Learning Prediction Suffix Trees with Winnow

Nikos Karampatziakis NK@CS.CORNELL.EDU
Dexter Kozen KOZEN@CS.CORNELL.EDU
Department of Computer Science, Cornell University, Ithaca, NY 14853 USA

Abstract

Prediction suffix trees (PSTs) are a popular tool for modeling sequences and have been successfully applied in many domains such as compression and language modeling. In this work we adapt the well studied Winnow algorithm to the task of learning PSTs. The proposed algorithm automatically grows the tree, so that it provably remains competitive with any fixed PST determined in hindsight. At the same time we prove that the depth of the tree grows only logarithmically with the number of mistakes made by the algorithm. Finally, we empirically demonstrate its effectiveness in two different tasks.

1. Introduction

Prediction suffix trees are a well studied and compact model for problems such as temporal classification and probabilistic modeling of sequences (Buhlmann & Wyner, 1999; Helmbold & Schapire, 1997; Pereira & Singer, 1999; Ron et al., 1996; Willems et al., 1995). Different variants of PSTs are also called context tree weighting (Willems et al., 1995) and variable length Markov Models (Buhlmann & Wyner, 1999). PSTs operate in a setting similar to online learning. The model observes symbols from a sequence, one at a time, and makes a prediction about the next symbol based on the symbols it has observed so far. PSTs typically use only a few recently observed symbols which are called the context of the prediction. In this sense they are making a Markovian assumption but, unlike most other Markov models, the number of symbols that are used to predict the next symbol depends on the specific context in which the prediction is made.

In this work, we show how to learn a PST by adapting the Winnow algorithm. Even though our theoretical results are somewhat similar to (Dekel et al., 2004), where the perceptron algorithm is used for the same task, we empirically show that the proposed algorithm consistently obtains lower error rates and grows smaller trees than the one in (Dekel et al., 2004). Our motivating application is monitoring processes in a computer system. Each process produces a sequence of system calls which request different services from the operating system. Our task is to model this sequence and maintain a compact profile of the application. In this setting, we expect a multiplicative update algorithm to be more appropriate than the algorithm in (Dekel et al., 2004) for two reasons. First, complex applications are likely to exhibit different behaviors at various points during their execution. A multiplicative algorithm can quickly modify its model to reflect this. Second, multiplicative algorithms cope better with many irrelevant attributes and this will turn out to be true when we formalize our problem.

The rest of the paper is organized as follows. In section 2 we review the balanced version of Winnow which forms the basis of our algorithm. Our main results are presented in section 3. In section 4 we present some empirical results and in section 5 we discuss related approaches. Section 6 states our conclusions.

2. Background

We start by describing Balanced Winnow (Littlestone, 1989). One way to express it is shown in Algorithm 1. At each round t, the algorithm updates a vector of parameters $\boldsymbol{\theta}_t \in \mathbb{R}^d$ and then uses it to construct its weight vector $\boldsymbol{w}_t$. Then a prediction is obtained as the inner product of $\boldsymbol{w}_t$ and the features $\boldsymbol{x}_t$. If the algorithm makes a mistake it updates its parameter vector $\boldsymbol{\theta}$ by adding to it a scaled version of the quantity $y_t \boldsymbol{x}_t$, $y_t \in \{-1, +1\}$. This type of update is similar to the perceptron (Rosenblatt, 1958), however that would use directly $\boldsymbol{\theta}_t$ instead of $\boldsymbol{w}_t$ for prediction. Selecting a value for the parameter α, known as the learning rate, is not trivial and we will always set it at the very end

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009.

Algorithm 1 Balanced Winnow Algorithm

$\boldsymbol{\theta}_1 \leftarrow \mathbf{0}$
for $t = 1, 2, \ldots, T$ **do**
  $w_{t,i} \leftarrow \frac{e^{\theta_{t,i}}}{\sum_{j=1}^{d} e^{\theta_{t,j}}}$
  $\hat{y}_t \leftarrow \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle$
  if $y_t \hat{y}_t \leq 0$ **then**
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha y_t \boldsymbol{x}_t$
  else
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t$
  end if
end for

so that it optimizes the quantities of interest.

The main characteristic of Balanced Winnow is that the features have the form $\boldsymbol{x}_t = [1, -1] \otimes \boldsymbol{x}_t^+$ where $\boldsymbol{x}_t^+$ is the original vector of features and $\boldsymbol{u} \otimes \boldsymbol{v}$ is the Kronecker product of $\boldsymbol{u}$ and $\boldsymbol{v}$. This form of $\boldsymbol{x}_t$ will later allow us to sparsify Balanced Winnow.

2.1. Mistake Bound for Balanced Winnow

In this section we will briefly discuss a bound on the number of mistakes made by Balanced Winnow. Throughout the paper we will assume, or construct our features such that, $||\boldsymbol{x}_t||_\infty \leq 1$ for all t to make the bounds simpler. In general, if the labels y_t are chosen uniformly at random or even adversarially, no algorithm could achieve any meaningful bound. Therefore, we assume that there exists a vector $\boldsymbol{u}$ such that for all t, $y_t \langle \boldsymbol{u}, \boldsymbol{x}_t \rangle > \delta$ for some $\delta > 0$.[1] The magnitude of δ quantifies how hard the problem is. Furthermore, since $\boldsymbol{w}_t$ is always inside the d-dimensional probabilistic simplex we also assume the same is true for $\boldsymbol{u}$.

