05, $max_l = 1$ and $max_w = M$.

Forest-fire planning. FeatureMine's original task was plan monitoring in stochastic domains. Probabilistic planners construct plans with a high probability of achieving their goal. The monitor's task is to watch the plan execute and predict whether the plan will succeed or fail, to facilitate replanning.

Plan monitoring (or monitoring any probabilistic process that we can simulate) is an attractive area for machine learning because an essentially unlimited supply of training data exists. Unfortunately, because the number of possible execution paths increases exponentially with the length of the plan or process, we cannot consider all possible execution paths. However, we can generate arbitrary numbers of new examples with Monte Carlo simulation. This reduces overfitting because we can test our hypotheses on "fresh" data sets.

Given a plan and goal, to build a monitor, we first simulate the plan repeatedly to generate execution traces. We label each execution trace with Success or Failure, depending on whether the goal is achieved in the simulation. These execution traces become the input to FeatureMine.

We constructed a simple forest-fire domain based loosely on the Phoenix fire simulator[9] (execution traces are available by e-mail, lesh@merl.com). We use a grid representation of the terrain. Each grid cell can contain vegetation, water, or a base. Figure 4 shows example terrain. At each simulation's beginning, fire starts at a random location. In each iteration of the simulation, the fire spreads stochastically. The probability of a cell igniting at time t is calculated based on the cell's vegetation, the wind direction, and how many of the cell's neighbors are burning at time $t - 1$. Additionally, the simulation uses bulldozers to contain the fire. For each example terrain, we hand-designed a plan for bulldozers to dig a fire line to stop the fire. A bulldozer's speed varies from simulation to simulation. Here's part of an example simulation:

> (time0 Ignite X3 Y7), (time0 MoveTo BD1 X3 Y4), (time0 MoveTo BD2 X7 Y4), (time0 DigAt BD2 X7 Y4), …, (time6 Ignite X4 Y8), (time6 Ignite X3 Y8), (time8 DigAt BD2 X7 Y3), (time8 Ignite X3 Y9), (time8 DigAt BD1 X2 Y3), (time8 Ignite X5 Y8), …, (time32 Ignite X6 Y1), (time32 Ignite X6 Y0), ….

We tag each plan with Success if no location with a base has been burned in the final state, or Failure otherwise. To train a plan monitor that can predict at time k whether or not the bases will ultimately be burned, we include only events that occur by time k in the training examples. Two features that FeatureMine produced for this domain are

- (MoveTo BD1 X2) → (time6) and
- (Ignite X2) → (time8 MoveTo Y3).

The first sequence holds if bulldozer BD1 moves to the second column before time 6. The second holds if a fire ignites anywhere in the second column and then any bulldozer moves to the third row at time 8.

Many correlations used by our second pruning rule (described in "Selection criteria for mining") arise in these data sets. For example, Y8 ⇝ Ignite arises in one of our test plans in which a bulldozer never moves in the eighth column.

For the fire data, 38 Boolean features describe each event. So, we search over $(38 \times 2)^{max_w \times max_l}$ composite features. In these experiments, *min_freq* = .2, $max_w = 3$, and $max_l = 3$.

Table 1. The average classification accuracy using different feature sets to represent the examples. TF indicates features obtained by the times × features approach, and FM indicates features produced by FeatureMine. FeatureMine produced the most accurate features. The standard deviations are in parentheses following each average, except for the spelling problems, for which we used only one test and training set.

Experiment	Winnow	WinnowTF	WinnowFM	Bayes	BayesTF	BayesFM
Random parity						
$N = 5, M = 3, L = 5$	.51 (.02)	N/A	.97 (.03)	.50 (.01)	N/A	.97 (.04)
$N = 3, M = 4, L = 8$	.49 (.01)	N/A	.99 (.04)	.50 (.01)	N/A	1.00 (0)
$N = 10, M = 4, L = 10$	.50 (.01)	N/A	.89 (.03)	.50 (.01)	N/A	.85 (.06)
Forest fire planning						
Time = 5	.60 (.11)	.65 (.12)	.79 (.02)	.69 (.02)	.72 (.02)	.81 (.02)
Time = 10	.60 (.14)	.77 (.07)	.85 (.02)	.68 (.01)	.68 (.01)	.75 (.02)
Time = 15	.55 (.16)	.74 (.11)	.89 (.04)	.68 (.01)	.68 (.01)	.72 (.02)
Spelling correction						
Their vs. there	.70	N/A	.94	.75	N/A	.78
I vs. me	.86	N/A	.94	.66	N/A	.90
Than vs. then	.83	N/A	.92	.79	N/A	.81
You're vs. your	.77	N/A	.86	.77	N/A	.86

Context-sensitive spelling correction. We tested our algorithm on the task of correcting spelling errors that result in valid words, such as substituting *there* for *their*.[10] For each test, we chose two commonly confused words and searched for sentences in the one-million-word Brown corpus[11] containing either word. We removed the target word and then represented each word by the word itself, the part-of-speech tag in the Brown corpus, and the position relative to the target word. For example, the sentence "And then there is politics" translates into (word=and tag=cc pos=–2) → (word=then tag=rb pos=–1) → (word=is tag=bez pos=+1) → (word=politics tag=nn pos=+2).

