Learning Multiple Concepts Efficiently: A Subset Learning Approach[1]

David Haines
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

Coins Technical Report 90-17
March 1, 1990

Abstract

This paper identifies two classes of problems that naturally give rise to learning situations whose instances are instances of several concepts simultaneously. It then analyzes two models of *subset learning:* an approach that allows more efficient concept learning in these situations than naively creating a number of single concept learning problems using all available features. Empirical results show that a system based on the additive subset learning model can learn concepts with generalization nearly as good as the naive learning method and at a much lower cost.

[1]This work has been supported by a grant from the Office of Naval Research through the University Research Initiative Program under contract number N00014-86-K-0764. It has benefited by discussions with Sharad Saxena, Richard Yee, Jamie Callan, Paul Utgoff, David Lewis, Tom Fawcett, Margie Connell, Carla Brodley and Jeff Clouse.

1 Introduction

Much of machine learning has been concerned with learning to recognize instances of single concepts. One common assumption has been that the given set of instances can be classified with disjoint labels; any one instance will have only one classification. Realistically, however, most entities have multiple, conjoint, classifications. While a *dog* can't also be a *cat* it can also be a mammal, a 4-legged animal, a house pet, a farm animal and many more things. Disjoint classification is appropriate for discriminating between instances but a single instance can provide information for many different concepts at the same time — information that should not be wasted.

The problem of learning these multiple classifications has not often been addressed in the machine learning literature. There has been some compatible work. Fisher's COBWEB [Fisher, 1987] system could potentially do multiple classification in the same manner it does single classification; attaching each classification as an additional feature and using the classification(s) of the best matches to the tested instance as predictions. Michalski [Michalski, 1983] points out the importance of being able to do multiple classification in medical diagnosis where a patient may have multiple diseases. His STAR methodology is capable of giving multiple classifications by treating multiple classification problems as a collection of single classification problems. In neither of these cases was there an attempt to try to exploit the characteristics of a multiple classification domain to do learning efficiently. Ben-Bassat [1980] has developed a Bayesian approach to the multiple classification problem. This current work differs by attempting to do multiple classification without requiring the calculation of probabilities.

One reason for the emphasis on single classification learning is that it isn't immediately clear under what circumstances does one actually want to use multiple classifications of an instance. Are there natural problems where we would both like to use machine learning methods and where we have conjoint classifications? Another reason for the emphasis is that there seems to be an obvious solution to the multiple classification problem — treat multiple classification as a collection of single classification problems and learn each concept as if it was the only concept.

This paper addresses both of these points. It points out situations where multiple classifications will naturally and usefully occur. It also shows that, in some circumstances, there is a more efficient method of learning that takes advantage of the fact of multiple classifications.

2 Why do Multiple Classification?

There are two areas where one can reasonably expect to want to learn a number of concepts from the same set of instances. In some domains there is a hierarchy of more and less specific concepts. These concepts will all share features (because they are in the same domain) but will be recognized from different subsets of features (because they have different extensions). One such domain is character recognition, where a letter can be decomposed into the combination of strokes that makes it up. Another type of domain is one where

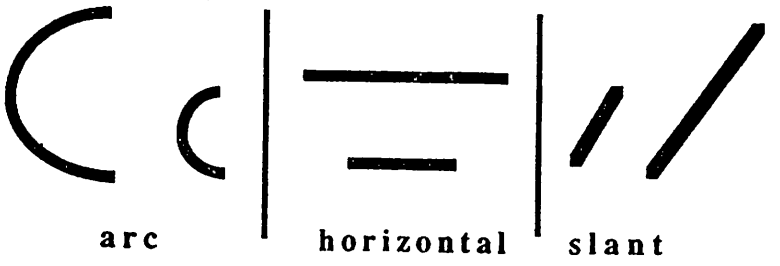

Figure 1: *The Stroke Concepts for the Character Recognition Task. Each concept will recognize a small and large version of its stroke and the absence of the stroke.*

objects are large and complex and so may be of interest in a variety of ways. These objects can contain much information and may provoke many questions. Document classification is an example of this kind of domain. Different people would want to look at the same document for a number of different reasons. If a system is to facilitate document retrieval it must be able to accommodate these multiple views of single documents.

