From: AAAI-98 Proceedings. Copyright © 1998, AAAI (www.aaai.org). All rights reserved.

Feature Generation for Sequence Categorization

Daniel Kudenko and Haym Hirsh
lastname@cs.rutgers.edu
Department of Computer Science
Rutgers University
Piscataway, NJ 08855

Abstract

The problem of sequence categorization is to generalize from a corpus of labeled sequences procedures for accurately labeling future unlabeled sequences. The choice of representation of sequences can have a major impact on this task, and in the absence of background knowledge a good representation is often not known and straightforward representations are often far from optimal. We propose a feature generation method (called FGEN) that creates Boolean features that check for the presence or absence of heuristically selected collections of subsequences. We show empirically that the representation computed by FGEN improves the accuracy of two commonly used learning systems (C4.5 and Ripper) when the new features are added to existing representations of sequence data. We show the superiority of FGEN across a range of tasks selected from three domains: DNA sequences, Unix command sequences, and English text.

Introduction

The *sequence categorization problem* is to take a collection of labeled sequences of variable length and form a procedure for accurately assigning labels in the future to otherwise unlabeled sequences. For example, in biological sequence recognition, when given a collection of nucleotide (or amino acid) sequences that are marked with class labels based on a structural or functional property, the goal is to find a hypothesis that enables a computer to classify future unlabeled sequences. Another common sequence categorization problem is e-mail filtering. Given a collection of e-mail texts with topic labels, the inductive learner has to generate a classifier which is able to assign a topic label to future incoming e-mail automatically.

Our goal is to develop a general method for sequence categorization tasks that works across many domains and does not need specialized knowledge for each domain. In particular, we focus on the class of inductive learning methods that apply when data can be described as *feature vectors*, i.e., as values assigned to a fixed set of attributes describing each object. Doing so requires representing sequences as feature vectors, and there are numerous ways to do this. For example, attributes could take the form of "The item in position i in the sequence", i.e., each position in the sequence gives a new attribute, whose value is the item (e.g., character) at that position. Or attributes could take the form of "The subsequence S occurs somewhere in the sequence", i.e., each such attribute tests for the presence or absence of a particular subsequence in the given sequence. This, of course, requires deciding what subsequences should be used in such features. For example, one could choose to create a feature for *every* possible subsequence up to a certain length k (i.e., make a feature for all n-grams for n ranging from 1 to k).

Unfortunately, such straightforward, albeit general, representations need not perform well. For example, Hirsh and Noordewier (Hirsh & Noordewier 1994) have shown that the choice of features for DNA sequence categorization problems can have a substantial impact on the accuracy of feature-based learners. Their features, based on domain knowledge, performed dramatically better than a straightforward positional encoding of data commonly used on this problem.

This paper proposes a method, called FGEN, to generate new feature-based sequence representations automatically from a collection of training data sequences. Each new feature is a kind of macro-feature that tests for the number of occurrences of each of a collection of subsequences within a sequence of interest. Since many learning methods involve forms of greedy search, such multi-sequence features may be selected for the hypothesis by such methods even when the individual sequences within them might not be selected by the learner when used in isolation, thereby leading to improved results in learning.

We evaluated FGEN representations on a wide range of sequence categorization tasks: English text, DNA sequences, and Unix command sequences. The evaluation was done by comparing the accuracy of C4.5 (Quinlan 1994) and Ripper (Cohen 1995) using straightforward sequence representations to the ac-

We thank William Cohen and our colleagues at Rutgers for many helpful discussions. The first author was partially supported by DIMACS, a cooperative project of Rutgers University, Princeton University, AT&T Labs, Bellcore, and Bell Labs.
Copyright ©1998, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

curacy of those learners using FGEN's features appended to these representations. Our results show that FGEN's features often improved the accuracy of the respective learning algorithms, and in those cases when they didn't, had only modest impact.

In the next section we describe straightforward approaches to representing sequences as feature vectors and the internal cross-validation approach to optimize the choice of representation. Afterwards we present the feature language of FGEN and the feature generation algorithm. Finally, test domains and the test strategy are described and the test results are shown.

