Very Predictive Ngrams for Space-Limited Probabilistic Models

Paul R. Cohen and Charles A. Sutton

Department of Computer Science,
University of Massachusetts, Amherst, MA 01002
{cohen,casutton}@cs.umass.edu

Abstract. In sequential prediction tasks, one repeatedly tries to predict the next element in a sequence. A classical way to solve these problems is to fit an order-n Markov model to the data, but fixed-order models are often bigger than they need to be. In a fixed-order model, all predictors are of length n, even if a shorter predictor would work just as well. We present a greedy algorithm, VPR, for finding variable-length predictive rules. Although VPR is not optimal, we show that on English text, it performs similarly to fixed-order models but uses fewer parameters.

1 Introduction

In *sequential prediction* tasks, one repeatedly tries to predict the next element in a sequence. More precisely, given the first $i \geq 0$ elements in a sequence, one tries to predict the next element x_{i+1}. After making a prediction, one is shown x_{i+1} and challenged to predict x_{i+2}. In general, it is very difficult to predict the first element x_0 in a sequence (imagine trying to predict the first letter in this paper). Often it is easier to predict x_1 having seen x_0, easier still to predict x_2, and so on. For instance, given the sequence $w\,a$ most English speakers will predict the next letter is s; indeed, in English text s is about seven times more likely to follow $w\,a$ than y, the next most likely letter. One might form a *prediction rule* $[wa \rightarrow s]$ that says, "When I am trying to predict the next letter in a corpus of English text and I have just observed $w\,a$, I should predict s." However, if there are N_{wa} occurrences of $w\,a$ in a corpus, then one must expect at least $N_{wa}/8$ incorrect predictions, because roughly one-eighth of the occurrences of $w\,a$ are followed by y, not s. In English, the sequence $l\,w\,a$ is almost always followed by y, so one can add the prediction rule $[lwa \rightarrow y]$ to "mop up" most of the prediction errors committed by the rule $[wa \rightarrow s]$.

This paper is concerned with finding the best k rules, the rules that together reduce prediction errors the most. We present a greedy algorithm, VPR, for finding sets of k very predictive rules. Although VPR finds suboptimal rules in some cases, it performs well on English-language text. If our algorithm always found the most predictive rules we would call it MPR; instead, we call it VPR, for Very Predictive Rules.

One motivation for VPR is from previous work on ngram models and Markov chains in which ngrams are of fixed length. ngrams are sequences $x_0 x_1 \ldots x_n$

that may be used as prediction rules of the form $x_0x_1 \ldots x_{n-1} \rightarrow x_n$, whereas Markov chains are conditional probability distributions which have the form $\Pr(X_n = x|X_0...X_{n-1})$. Typically, n is fixed, but fixed-order models are often bigger than they need to be. Consider the case of the letter q in English. q is invariably followed by u, so the prediction rule, $[q \rightarrow u]$ has a very low error rate. If we insist that all prediction rules be of length three (say), then we would have to write additional rules — $[aq \rightarrow u]$, $[bq \rightarrow u]$, $[cq \rightarrow u]$ ... — none of which reduces the errors made by the original, shorter rule. Similarly, an order-3 Markov chain requires conditional probabilities $\Pr(X_2 = u|X_0, X_1 = q)$ though none of these is different from $\Pr(X_2 = u|X_1 = q)$.

Reducing the number of predictors is good both for minimizing space and increasing accuracy. For a given amount of space, one can increase accuracy by using variable-length predictors, because one short rule, like $[q \rightarrow u]$, can do the work of many longer ones. And also, for a desired level of accuracy, having few predictors makes the parameter learning problem easier. A lot of data is needed to train a full 7-gram model; considerably less is needed to train a model that has mostly 3- and 4-grams, but a few highly predictive 7-grams.

The VPR algorithm finds k very predictive rules which are no longer than they should be. The rule $[aq \rightarrow u]$ will not be among these k because it makes identical predictions with, so cannot reduce more errors than, the shorter rule $[q \rightarrow u]$. In contrast, the rule $[lwa \rightarrow y]$ was shown to reduce errors that would be committed by $[wa \rightarrow s]$, but it might not reduce enough errors to warrant including it in the k rules found by VPR.

