From "Proceedings of the Thirteenth National Conference on Artificial Intelligence" (AAAI-96)

Learning Trees and Rules with Set-valued Features

William W. Cohen
AT&T Laboratories
600 Mountain Avenue Murray Hill, NJ 07974
wcohen@research.att.com

Abstract

In most learning systems examples are represented as fixed-length "feature vectors", the components of which are either real numbers or nominal values. We propose an extension of the feature-vector representation that allows the value of a feature to be a set of strings; for instance, to represent a small white and black dog with the nominal features `size` and `species` and the set-valued feature `color`, one might use a feature vector with `size=small`, `species=canis-familiaris` and `color={white,black}`. Since we make no assumptions about the number of possible set elements, this extension of the traditional feature-vector representation is closely connected to Blum's "infinite attribute" representation. We argue that many decision tree and rule learning algorithms can be easily extended to set-valued features. We also show by example that many real-world learning problems can be efficiently and naturally represented with set-valued features; in particular, text categorization problems and problems that arise in propositionalizing first-order representations lend themselves to set-valued features.

Introduction

The way in which training examples are represented is of critical importance to a concept learning system. In most implemented concept learning systems an example is represented by a fixed-length vector, the components of which are called *attributes* or *features*. Typically, each feature is either a real number or a member of a pre-enumerated set; the latter is sometimes called a *nominal* feature.

Clearly, fixed-length feature vectors are of limited expressive power. Because of this, numerous previous researchers have proposed learning methods that employ more expressive representations for examples. For instance, many "inductive logic programming" systems (Quinlan 1990b; Muggleton 1992) can be viewed as learning from examples that are represented as saturated Horn clauses (Buntine 1988); in a similar vein, KL-one type languages (Cohen and Hirsh 1994; Morik 1989) and conceptual dependency structures (Pazzani 1990) have also been used in learning.

While some successes have been recorded, these alternative representations have seen quite limited use. In light of this, it is perhaps worthwhile to review some of the practical advantages of the traditional feature-vector representation over more expressive representations. One advantage is *efficiency*. The rapid training time of many feature vector systems greatly facilitates systematic and extensive experimentation; feature-vector learners also have been used on very large sets of training examples (Catlett 1991; Cohen 1995a). A second important advantage is *simplicity*. The simplicity of the representation makes it easier to implement learning algorithms, and hence to refine and improve them. A simple representation also makes it easier for non-experts to prepare a set of training examples; for instance, a dataset can be prepared by someone with no background in logic programming or knowledge representation. The efficiency and simplicity of the feature-vector representation have doubtless contributed to the steady improvement in learning algorithms and methodology that has taken place over the last several years.

In this paper, we propose an alternative approach to generalizing the feature vector representation that largely preserves these practically important advantages. We propose that the feature-vector representation be extended to allow *set-valued features*, in addition to the usual nominal and continuous features. A set-valued feature is simply a feature whose value is a set of strings. For instance, to represent a small white and black dog with the nominal features `size` and `species` and the set-valued feature `color`, one might use a vector with `size=small`, `species=canis-familiaris` and `color={white,black}`.

Importantly, we will *not* assume that set elements (the colors in the example above) are taken from some small, pre-enumerated set, as is typically the case with nominal values. We will show that a connection can be established between set-valued features in this setting and the "infinite attribute model" (Blum 1992).

In the remainder of the paper, we first define set-valued features precisely. We then argue that many decision tree and rule learning algorithms can be easily extended to set-valued features, and that including set-valued features should not greatly increase the number of examples required to learn accurate concepts, relative to the traditional feature-vector representation.

We next present some examples of problems that naturally lend themselves to a set-valued representation, and argue further that for some of these problems, the use of set-valued representations is essential for reasons of efficiency. Finally we summarize our results and conclude.

Set-valued features

In the interests of clarity we will now define our proposed extension of the feature vector representation more precisely. A *domain* $D = \langle n, \vec{t}, \vec{u}, \vec{V}, Y \rangle$ consists of a *dimension* n, a *type vector* $\vec{t} = \langle t_1, \ldots, t_n \rangle$, a *name vector* $\vec{u} = \langle u_1, \ldots, u_n \rangle$, a *value vector* $\vec{V} = \langle V_1, \ldots, V_n \rangle$, and a *class set* $Y = \{y_1, \ldots, y_k\}$. Each component t_i of the type vector $\vec{t}$ must be one of the symbols `continuous`, `nominal`, or `set`. Each component u_i of the name vector $\vec{u}$ must be a string over the alphabet Σ. (Here Σ is a fixed alphabet, such as $\{0, 1\}$ or $\{a, \ldots, z\}$.) The class set Y and the components V_i of the value vector $\vec{V}$ are sets of strings over Σ. Intuitively, a "domain" formalizes the sort of information about a learning problem that is recorded by typical feature-vector learners such as C4.5 (Quinlan 1994)—with the addition of the new feature type `set`.