2.1 Hierarchical Domains

Consider a character recognition task where one needs to classify some inputs according to the letters that they represent. One approach is to learn a separate concept for each letter in the alphabet; memorize each of the letters separately. Another approach is to decompose the learning task by decomposing the original concepts into subconcepts[1] that are smaller and therefore easier to learn. In the case of character recognition one would decompose each letter into a collection of strokes (see figure 1). The definition of a letter becomes a record of the strokes, and their locations, in the letter. The learning task would be learning the subconcepts defining each stroke.

The advantages of a decomposition approach are threefold.

- **more instances** — Each instance will composed of several subconcepts. Therefore we get an increase in the number of instances for each subconcept. Each letter is an instance for several strokes. The system therefore will require fewer instances because each one can be used for more than one subconcept.

- **more generality** — If there is a decomposition into fewer subconcepts than top-level concepts then the subconcepts will be more general that the original concept. When learning a number of sub-concepts the system will be able to transfer the knowledge it gained from one instance of a top-level concept to the same subconcept used in another top-level concept. For instance the only instance for the letter **R** is **R**. But there are a lot of letters with examples of a vertical long stroke (**B**,**E**,**F**), or short slant strokes (**K**,**Q**) or small arcs (**B**,**P**,**J**). By the time we get to the letters late in the learning sequence we are likely to know a lot about the subconcepts that

[1]This decomposition step can be seen as either producing subconcepts that are easier to learn or intermediate features that simplify learning the original concept. Since the machine learning literature usually takes "feature" as information provided to the system and "concept" as a classifier to be learned the intermediate level concepts are appropriately called "subconcepts." I'll restrict the use of "feature" to refer to the easily evaluated information provided with instances.

make it up. This cross-concept transfer can decrease the effort required to learn the full set of concepts.

- **less complexity** — Since each of the original, top-level, concepts is composed of several subconcepts we can expect each subconcept to be less complex than the entire top-level concept. Some defining parts of each subconcept must be independent or these subconcepts couldn't be simultaneously represented in a single high level concept. In the character recognition domain the subconcepts for the strokes need not contain any information about location or associated strokes. However the full definition of a letter must note all that information to be complete.

2.2 Multiple Interest Domains

An example of a domain with complex objects and many possible ways of classifying instances is document retrieval[2]. A text document is very complex, containing much information. There is not a clear understanding of how to mechanically extract much of that information. Even so there is some information we can extract. There are classification schemes for documents that attempt to make retrieval easier by providing at least some indication of the areas that are relevant to the document.

The learning task is to build classifiers for each topic that would indicate if an unseen document is relevant to a particular topic. Examples are provided by having experts classify documents as to their relevance to predetermined topics. Each document is relevant to more than one topic: Consider this paper — it is marked as relevant to *machine learning, feature selection,* and *inductive learning.* Inducing rules for automating the classification of documents would save a lot of time and work.

2.3 Implications

One price for the hierarchical approach is that we have to give the learning system more than just instances and the original classes. When decomposing a concept we need to show how to do the decomposition. The information to extract out the strokes from the letter is given — not learned. If this information is not available then a decomposition approach will not be applicable. Often this information is available. For instance experts can often decompose problems into smaller pieces that could be learned more easily. This approach could automate the lower level of knowledge acquisition.

One natural situation for multiple classification then is when a concept can be decomposed by either theory or observation, and the decomposed pieces of the concept can be learned separately. Another is when an instance may be used in different ways in different situations; when no one concept provides an adequate summary of the instances.

[2]See [van Rijsbergen, 1979] for an introduction.

3 Learning Efficiently

The second point of this paper is that multiple classifications can be learned efficiently. In both the hierarchical and the multiple interest problems each individual concept will require only a subset of the features available in the domain.

