00 01 02 03 04 05 06 07 08 09 0A

In *Discovery Science 99, LNAI 1721, Tokyo, Japan, 1999.* Revised November 5, 1999, with correct accuracy results.

CAEP: Classification by Aggregating Emerging Patterns

Guozhu Dong[1] and Xiuzhen Zhang[2] and Limsoon Wong[3] and Jinyan Li[2]

[1] Department of CSE, Wright State University, OH 45435, USA.
gdong@cs.wright.edu
[2] Department of CSSE, Univ of Melbourne, Vic 3052, Australia.
{xzhang,jyli}@cs.mu.oz.au
[3] Kent Ridge Digital Labs, Singapore. limsoon@krdl.org.sg

Abstract. Emerging patterns (EPs) are itemsets whose supports change significantly from one dataset to another; they were recently proposed to capture multi-attribute contrasts between data classes, or trends over time. In this paper we propose a new classifier, CAEP, using the following main ideas based on EPs: (i) Each EP can sharply differentiate the class membership of a (possibly small) fraction of instances containing the EP, due to the big difference between its supports in the opposing classes; we define the differentiating power of the EP in terms of the supports and their ratio, on instances containing the EP. (ii) For each instance t, by aggregating the differentiating power of a fixed, automatically selected set of EPs, a score is obtained for each class. The scores for all classes are normalized and the largest score determines t's class. CAEP is suitable for many applications, even those with large volumes of high (e.g. 45) dimensional data; it does not depend on dimension reduction on data; and it is usually equally accurate on all classes even if their populations are unbalanced. Experiments show that CAEP has consistent good predictive accuracy, and it almost always outperforms C4.5 and CBA. By using efficient, border-based algorithms (developed elsewhere) to discover EPs, CAEP scales up on data volume and dimensionality. Observing that accuracy on the whole dataset is too coarse description of classifiers, we also used a more accurate measure, *sensitivity* and *precision*, to better characterize the performance of classifiers. CAEP is also very good under this measure.

1 Introduction

Classification is an important problem in data mining and machine learning, aimed at building a classifier from training instances for predicting the classes of new instances. Recently, datasets are becoming increasingly larger in both volume and dimensionality (number of attributes); a new challenge is the ability to efficiently build highly accurate classifiers from such datasets. In this paper we propose a new classifier, CAEP (Classification by Aggregating Emerging Patterns), which is suitable for many applications, even those with large volumes of high dimensional data. The classifier is highly accurate, and is usually equally accurate on all classes even if their populations are unbalanced. These advantages are achieved without dimension reduction on data.

CAEP is based on the following two main new ideas:

(i) We use a new type of knowledge, the so-called *emerging patterns* (EPs), recently proposed in [4], to build CAEP. Roughly speaking, EPs are those itemsets whose supports (i.e. frequencies) increase significantly from one class of data to another. For example, the itemset {odor=none, stalk-surface-below-ring = smooth, ring-number=one} in the Mushroom dataset [12] is a typical EP, whose support increases from 0.2% in the poisonous class to 57.6% in the edible class, at a growth rate of 288 ($= \frac{57.6\%}{0.2\%}$). For us, an item is a simple test on an attribute, and an EP is a multi-attribute test. Each EP can have very strong power for differentiating the class membership of some instances. Indeed, if a new instance s contains the above EP, then with odds of 99.6% we can claim that s belongs to the edible class. In general, the differentiating power of an EP is roughly proportional to the growth rate of its supports and its support in the target class.

(ii) An individual EP is usually sharp in telling the class of only a very *small* fraction (e.g. 3%) of all instances, and thus it will have very poor overall classification accuracy if it is used by itself on all instances. To build an accurate classifier, we first find, for each class $\mathcal{C}$, all the EPs meeting some support and growth rate thresholds, from the (opponent) set of all none-$\mathcal{C}$ instances to the set of all $\mathcal{C}$ instances. Then we aggregate the power of the discovered EPs for classifying an instance s: We derive an aggregate differentiating score for each class $\mathcal{C}$, by summing the differentiating power of all EPs of $\mathcal{C}$ that occur in s; the score for $\mathcal{C}$ is then normalized by dividing it by some base score (e.g. median) of the training instances of $\mathcal{C}$. Finally, we let the largest normalized score determine the winning class. Normalization is done to reduce the effect of unbalanced distribution of EPs among the classes (classes with more EPs frequently give higher scores to instances, even to those from other classes).

CAEP achieves very good predictive accuracy on *all* the datasets we tested, and it outperforms the best of five classifiers on 5 out of 9 datasets (see Table 1). We believe that the high accuracy is achieved because we are using a new high dimensional method to solve a high dimensional problem: Each EP is a multi-attribute test and CAEP is using the combined power of an unbounded set of EPs to arrive at a classification decision.

Being equally accurate on all classes is very useful for many applications, where there are a dominant class (e.g. 98% of all instances) and a minority class, and the sole purpose of classification is to accurately catch instances of the minority class. Classification accuracy is not the desired measure, as we would then consider the classifier which classifies all instances as in the dominant class a very good classifier. In this paper we also measure classifiers using *sensitivity* and *precision*, which reward classifiers that correctly label more minority instances and do not mislabel many other instances.

The CAEP classifier can be efficiently built for large high dimensional training datasets in a scalable way, since EPs can be discovered efficiently, using border-based algorithms [4] and Max-Miner [1]. We can quickly produce CAEP classifiers for datasets such as Mushroom, whose records consist of 21 attributes. (Figure 9 of [1] shows that CPU time is 100 seconds for support threshold of 0.1%. Border-based algorithms [4] then can find the EPs in around 0.5 hour.)