Using domains we can define the notion of a "legal example". A *legal example* of the domain $D = \langle n, \vec{t}, \vec{u}, \vec{v}, Y \rangle$ is a pair $\langle \vec{x}, y \rangle$ where $\vec{x}$ is a *legal instance* and $y \in Y$. A *legal instance* of the domain D is a vector $\vec{x} = \langle x_1, \ldots, x_n \rangle$ where for each component x_i of $\vec{x}$, (a) if $t_i =$ `continuous` then x_i is a real number; (b) if $t_i =$ `nominal` then $x_i \in V_i$ (otherwise, if $t_i \neq$ `nominal`, the value of V_i is irrelevant); (d) if $t_i =$ `set` then x_i is a set of strings over Σ, *i.e.*, $x_i = \{s_1, \ldots, s_w\}$.

Finally, we will define certain *primitive tests* on these instances. If u_i is a component of the name vector $\vec{u}$, r is a real number and s is a string, then the following are all *primitive tests* for the domain D: $u_i = s$ and $u_i \neq s$ for a nominal feature u_i; $u_i \leq r$ and $u_i \geq r$ for a continuous feature u_i; and $s \in u_i$ and $s \notin u_i$ for a set-valued feature u_i. The semantics of these primitive tests are all defined in the obvious way—for instance, if $u_3 =$ `color`, then "`puce` $\in$ `color`" denotes the set of legal instances $\vec{x} = \langle x_1, \ldots, x_n \rangle$ such that `puce` $\in x_3$.

Boolean combinations of primitive tests are also defined in the obvious way. We can now speak precisely of representations such as DNF, decision trees, or decision lists over set-valued representations.

Implementing Set-Valued Features

Let us now consider the problem of learning decision trees for the representation described above—that is, for feature vector examples that contain a mix of set-valued, nominal and continuous features. As a concrete case, we will consider how the ID3 algorithm (Quinlan 1990a) can be extended to allow internal nodes to be labeled with *element-of* tests of the form $s \in u_i$ and $s \notin u_i$ on set-valued features u_i, in addition to the usual tests on continuous or nominal features.

```
function MaxGainTest(S, i)
1   Visited := ∅;
2   TotalCount[+]:=0; TotalCount[−]:=0;
3   for each example ⟨x⃗, y⟩ in the sample S do
4       for each string s ∈ x_i do
5           Visited := Visited ∪{s};
6           ElemCount[s, y] := ElemCount[s, y]+1;
7       endfor
8       TotalCount[y] := TotalCount[y]+1;
9   endfor
10  BestEntropy = −1;
11  for each s ∈ Visited do
12      p := ElemCount[s,+]; n := ElemCount[s,−];
13      if (Entropy(p, n) > BestEntropy) then
14          BestTest := "s ∈ u_i";
15          BestEntropy := Entropy(p, n);
16      endif
17      p' := TotalCount[+] − ElemCount[s,+];
18      n' := TotalCount[−] − ElemCount[s,−];
19      if (Entropy(p', n') > BestEntropy) then
20          BestTest := "s ∉ u_i";
21          BestEntropy := Entropy(p', n');
22      endif
23      ElemCount[s,+] := 0;
24      ElemCount[s,−] := 0
25  endfor
26  return BestTest
```

Figure 1: Finding the best element-of test

To implement this extension of ID3, it is clearly necessary and sufficient to be able to find, for a given set-valued feature u_i and a given set of examples S, the element-of test $s \in u_i$ or $s \notin u_i$ that maximizes entropy on S. Figure 1 presents a function `MaxGainTest` that finds such a maximizing test. For simplicity, we have assumed that there are only two classes.

Lines 1-9 of the function loop over the examples in the sample, and record, for each string s that appears as an element of the i-th feature of an example, the number of times that s appears in a positive example, and the number of times that s appears in a negative example. These counts are stored in `ElemCount[`$s, +$`]` and `ElemCount[`$s, -$`]`. (These counters are assumed to be initialized to zero before the routine is called; they are reset to zero at line 20.) Additionally, a set `Visited` of all the elements s that appear in feature i of the sample is maintained, and the total number of positive and negative examples is recorded in `TotalCount[+]` and `TotalCount[−]`.

Lines 10-25 make use of these counts to find the best test. For a given set element s, the number of examples of class y covered by the test $s \in u_i$ is simply `ElemCount[`s, y`]`; similarly the number of examples of class y covered by the test $s \notin u_i$ is given by `TotalCount[`y`]-ElemCount[`s, y`]`. The maximal entropy test can thus be found by looping over the elements s in the set `Visited`, and computing the entropy of each test based on these formulae.