In the case of hierarchical concepts each different subconcept is likely to require a few features of its own to allow it to be expressed and discriminated. The higher level concepts in a hierarchy will cover a large number of specific instances. There are, for instance, many more vertebrates than mammals. To adequately describe all vertebrates requires enough features to distinguish snakes and squirrels, not just dogs and cats. Given a hierarchy we need a feature language big enough to describe and discriminate all the instances covered by the high level concepts. Not all of these features will be important to every instance.

In the multiple interest problems the concepts are independent. The concepts may overlap but it is very unlikely that all these concepts will require all of the features available from the document. For example, in computer science, papers on operating systems are unlikely to be concerned with concept learning and vice versa. The broader the range of instances that the documents cover the more likely it is that any particular document classification will require only a small portion of the available features.

Additionally some of the multiple classification problems come in domains where the features can be automatically generated. For these problems it is unlikely that there could be a careful selection of exactly the appropriate features needed for good learning.

Therefore multiple-classification domains will tend to have many features: but we can expect the number of features required for most of the *individual* concepts to be less than the total number of features required for the whole domain. Since the number of features required to represent the subconcepts is less than that required to represent all instances it should be less costly to learn a concept if we don't use unnecessary features. Ideally the system could do *subset learning:* learning with only an appropriate subset of the total set of features.

This leaves us with a feature selection problem — how can the system find a useful set? Under what conditions will it be cheaper to go through a feature selection process and then learn with this subset that to simply learn each concept separately using all the features available?

4 Subset Learning

There are two ways to model a combined feature selection and learning process. One is an *additive* model where there is a single feature selection process and then a single learning step. The other is a *multiplicitive* model where there are a number of iterations of feature selection and learning that attempt to converge on the proper set.

The following analysis of these two models find conditions where learning with a small number of features is advantageous. Because they involve order (O) calculations neither analysis will give us exact numeric predictions but they indicate if and when an approach can be appropriate.

For the following discussion assume the following definitions:

- k is the cost of selecting a set of features.
- n is the number of instances provided.
- f is the total number of features provided.
- fl is the size of a subset of the features with l ($0 \leq l \leq 1$) as a multiplier that defines the size of the subset.
- $C(n, f)$ is the cost of learning a concept given n instances and f features.

The cost of the obvious approach to learning in a multiple-classification situation would be to learn each concept separately using all the features — giving a cost for each concept of $C(n, f)$. The cost of simply selecting all of the features is 0.

4.1 The Additive Model

In this model there are 2 sequential steps to learning. In the first step a set of features is selected producing a subset of the features of size fl. In the second step learning is done with these features. The total cost of learning a concept then is $k + C(n, fl)$. Subset learning will be useful only if it is less expensive than learning with all the features: therefore subset learning will be useful only when

$$C(n, f) > k + C(n, fl)$$

or

$$C(n, f) - C(n, fl) > k.$$

To examine this further we need to select a particular learning method. In this paper ID3[Quinlan, 1986] is used as the learning algorithm. ID3 has a worst case complexity of $O(nf^2)$[Utgoff, 1989]. Substituting this in for $C(n, f)$ the above condition (up to constants) becomes

$$nf^2 - n(fl)^2 > k_1.$$

Simplified this is

$$nf^2(1 - l^2) > k_1.$$

The important point is that the contribution of l is squared in its effect. When l is 0.5 (when the number of features required is only half the number provided) the value of k_1 must be no more than 0.75 the value of nf^2 in order for this to be a useful alternative. When l is 0.25 (when the number of features required is a quarter of the total provided) the value of k_1 can be as high as 0.93 of the cost of nf^2. This indicates that as the concept size decreases we can accept larger and larger costs for selecting the features. It is worth a lot of work to select the right few features for learning. An experimental evaluation of this approach is reported later in the paper.

4.2 The Multiplicative Model

Another approach to finding the features would be to intermix feature selection and learning. In other words we could select some features, learn a concept using them, then use the results of that learning to help select other features. Pagallo and Haussler [1989] have used this approach in the Fringe system. Their goal is different however. They use multiple iterations to suggest new combined features for building more compact decision trees. The approach here is to find a method for building concepts more quickly, not for refining their structure.