There are several parameters that need to be selected. All these are done automatically, using the performance of the resulting classifier on the training instances as guidance. Because of the use of aggregation and perhaps normalization, we have not encountered the traditional overfitting problem in our experiments.

Organization: We compare CAEP with related work below. §2 introduces EPs and preliminaries. §3 presents our main ideas on how to build CAEP. §4 discusses how to efficiently discover the EPs, their supports and growth rates. §5 discusses how to reduce the number of EPs and §6 on the automatic selection of CAEP's parameters. §7 discusses the rice-DNA dataset and the need to use *sensitivity* and *precision* to measure classifiers. §8 contains the experimental results, and §9 offers some concluding remarks.

Related Work: CAEP is fundamentally different from previous classifiers in its use of the new knowledge type of EPs. To arrive at a score for decision making, CAEP uses a set of multi-attribute tests (EPs) for each class. Most previous classifiers consider only one test on one attribute at a time; a few exceptions, X-of-N [18], CBA [11] and linear decision trees [2], consider only one multi-attribute test to make a decision.

Aggregation of the differentiating power of EPs is different from bagging or boosting [17], which manipulate the training data to generate different classifiers and then aggregate the votes of several classifiers. With CAEP, each EP is too weak as a classifier and all the EPs are more easily obtained.

Loosely speaking, our aggregation of the power of EPs in classification is related to the Bayesian prediction theory. For an instance t viewed as an itemset, Bayesian prediction would label t as $\mathcal{C}_k$, where the probability $Pr(t|\mathcal{C}_k) * Pr(\mathcal{C}_k)$ is the largest among the classes. The optimal Bayesian classifier needs to "know" the probability $Pr(t|\mathcal{C}_k)$ for all possible t, which is clearly impractical for high dimensional datasets. Roughly speaking, CAEP "approximates" $Pr(t|\mathcal{C}_k) * Pr(\mathcal{C}_k)$ using the normalized score.

CAEP is the first application of EPs to classification. Partially influenced by CAEP, [10] proposes a different classifier, JEP-Classifier, also based on aggregated power of EPs. Major differences include: (i) CAEP uses general EPs, whereas JEP-Classifier uses exclusively jumping EPs (i.e. EPs whose support increases from zero in one dataset to non-zero in the other dataset). (ii) For datasets with more than two classes, CAEP uses the classes in a symmetric way, whereas JEP-Classifier uses them in an ordered way. (iii) In aggregating the differentiating power of all EPs, CAEP uses factors based on both support and support growth rate, whereas JEP-Classifier uses only the supports. (iv) As CAEP uses EPs with mixed growth rates, the reduction of the EPs is more complicated; for JEP-Classifier, all jumping EPs have infinite growth rates and the reduction is simpler. (v) The normalization idea is used in CAEP but not in

JEP-Classifier. The two classifiers offer their own advantages: CAEP is better for cases with few or even no jumping EPs whose supports meet a reasonable threshold (such as 1%), whereas JEP-Classifier is better when there are many jumping EPs. Each of them is almost consistently better than C4.5 and CBA.

2 Emerging Patterns and Preliminaries

Assume the original data instances have m attribute values. Each instance in the training dataset $\mathcal{D}$ is associated with a class label, out of a total of p class labels: $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_p$. We partition $\mathcal{D}$ into p sets, $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_p$, with $\mathcal{D}_i$ containing all instances of class $\mathcal{C}_i$.

Emerging patterns are defined for binary transaction databases. To find them, we may need to encode a raw dataset into a binary one: We discretize the value range of each continuous attribute into intervals [9]. Each $(attribute, interval)$ pair is called an *item* in the binary (transaction) database, which will be represented as an integer for convenience. An instance t in the raw dataset will then be mapped to a transaction of the binary database: t has the value 1 on exactly those items (A, v) where t's A-value is in the interval v. We will represent this new t as the set of items for which it takes 1, and we will assume henceforth the datasets $\mathcal{D}$, $\mathcal{D}_1$, ..., $\mathcal{D}_p$ are binary.

Let I be the set of all items in the encoding. An *itemset* X is a subset of I, and its *support* in a dataset $\mathcal{D}'$, $supp_{\mathcal{D}'}(X)$, is $\frac{|\{t \in \mathcal{D}' | X \subseteq t\}|}{|\mathcal{D}'|}$. Given two datasets $\mathcal{D}'$ and $\mathcal{D}''$, the *growth rate* of an itemset X from $\mathcal{D}'$ to $\mathcal{D}''$ is defined as $growth\_rate_{\mathcal{D}' \to \mathcal{D}''}(X) = \frac{supp_{\mathcal{D}''}(X)}{supp_{\mathcal{D}'}(X)}$ if $supp_{\mathcal{D}'}(X) \neq 0$; $= 0$ if $supp_{\mathcal{D}'}(X) = supp_{\mathcal{D}''}(X) = 0$; and $= \infty$ if $supp_{\mathcal{D}'}(X) = 0 \neq supp_{\mathcal{D}''}(X)$.

Emerging patterns [4] are itemsets with large growth rates from $\mathcal{D}'$ to $\mathcal{D}''$.