A few points regarding this procedure bear mention.

Efficiency. Defining the size of a sample in the natural way, it is straightforward to argue that if accessing `ElemCount` and the `Visited` set requires constant time,[1] then invoking `MaxGainTest` for *all* set-valued features of a sample only requires time *linear* in the total size of the sample.[2] Hence finding maximal-entropy element-of tests can be done extremely efficiently. Notice that this time bound is independent of the number of different strings appearing as set elements.

"Monotone" element-of tests. One could restrict this procedure to generate only set-valued tests of the form "$s \in u_i$" by simply removing lines 15-19. Henceforth, we will refer to these tests as *monotone element-of tests*. Theory, as well as experience on practical problems, indicates that this restriction may be useful.

Generality. This routine can be easily adapted to maximize a metric other than entropy, such as the GINI criteria (Brieman *et al.* 1984), information gain (Quinlan 1990b), predictive value (Apté *et al.* 1994), Bayes-Laplace corrected error (Clark and Niblett 1989), or LS-content (Ali and Pazzani 1993). In fact, any metric that depends only on the empirical performance of a condition on a sample can be used. *Hence it is possible to extend to set-valued features virtually any top-down algorithm for building decision trees, decision lists, or rule sets.*

A Theory Of Set-Valued Features

Given that it is possible to extend a learning system to set-valued features, the question remains, is it useful? It might be that few real-world problems can be naturally expressed with set-valued features. More subtly, it might be that learning systems that use set-valued features tend to produce hypotheses that generalize relatively poorly.

[1]In a batch setting, when all the string constants are known in advance, is it trivial to implement constant-time access procedures for `ElemCount` and `Visited`. One simple technique is to replace every occurrence of a string s in the dataset with a unique small integer, called the *index of s*. Then `ElemCount` can be a $r \times k$ matrix of integers, where r is the largest index and k is the number of classes. Similarly, `Visited` can be a single length-r array of flags (to record what has been previously stored in `Visited`) and another length-r array of indices.

[2]A brief argument: any invocation of `MaxGainTest`, the number of times line 5 is repeated is bounded by the total size of the i-th features of examples in the sample, and the number of iterations of the **for** loop at lines 10-21 is bounded by the size of `Visited`, which in turn is bounded by the number of repetitions of line 5.

The former question will be addressed later. In this section we will present some formal results that suggest that extending a boolean hypothesis space to include element-of tests on set-valued features should not substantially increase the number of examples needed to learn accurate concepts. In particular, we will relate the set-valued attribute model to Blum's (1992) "infinite attribute space" model, thus obtaining bounds on the sample complexity required to learn certain boolean combinations of element-of tests.

In the infinite attribute space model of learning, an large (possibly infinite) space of boolean attributes A is assumed. This means that an instance can no longer be represented as a vector of assignments to the attributes; instead, an instance is represented by a list of all the attributes in A that are true for that instance. The *size* of an instance is defined to be the number of attributes in this list.

One can represent an instance I in the infinite attribute model with a single set-valued feature `true_attribs`, whose value is the set of attributes true for I. If I' is the set-valued representation of I, then the element-of test "$a_j \in$ `true_attribs`" succeeds for I' exactly when the boolean attribute a_j is true for I, and the test "$a_j \notin$ `true_attribs`" succeeds for I' exactly when a_j is false for I.

Conversely, given an instance I represented by the n set-valued features $u_1, \ldots, u_n$, one can easily construct an equivalent instance in the infinite attribute model: for each set-valued feature u_i and each possible string $c \in \Sigma^*$, let the attribute $s\_in\_u_i$ be true precisely when the element-of test "$s \in u_i$" would succeed. This leads to the following observation.

Observation 1 *Let $D = \langle n, \vec{t}, \vec{u}, \vec{v}, Y = \{+,-\}\rangle$ be a domain containing only nominal and set-valued features, and let $\mathcal{L}$ be any language of boolean combinations of primitive tests on the features in $\vec{u}$.*

Then there exists a boolean language $\mathcal{L}'$ in the infinite attribute model, a one-to-one mapping f_I from legal instances of D to instances in the infinite attribute model, and a one-to-one mapping f_C from concepts in $\mathcal{L}$ to concepts in $\mathcal{L}'$ such that

$$\forall C \in \mathcal{L}, \quad (I \in C) \Leftrightarrow (f_I(I) \in f_C(C))$$

In other words, if one assumes there are only two classes and no continuous features, then every set-valued feature domain D and set-valued feature language $\mathcal{L}$ has an isomorphic formulation in the infinite attribute model.