Baseline Sequence Representations

There are many ways to formulate a sequence classification problem as one to which feature-based learners can be applied. This section presents the two general approaches to representing sequences as feature vectors that we use as baselines in this work. Note that although in some domains, notably text, the sequences can be broken up into well-defined tokens (e.g., words), each of which can serve as a feature, we do not consider such representations here due to our interest in more general approaches to sequence categorization that apply even when there are no a priori well-defined (and domain-specific!) tokenizations of sequences. Similarly, we do not consider numerical features that measure, for example, the proportion of sequence location occupied by a particular item (e.g., letter), out of a belief that, although general, the range of problems to which such representations would be relevant are fairly limited.

For notation, we write a vector of n features as an n-tuple, where each element is of the form (*feature-name feature-value*). A vector representation with 3 features F1, F2, and F3 having values V1, V2, and V3 respectively would therefore be described as `((F1 V1) (F2 V2) (F3 V3))`.

Positional Representations

In the positional representation of sequences each feature denotes a certain position of the sequence. For example, if feature Pi denotes the feature corresponding to position i in a sequence, the sequence ABBB would be represented `((P1 A) (P2 B) (P3 B) (P4 B))`, i.e., that there is an A in position 1, a B in position 2, etc. This is the simple positional representation for sequences that we will be using in the experiments that we discuss later in this paper.

To represent a collection of sequences, each sequence would be re-encoded as a vector of position/value pairs. However, although this is a fairly general way to represent sequences, it only works as described when all the data sequences are of the same length, since feature-based learners require all data to be described using identical sets of attributes. In cases where sequences can vary in length, this representation can still be used, although less elegantly, by "padding" all shorter sequences with repetitions of some "filler" character so that the same-vector-length requirement of this representation is satisfied.

Subsequence Representations

The preceding positional representation has the limitation that information concerning the adjacency of elements of a sequence are much more difficult to identify and represent. Consider, for example, what it would take to say that there is an A immediately adjacent to a B in the above positional representation when sequences are of length 1000.

An alternative approach to representing sequences considers the presence or number of occurrences of various subsequences in a sequence. For example, a simple representation would be to generate a feature for each n-gram that can occur in a sequence. Thus, if using a 2-gram representation and sequences are over the three-letter alphabet {A, B, C}, the sequence ABBB would represented as: `((AA 0) (AB 1) (AC 0) (BA 0) (BB 2) (BC 0) (CA 0) (CB 0) (CC 0))`. This representation says that the 2-gram AB occurs once and BB occurs twice in the sequence ABBB while all other 2-grams do not. We also consider a closely related representation, in which each potential subsequence is a Boolean variable rather than a count — the feature is true if the subsequences it represents are present in a given sequence. We call the above two representations the subsequence frequency and presence representations, respectively.

In the results to be described shortly, we use as features the set of all n-grams that occur in any sequence in the data (there is no need to enumerate all n-grams, since only those occurring somewhere in the data would ever be used), for n ranging from 1 to some value k. Although in some cases this is a general approach to representing sequences, one of its major limits is precisely when positional representations would excel — when the exact location of some sequence element is crucial for the classification task.

The FGEN Representation Formalism

FGEN takes a collection of sequence data and creates a set of new Boolean features out of the data. The idea is that each such feature tests that, for each of a set of subsequences, the number of times it occurs in a sequence is at least some minimum count. For instance, an FGEN feature could specify that it is true for a sequence if and only if it has at least two occurrences of the subsequence AC and at least three occurrences of the subsequence CB. One intuition behind considering such features is that they are *macro*-features, i.e., they represent information concerning more than a single subsequence. This combination can contain information that stays hidden from a learning algorithm that works in a greedy fashion.