Definition 1. *Given $\mathcal{D}'$, $\mathcal{D}''$ and a growth rate threshold $\rho > 1$, an* emerging pattern *(ρ-EP or simply EP) from $\mathcal{D}'$ to $\mathcal{D}''$ is an itemset e where $growth\_rate_{\mathcal{D}' \to \mathcal{D}''}(e) \geq \rho$.*

Example 1. Consider the following training dataset with two classes, $\mathcal{P}$ and $\mathcal{N}$ (this is actually an encoding of the Saturday morning activity example from [15]).

$\mathcal{P}$			$\mathcal{N}$	
{ 2,6,7,10 }	{ 3,5,7,10 }	{ 3,4,8,10 }	{ 1,6,7,10 }	{ 1,6,7,9 }
{ 2,4,8,9 }	{ 1,4,8,10 }	{ 3,5,8,10 }	{ 3,4,8,9 }	{ 1,5,7,10 }
{ 1,5,8,9 }	{ 2,5,7,9 }	{ 2,6,8,10 }	{ 3,5,7,9 }	

Then $\{1, 9\}$ is an EP from class $\mathcal{P}$ to class $\mathcal{N}$ with a growth rate $\frac{9}{5}$; it is also an ρ-EP for any $1 < \rho \leq \frac{9}{5}$. Some other EPs are given later.

3 Classification by Aggregating EPs

We now describe the major ideas and components of the CAEP classifier: (1) how to partition the dataset to derive the EPs for use in CAEP, (2) how individual EPs can differentiate class memberships, (3) how to combine the contribution of individual EPs to derive the *aggregate scores*, and (4) how to normalize the aggregate scores for deciding class membership. We also give an overview on how to construct and use CAEP.

3.1 Partitioning dataset to get EPs of classes

For each class $\mathcal{C}_k$, we will use a set of EPs to contrast its instances, $\mathcal{D}_k$, against all other instances: We let $\mathcal{D}'_k = \mathcal{D} - \mathcal{D}_k$ be the opposing class, or simply opponent, of $\mathcal{D}_k$. We then mine (discussion on how is given later) the EPs from $\mathcal{D}'_k$ to $\mathcal{D}_k$; we refer to these EPs as *the EPs of class* $\mathcal{C}_k$, and sometimes refer to $\mathcal{C}_k$ as the *target* class of these EPs.

For Example 1, some EPs of class $\mathcal{N}$ (i.e. from $\mathcal{P}$ to $\mathcal{N}$) are $(e : \{1\}, supp_{\mathcal{N}}(e) : 0.6, growth\_rate_{\mathcal{P}\to\mathcal{N}} : 2.7)$, $(\{1,7\}, 0.6, \infty)$, $(\{1,10\}, 0.4, 3.6)$, $(\{3,4,8,9\}, 0.2, \infty)$. Similarly, some EPs of $\mathcal{P}$ are $(\{2\}, 0.44, \infty)$, $(\{8\}, 0.67, 3.33)$, $(\{4,8\}, 0.33, 1.67)$.

3.2 Differentiating power of individual EPs

Each EP can sharply differentiate the class membership of a fraction of instances which contain the EP, and this sharp differentiating power is derived from the big difference between its supports in the opposing classes. Continuing with Example 1, consider the EP $(\{1,10\}, 0.40, 3.60)$ for class $\mathcal{N}$. Suppose s is an instance containing this EP. What is the odds that s belongs to $\mathcal{N}$, given that s contains this EP? To simplify the discussion, we assume all classes have roughly equal population counts; then the answer is $\frac{supp_{\mathcal{N}}}{supp_{\mathcal{N}}+supp_{\mathcal{P}}} = \frac{3.60*supp_{\mathcal{P}}}{3.60*supp_{\mathcal{P}}+supp_{\mathcal{P}}} = \frac{3.60}{3.60+1} = 78\%$, since $supp_{\mathcal{N}} = 3.60 * supp_{\mathcal{P}}$. Without this assumption, we need to replace supports (e.g. $supp_{\mathcal{N}}$) by counts (e.g. $supp_{\mathcal{N}}*count_{\mathcal{N}}$ where $count_{\mathcal{N}}$ is the number of instances of class $\mathcal{N}$), and similar odds can be obtained. Observe that this EP has no differentiating power on instances s' that do not contain the EP. Therefore, assuming the population ratio in the training data accurately reflects the ratio in test instances and all classes have roughly equal population counts, this EP can be used to differentiate the class membership with the probability of 78% for roughly $\frac{supp_{\mathcal{N}}+supp_{\mathcal{P}}}{2} = 0.5 * (1 + \frac{1}{3.60}) * supp_{\mathcal{N}} = 25\%$ of the total population.

The fraction of instances which contain an EP may be a very small fraction (25% above, but much smaller, e.g. 3%, in many examples) of all instances. Hence, it cannot yield very accurate predictions if it is used by itself on all instances. For example, if we applied the above EP on all instances, we would arrive at an overall predictive accuracy of roughly $0.25 * 0.78 = 19.5\%$. This would be much lower if coverage is only 3%.

3.3 Better overall accuracy by aggregated score

We noticed above that a single EP is sharp on predicting class membership of a small fraction of instances, but not on all instances. We now show how to combine the strength of a set of EPs in order to produce a classifier with good overall accuracy.

Roughly speaking, given a test instance s, we let all the EPs of a class $\mathcal{C}_i$ that s contains contribute to the final decision of whether s should be labelled as $\mathcal{C}_i$. This gives us the advantage of covering more cases than each single EP can cover, because different EPs complement each other in their applicable populations. To illustrate, consider Example 1. The largest fraction of population that a single EP (e.g. $\{8\}$) can cover is around 50%, whereas the seven EPs given above in §3.1 have a much larger combined coverage, around $\frac{12}{14} \approx 85.7\%$.