This observation allows one to immediately map over results from the theory of infinite attributes, such as the following:

Corollary 2 *Let D be a two-class domain containing only set-valued features, but containing any number of* *these. Let n be an upper bound on the size of legal instances of D. Let $\mathcal{L}_k$ be the language of conjunctions of at most k element-of tests, let $\mathcal{M}_k$ be the language of conjunctions of at most k monotone element-of tests, and let VCdim$(*)$ denote the Vapnik-Chervonenkis (V-C) dimension of a language. Then*

- *VCdim$(\mathcal{L}_k) \leq (n+1)(k+1)$;*
- *VCdim$(\mathcal{M}_k) \leq n+1$, irrespective of k.*

Proof: Immediate consequence of the relationships between mistake bounds and VC-dimension established by Littlestone (1988) and Theorems 1 and 2 of Blum (1992). ■

Together with the known relationship between VC-dimension and sample complexity, these results give some insight into how many examples should be needed to learn using set-valued features. In the monotone case, conjunctions of set-valued element-of tests for instances of size n have the same VC-dimension as ordinary boolean conjunctions for instances of size n. In the non-monotone case, set-valued features are somewhat more expressive than non-monotone boolean features. This suggests that negative element-of tests should probably be used with some care; although they are computationally no harder to find than monotone element-of tests, they are an intrinsically more expressive representation (at least when large conjunctions are possible), and hence they may require more examples to learn accurately.

Using Corollary 2 and other general results, bounds on the V-C dimension of related languages can also easily be established. For example, it is known that if $\mathcal{L}$ is a language with V-C dimension d, then the language of ℓ-fold unions of concepts in $\mathcal{L}$ has V-C dimension of at most $2\ell d \log(e\ell)$ (Kearns and Vazarani 1994, p. 65). Applying this result to $\mathcal{L}_k$ immediately yields a polynomial upper bound on DNF over set-valued features, which includes as a subset decision trees over set-valued features.

Alternatives to set-valued features

In the preceding section we showed that if continuous attributes are disallowed then the set-valued feature model is equivalent to the infinite attribute model. Another consequence of this observation is that in a batch setting, in which all examples are known in advance, set-valued features can be replaced by ordinary boolean features: one simply constructs a boolean feature of the form $s\_in\_u_i$ for every string s and every set-valued feature u_i such that s appears in the i-th component of some example. Henceforth, we will call this the *characteristic vector* representation of a set-valued instance.

One drawback of the characteristic vector representation is that if there are m examples, and d is a bound on the total size of each set-valued instance, then the construction can generate md boolean features. This means that the size of the representation can grow in the worst case from $O(md)$ to $O(m^2d)$—*i.e.*, quadratically in the number of examples m.

We will see later that some natural applications do indeed show this quadratic growth. For even moderately large problems of this sort, it is impractical to use the characteristic vector representation if vectors are implemented naively (*i.e.*, as arrays of length n, where $n < md$ is the number of features). However, it may still be possible to use the characteristic vector representation in a learning system that implements vectors in some other fashion, perhaps by using a "sparse matrix" to encode a set of example vectors.

Hence, it is clear that there are (at least) two other ways in which we could have described the technical contributions of this paper: as a scheme for extending top-down decision trees and rule learning algorithms to the infinite attribute model; or as a specific data structure for top-down decision tree and rule learning algorithms to to be used in domains in which the feature vectors are sparse.

We elected to present our technical results in the model of set-valued features because this model enjoys, in our view, a number of conceptual and pedagogical advantages over the other models. Relative to the infinite-attribute model, set-valued features have an advantage in that they are a *strict generalization* of the traditional feature-vector representation; in particular, they allow ordinary continuous and nominal features to co-exist with "infinite attributes" in a natural way. Additionally, the nature of the generalization (adding a new kind of feature) makes it relatively easy to extend existing learning algorithms to set-valued features. We note that to our knowledge, the infinite attribute model has seldom been used in practice.

There are also certain conceptual advantages of the set-valued feature model over using a sparse implementation of the characteristic vector representation. For instance, the set-valued feature model lends itself naturally to cases in which some features require dense encoding and others require a sparse encoding. Also, the same learning system also be used without significant overhead on problems with either sparse or non-sparse feature vectors.

A more subtle advantage is that for set-valued features, the representation *as perceived by the users and designers of a learning system* closely parallels the actual implementation. This has certain advantages when selecting, designing, and implementing learning algorithms. For example, set-valued features share with traditional (non-sparse) feature vectors the property that the size of an example is closely related to the V-C dimension of the learning problem. This is not the case for a sparse feature vector, where the number of components in the vector that represents an example depends both on the example's size and on the size of a dataset. One can easily imagine a user naively associating the length of a feature vector with the difficulty of a learning problem–even though long feature vectors may be caused by either large amounts of data (which is of course helpful in learning) or by long documents (which is presumably *not* helpful in learning.)