One method of selecting the features is to use the performance of the learned concept to assess the value of the features used for learning. If a concept performs well then we can believe that it uses appropriate features. If it performs poorly then we can believe that other features should be selected. In this case the factor k is the combined cost of allocating features, the cost of evaluating a concept, and the number of repetitions of the application of the learning method in order to find good concepts. The cost for learning a concept then is $kC(n, fl)$.

We can assess when the multiplicative method will be cheaper than learning with the full set of features by looking at the relation between the costs of these two methods. In other words we want to know under what conditions

$$\frac{C(n,f)}{kC(n,fl)} > 1.$$

Equivalently we see that the method that learning using the multiplicative model is better when

$$\frac{C(n,f)}{C(n,fl)} > k.$$

To analyze further we need to know more about the cost of the particular learning methods that we care to use. As with the additive model this analysis proceeds using ID3 as the learning method. The equation above becomes

$$\frac{nf^2}{n(fl)^2} > k_2$$

or

$$\frac{1}{l^2} > k_2.$$

This again demonstrates we can tolerate costly feature selection mechanisms (k) as long the number of features required to learn the concept is small. Experimental examination of this approach will be left to a later paper.

5 The Experiments

The additive approach was tested in two domains. The first was character recognition, the second was document retrieval. In both cases subset learning was compared with the same programs and same data run using all the features available.

5.1 Character Recognition

The characters tested were selected from the English capital letters (A through K) and encoded into a 9 by 9 bit array. Each of the bits was then treated as an attribute with the values T or F. Each letter was decomposed into the stroke features shown in figure 1. The information on the location and identity of strokes in a letter was provided to the system. When instances were provided to ID3 for each of the 3 stroke subconcepts (arc, slant, horizontal) the bitmaps were appropriately rotated and translated to locate the relevant strokes in the same region of the array. The letters might, for instance, be flipped so that the \ stroke could be learned as a / stroke. Each stroke was an instance for each subconcept. If the transformed bitmap contained a stroke for that subconcept it was a positive instance, otherwise it was a negative instance.

The performance of the system was tested by recording what letters it would (and would not) recognize. It clearly should recognize the letters that it had trained on. Equally important it should *not* incorrectly recognize letters or meaningless stroke combinations. To verify this we tested that each real letter wouldn't be recognized as some other letter and that no *false letters* would be recognized. False letters were constructed by combining the strokes that define a letter from one letter with the bit array from another. The justification for this is that no two different letters should have exactly the same set of strokes — so that a bit pattern for one letter should never be recognized using the strokes for another.

The feature selection method was random and used no domain specific information. Each stroke concept was given 27 randomly selected features. These features were extracted from each instance and presented to ID3.

The time to learn each subconcept was recorded. The percentage of correct identifications was only recorded for the letters, not for each of the stroke subconcepts.

The table in table 1 and graph in figure 2 summarizes the learning times for the character recognition problem. The "complexity" column in the table is the number of test nodes in the decision tree generated using all feature values. This was taken as a measure of how much information would be required to correctly represent the concept. For both learning conditions the values reported are the average of 10 runs. For each of the three subconcepts the time required to learn using a subset of the features is less than 70% of the time required to learn with the full set of features. The accuracy of classification only decreases by 1.57% (from 97.52% using all features to 95.95% using a subset of the features). This subset learning approach leads to nearly as good performance at much less expense than using the full set of features.

5.2 Document Retrieval

The problem addressed here was the classification of documents as belonging to particular interest categories. The approach was to take the stemmed content words that appeared in the title and abstract of a database of documents[3] and use their presence or absence as

[3]The database was the CACM collection available from the Machine Learning database archive at the University of California, Irvine.

concept	complexity	feature selection		% difference
		subset	full	
slant	3	2.28	8.98	80.85
arc	6	13.55	57.14	76.29
horizontal	7	80.15	295.71	72.25

Table 1: *Time, in CPU seconds on an Explorer II, to learn concepts for the character recognition domain.*