How do we combine the differentiating power of a set of EPs? A natural way is to sum the contributions of the individual EPs. (Other possibilities exist, but are beyond the scope of this paper.) Now, how do we formulate the contribution of a single EP? Roughly, we use a product of the odds discussed earlier and the fraction of the population of the class that contain the EP. More specifically, let e be an EP of class $\mathcal{C}$, we let e's contribution be given by $\frac{growth\_rate(e)}{growth\_rate(e)+1} * supp_{\mathcal{C}}(e)$. Observe that the first term is roughly the conditional probability that an instance is in class $\mathcal{C}$ given that the instance contains this EP e, and the second term is the fraction of the instances of class $\mathcal{C}$ that this EP applies. The contribution is proportional to both $growth\_rate(e)$ and $supp_{\mathcal{C}}(e)$. We now define scores of instances for the classes.

Definition 2. *Given an instance* s *and a set $E(\mathcal{C})$ of EPs of a class $\mathcal{C}$ discovered from the training data, the* aggregate score *(or* score*) of* s *for $\mathcal{C}$ is defined as*

$$score(s, \mathcal{C}) = \sum_{e \subseteq s, e \in E(\mathcal{C})} \frac{growth\_rate(e)}{growth\_rate(e) + 1} * supp_{\mathcal{C}}(e).$$

We now illustrate the calculation of contributions of EPs and scores of instances using Example 1 and the instance $s = \{1, 5, 7, 9\}$. Among EPs of the growth rate threshold of 1.1, s contains 2 of class $\mathcal{P}$: ($\{5\}$, 44%, 1.11), ($\{1,5,9\}$, 11%, ∞); it contains 10 of class $\mathcal{N}$: ($\{1\}$, 60%, 2.7), ($\{7\}$, 80%, 2.4), ($\{1,5\}$, 20%, 1.8), ($\{1,7\}$, 60%, ∞), ($\{1,9\}$, 20%, 1.8), ($\{5,7\}$, 40%, 1.8), ($\{7,9\}$, 40%, 3.6), ($\{1,5,7\}$, 20%, ∞), ($\{1,7,9\}$, 20%, ∞), ($\{5,7,9\}$, 20%, 1.8). The aggregate score of s for $\mathcal{P}$ is: $score(s, \mathcal{P}) = \frac{1.11}{1.11+1} * 0.44 + \frac{\infty}{\infty+1} * 0.11 = 0.52 * 0.44 + 1 * 0.11 = 0.33$. Similarly, the contributions of the 10 EPs for $\mathcal{N}$ are respectively 0.41, 0.56, 0.12, 0.60, 0.12, 0.24, 0.31, 0.20, 0.20, 0.12, and their sum is $score(s, \mathcal{N}) = 2.88$.

3.4 Normalizing the scores to make decision

For each instance s, how do we use the p scores for all classes to predict its class?

One might be tempted to assign to s the class label $\mathcal{C}$, for which the score of s is the largest. This turns out to be a bad strategy. The main reason for this is that the numbers of EPs for different classes may not be balanced, which is a frequent scenario for applications where some classes may have more random (uniform) distributions of values and consequently fewer EPs. If a class $\mathcal{C}$ has many more EPs than another class $\mathcal{C}'$, then instances usually get higher scores for $\mathcal{C}$ than for $\mathcal{C}'$, even for training instances of class $\mathcal{C}'$. This indeed happens, for example in the rice-DNA dataset (see §6), which consists of a positive class and a negative class. The negative class contains mostly "random" instances, and the ratio of the number of EPs of the positive to that of the negative is 28:1 when the support threshold is 3% and the growth rate threshold is 2.

Our solution to this problem is to "normalize" the scores, by dividing them using a score at a fixed percentile for the training instances of each class. More specifically, a *base score* for each class $\mathcal{C}$, $base\_score(\mathcal{C})$, should be first found from the training instances of the class. The *normalized score* of an instance s for $\mathcal{C}$, $norm\_score(s, \mathcal{C})$, is defined as the ratio $score(s, \mathcal{C})/base\_score(\mathcal{C})$. (Observe that our use of the term "normalized" is a slight abuse, since our normalized scores may be > 1.) Instead of letting the class with the highest raw score win, we let the class with the largest normalized score win. (We break tie by letting the class with the largest population win.)

How do we determine the base scores? We can let $base\_score(\mathcal{C})$ be the median of the scores of the training instances class $\mathcal{C}$; that is, exactly 50% of the training instances of $\mathcal{C}$ have scores larger than or equal to $base\_score(\mathcal{C})$. We do not have to use 50%; in fact, other percentiles between 50%–85% give roughly similar results. The CAEP construction program should automatically choose a good percentile in this range, by testing the performance of the constructed classifier on the training instances. We do not want to use percentage on the two extreme ends (e.g. 3%), because the training instances usually contain some outliers, and if we use such a choice we let the outliers give too much influence.