Two features that FeatureMine produced are (pos=+3) → (word=the), indicating that *the* occurs at least three words after the target word, and (pos=–4) → (tag=nn) → (pos=+1), indicating that a noun occurs within three words before the target word. These features (for reasons not obvious to us) correlated significantly with either *there* or *their* in the training set.

The *I* versus *me* data set had 3,802 training examples, 944 test examples, and 4,692 feature–value pairs. The *there* versus *their* data set had 2,917 training examples, 755 test examples, and 5,663 feature–value pairs. The *than* versus *then* data set had 2,019 training examples, 494 test examples, and 4,331 feature–value pairs. The *you're* versus *your* data set had 647 training examples, 173 test examples, and 1,646 feature–value pairs. If N is the number of feature–value pairs, we search over an $N^{max_w \times max_l}$ pattern space. In these experiments, $min\_freq = .05$, $max_w = 3$, and $max_l = 2$.

Table 2. The number of features that FeatureMine considered and returned.

Experiment	Features Evaluated	Features Selected
Random parity, $N = 10, M = 4, L = 10$	7,693,200	196
Forest fire planning, time = 10	64,766	553
Spelling correction, there vs. their	72,264	318

Results. For each test in the parity and fire domains, we generated 7,000 random training examples. We mined features from 1,000 examples, pruned features that did not pass a chi-squared significance test (for correlation to a class in which the feature was frequent) in 2,000 examples, and trained the classifier on the remaining 5,000 examples. We then tested on 1,000 additional examples. The results in Tables 1 and 2 for these domains are averages from 25 to 50 such tests.

For spelling correction, we used all the examples in the Brown corpus, roughly 1,000 to 4,000 examples per word set, split 80–20 (by sentence) into training and test sets. We mined features from 500 sentences and trained the classifier on the entire training set.

Table 1 shows that the features produced by FeatureMine improved classification performance. We compared using the feature set produced by FeatureMine with using only the primitive features themselves—that is, length-one features. In the forest-fire domain, we also evaluated the feature set containing a feature for each primitive feature at each time step (this is the feature set of size $3N$ described in "Poker and feature mining"). Both Winnow and Naive Bayes performed much better with the features produced by FeatureMine. In the parity exper-

Table 3. The running time and nodes visited for FeatureMine with and without $A \rightsquigarrow B$ pruning. The results are from one data set for each example. "All pruning" means both pruning rules were applied.

Experiment	CPU seconds: No pruning	CPU seconds: $A \rightsquigarrow B$ pruning	CPU seconds: All pruning	Features examined: No pruning	Features examined: $A \rightsquigarrow B$ pruning	Features examined: All pruning
Random parity	320	337	337	1,547,122	1,547,122	1,547,122
Forest fire planning	5.8 hours	560	559	25,336,097	511,215	511,215
Spelling correction	490	407	410	1,126,114	999,327	971,085

iments, the mined features dramatically improved the performance of the classifiers, and in the other experiments, the mined features significantly improved the accuracy of the classifiers, often by more than 20%.

Table 2 shows the number of features evaluated and the number returned, for three problems. For the random-parity problem, there were 100 million possible features. (There are 50 Boolean features, giving rise to 100 feature–value pairs. We searched to a depth of four because $M = 4$.) However, the pruning rules implicitly rejected most.

Table 3 shows the $A \rightsquigarrow B$ pruning rule's impact. The results are from one data set from each domain, with slightly higher values for max_l and max_w than in the previous experiments. The pruning rule did not always improve mining time, but it made a tremendous difference in the forest-fire problems, where the same event descriptors often appear together. Without $A \rightsquigarrow B$ pruning, the forest-fire problems are essentially unsolvable because FeatureMine finds more than 20 million frequent sequences.

A related task is to predict important events in a continuous stream of observations, rather than in the sequences with clear beginnings and ends that we considered here. One example of such a task is to predict when a computer server is likely to crash based on observations of network traffic. Another way in which this work could be extended is to involve human experts in the selection of features. That is, the features selected by our algorithm could be viewed as suggestions to be reviewed by a human user.