Applications

In this section we will present some results obtained by using set-valued features to represent real-world problems. The learning system used in each case is a set-valued extension of the rule learning system RIPPER (Cohen 1995a), extended as suggested above.

To date we have discovered two broad classes of problems which appear to benefit from using a set-valued representation. The first class is learning problems derived by propositionalizing first-order learning problems. The second is the class of text categorization problems, *i.e.*, learning problems in which the instances to be classified are English documents.

First-order learning

A number of theoretical results have been presented which show that certain first-order languages can be converted to propositional form (Lavrač and Džeroski 1992; Džeroski *et al.* 1992; Cohen 1994). Further, at least one practical learning system (LINUS) has been built which learns first-order concepts by propositionalizing the examples, invoking a propositional learning system on the converted examples, and then translating the resulting propositional hypothesis back to a first-order form (Lavrač and Džeroski 1994).

There are several reasons why a LINUS-like system might be preferred to one that learns first-order concepts in a more direct fashion. One advantage is that it allows one to immediately make use of advances in propositional learning methods, without having to design and implement first-order versions of the new propositional algorithms. Another potential advantage is improved efficiency, since the possibly expensive process of first-order theorem-proving is used only in translation.

A disadvantage of LINUS-like learning systems is that some first-order languages, when propositionalized, generate an impractically large number of features. However, often only a few of these features are relevant to any particular example. In this case, using set-valued features to encode propositions can dramatically reduce storage space and CPU time.

Bias	m	RIPPER		Grendel2	
		Time	Error(%)	Time	Error(%)
2-local	100	0.4	2.3	10.3	2.8
	500	2.0	0.0	41.0	0.0
4-local	100	1.9	3.5	88.8	2.7
	500	9.1	0.0	376.0	0.0

Table 1: k-local bias: direct *vs.* set-valued feature implementations on Pazzani and Brunk's student loan prediction. The column labeled m lists the number of training examples. CPU times are on a Sun Sparcstation 20/60 with 96Mb of memory.

We will illustrate this with the problem of predicting when payment on a student loan is due (Pazzani and Brunk 1991). In Pazzani and Brunk's formulation of this problem, the examples are 1000 labeled facts of the form `no_payment_due`(p), where p is a constant symbol denoting a student. A set of *background predicates* such as `disabled`(p) and `enrolled`($p, school, units$) are also provided. The goal of learning is to find a logic program using these background predicates that is true only for the instances labeled "+".

Previous experiments (Cohen 1993) have shown that a first-order learning system that hypothesizes "k-local" programs performs quite well on this dataset. It is also a fact that any non-recursive logic program that is "k-local" can be emulated by a monotone DNF over a certain set of propositions (Cohen 1994). The set of propositions is typically large but polynomial in many parameters of the problem, including the number of background predicates and the number of examples.[3] For the student loan problem with $k = 2$, for instance, some examples of the propositions generated would be $p_{132}(A) \equiv$ true iff $\exists$`B :enlist(A,B)` $\wedge$ `peace_corps(B)` and $p_{39}(A) \equiv$ true iff $\exists$`B : longest_absence_from_school(A,B)`$\wedge$ `lt(B,4)`. Often, however, relatively few of these propositions are true for any given example. This suggests giving using a set-valued feature to encode, for a given example, the set of all true constructed propositions which are true of that example.

We propositionalized the student loan data in this way—using set-valued features to encode the propositions generated by the k-local conversion process—for various values of k. Propositions were limited to those that satisfied plausible typing and mode constraints. We then ran the set-valued version of RIPPER on this data, and compared to Grendel2 (Cohen 1993) configured so as to directly implement the k-local bias. Since there is no noise in the data, RIPPER's pruning algorithm was disabled; hence the learning system be-

[3] It is exponential only in k (the "locality" of clauses) and the arity of the background predicates.

ing investigated here is really a set-valued extension of propositional FOIL. Also, only monotone set-valued tests were allowed, since monotone DNF is enough to emulate the k-local bias. For each number of training examples m given, we report the average of 20 trials. (In each trial a randomly selected m examples were used for training, and the remainder were used for testing.)

The results are shown in Table 1. None of the differences in error rates are statistically significant; this is expected, since the learning algorithms are virtually identical. However, the set-valued RIPPER is substantially faster than the first-order system Grendel2. The speedup in learning time would more than justify the cost of converting to propositional form, if any moderately substantial cross-validation experiment were to be carried out;[4] for the larger problems even a single learning run is enough to justify the use of set-valued RIPPER. (Additionally, one would expect that RIPPER would show an improvement in error rate on a noisy dataset, since Grendel2 does not include any pruning mechanisms.)