Example 2. For a simple illustration of the decision process, assume there are 5 training instances from each of the positive (+ve) and negative (-ve) classes, and their scores are:

+ve training instances		*-ve training instances*	
score(s,+ve)	score(s,-ve)	score(s,+ve)	score(s,-ve)
18.44	0.31	4.89	5.51
16.65	0.39	8.37	5.47
15.76*	0.05	2.8	5.4*
15.28	0.21	9.93	4.97
14.52	0.41	10.31	4.8

The (median) base scores for the positive and negative classes are respectively 15.76 and 5.4. Given a test instance s (known to be from the negative class) with scores 7.07 and 4.82 for the positive and negative classes respectively, we have
norm_score(s,+ve)=7.07/15.76=0.45   norm_score(s,-ve)=4.82/5.4=0.89
s is thus labelled as negative. Observe that this decision is made even when s has a higher raw score for the positive class.

3.5 The entire process

The entire process for building and using CAEP is summarized below, assuming that the original dataset is partitioned according to the class labels.

```
CAEP (training datasets $\mathcal{D}_1, \ldots, \mathcal{D}_p$ for $p$ classes $\mathcal{C}_1, \cdots, \mathcal{C}_p$)
;; training phase
1) Mine the EP set $E_i$ from $\cup_{j=1}^{p} \mathcal{D}_j - \mathcal{D}_i$ to $\mathcal{D}_i$ for each $1 \leq i \leq p$;
   ;; A growth rate threshold is given, or set to a default e.g. 2
2) Optionally, reduce the number of EPs in each of $E_1, \cdots, E_p$;
3) Calculate the aggregate scores of all training instances for all classes;
4) Get the base scores $base\_score(\mathcal{C}_i)$ for each class $\mathcal{C}_i$;
;; testing phase
5) For each test instance $s$ do:
6)     Calculate aggregate and normalized scores of $s$ for each class $\mathcal{C}_i$;
7)     Assign to $s$ the class $\mathcal{C}_j$ for which $s$ has the largest normalized score.
```

4 Efficient Mining of EPs

For the discovery of EPs we will be using methods introduced in [4]. A key tool used by the efficient methods of [4] is that of borders, useful for the concise representation and efficient manipulation of large collections of itemsets.

Example 3. An example border is $<\mathcal{L} = \{\{22\}, \{57\}, \{61\}\}, \mathcal{R} = \{\{22, 34, 36, 57, 61, 81, 85, 88\}\}>$. The collection of itemsets represented by this border is $\{Y \mid \exists X \in \mathcal{L}, \exists Z \in \mathcal{R}$ such that $X \subseteq Y \subseteq Z\}$. Representative itemsets covered in the border include $\{22\}, \{22, 57\}, \{36, 57, 81, 88\}$. For interested readers, this is actually a border for the EPs from the edible to the poisonous class, of an encoding of the Mushroom dataset, at support threshold $\delta = 40\%$ in the poisonous and growth rate threshold $\rho = 2$.

To calculate the aggregate score contributed by all the EPs (meeting some thresholds) of a class $\mathcal{C}_i$, we need to (i) find the EPs of $\mathcal{C}_i$ and (ii) discover their supports and growth rates. We now list two possible methods:

The large-border based approach: Max-Miner [1] is first used to efficiently discover the border of the large itemsets from $\mathcal{D}_i$. (Such a border is called a large border, hence the word "large" in the title of this approach.) If the large itemsets represented by the border can be enumerated in memory, then with one more scan of $\mathcal{D}_i$ and $\mathcal{D}'_i$ we can get the supports and growth rates of the EPs of $\mathcal{C}_i$. If it can be applied, this approach can discover all EPs whose supports in $\mathcal{D}_i$ are larger than the given support threshold. However, because some larger borders may represent "exponentially" many candidate itemsets, only a small portion of these candidates can be held in memory; we need then to use the next approach.

The border differential based approach: We first use Max-Miner [1] to discover the two large borders of the large itemsets in $\mathcal{D}_i$ and the opponent $\mathcal{D}'_i$ having certain support thresholds. Then we use the *MBD-LLborder* (multiple-border differential) algorithm of [4] to find all the EP borders. Finally, we enumerate the EPs contained in the EP borders, and go through $\mathcal{D}_i$ and $\mathcal{D}'_i$ to check their supports and growth rates. With 13 EP borders of the Mushroom dataset for some support thresholds, using this approach we quickly found the supports and growth rates of 4692 EPs. Since *MBD-LLborder* only finds EPs whose supports in the second dataset are $\geq$ one support threshold and in the first dataset are $<$ another support threshold, we need to apply this method multiple times on multiple pairs [4] of large borders, or combine it with the previous method, to get the important EPs satisfying the given support and growth rate thresholds.

5 Reduction of EPs Used

Given a class $\mathcal{C}$, we would like to find as many EPs as possible to give good coverage of the training instances; at the same time, we prefer EPs that have relatively large supports and growth rates, as these characteristics correspond to larger coverage and stronger differentiating power. Very often, many of the EPs can be removed without loss of too much accuracy, by exploiting relationships between the EPs. Reduction can increase understandability of the classifier, and it may even increase predictive accuracy.

The reduction step is optional, and it should not be done if it leads to poor classification of the training instances. This is a training time decision.

Our method to reduce the number of EPs uses these factors: the absolute strength of EPs, the relationships between EPs, and the relative difference between their supports and growth rates. We measure the absolute strength of EPs using a new growth rate threshold ρ', which should be larger than the growth rate threshold ρ for the EPs. The main idea is to select the strong EPs and remove the weaker EPs which have strong close relatives. We will refer to the selected EPs as the essential EPs.