THE RESULTS SHOW THAT WE CAN construct problems in which classifiers perform no better than random guessing using the original features but perform with nearly perfect accuracy when using the features produced by FeatureMine. More generally, this work shows that we can apply classification algorithms to domains that contain no obvious small set of features for describing examples but that contain a large space of combinations of primitive features that probably include some useful features. Future work could involve applying these ideas to the classification of, for example, images or audio signals. We also plan to test the mined features for clustering tasks. For background on related research on feature mining, see the sidebar. ■

Related work

Feature-subset selection has experienced a great deal of research, motivated by the observation that classifiers can perform worse with feature set $\mathcal{F}$ than with some $\mathcal{F}' \subset \mathcal{F}$.[1] Such algorithms explore the exponentially large space of all subsets of a given feature set. In contrast, we explore exponentially large sets of potential features but evaluate each feature independently. The feature-subset approach seems infeasible for the problems we consider, which contain hundreds of thousands to millions of potential features.

Andrew Golding and Dan Roth applied a Winnow-based algorithm to context-sensitive spelling correction.[2] They used sets of 10,000 to 40,000 features and either used all those features or pruned some based on the classification accuracy of the individual features. They obtained higher accuracy than we did. Their approach, however, involved an ensemble of Winnows, combined by majority weighting, and they took more care in choosing good parameters for this specific task. Our goal was simply to demonstrate that the features produced by FeatureMine improve classification performance.

Data mining algorithms have often been applied to classification.[3] Bing Liu, Wynne Hsu, and Yimin Ma built decision lists out of patterns found by association mining, which is the nonsequential version of sequence mining. Additionally, while previous work has explored new methods for combining association rules to build classifiers, the thrust of our work has been to leverage and augment standard classification algorithms. Our pruning rules resemble ones used by Richard Segal and Oren Etzioni, who also have employed data mining to construct decision lists.[4] Previous work on using data mining for classification has focused on combining highly accurate rules. In contrast, our classification algorithms can weigh evidence from many features that each have low accuracy to classify new examples.

Our work is close in spirit to that of Daniel Kudenko and Haym Hirsh,[5] who also constructed a set of sequential, Boolean features for use by classification algorithms. They employed FGEN, a heuristic search algorithm that incrementally generalizes features to cover more and more of the training examples, based on its classification performance on a holdout set of training data. In contrast, we perform an exhaustive search (to some depth) and accept all features that meet our selection criteria. Also, we use a different feature language and have tested our approaches on different classifiers.

References

1. R. Caruana and D. Freitag, "Greedy Attribute Selection," *Proc. ML '94: 11th Int'l Conf. Machine Learning*, Morgan Kaufmann, San Francisco, 1994, pp. 28–36.
2. A. Golding and D. Roth, "Applying Winnow to Context-Sensitive Spelling Correction," *Proc. ML '96: 13th Int'l Conf. Machine Learning*, Morgan Kaufmann, San Francisco, 1996, pp. 180–190.
3. B. Liu, W. Hsu, and Y. Ma, "Integrating Classification and Association Rule Mining," *Proc. Fourth Int'l Conf. Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, Calif., 1998, pp. 80–86.
4. R. Segal and O. Etzioni, "Learning Decision Lists Using Homogeneous Rules," *Proc. AAAI '94: 12th Nat'l Conf. Artificial Intelligence*, AAAI Press, Menlo Park, Calif., 1994, pp. 619–625.
5. D. Kudenko and H. Hirsh, "Feature Generation for Sequence Categorization," *Proc. AAAI '98: 15th Nat'l Conf. Artificial Intelligence*, AAAI Press, Menlo Park, Calif., 1998, pp. 733–739.

Acknowledgment

Mitsunori Ogihara is supported in part by National Science Foundation grants CCR-9701911, CCR-9725021, and INT-9726724 and by DARPA grant F30602-98-2-0133.

References

1. R. Segal and O. Etzioni, "Learning Decision Lists Using Homogeneous Rules," *Proc. AAAI '94: 12th Nat'l Conf. Artificial Intelligence*, AAAI Press, Menlo Park, Calif., 1994, pp. 619–625.
2. B. Liu, W. Hsu, and Y. Ma, "Integrating Classification and Association Rule Mining," *Proc. Fourth Int'l Conf. Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, Calif., 1998, pp. 80–86.
3. W. Lee, S. Stolfo, and K. Mok, "Mining Audit Data to Build Intrusion Detection Models," *Proc. Fourth Int'l Conf. Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, Calif., 1998, pp. 66–72.
4. R. Caruana and D. Freitag, "Greedy Attribute Selection," *Proc. ML '94: 11th Int'l Conf. Machine Learning*, Morgan Kaufmann, San

2000 Editorial Calendar

JAN/FEB — Cross-Pollinating Disciplines
Ideally, every engineering discipline should learn from each other's strengths to minimize the risk to public safety. However, specialization within each discipline often inhibits mutual learning. This issue offers lessons the software industry can learn from other disciplines and vice versa.