More formally, each FGEN feature can be written as a function from the set of all sequences to $\{T, F\}$. The function denoted by a feature F can be

```
/* main function that computes the set of FGEN features.
   Ti union Pi are training examples of class i.
   Pi is used as a holdout set for evaluation purposes
   (usually 1/3 of the training set).  */
GenerateFeatures(T1,T2,P1,P2) {
  Seeds = T1;   /* set of potential seeds */
  Features = {};
  while (Seeds not empty) {
    Let s be the sequence in Seeds which starts the generation of
       the feature that subsumes most sequences in T1;
    F = ComputeFeature(s,T1,T2,P1,P2);  /* let F be such a feature */
    Features = Features union {F};
    C = set of sequences in T1 that are subsumed by F;
    Seeds = Seeds - C;   /* remove all elements of C from Seeds */
  }
  return Features;
}

/* compute a feature corresponding to a cluster built around seed s */
ComputeFeature(s,T1,T2,P1,P2) {
  F = feature denoting the minimum frequency restrictions on all
      subsequences up to length 8 of s;
  OldValue = GetValue(F,P1,P2); /* compute heuristic that evaluates F */
  repeat
    Let s' be the sequence that is ''most similar'' to F;
    Remove s' from T1;
    FG = least generalization of F that subsumes s';
    NewValue = GetValue(FG,P1,P2);
    if (NewValue >= OldValue) then F = FG;
  until NewValue < OldValue;  /* Hillclimbing on heuristic evaluation of F */
  return F;
}


/* heuristic evaluation function for features:
   proportional classification accuracy on the holdout set*/
GetValue(F,P1,P2) {
  p = number of sequences in P1 that are subsumed by F;
  n = number of sequences in P2 that are subsumed by F;
  P = number of sequences in P1;
  N = number of sequences in P2;
  return ((p*N/P)+(N-n))/(2*N);
}
```

Figure 1: FGEN feature building algorithm

written as: $F(S) = s_1^{n_1}(S) \wedge s_2^{n_2}(S) \wedge \ldots \wedge s_k^{n_k}(S)$ where $s_i^{n_i}$ is a boolean function and $s_i^{n_i}(S) = True$ for a sequence S if the sequence s_i occurs at least n_i times in S. ($s_i^{n_i}$ is called a minimum frequency restriction of F.) For example, if $F = AC^2 \wedge CB^3$ then $F(ACBBACBCB) = T$ and $F(ACBAC) = F$.

We say that a feature F subsumes a sequence S, if $F(S) = T$, i.e. S is compatible with all subsequence frequency restrictions of F. Furthermore, a feature F_1 is more general than a feature F_2, if the set of sequences subsumed by F_1 is a superset of the set of sequences subsumed by F_2. In other words, F_1 contains less restrictions than F_2.

FGEN features can be efficiently implemented using a data structure based on suffix trees, which has been described in detail in (Hirsh & Kudenko 1997).

The FGEN algorithm

The input to FGEN are pre-classified training examples. The output is a set of features as described above, which can be used to map each training and test example into a Boolean feature vector. FGEN generates a set of features for each class separately, and returns the union of the sets. Although the current implementation of FGEN only works on two-class learning problems, it can be easily extended for multi-class settings.

The basic idea behind the FGEN feature generation algorithm is that a representation which focuses on combinations of subsequences that occur mostly in training examples of one class and not in the examples of the other class will improve classification accuracy.

The FGEN algorithm consists out of two main parts: feature building (shown in Figure 1) and generalization (Figure 2).

```
prune(F,P1,P2) {
  OldValue = GetValue(F,P1,P2);
  repeat
    Decrement a frequency restriction in F;
    Let FP be the feature resulting from the
       above operation for which GetValue(FP,P1,P2)
       is maximized;
    NewValue = GetValue(FP,P1,P2);
    if (NewValue >= OldValue) then F = FP;
  until NewValue < OldValue;
  return F;
}
```

Figure 2: Feature Generalization Algorithm

Features are built incrementally, starting with a feature that corresponds to just a single example of the target class (the seed around which a feature is generated). This seed feature denotes the minimum frequency restrictions on all subsequences of the seed sequence.[1] For example, if the seed sequence s is equal to ABB, then the corresponding seed feature is defined as follows: $F_s = A^1 \wedge B^2 \wedge AB^1 \wedge BB^1 \wedge ABB^1$.