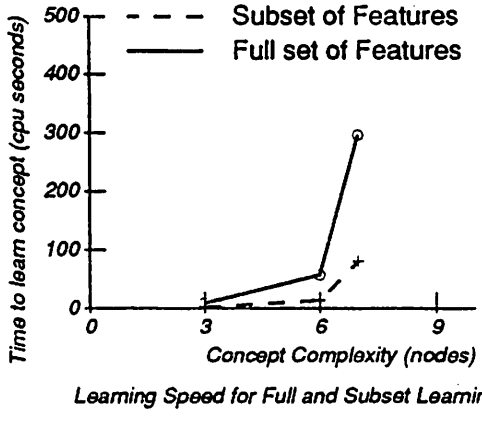

Figure 2: *Performance in the Character Recognition task*

a binary feature. This type of data has some interesting characteristics. First, since each word is potentially a feature, there are a huge number of features. In fact there were many more features available ($\approx$ 6000) than instances ($\approx$ 1500). Second most of the features would appear in only a few documents. There would be large number of features that wouldn't be relevant to general classifiers.

Space restrictions forced us to use as features only words that appeared in between 5 and 1000 documents (1522 words). 500 documents were randomly chosen to be the training set. A different 500 documents were chosen to be the test set.

The feature selection method had three phases. A portion of the features were chosen at random. Other features were added on the basis of domain information: All words that appeared in between 100 and 1000 documents (62 words) were added. This subset of features was then tested to see if allowed a consistent labeling of the instances; the features in the subset must be sufficient to prevent any two instances with the same values for all the selected features in the subset from having different classifications. If the labeling wasn't consistent then more high frequency words were added (the lower limit on the frequency of documents a word could appear in was lowered by a factor of 0.8) and additional randomly chosen features were added. This cycle continued until there was a consistent labeling of all the training instances.

Tables 5.2 and 5.2 summarize the learning time and classification accuracy for each of the eight concepts learned. All figures are the average of 10 runs. The same data are

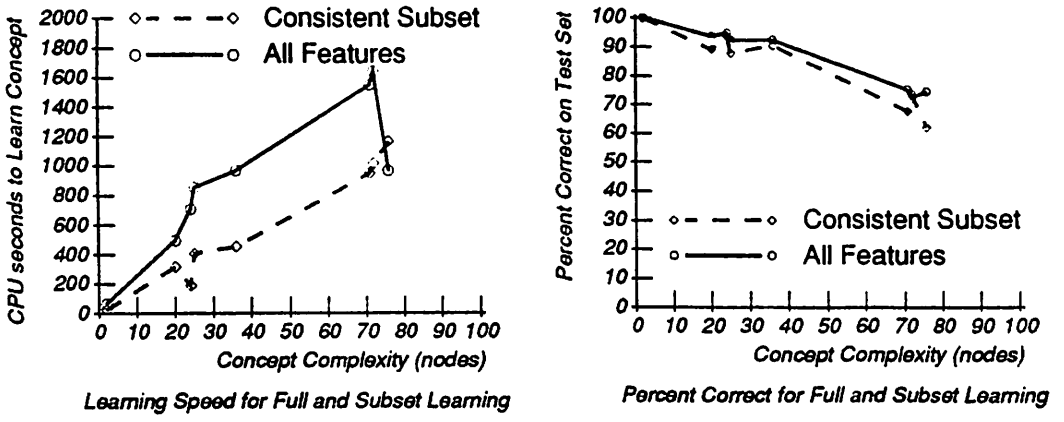

Figure 3: *Performance in the Document Classification task*

concept	complexity	feature selection		% difference
		subset	full	
c7	2	23.61	66.41	64.45
c1	20	316.14	497.53	36.46
c8	24	187.98	710.49	47.08
c2	25	408.47	855.73	52.27
c6	36	453.89	969.46	53.18
c5	71	947.53	1551.54	38.93
c4	72	1015.08	1645.38	38.31
c3	76	1164.84	964.96	-20.71