2 Reducing Errors by Extending Rules

It is best to introduce the algorithm with a series of examples. Suppose someone opens an English text to a random location and says to you, "I am looking at the letter n, predict the letter that follows it." If the text is the first 50 000 letters in George Orwell's *1984* (the source of all examples and statistics in this article), then your best guess is t, because $n\,t$ appears 548 times in the text, more than any other subsequence of length two that begins with n. A prediction rule $[X \rightarrow Y]$ is called *maximal* if XY is more frequent in a corpus than $XZ, (Z \neq Y)$. The rule $[n \rightarrow t]$ is maximal because the sequence $n\,t$ occurs 548 times whereas $n\,d$, the next most frequent subsequence that begins with n, occurs only 517 times. Maximal prediction rules incur fewer errors; for example, $[n \rightarrow t]$ incurs $548 - 517 = 31$ fewer errors than $[n \rightarrow d]$.

Although it is maximal, $[n \rightarrow t]$ incurs *many* errors: because n occurs 3471 times, and $n\,t$ only 548 times, the rule is wrong 2923 times. To cover some of these errors, we can *extend* the maximal rule, by adding an additional letter to the left-hand side, producing a new rule of the form $[\alpha n \rightarrow \beta]$. Of course, then both $[n \rightarrow t]$ and $[\alpha n \rightarrow \beta]$ will match the sequence $a\,n$, so we need a *precedence rule* to decide which to use. Since the longer rule is designed to be an exception to the shorter one, the precedence rule we use is: Always use the rule with the

longest left-hand side. It turns out that the best extension to $[n \rightarrow t]$ is $[in \rightarrow g]$. This rule eliminates 341 errors, more than any other extension of $[n \rightarrow t]$.

The reduction comes from two sources, one obvious, the other less so, as shown in Table 1. The first two rows of the table show what we already know, that there are 548 instances of $n\,t$ and 3471 instances of n, so predicting t following n incurs $3471 - 548 = 2923$ errors. The next two lines show the number of errors incurred by the maximal rule $[in \rightarrow g]$. There are 1095 occurrences of $i\,n$ in the corpus and 426 occurrences of $i\,n\,g$, so $1095 - 426 = 669$ errors are incurred by this rule. The surprise comes in the final two lines of Table 1. The rule $[in \rightarrow g]$ applies whenever one sees the sequence $i\,n$, but what if one sees, instead, $o\,n$? Should we continue to use the rule $[n \rightarrow t]$ in this case? No. Once we have a rule $[in \rightarrow g]$, we can distinguish two kinds of sequence: those that include n and start with i and those that include n and start with something other than i, denoted $\neg i$ in Table 1. To treat these cases as identical, to use $[n \rightarrow t]$ for both, is to ignore the information that we're looking at n and the previous letter was *not* i. This information can reduce errors. Table 1 shows 2376 instances of $\neg in$ and in these cases the maximal rule predicts not t but d. Whereas $[n \rightarrow t]$ incurs 2923 errors, the pair of rules $[in \rightarrow g]$ and $[\neg in \rightarrow d]$ incur $669 + 1913 = 2582$ errors. The reduction in errors by substituting these rules for the original one, $[n \rightarrow t]$, is 341 errors. As it happens, this substitution is the best available for the rule $[n \rightarrow t]$, the one that reduces errors the most.