A feature is built step by step by making it more general to subsume at least one additional sequence from the target class in each iteration. The additional sequence is one that requires the fewest changes to the current feature to subsume the additional sequence. In other words, the additional sequence is "most similar" to the feature.

After each iteration the new feature is heuristically evaluated. As a heuristic we use classification accuracy of the feature on a holdout set, which is one third of the training set (since FGEN features are Boolean they can be interpreted as two-class classifiers). The generalization steps are continued, until the resulting feature has a lower heuristic value than the feature in the previous step.

This feature generation algorithm creates features that tend to over-fit the training data. In order to solve this problem each computed feature is subsequently generalized after it is created. The algorithm is shown in Figure 2. FGEN's generalization step removes subsequence restrictions one by one from a feature. This is done by decrementing a minimum frequency restriction on some subsequence. Once a minimum frequency restriction has reached 0, it can be removed completely, since it does not contain any information.[2]

[1] In order to limit the complexity we arbitrarily chose to restrict the length of such subsequences to 8. This choice seems reasonable, because we never encountered a domain where subsequences of length greater than 5 had a considerable impact on classification accuracy. Furthermore, a high value of n implies a sparse distribution of n-grams.

[2] In the actual implementation of FGEN we restrict the decrementing of frequency restrictions to single characters for efficiency reasons. The changes in frequency restrictions are propagated to all subsequences that start with the given character. Furthermore, we allow the frequency restriction of certain subsequences to be set to 0 in one step. These feature simplification operations can be computed efficiently on the data structure for FGEN features, which is based on suffix trees (Hirsh & Kudenko 1997).

The generalization is performed in a greedy hill-climbing fashion by trying to increase the heuristic value of the pruned feature as much as possible and stopping the generalization as soon as a local maximum is reached (i.e. the heuristic value of the current feature is lower than the value of the feature of the last iteration).

Evaluation

This section describes the empirical tests we performed to show that the FGEN feature representation improves accuracy of feature-based learners. We chose C4.5 and Ripper as two common and representative systems. C4.5 is a widely used decision-tree learning algorithm, in which trees are built in a divide-and-conquer fashion using an information-gain heuristic. Ripper, on the other hand, forms rules in a method similar to that used in inductive logic programming, and has been successfully used in many applications, showing a comparably high accuracy.

There are certain complexity limits to the straightforward baseline representations that were described previously in this paper. When the alphabet is large, the representations based on the presence or frequency of subsequences can result in a very large number of features.[3] In such cases C4.5 required a reduction in the number of features. For representations based on the presence of subsequences we chose to retain the 1000 most frequent features (i.e., subsequences), and did not use representations based on subsequence frequencies for datasets with large alphabets (English texts and Unix command sequences).

On the other hand, Ripper allows features to be set-valued, which can be exploited to simplify the baseline representation based on the presence of subsequences. In Ripper a sequence S can be represented as a single set-valued feature F that contains all subsequences of S of length less than or equal to some n. The sequence ABBB can therefore be represented in the form ((F {A,B,AB,BB})) for n = 2. This made it possible to use the full set of features.

In our empirical tests we use as a baseline whatever representation appeared to be best on the training data. The representations that were tried were the pure positional representation and subsequence presence and frequency representations for subsequences up a certain length n for $n \in \{2, 3, 4, 5\}$.[4] For each run the learner used 10-fold cross-validation on the training set to find the "best" baseline representation amongst those described above (i.e., the one yielding the lowest cross-validation error rate on the training set). Although this method will not always choose the optimal

[3] The number of features is in the order of $|\Sigma|^n$, where Σ is the alphabet and n is the length of the subsequences underlying the representation.

[4] For tractability reasons we restricted n to 5 or less.

representation, it will yield the better performance on average, as has been shown in (Schaffer 1993). Afterwards, the training and test data is represented in the winning format and the error rate with this representation is reported.