To reduce the set of EPs, we first sort the mined EPs into a list E, in decreasing order on $(growth\_rate, support)$. The set of essential EPs, *essE*, is initialized to contain the first EP in E. For each next EP e in E we do 1 and then 2:

1. For each EP x in *essE* such that $e \subset x$, replace x by e if 1.a or 1.b is true:
   1.a. $growth\_rate(e) \geq growth\_rate(x)$
   1.b. $supp(e) >> supp(x)$ and $growth\_rate(e) \geq \rho'$
2. Add e to *essE* if both 1.a and 1.b are false, and e is not a superset of any x in *essE*.

We select EPs this way because: When condition 1.a is true, e definitely covers more instances than x since $e \subset x$, and e has a stronger differentiating power than x because e has a higher growth rate. A typical situation captured by condition 1.b is when x is an EP with growth rate ∞ but a very small support, whereas e is an EP whose growth rate is less than that of x but e has a much larger support than x. In this case, we prefer to have e since it covers many more cases than x and since it has a relatively high differentiating power already due to its growth rate being larger than ρ'. To illustrate this point, consider these two EPs of the *Iris-versicolor* class from the Iris dataset [12]:

$e_1 = (\{1, 5, 11\}, 3\%, \infty) \qquad e_2 = (\{11\}, 100\%, 22.25)$

e_2 is clearly more useful than e_1 for classification, since it covers 32 times more instances and its associated odds, 95.7%, is also very near that of the other EP, e_1. In our experiments, for 1.b, we set the default value of ρ' to 20 and the default interpretation of the condition "$supp(e) >> supp(x)$" is $\frac{supp(e)}{supp(x)} \geq 30$. These parameters can be tuned based on coverage on training instances.

6 Selection of Thresholds and Base Scores

To build a classifier from a training dataset, we need to select two thresholds (one for support and one for support growth rate), and a base score for each class. The selection of the base scores was discussed in section 3.4.

The selection of these can be done *automatically* with the guidance of the *training* data, although not done in the experiments reported later: We start with some default thresholds and percentiles for the base scores. Then a classifier is built and its performance on the training instances is found. We then let the program try several alternatives and see if significant improvements are made. The best choice is then selected.

We observe from our experiments that the lower the support threshold δ, the higher predictive accuracy the classifier achieves; and for each support threshold, the higher the growth rate threshold, the higher predictive accuracy the classifier achieves. Once δ is lowered to 1%-3%, the classifier usually becomes stable in predictive accuracy. In our experiments, δ is usually between 1%–3%.

The growth rate threshold also has strong effect on the quality of the classifier produced. Our general principle is to (a) mine EPs with a small initial growth rate threshold such as 2, and (b) automatically select a larger final growth rate threshold guided by the coverage of selected EPs on the training instances. Generally, if the growth rate threshold is too high, the classifier would contain too few EPs and the classifier may have low accuracy because of poor coverage of the training instances. (Coverage of a set of EPs is measured by the number of zero scores it produces on the training instances: The fewer the number of zero scores the better the coverage.) On the other hand, if it can be done without lowering the coverage of training instances, raising the growth rate threshold would always results in a classifier with higher predictive accuracy. Our experiments show that with support threshold 1%–3%, the UCI datasets usually yield a huge number of EPs with growth rates from 1 to ∞ (Rice-DNA data – see the next section for its description – is an exception). The automatically chosen growth rate threshold is usually around 5.

7 Rice-DNA, Sensitivity and Precision

Our motivation for more accurate measure of classifiers comes from the rice-DNA dataset (available at http://adenine.krdl.org.sg:8080/limsoon/kozak/rice), containing rice-DNA Kozak sequences.

A genomic DNA is a string over the alphabet of {A, C, T, G}. The context surrounding the protein translation start site of a gene is called the Kozak sequence [7]. Correct identification of such start sites from a long genomic DNA sequence can save a lot of labor and money in identifying genes on that sequence. The start site is always the A-T-G sequence. The context surrounding the A-T-G has been the most important information to distinguish real start sites from non-start sites. A context is typically taken from up to 15 bases up stream of A-T-G to 10 bases down stream. So a Kozak sequence—for the purpose of this work—consists of 25 letters (excluding the A-T-G).

In the genomic DNA sequences, non-start sites (negative) overwhelm real start sites (positive) typically at a ratio of 24:1 or more. So a distinctive feature of the rice-DNA dataset is that the number of instances of the two datasets are very unbalanced. What makes the treatment of this dataset more difficult is that the number of positives, which is more important in reality, is far more the minority. With this very unbalanced dataset, even the *just-say-no* classifier, which always predicts an instance to be negative, will have an overall accuracy of $\frac{24}{25} = 96\%$. Unfortunately, the fact is that not a single real start site has been identified, which is against our aim of classification.

From this analysis we can see that in evaluating a classification method more meaningful measures than accuracy or error rate on the whole dataset are desirable. We will use a measure in terms of two parameters, namely *sensitivity* and *precision*, for each class, which have long been used in the signals world and in information retrieval [5].

Definition 3. *Given N instances whose class is known to be $\mathcal{C}$, for a classifier P, if P labels N' instances as of class $\mathcal{C}$, of which N_1 are indeed to be of class $\mathcal{C}$, then N_1/N is called P's* sensitivity *on $\mathcal{C}$, denoted* $\mathrm{sens}(\mathcal{C})$*, and N_1/N' is called P's* precision *on $\mathcal{C}$, denoted* $\mathrm{prec}(\mathcal{C})$*. For the special case of $N' = 0$, we define $sens(\mathcal{C}) = 0, prec(\mathcal{C}) = 0$.*

From the above definition we can see that *sensitivity* on a class $\mathcal{C}$ is an indication of the strength of a classifier on $\mathcal{C}$ and *precision* tells us how much confidence the classifier has on $\mathcal{C}$. Using the "minority" vs "majority" terminology, sensitivity for the minority class represents the percentage of the minorities caught by the classifier, precision for the minority class represents the percentage of those claimed to be of the minority class are indeed in the minority class. Clearly, these are a more accurate description of the performance of the classifier on class $\mathcal{C}$.