LHS x	Prediction α	Frequency $N(x)$	Frequency $N(x\alpha)$	Error
n	t	3471	548	2923
in	g	1095	426	669
¬ i n	d	2376	463	1913

Table 1. How error rates change when maximal rules are extended

Although $[in \rightarrow g]$ is the best extension of $[n \rightarrow t]$, it is not the only one, and others may reduce errors, too. In Figure 1 we show how a second extension of $[n \rightarrow t]$ is added to a tree of prediction rules. Panel A shows the rule $[n \rightarrow t]$. Panel B shows the substitution of two rules, $[in \rightarrow g]$ and $[\neg(i)n \rightarrow d]$ for the original one. The original 3471 instances of n are broken into 1095 occurrences of in and 2376 instances of $\neg(i)n$. Correspondingly, the best prediction for $\neg(i)n$ switches from t to d and the errors for both rules sum to 2582, a reduction of 341 over the single rule in panel A. A further substitution is shown in panel C. This time $[\neg(i)n \rightarrow d]$ is replaced by two rules, $[\neg(i,a)n \rightarrow t]$ and $[an \rightarrow d]$. The 2376 cases of $\neg(i)n$ are split into 1704 occurrences of $\neg(i,a)n$ and 672 occurrences of an. By pulling the an cases out of the $\neg(i)n$ cases, we gain information that helps to reduce errors. If we observe an and predict d we make 328 errors ($[an \rightarrow d]$). If we observe n preceded by *neither* i nor a and predict t we make 1359 errors ($[\neg(i,a)n \rightarrow t]$). The two new rules together incur 228 fewer errors than $[\neg(i)n \rightarrow d]$.

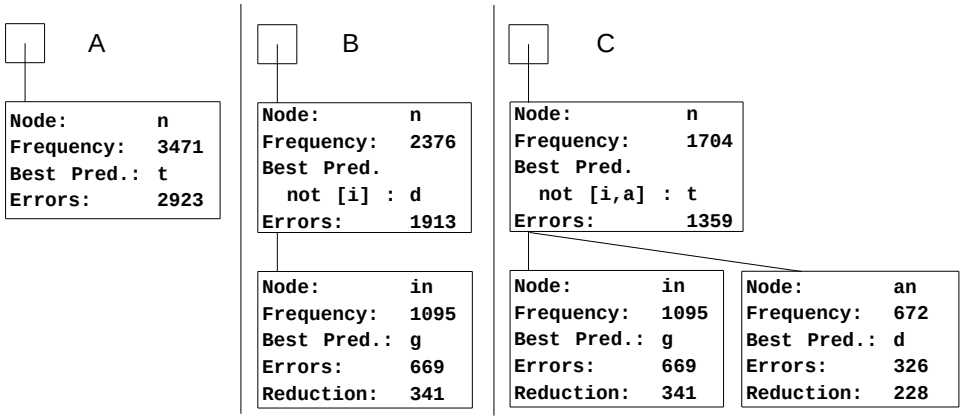

Fig. 1. The evolution of a tree of prediction rules

3 The Space-Limited-Ngrams Problem

Now we formally define the problem. Let X be a string of length n with alphabet Σ. We define a *prediction set* P for X as a set of rules $[r \to \alpha]$ for ngrams r and characters α. We require that P contain a rule for the empty string ϵ; this is the rule of last resort. To add an ngram r to P is to add the rule $[r \to \gamma]$, where $r\gamma$ is maximal. An extension of the rule $[r \to \gamma]$ is any rule $[\alpha r \to \beta]$.

We can organize a prediction set P as a tree, as shown in Figure 1, where each ngram has its extensions as descendants. For example, the tree in Figure 1(C) corresponds to the prediction set $\{[\neg(i,a)n \to t], [in \to g], [an \to d]\}$. Using this tree it is easy to find the longest rule that matches a given sequence. One simply "pushes" the sequence into the tree. Suppose the tree is as shown in Figure 1(C) and the sequence is $o\,n$: This sequence "stops" at the rule $[\neg(i,a)n \to t]$ because on matches $\neg(i,a)n$. Alternatively, if the sequence were $i\,n$ it would stop at the rule $[in \to g]$.

The error $E(P)$ of a prediction set is the number of mistakes it makes in predicting all the characters of X. More formally, for a string Y, let $f_P(Y)$ be the prediction made by the longest rule in P that matches Y. Let $X(i) = x_0 x_1 \ldots x_i$. Let $\overline{\delta}(a,b)$ be the function that is 0 if its arguments are equal, and 1 otherwise. Then we can write

$$E(P) = \sum_i \overline{\delta}(f_P(X(i)), x_i). \tag{1}$$

For trees like those in Figure 1, we can compute $E(P)$ by summing the `Errors:` field for all the nodes.