In this case the number of propositional features can be bounded independently of the number of examples. However, other first-order learning systems such as FOIL (Quinlan 1990b) and Progol (Muggleton 1995) allow constant values to appear in learned clauses, where the constant values are derived from the actual training data. If such a first-order language were propositionalized, then this would certainly lead to a number of features linear in the number of examples, causing quadratic growth in the size of the propositionalized dataset.

Text categorization

Many tasks, such as e-mail filtering and document routing, require the ability to classify text into predefined categories. Because of this, learning how to classify documents is an important problem.

In most text categorization methods used in the information retrieval community, a document is treated as an unordered "bag of words"; typically a special-purpose representation is adopted to make this efficient. For shorter documents a "set of words" is a good approximation of this representation. This suggests representing documents with a single set-valued feature, the value of which is the set of all words appearing in the document.

Traditional feature-vector based symbolic learning methods such as decision tree and rule induction can be and have been applied to text categorization (Lewis and Ringuette 1994; Lewis and Catlett 1994; Apté *et al.* 1994; Cohen 1995b). A number of representations for symbolic learning methods have been explored, but generally speaking, features correspond to words or phrases. Since the number of distinct words that appear in a natural corpus is usually large, it is usually necessary for efficiency reasons to select a relatively small set of words to use in learning.

An advantage of the set-valued representation is that it allows learning methods to be applied *without* worrying about feature selection (at least for relatively short documents). We note that the feature selection process can be complex; for instance one set of authors (Apté *et al.* 1994) devoted four pages of a paper to explaining the feature selection process, as compared to five pages to explaining their rule induction program. It is also sometimes the case that the number of features must be limited for efficiency reasons to fewer than would be optimal. For instance, Lewis and Ringuette (1994) report a case in which the performance of a decision tree learning method continued to improve as the number of features was increased from 1 to 90; presumably on this problem still more features would lead to still better performance.

The following section describes an evaluation of the set-valued version of RIPPER on text categorization problems.

The text categorization problems The benchmark we will use is a corpus of AP newswire headlines, tagged as being relevant or irrelevant to topics like "federal budget" and "Neilsens ratings" (Lewis and Gale 1994; Lewis and Catlett 1994). The corpus contains 319,463 documents in the training set and 51,991 documents in the test set. The headlines are an average of nine words long, with a total vocabulary is 67,331 words. No preprocessing of the text was done, other than to convert all words to lower case and remove punctuation.

In applying symbolic learning system to this problem, it is natural to adopt a characteristic vector version of the set-of-words representation—*i.e.*, to construct for each word w one boolean feature which is true for a document d iff w appears in d. This representation is not practical, however, because of the size of the dataset: Lewis and Catlett (1994) estimated that storing all 319,463 training instances and all 67,331 possible word-features would require 40 gigabytes of storage.

However, the set-valued extension of RIPPER can be easily run on samples of this size. Table 2 summarizes monotone RIPPER's performance, averaged across nine of the ten categories, and compares this to a learning algorithm that uses a representation optimized for text—Rocchio's algorithm, which represents a document with term frequency/inverse document frequency weights (TF-IDF). The implementation used here fol-

[4] The time required to convert to propositional form is 35 seconds for $k = 2$ and 231 seconds for $k = 4$. A total of 139 propositions are generated for $k = 2$ and 880 for $k = 4$.

Domain	Rocchio #errors	recall	precis	RIPPER #errors	recall	precis	time
bonds	31.00	50.00	96.77	34.00	46.67	93.33	1582
boxoffice	26.00	52.38	78.57	20.00	64.29	84.38	2249
budget	170.00	35.53	61.95	159.00	32.99	70.65	2491
burma	46.00	55.91	91.23	33.00	69.89	92.86	2177
dukakis	107.00	0.00	100.00	112.00	17.76	44.19	3593
hostages	212.00	37.72	55.13	206.00	44.30	56.11	4795
ireland	106.00	32.48	58.46	97.00	27.35	72.73	1820
nielsens	49.00	52.87	85.19	35.00	72.41	85.14	10513
quayle	73.00	81.20	69.23	65.00	87.22	70.73	2416
average	91.11	44.23	77.39	84.56	51.43	74.46	3652.50