Example 4. Given the ratio of negative to positive instances of 24:1, let us examine the performance of the *just-say-no* classifier on the rice-DNA dataset:

$$\begin{array}{ll} sens(+ve) = 0 & prec(+ve) = 0 \\ sens(-ve) = \frac{24}{24} = 100\% & prec(-ve) = \frac{24}{25} = 96\% \end{array}$$

Since the sensitivity and precision on the positive class are more important for the rice-DNA dataset, we can conclude that the *just-say-no* classifier performs very badly on the rice-DNA dataset and we should seek better classifiers.

8 Experimental Results

We compare CAEP with five state-of-the-art classifiers: C4.5 (without discretization), CBA, Naive Bayes (NB) [3], TAN [6] which is a state-of-the-art extension of NB shown to outperform NB and many Bayesian Network approaches, and LB [13], a recently proposed classifier which uses long itemsets as the basis of classification.

Except for rice-DNA (see §6 for its description), the datasets we use are from the UCI machine learning repository [12]. Furthermore, we test CAEP on datasets with a large number of records and where each record is long, where no results of C4.5 or CBA are known to us.

By default, numerical attributes are discretized into a number (default=5) of bins using the "equal bin population" method[1]: For each attribute, we first obtain the count of occurrences of each value in the training instances; then each value v is mapped to an interval $[0.5(v + v_l), 0.5(v + v_r))$, where v_l is the largest among all values that are less than v and v_r is the smallest among all values that are larger than v; then iteratively we combine a consecutive pair of intervals which when combined give the smallest (in terms of number of occurrences in instances) new combined interval. For starred results discretization was done by

[1] We are extremely grateful to Tim Hansell for helping to implement this method.

using the entropy method [9], whose code is available from the MLC++ machine learning library [8].

The choice of parameters is as follows. The support threshold (for the target classes) is $\frac{5}{N}$ if $N < 50$, and it is $max(1\%, \frac{12}{N})$ otherwise, where N is the total number of tuples in the target class under consideration. The growth rate threshold is 5 by default, but it is 2 for rice since there are few EPs there. The base score is set to 85% by default, but it is set to 50% for vehicle.

Results of the other classifiers for the first nine datasets are quoted from [13], where ten-fold cross validation (CV-10) is used for all except waveform (where one-fold cross validation is used). Results of C4.5 and CBA on wine, ionosphere, iris and tic-tac-toe were quoted from [11] (CV-10). All our results are also obtained by CV-10.

Table 1 compares the overall predictive accuracy of CAEP, C4.5 (without discretization), CBA, NB, TAN, and LB. Dashes indicate that results are unavailable. (A recent test of C4.5 on rice shows that it has essentially 0% sensitivity and 0% precision – it is essentially the "just-say-negative" classifier which claims that everything is negative; observe that this gives an accuracy of about 96%.) Columns 2, 3 and 4 describe the datasets: the numbers of records, of attributes and of classes respectively. Rice-dna and mushroom are challenging datasets, having both a large number of instances and high dimensionality. Observe that datasets of 2, 3 and even more classes are included, and that CAEP performs equally well. Columns 5 to 10 give the average predictive accuracy of the classifiers.

Table 2 show the accuracy of CAEP, before and after reduction on several datasets. It can be seen that although the number of EPs has been dramatically reduced after the reduction process, there is no big loss in predictive accuracy and often there is an increase in accuracy.

It takes the CAEP classifier almost no time to decide the class of an instance; e.g. only 0.01 second for a classifier with 10000 EPs.

Table 3 gives a more detailed characterization of CAEP on the datasets; sensitivity and precision on each class, before and after reduction, are listed. It shows that CAEP generally has good sensitivity and precision on each class.

The positive sensitivity and precision on the rice-DNA dataset are better than the best neural network (NN) results known to us. Since we are not aware of any NN result on rice, we give an estimate using the NetStart NN [14] on the model dicot *Arabidopsis Thaliana*, which is also a higher plant and whose sequence complexity is comparable to that of rice. On the *Arabidopsis* genomic DNA from Entrez as of 1 April, 1999, the actual observed accuracy of NetStart is precision = 2.1% and sensitivity = 68%. (NetStart has a much higher accuracy on cDNAs; it achieved a 26% precision at 88% sensitivity, for a small *Arabidopsis* cDNA subset.)

9 Conclusion

In this paper we have proposed a classification method which is fundamentally different from previous classifiers, including C4.5 or CBA. It is based on a new kind of knowledge mined from the training dataset: *emerging patterns* (EPs).