This error metric is an interesting one for judging rules because it combines precision and recall. Rules with longer antecedents are correct more often when they apply, but they apply less frequently. So a rule that is very precise but is

uncommon, such as $[declaratio \rightarrow n]$, will not reduce the error of P as much as a more fallible rule like $[th \rightarrow e]$.

The SPACE-LIMITED-NGRAMS problem is, given an input string X and an integer k, to return the prediction set P^* of size k that minimizes $E(P^*)$. Although we do not have a polynomial time algorithm for this problem, we have a greedy polynomial-time approximation, which we call VPR.

The VPR algorithm, shown in Figure 2, repeatedly adds the ngram to P that would most reduce $E(P)$. With some bookkeeping we can quickly compute the error after adding an ngram (line 4). We do this by maintaining for each node $q \in P$ its frequency $N(q)$ (that is, the number of times it is needed to make predictions), and its follower table $\boldsymbol{v}(q)$, that is, for each $\alpha \in \Sigma$, the number of times α follows q. When we add an ngram r that extends q, we modify the frequency and follower count so that they do not include the portions of the string that are handled by the exception. More specifically, note that every point in the string X that r predicts was previously predicted by q, for q is a suffix of r. So the new frequency $N'(q)$ is simply $N(q) - N(r)$. And the new follower table $\boldsymbol{v}'(q)$ is $\boldsymbol{v}(q) - \boldsymbol{v}(r)$.

The running time of VPR depends on the loop in line 4 in Figure 2, that is, how large the set Q can be. For each ngram we add to P, we add at most Σ new ngrams to Q, so the size of Q is at most $k\Sigma$. We can cache the frequencies and follower counts of the ngrams using a linear preprocessing step over the string. So the running time of VPR is $O(n + k^2\Sigma)$.

Note that VPR does not necessarily find the optimal prediction set. This can happen if, for example, none of the bigrams appear very predictive, but some trigrams are highly predictive. An example is the string "abccbbacaabccaacbaabccbbbac," where the bigrams are equally distributed, but the trigrams are not.

An alternate greedy algorithm, VPR*, is given in Figure 3. Although slower, VPR* finds better predictive sets. Whereas VPR chooses greedily among one-step extensions of existing rules, VPR* chooses greedily among all the ngrams that occur in the corpus. Clearly, VPR* can perform no worse than VPR, but this comes at a price: in the worst case, VPR* will iterate over all the ngrams that appear in the corpus to select each of the k prediction elements. At most $\binom{n}{2}$ distinct ngrams occur in the corpus—n of length 1, $n-1$ of length 2, and so on. So VPR* has a worst-case running time of $O(kn^2)$.

4 Experiments

Although VPR does not always find the optimal predictor set, we believe that it performs well on naturally occurring sequences. To demonstrate this, we compared VPR to fixed-order classifiers on the first 50 000 characters of George Orwell's *1984*. Each algorithm was trained on the entire corpus, and then the resulting classifier was used to predict each character in the corpus, given the preceding characters. This task is challenging only because k is small; if k were 50 000, for example, it would be trivial to find an optimal predictor set.

VPR(seq, k)
1 $P \leftarrow$ an ngram-set predictor, initially $\{\epsilon\}$
2 $Q \leftarrow$ a list of all 1-grams in seq
3 **repeat** k times
4 $r \leftarrow \arg\min_{r \in Q} E(P \cup \{r\})$
5 Add r to P
6 Remove r from Q
7 Add all one-character extensions of r (that is, αr for all $\alpha \in \Sigma$) to Q
8 **return** P

Fig. 2. The VPR algorithm

VPR*(seq, k)
1 $P \leftarrow$ a rule set, initially $\{\epsilon\}$
2 $Q \leftarrow$ the list of all ngrams in seq
3 **repeat** k times
4 $r \leftarrow \arg\max_{r \in Q} E(P \cup \{r\})$
5 Add r to P
6 Remove r from Q
7 **return** P