Table 2: RIPPER and Rocchio's algorithm on AP titles with full sample

Learner	#errors	recall	precision	$F_{\beta=1}$
Rocchio	91.11	44.23	77.39	0.52
Prob. class.				0.41
RIPPER				
w/ negation	86.00	60.12	72.26	0.64
RIPPER				
all words	84.56	51.43	74.46	0.59
10,000 words	85.11	51.61	73.62	0.59
5,000 words	85.22	50.95	73.84	0.59
1,000 words	85.56	49.64	74.17	0.58
500 words	86.67	50.72	72.51	0.58
10 words	87.78	52.80	72.07	0.59
50 words	91.78	44.39	73.17	0.52
10 words	98.56	35.12	72.06	0.41
5 words	109.33	17.94	85.61	0.23
1 word	118.22	0	100.00	0.00

Table 3: Effect of entropy-driven feature selection.

lows Ittner *et al.* (1995).[5] Although both algorithms are attempting to minimize errors on the test set, we also record the widely used measurements of recall and precision.[6] RIPPER achieves fewer errors than Rocchio on 7 of the 9 categories, and requires a reasonable amount of time (given the size of the training set.)

Table 3 gives some additional points of reference on this benchmark. All entries in the table are averages over all nine problems (equally weighted). So that we can compare earlier work, we also record the value of the F-measure (Van Rijsbergen 1979, pages 168–176) at $\beta = 1$. The F-measure is defined as $F_\beta \equiv \frac{(\beta^2+1) precision \cdot recall}{\beta^2 precision + recall}$ where β controls the importance given to precision relative to recall. A value of $\beta = 1$ corresponds to equal weighting of precision and recall, with higher scores indicating better performance. The first few rows of the table show the average performance of Rocchio's algorithm, a probabilistic classifier used by Lewis and Gale (1994), and non-monotone RIPPER (*i.e.*, RIPPER when tests of the form $e \notin S$ are allowed.)

So far, we have demonstrated that good performance can be obtained without using feature selection by using set-valued features. We will now make a stronger claim: that feature selection is actually harmful in this domain. The final rows of Table 3 show the performance of monotone RIPPER when feature selection is applied. We used the strategy employed by Lewis and Ringuette (1994) and also Apte *et al.* (1994) in a similar context: in each learning problem the mutual information of each word and the class was computed, and the k words that scored highest were retained as features. In our experiments, we implemented this by removing the low-information words from the sets that represent examples. Aside from efficiency issues, this is equivalent to using the k retained words as binary features; however, by using set-valued features we were able to explore a much wider range of values of k than would be otherwise be possible.

To summarize the results, although around 100 features does give reasonably good performance, more features always lead to better average performance (as measured by error rate). This result might be unexpected if one were to think in terms of the 66197-component characteristic vector that is used for these problems—one would think that feature selection would surely be beneficial in such a situation. However, the result is unsurprising in light of the formal results. Because the documents to be classified are short (aver-

[5] Very briefly, each document is represented as a (sparse) vector, the components of which correspond to the words that appear in the training corpus. For a document d, the value of the component for the word w_i depends on the frequency of w_i in d, the inverse frequency of w_i in the corpus, and the length of d. Learning is done by adding up the vectors corresponding to the positive examples of a class C and subtracting the vectors corresponding to the negative examples of C, yielding a "prototypical vector" for class C. Document vectors can then be ranked according to their distance to the prototype. A novel document will be classified as positive if this distance is less than some threshold t_C. In the experiments, t_C was chosen to minimize error on the training set.

[6] *Recall* is the fraction of the time that an actual positive example is predicted to be positive by the classifier, and *precision* is the fraction of the time that an example predicted to be positive is actually positive. We define the precision a classifier that never predicts positive to be 100%.

aging only nine words long) the VC-dimension of the hypothesis space is already quite small. Put another way, a powerful type of "feature selection" has already been performed, simply by restricting the classification problem from complete documents to the much shorter *headlines* of documents—as a headline is by design a concise and informative description of the contents of the document.

Other results Although space limitations preclude a detailed discussion, experiments have also been performed (Cohen and Singer 1996) with another widely-used benchmark, the Reuters-22173 dataset (Lewis 1992). Compared to the AP titles corpus, this corpus has fewer examples, more categories, and longer documents. The stories in the Reuters-22173 corpus average some 78 words in length, not including stopwords. The vocabulary size is roughly comparable, with 28,559 words appearing in the training corpus. Although the longer documents have a larger effective dimensionality, set-valued RIPPER without feature selection also seems to achieve good performance on this dataset. For instance, following the methodology of Apte *et al.*, RIPPER's "micro-averaged breakeven point" for this benchmark is 80.9%, slightly better than the best reported value of 80.5% for SWAP-1; following the methodology of Lewis and Ringuette (1994), a micro-averaged breakeven point of 71.9% was obtained, again bettering the best previously reported value of 67%. Set-valued RIPPER averages a little over 5 minutes of CPU time to learn from the 15,674-example training sets used by Lewis.