Table 1: Accuracy Comparison

Dataset	#records	#attributes	#classes	NB	C4.5	TAN	CBA	LB	CAEP	#Competitors
australian	690	14	2	.8565	.8428	.8522	.8551	.8565	**.8621***	5
german	999	20	2	.741	.717	.727	.732	**.748**	.7250*	5
heart	270	13	2	.8222	.7669	.8333	.8187	.8222	**.8370**	5
pima	768	8	2	**.759**	.711	.7577	.7303	.7577	.75	5
vehicle	846	18	4	.6112	.6982	**.7092**	.6878	.688	.6632*	5
waveform	5000	21	3	.7851	.704	.7913	.7534	.7943	**.8468**	5
breast	699	10	2	.97	.9542	—	.9528	.9686	**.9728**	4
cleve	303	13	2	.8278	.7229	—	.7724	.8219	**.8325**	4
hepatitis	155	19	2	.8392	.8	—	.802	**.845**	.8303	4
wine	178	13	3	—	.927	—	.916	—	**.9711**	2
ionosphere	351	34	2	—	.9000	—	**.918**	—	.9004*	2
iris	150	4	3	—	**.953**	—	.929	—	.9467	2
tic-tac-toe	958	9	2	—	.994	—	**1.00**	—	.9906	2
rice-DNA	15764	25	2	—	—	—	.553	—	**.7087**	1
mushroom	8124	22	2	—	—	—	—	—	.9882	0

Table 1. The datasets are grouped according to the number of results available for the other classifiers. Best result for each dataset is in bold. CAEP outperforms the best of NB, C4.5, TAN, CBA, LB on 5 out of the 9 datasets where at least four results are available. CAEP improves the best previous results by 6.6% for Waveform and by 4.7% for Wine.

Table 2: Effect of EP Reduction

Dataset	#records	#attributes	#classes	CAEP Accuracy w/o red.	CAEP Accuracy red.	#EPs/class w/o red.	#EPs/class red.
mushroom	8124	22	2	98.82%	98.93%	823649	2738
rice-DNA	15760	28	2	70.87%	75.63%	30555	24449
tic-tac-toe	958	9	2	99.06%	96.87%	5707	682

Specifically, our classifier CAEP is based on aggregating the contribution of all EPs for differentiating instances of different classes. The aggregate score is proposed to quantify the overall contribution of all the EPs. The contribution of an EP takes into account both the support and growth rate of the EP. We then normalize the aggregate scores to classify instances, by dividing the scores by the base scores (chosen at a certain percentile of training instance scores) of the corresponding classes. The resulting classifier CAEP is in general more accurate than C4.5 and CBA, and is a lot more accurate than them over datasets where they do not have good performance. CAEP is equally accurate on all classes, and can be built efficiently from large, even high dimensional datasets. Observing that accuracy on the whole dataset is too coarse description of classifiers, we also used a more accurate measure, *sensitivity* and *precision*, to better characterize the performance of classifiers. CAEP is also very good under this measure.

Table 3: Precision and Sensitivity of CAEP

Dataset	class(instance dist.)	sensitivity		precision	
		w/o red.	red.	w/o red.	red.
mushroom	edible (52%)	99.43%	99.61%	98.32%	98.46%
	poisonous (48%)	98.16%	98.35%	99.38%	99.34%
rice-DNA	positive (4%)	77.01%	73.91%	9.58%	10.95%
	negative (96%)	70.62%	75.70%	98.71%	98.63%
tic-tac-toe	positive (65%)	99.52%	96.01%	99.07%	99.21%
	negative (35%)	98.19%	98.49%	99.13%	93.12%

References

1. Roberto J. Bayardo. Efficiently mining long patterns from databases. In *Proceedings of the 1998 ACM-SIGMOD International Conference on Management of Data*, pages 85–93, 1998.
2. L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth International Group, 1984.
3. R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* New York: John Wiley & Sons, 1973.
4. Guozhu Dong and Jinyan Li. Efficient mining of emerging patterns: Discovering trends and differences. To appear in *ACM KDD'99*, August 1999.
5. William B. Frakes and Ricardo Baeza-Yates. *Information Retrieval: Data Structures and Algorithms.* Prentice Hall, 1992.
6. N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. In *Machine Learning*, volume 29, pages 131–163, 1997.
7. Mary Kozak. An analysis of 5'-noncoding sequences from 699 vertebrate messenger RNAs. *Nucleic Acids Research*, 15:8125–8148, 1987.
8. R. Kohavi, G. John, R. Long, D. Manley, and K. Pfleger. MLC++: a machine learning library in C++. In *Tools with artificial intelligence*, pages 740 – 743, 1994.
9. R Kohavi, M Sahami. Error-based and Entropy-based Discretization of Continuous Features. In *KDD'96*, 1996.
10. J. Li, G. Dong, and K. Ramamohanarao. JEP-Classifier: Classification by Aggregating Jumping Emerging Patterns. Tech report, Univ of Melbourne, 1999.
11. B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proc. of 4th KDD*, 1998.
12. P.M. Murphy and D.W. Aha. UCI repository of machine learning database. In [http://www.cs.uci.edu/ mlearn/mlrepository.html].
13. Dimitris Meretakis and Beat Wuthrich. Extending naive bayes classifiers using long itemsets. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego*, pages 165–174, 1999.
14. A. G. Pedersen and H. Nielsen. Neural network prediction of translation initiation sites in eukaryotes: Perspectives for EST and genome analysis. *Intelligent Systems for Molecular Biology*, 5:226–233, 1997.
15. J R Quinlan. Induction of decision trees. In *Machine Learning*, Vol 1, pages 81–106, 1986.
16. J.R. Quinlan. *C4.5: program for machine learning.* Morgan Kaufmann, 1992.
17. R.E. Schapire. *The strength of weak learnability.* Machine Learning, 5(2):197-227, 1990.
18. Z. Zheng, Constructing X-of-N attributes for decision tree learning. *To appear in Machine Learning*, 1999.