MAR/APR — Cobol
Now that Y2K preparations are over, what is Cobol's role from here on? This issue will present some innovative but practical answers, as well as leads on related tools.

MAY/JUN — Process Diversity
People who build software use a myriad of different approaches for development, from extreme programming to CMM, depending on factors such as application domain, system size, architecture, project size, reliability, and safety requirements. This focus looks at the diversity range of approaches and where to get more information about each.

Requirements Engineering
This issue highlights the best papers from ICRE '99, contributions that address changes in people's expectations, in business practices, in social forces, in enabling technology opportunities. The magazine will be bundled with the conference proceedings and distributed to all attendees.

JUL/AUG — Corporate-Level Implementations of CMM
Only a few have made it to Level 5. Who are they? Where have they succeeded? Where do they go from here? Is anything still missing?

Software Engineering in the Small
Small groups, especially those developing mass-market and Internet software products, rarely work according to the published, formal methods. This issue deals with the particular needs of small groups working on complex projects—realistic best practices, failure and success stories, and helpful tools.

SEP/OCT — Viruses, Intrusion Detection, and Security
What are the big shops and government labs doing? What can all of us do to protect our information and our organizations? This issue will also provide a full list of security products.

Human Factors
What is being done to make computers adapt to the needs of their human counterparts? How can we change our systems to help our teams work better? What resources are available to improve relations between users and their computers.

Estimation
What is the latest on software project estimation approaches, techniques, and tools? This issue will cover today's practices as well as research that might lead to tomorrow's advances.

NOV/DEC — Why2K? Lessons Learned
We've had time to reflect, evaluate, and look ahead. What have we learned? Where do similar dangers lie ahead?

Multisite/Multicultural Project Ware Stories
What works and what doesn't when a team is distributed across time zones, continents, and cultures?

IEEE Software

computer.org/software

Francisco, 1994, pp. 28–36.

5. M.J. Zaki, "Efficient Enumeration of Frequent Sequences," *Proc. CIKM '98: Seventh Int'l Conf. Information and Knowledge Management*, ACM Press, New York, 1998, pp. 68–75.
6. N. Littlestone, "Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm," *Machine Learning*, Vol. 2, No. 4, 1987, pp. 285–318.
7. R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York, 1973.
8. P. Domingos and M. Pazzani, "Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier," *Proc. ML '96: 13th Int'l Conf. Machine Learning*, Morgan Kaufmann, San Francisco, 1996, pp. 105–112.
9. D.M. Hart and P.R. Cohen, "Predicting and Explaining Success and Task Duration in the Phoenix Planner," *Proc. First Int'l Conf. Artificial Intelligence Planning Systems*, AAAI Press, Menlo Park, Calif., 1992, pp. 106–115.
10. A. Golding and D. Roth, "Applying Winnow to Context-Sensitive Spelling Correction," *Proc. ML '96: 13th Int'l Conf. Machine Learning*, Morgan Kaufmann, San Francisco, 1996, pp. 180–190.
11. H. Kucera and W.N. Francis, *Computational Analysis of Present-Day American English*, Brown Univ. Press, Providence, R.I., 1967.

Neal Lesh is a research scientist at the Mitsubishi Electric Research Lab. His research interests focus on plan recognition, collaborative planning, and probabilistic-inference tasks. He received his BS in computer science from Brown University and his PhD in computer science from the University of Washington. Contact him at MERL, 201 Broadway, Cambridge, MA 02139; lesh@merl.com.

Mohammed J. Zaki is an assistant professor of computer science at Rensselaer Polytechnic Institute. His research interests focus on developing efficient, scalable, parallel, and interactive algorithms for data mining and knowledge discovery. He received his BS in computer science from Angelo State University and his MS and PhD in computer science from the University of Rochester. Contact him at the Computer Science Dept., Rensselaer Polytechnic Inst., Troy, NY 12180; zaki@cs.rpi.edu.

Mitsunori Ogihara (also known as Ogiwara) is an associate professor in, and the chair of, the Computer Science Department at the University of Rochester. His research interests are computational complexity, DNA computing, and data mining. He received his BS, MS, and PhD in information sciences from the Tokyo Institute of Technology. Contact him at the Computer Science Dept., Univ. of Rochester, Rochester, NY 14627; ogihara@cs.rochester.edu.