Table 2: *Time, in CPU seconds on an Explorer II, to learn concepts for the document classification domain.*

concept	complexity	feature selection		% difference
		subset	full	
c7	2	99.80	100.00	0.20
c1	20	89.13	93.74	4.61
c8	24	93.73	94.48	0.75
c2	25	87.77	92.27	4.50
c6	36	90.40	92.27	1.87
c5	71	67.75	75.21	7.46
c4	72	74.07	72.27	-1.80
c3	76	62.05	74.60	12.55

Table 3: *Percent correct on novel test set for the document classification domain.*

presented graphically in figure 3. The average decrease in learning time using the subset method is 38.75%. The average decrease in classification accuracy is 3.77%. The results are similar for each concept except for C3. C3 is learned more quickly and much more accurately when all the features are used. We don't have a full explanation for this but this is the result expected for a concept that had some non-redundent and important features. If a concept could not be learned well without those particular features and there is not sufficient domain information to identify them then learning with all features would be more efficient that repeatedly trying to build a consistent subset of features using random selection. However even including C3 the *overall* result is the same as for the character recognition domain: subset learning gives us quicker learning than with the full set of features and gives nearly as good classification.

6 Discussion and Future Work

The empirical results are quite strong: the additive approach to subset learning is providing nearly equal accuracy at much less cost than the simpler approach of learning the concepts with every available feature. It isn't surprising that this subset learning approach is appropriate in some domains. However is surprising that feature selection for the subset appears to be very easy and that the speed up that comes with a subset leads to little degradation is the quality of the concepts learned.

This makes the additive method attractive for problems with large feature spaces. Of course empirical exploration of the multiplicitive model for subset learning has not be done and should be tested. It is most likely to be useful in domains where we have enough knowledge about evaluation to let the results of evaluation of a concept guide further feature selection.

The question deserving most study however is: *why is feature selection so easy?* Features selection should be a difficult process[Kittler, 1986]. Is it a characteristic of the learning method that makes feature selection easy? Is it some characteristic of these domains?

It might be that the learning method (ID3) is not very sensitive to the exact set of features selected for learning. If this is true then knowing why this was so would be very important. Knowing why should give us a deeper understanding of how the learning method is actually working.

If the ease of feature selection is a characteristic of the domains chosen then we need to find the aspects of the domains that lead to such good performance. The arguments advanced in section 4 on the size of the feature space don't give predictions as to how easy finding a good subset of features should be, only that learning the concept should be easier if we do find the right subset. One possible explanation for the success of this procedure in these domains is that when there are multiple concepts and abundant features many of the features are likely to be either irrelevant or redundant for any one concept. Irrelevant features will not be helpful additions to the set of features for learning. If there is redundancy then it doesn't matter which exact set of features are selected, making it easier to find a useful feature. Since both of these domains use automatically generated features

they have many features and, as this set of features that has *not* been carefully hand-crafted, it is unlikely that all of them must be essential to the concepts of interest. These arguments suggest that the subset learning would an appropriate technique for domains that have a far larger number of features than the typical domain used in machine learning. This, in turn, might make feature generation an easier task. Less attention would need to be paid to producing a small number of features and more reliance could be put on allowing an inductive component to choose among many possible features.

References

[Ben-Bassat, 1980] Moshe Ben-Bassat. Multimembership and multiperspective classification: Introduction, applications, and a bayesian model. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-10(6):331–336, 1980.

[Fisher, 1987] Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, pages 139–172, 1987.

[Kittler, 1986] J. Kittler. Feature selection and extraction. In Tzan Y. Young and King-Sun Fu, editors, *Handbook of Pattern Recognition and Image Processing*, chapter 3. Academic Press, 1986.

[Michalski, 1983] Ryszard S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, pages 111–161, 1983.

[Pagallo and Haussler, 1989] Giulia Pagallo and David Haussler. Two algorithms that learn dnf by discovering relevant features. In *Proceedings of the Sixth International Workshop on Machine Learning*. Morgan Kaufmann Publishers, Inc., 1989.

[Quinlan, 1986] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Utgoff, 1989] Paul E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.

[van Rijsbergen, 1979] C. J. van Rijsbergen. *Information Retrieval.* Butterworths, Boston, 1979.