The second strategy works like the first with the exception that FGEN features are appended to the winning representation and the error rate for this extended representation is reported. Since FGEN's features are created independently of which representation is selected by the internal representation-selecting cross-validation runs, it is computed separately, and then added on once the baseline representation selection is made.

Datasets

In order to show the applicability of FGEN across a wide range of sequence categorization tasks we tested it on three domains: English text (3 sub-domains), DNA sequences (3 sub-domains), and Unix Command sequences (1 sub-domain). We describe each domain in more detail.

DNA Sequences

Promoter: This dataset contains 600 DNA sequences of length 100; 300 of them classified as Promoters and 300 as Non-Promoters. All DNA sequences are over a four letter alphabet.

A/B-DNA: This dataset contains three types of DNA: sequences in A, B, and Z confirmation. Since there were only a small number of Z-DNA sequences, the two learning tasks were to learn to distinguish A-DNA from the union of B-DNA and Z-DNA data, and B-DNA from the union of A-DNA and Z-DNA data. This dataset contains 138 relatively short sequences (between 4 and 12 characters per sequence), most of which are A-DNA and B-DNA in equal proportions.

Human/Phage DNA: We generated this dataset by extracting 300 Human and 300 Phage DNA sequences of length 100 from GenBank.[5] The goal is to learn to distinguish Human from Phage DNA.

Unix command sequences

The goal of these datasets is to identify a user through Unix command sequences. The datasets contain 225 non-overlapping Unix command sequences of length 20 from three users (75 sequences from user 1, 101 from user 2, and 49 from user 3). The alphabet size (i.e., the number of different commands used in the sequences) varied from 40 to 90.[6] One of the users forms the target class (a different one for each dataset) while the other two form the background class.

English Text

Newsgroup topic spotting: We generated three datasets by extracting postings from three different newsgroups: comp.os.ms-windows, rec.games.backgammon, and sci.classics. Each dataset has one of the three topics as the target class and the other two as the background class. The size of the datasets ranges from 350 to 550, with 30% to 50% of the data being texts from the target class. For complexity reasons the length of individual postings has been restricted to the first 500 characters.

Email topic spotting: These four datasets are extracted from four different personal mail folders. The target class are talk announcement messages.[7] The datasets contain between 150 and 600 examples, 10–50% of which are from the target class. For complexity reasons the length of individual postings has been restricted to the first 500 characters.

Book passage categorization: This dataset contains 600 character sequences of length 100 from two books (300 from each).[8] The goal of this dataset is to learn to identify the book given the character sequence.

Test Results

Table 1 shows the error rates on each single dataset of the three domains. The winning error rate is shown in bold. All error rates have been obtained by performing 10-fold cross-validation. (Note that this is separate from the cross-validation being performed to select the best representation for learning — each of the 10-fold cross-validation runs itself involves a 10-fold cross-validation run on the respective training set to select that run's representation.)

As can be seen in the table FGEN features help in many cases and hurt only in very few (11 wins, 2 ties, 2 losses with Ripper, and 8 wins, 2 ties, 5 losses with C4.5). Overall, the wins outweigh the losses considerably. Furthermore, the losses are relatively small in most cases, while some of the wins are dramatic. The results reflect our intuition that the learning systems will not use FGEN features in their hypothesis when they would increase the error rate considerably.

Note that the number of features generated by FGEN never exceeded 20 on any dataset. This is remarkable, given the large size of the space of potential features.

Related Work

Previous work in sequence categorization includes Hidden Markov Models (Rabiner 1989), learning of finite automata, and entropy-based induction (LLLAMA)

[5]The choice of sequence length 100 was arbitrary, as is the case for the book sequence dataset and the Unix command sequence dataset.

[6]The superset of this data has been collected for experiments in Unix command prediction (Davison & Hirsh 1997).

[7]This data was previously used in a previous study on e-mail filtering (Cohen 1996; Cohen & Kudenko 1997).

[8]The book texts have been taken from the Calgary Corpus, a collection of files for compression benchmarks.