Fig. 3. The VPR* algorithm. This differs from VPR* in that it considers all possible rules, not just one-step extensions of current rules

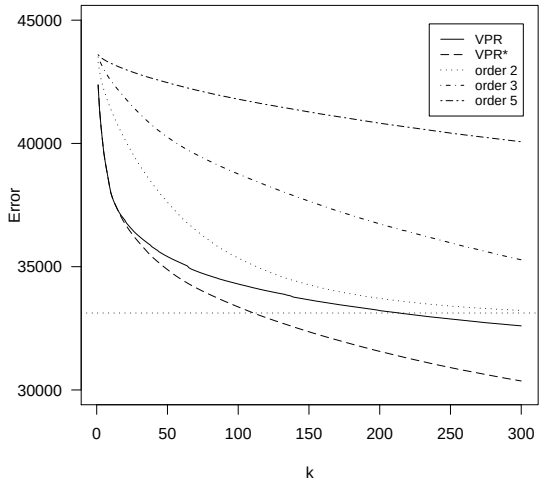

Fig. 4. Comparison of VPR, VPR*, and space-limited fixed-order models on the Orwell data set. The horizontal line is the number of errors made by a full bigram model

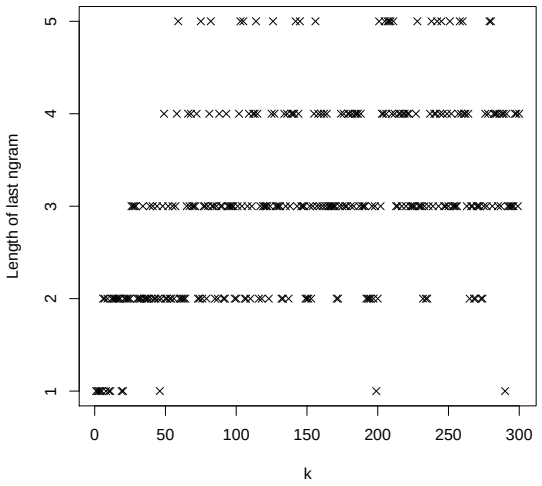

Fig. 5. The length of the k-th ngram chosen by VPR, for each k

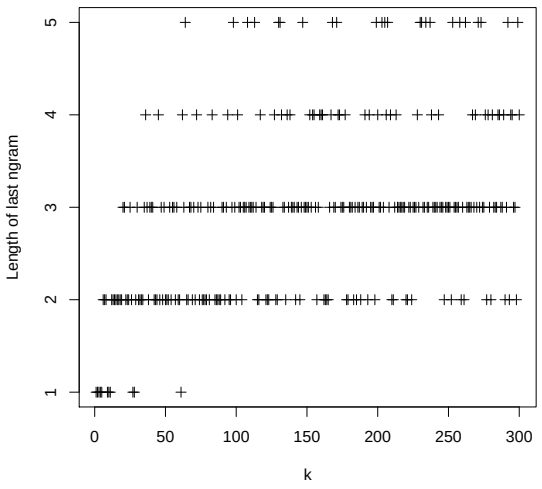

Fig. 6. The length of the k-th ngram chosen by VPR*, for each k

We ran VPR and VPR* with k ranging from 1 to 300. For the fixed-order classifiers, we trained an order-n Markov chain on the text, for n ranging from 1 to 7. But using a full fixed-order model would not be fair to VPR, because a full bigram model, for example, contains 676 predictors, while the largest variable-length model we built has only 300. We made the fixed-order models space-limited by selecting the k fixed-length ngrams that most reduced error. Figure 4 shows the number of errors out of 50 000 made by VPR, VPR*, and the space-limited fixed-length models for different values of k.

With 300 ngrams, VPR makes 32 600 errors on the Orwell corpus out of 50 000 characters. VPR* makes 30 364 errors, about 7% better. By comparison, a full bigram model makes 33 121 errors but uses 676 ngrams. VPR gets that level of performance with 214 ngrams.