Conclusions

The feature vector representation traditionally used by machine learning systems enjoys the practically important advantages of efficiency and simplicity. In this paper we have explored several properties of *set-valued features*, an extension to the feature-vector representation that largely preserves these two advantages.

We showed that virtually all top-down algorithms for learning decision trees and rules can be easily extended to set-valued features. We also showed that set-valued features are closely related to a formal model that allows an unbounded number of boolean attributes. Using this connection and existing formal results, we argued that the sample complexity of set-valued feature learners should be comparable to that of traditional learners with comparably sized examples.

Finally, we demonstrated that two important classes of problems lend themselves naturally to set-valued features: problems derived by propositionalizing first-order representations, and text categorization problems. In each case the use of set-valued features leads to a reduction in memory usage that can be as great as quadratic. This dramatic reduction in memory makes it possible to apply set-valued symbolic learners to large datasets—ones that would require tens of thousands of features if traditional representations were used—without having to perform feature selection.

References

Kamal Ali and Machael Pazzani. HYDRA: A noise-tolerant relational concept learning algorithm. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, 1993.

Chidanand Apté, Fred Damerau, and Sholom M. Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 12(3):233–251, 1994.

Avrim Blum. Learning boolean functions in an infinite attribute space. *Machine Learning*, 9(4):373–386, 1992.

L. Brieman, J. H. Friedman, R.A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, Belmon, CA, 1984.

Wray Buntine. Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36(2):149–176, 1988.

Jason Catlett. Megainduction: a test flight. In *Proceedings of the Eighth International Workshop on Machine Learning*, Ithaca, New York, 1991. Morgan Kaufmann.

P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(1), 1989.

William W. Cohen and Haym Hirsh. Learning the CLASSIC description logic: Theoretical and experimental results. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR94)*. Morgan Kaufmann, 1994.

William W. Cohen and Yoram Singer. Context-sensitive learning methods for text categorization. To appear in SIGIR-96, 1996.

William W. Cohen. Rapid prototyping of ILP systems using explicit bias. In *Proceedings of the 1993 IJCAI Workshop on Inductive Logic Programming*, Chambery, France, 1993.

William W. Cohen. Pac-learning nondeterminate clauses. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Seattle, WA, 1994.

William W. Cohen. Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

William W. Cohen. Text categorization and relational learning. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

Sašo Džeroski, Stephen Muggleton, and Stuart Russell. Pac-learnability of determinate logic programs. In *Proceedings of the 1992 Workshop on Computational Learning Theory*, Pittsburgh, Pennsylvania, 1992.

David J. Ittner, David D. Lewis, and David D. Ahn. Text categorization of low quality images. In *Symposium on Document Analysis and Information Retrieval*, pages 301–315, Las Vegas, NV, 1995. ISRI; Univ. of Nevada, Las Vegas.

Michael Kearns and Umesh Vazarani. *An introduction to computational learning theory*. The MIT Press, Combridge, Massachusetts, 1994.

Nada Lavrač and Sašo Džeroski. Background knowledge and declarative bias in inductive concept learning. In K. P. Jantke, editor, *Analogical and Inductive Inference: International Workshop AII'92*. Springer Verlag, Daghstuhl Castle, Germany, 1992. Lectures in Artificial Intelligence Series #642.

Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chichester, England, 1994.

David Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Machine Learning: Proceedings of the Eleventh Annual Conference*, New Brunswick, New Jersey, 1994. Morgan Kaufmann.

David Lewis and William Gale. Training text classifiers by uncertainty sampling. In *Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1994.

David Lewis and Mark Ringuette. A comparison of two learning algorithms for text categorization. In *Symposium on Document Analysis and Information Retrieval*, Las Vegas, Nevada, 1994.

David Lewis. Representation and learning in information retrieval. Technical Report 91-93, Computer Science Dept., University of Massachusetts at Amherst, 1992. PhD Thesis.

Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4), 1988.

Katharina Morik. A bootstrapping approach to conceptual clustering. In *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, New York, 1989. Morgan Kaufmann.

Stephen H. Muggleton, editor. *Inductive Logic Programming*. Academic Press, 1992.

Stephen Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13(3,4):245–286, 1995.

Michael Pazzani and Clifford Brunk. Detecting and correcting errors in rule-based expert systems: an integration of empirical and explanation-based learning. *Knowledge Acquisition*, 3:157–173, 1991.

Michael Pazzani. *Creating a Memory of Causal Relationships*. Lawrence Erlbaum, 1990.

J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1), 1990.

J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3), 1990.

J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1994.

C. J. Van Rijsbergen. *Information Retrieval*. Butterworth, London, second edition, 1979.