Table 1: Error Rates on Test Domains

Dataset	Ripper	Ripper+FGEN	C4.5	C4.5+FGEN
Promoter	18.9	**16.0**	20.8	**16.7**
A-DNA	20.0	**16.7**	13.3	**11.1**
B-DNA	**7.1**	**7.1**	**3.8**	**3.8**
Hum/Pha DNA	11.2	**5.8**	8.3	**6.0**
Unix User 1	13.3	**7.1**	**6.7**	9.5
Unix User 2	6.3	**5.5**	9.7	**7.4**
Unix User 3	4.7	**1.7**	3.4	**0.9**
News 1	13.8	**9.4**	13.4	**9.6**
News 2	**8.8**	10.3	9.7	**9.1**
News 3	17.8	**14.4**	**12.4**	13.5
Talks 1	**10.1**	10.2	**15.4**	15.7
Talks 2	5.3	**5.1**	**4.1**	4.2
Talks 3	**4.0**	**4.0**	**4.7**	**4.7**
Talks 4	7.8	**3.9**	**5.1**	5.5
Books	23.8	**20.3**	16.2	**12.7**

(Loewenstern, Berman, & Hirsh 1998). Hidden Markov Models require an initial model, which is usually based on domain knowledge. LLLAMA has been designed mainly with biological sequences in mind. This is contrary to the philosophy of FGEN which has been designed to be independent of domain knowledge and to be applied across a wide range of domains. Learning of finite automata has been mainly applied to datasets that have been generated by finite automata in the first place.

Feature-based learners have been applied to sequence categorization using either one of the baseline representations presented in this paper, or using features that have been suggested by domain experts (Hirsh & Noordewier 1994; Salzberg 1995; Norton 1994).

An overview of the area of constructive induction, which deals with the automatic generation of representations for learning, is presented in (Saitta 1996). We are not aware of any constructive induction research for sequence categorization.

Final Remarks

We presented a feature generation method for sequence categorization that creates boolean features based on minimum subsequence frequencies. We have shown empirically that the representation computed by this method improves learning accuracy across a wide range of domains, compared to straightforward representations.

It is interesting to note that FGEN follows a bottom-up approach, starting with a specific feature and then generalizing it to compute the result. This is in contrast to C4.5 and Ripper, which both use a top-down approach. They start with an empty hypothesis and make it more specific. Using the FGEN representation results in a hybrid approach, which might be one of the reasons for the success of FGEN. It would be interesting to further investigate the reasons for this success. We plan to examine the properties of datasets in which the FGEN representation succeeds and in which it fails.

References

Cohen, W., and Kudenko, D. 1997. Transferring and retraining learned information filters. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence.*

Cohen, W. 1995. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning.*

Cohen, W. 1996. Learning rules that classify e-mail. In *Proceedings of the 1996 AAAI Spring Symposium on Machine Learning in Information Access.*

Davison, B., and Hirsh, H. 1997. Toward an adaptive command line interface. In *Seventh International Conference on Human-Computer Interaction.*

Hirsh, H., and Kudenko, D. 1997. Representing sequences in description logics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence.*

Hirsh, H., and Noordewier, M. 1994. Using background knowledge to improve inductive learning of DNA sequences. In *Proceedings IEEE Conference on AI for Applications.*

Loewenstern, D.; Berman, H.; and Hirsh, H. 1998. Maximum a posteriori classification of DNA structure from sequence information. In *Proceedings of PSB-98.*

Norton, S. W. 1994. Learning to recognize promoter sequences in E.coli. In *Proceedings of the Eleventh National Conference on Artificial Intelligence.*

Quinlan, J. R. 1994. *C4.5: programs for machine learning.* San Mateo, CA: Morgan Kaufmann.

Rabiner, L. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings IEEE.*

Saitta, L. 1996. Representation change in machine learning. *AI Communications* 9:14–20.

Salzberg, S. 1995. Locating protein coding regions in human DNA using a decision tree algorithm. *Journal of Computational Biology* 2(3).

Schaffer, C. 1993. Selecting a classifier method by cross-validation. *Machine Learning* 13.