To see how useful variable-length ngrams are, we plotted for VPR and VPR*, the length of the k-th ngram added for each k, as shown in Figures 5 and 6. After an initial period of adding 1-grams and 2-grams, the algorithm choose 3-, 4-, and 5-grams alternately. In fact, many 5-grams are chosen before the last few 1-grams.

These results suggest that using variable-length predictors can generate classifiers with the same predictive power but fewer parameters. In addition, the fact that VPR* performs only marginally better than VPR despite having more flexibility suggests that VPR performs well for natural inputs.

5 Related Work

Several authors have used suffix trees to represent variable-memory Markov probabilistic models: Ron et al. [6] call them Probabilistic Suffix Trees (PSTs), and Laird and Saul [4] call them Transition Directed Acyclic Graphs (TDAGs). The rule sets we generate are essentially deterministic versions of these, although we can use the follower tables to build a probabilistic classifier. Ron et al. apply PSTs to finding errors in corrupted text and identifying coding sequences in DNA. PSTs have also been applied to protein family modeling [2]. Laird and Saul describe applications to text compression, dynamic optimization of Prolog programs, and prefetching in mass-storage systems.

Given a corpus generated by a PST T,[1] Ron et al. [6] give a polynomial-time algorithm that with probability $1 - \delta$, returns a PST $\hat{T}$ such that the KL-distance between T and $\hat{T}$ is less than ϵ, for any choices of δ and ϵ, assuming that the sample size is sufficiently large. They give an upper bound on the number of states in $\hat{T}$, but in terms of the size of the unknown generator T. A later algorithm for PST-learning has linear time and space complexity [1].

Laird and Saul [4] construct a full suffix tree up to a fixed depth, but omit states whose relative frequency are below a user-specified threshold.

Both of these algorithms use variable-memory models to reduce the space used by the model while retaining predictive power, as does ours. Our approach

[1] There is actually a minor restriction on the source PST.

differs from these in that VPR takes the number of predictors k as a parameter, and guarantees that the model it returns will not exceed this. Furthermore, the question of what is the best accuracy that can be attained with a fixed predictor size has not been explored in the previous work.

Our error metric $E(P)$, the number of errors P would make in predicting the input, is also novel. Ron et al. choose to add an ngram if its follower distribution is sufficiently different from its parent's. Laird and Saul include an ngram in a model if it occurs a minimum number of times and its probability of being reached, no matter the confidence in its prediction, is above some threshold. A comparison of these methods is left for future work.

Many methods exist for fixed-memory sequence prediction. Commonly-used techniques include Markov chains, HMMs [5], and recurrent neural networks [3].

6 Conclusion

We have given a greedy algorithm for sequential prediction tasks that uses variable-sized predictors, rather than the fixed-size predictors used in classical Markov models. By doing this, we can build predictors that are equally good as Markov models, but have a smaller number of predictors.

Although both VPR and VPR* require time polynomial in the length n of the corpus, neither of them is optimal. We do not have a polynomial-time optimal algorithm for the SPACE-LIMITED-NGRAMS problem. Future work includes either giving a polynomial time algorithm or showing that the associated decision problem is **NP**-complete. If the problem is **NP**-complete, then it would be good to derive an approximation ratio for VPR.

It is also natural to extend the SPACE-LIMITED-NGRAMS problem so that the input is a sample drawn from a language. Then the problem would be to learn a predictor set from the sample that has low expected error on the language. We suspect, but have not demonstrated, that similar algorithms would be useful in this case.

References

1. Alberto Apostolico and Gill Bejerano. Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. In *RECOMB*, pages 25–32, 2000.
2. G. Bejerano and G. Yona. Variations on probabilistic suffix trees: statistical modeling and the prediction of protein families. *Bioinformatics*, 17(1):23–43, 2001.
3. Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley, 2nd edition, 2001.
4. Philip Laird and Ronald Saul. Discrete sequence prediction and its applications. *Machine Learning*, 15:43–68, 1994.
5. Lawrence Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, 1989.
6. Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25(2-3):117–149, 1996.