The general strategy for proving bounds for online learning algorithms is to define some measure of progress $\Phi(\boldsymbol{w}_t)$ towards $\boldsymbol{u}$ and show that with each mistake the algorithm makes, $\boldsymbol{w}_t$ comes closer to $\boldsymbol{u}$. A natural measure of progress is the relative entropy between $\boldsymbol{w}_t$ and $\boldsymbol{u}$:

$$\Phi(\boldsymbol{w}_t) = D(\boldsymbol{u}||\boldsymbol{w}_t) = \sum_{i=1}^{d} u_i \log \frac{u_i}{w_{t,i}}$$

which is always nonnegative. Then we can show that:

Theorem 1. *Let $\{(\boldsymbol{x}_t, y_t)\}$, $t = 1, \ldots, T$ be a sequence of input output pairs with $\boldsymbol{x}_t \in \mathbb{R}^d$ and $y_t \in \{-1, 1\}$. Let $\boldsymbol{u}$ be a a vector such that $u_i \geq 0$ for all i, $\sum_i \boldsymbol{u}_i = 1$, and $y_t \langle \boldsymbol{u}, \boldsymbol{x}_t \rangle \geq \delta$ for some $\delta > 0$. Then Balanced Winnow will make at most $\frac{2 \log d}{\delta^2}$ mistakes.*

[1] This assumption can be relaxed. See also theorem 2.

Proof. We present a sketch here because this has already been proved elsewhere in more general form (e.g. (Shalev-Shwartz, 2007)) and the proof of our main result in theorem 2 has a similar structure. The proof proceeds by upper bounding the initial potential and lower bounding $\Phi(\boldsymbol{w}_t) - \Phi(\boldsymbol{w}_{t+1})$ using lemma 4 and the fact that when a mistake is made $y_t \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle \leq 0$. Combining the upper and lower bounds leads to

$$M \leq \frac{\log d}{(\alpha\delta - \log \cosh(\alpha))}$$

where M is the number of mistakes made. This can be upper bounded using lemma 5. The optimal value of α is δ and this yields the bound in the theorem. □

3. Learning Prediction Suffix Trees

PSTs are popular models for sequential prediction. A PST is a tree data structure that can compactly store a set S of strings and all their suffixes. Each $s \in S$ corresponds to a unique node in the tree. In this node we keep a model for predicting the next symbol when s is a suffix of the current sequence. A PST makes a prediction using a weighted sum of the models in the nodes that match the suffixes of the current sequence. Section 3.2 gives a more detailed description.

The advantage of PSTs compared to other Markov models is that the number of symbols used to predict the next symbol depends on the specific context in which the prediction is made. But how much context should be used for each prediction? When learning a PST one is faced with the following dilemma. If we consider contexts with more symbols than necessary we must estimate more parameters and we are likely to hurt the generalization performance of the algorithm. On the other hand underestimating the right amount of context will typically lead to many mistakes.

Recently, (Dekel et al., 2004) proposed an algorithm that learns a PST of appropriate size by balancing the above tradeoff. The algorithm in (Dekel et al., 2004) uses the perceptron algorithm to learn a PST. The basic idea is that avoiding to grow the tree is like introducing a noise term in the perceptron update. We use the same idea here for Balanced Winnow but many details are different, especially how much noise is tolerated before the tree is grown. Before we discuss how Balanced Winnow can be adapted to work for PSTs we digress to discuss how to modify Balanced Winnow so that it effectively ignores some of its inputs.

3.1. Sparse Balanced Winnow

Algorithm 1 assigns a nonzero weight to every feature, even if some of these weights may be exponentially

small. Here we explain why Balanced Winnow could be modified so that its predictions can ignore the values of some attributes. Then we can say that the algorithm has assigned zero weight to those attributes and our hypothesis is sparse. This may at first seem hard given that the elements of the weight vector are by construction always positive and that the proof of theorem 1 hinges on the use of relative entropy which becomes infinite if one of the entries of the weight vector is zero. However, if one considers the form of $\boldsymbol{x}_t = [1, -1] \otimes \boldsymbol{x}_t^+$ and notices that

$$\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle = \langle \boldsymbol{w}_t^+, \boldsymbol{x}_t^+ \rangle - \langle \boldsymbol{w}_t^-, \boldsymbol{x}_t^+ \rangle = \langle \boldsymbol{w}_t^+ - \boldsymbol{w}_t^-, \boldsymbol{x}_t^+ \rangle$$

where $\boldsymbol{w}_t^+$ corresponds to the first half of the weight vector $\boldsymbol{w}_t$ and $\boldsymbol{w}_t^-$ corresponds to its second half, then if $w_{t,i}^+ = w_{t,i}^-$ any decision of the form $\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle$ will effectively ignore attribute $x_{t,i}$. We can have a better insight on how the algorithm operates with inputs of the form $[1, -1] \otimes \boldsymbol{x}_t^+$ if we rewrite the parameters $\boldsymbol{\theta}_t$ in the same manner as the weight vector ie. $\boldsymbol{\theta}_t = [\boldsymbol{\theta}_t^+, \boldsymbol{\theta}_t^-]$. The following lemma shows the relationship between the two vectors $\boldsymbol{\theta}_t^+$ and $\boldsymbol{\theta}_t^-$

Lemma 1. *In Balanced Winnow, for all t* $\boldsymbol{\theta}_t^- = -\boldsymbol{\theta}_t^+$.

Proof. We proceed by induction. For $t = 1$ we have $\boldsymbol{\theta}_1 = \mathbf{0}$ so $\boldsymbol{\theta}_t^+ = \boldsymbol{\theta}_t^- = \mathbf{0}$ and the claim is true. If at time $t + 1$ there was no mistake the claim is true. If there was a mistake then the parameters were updated as follows $\boldsymbol{\theta}_{t+1}^+ = \boldsymbol{\theta}_t^+ + \alpha y_t \boldsymbol{x}_t^+$ and $\boldsymbol{\theta}_{t+1}^- = \boldsymbol{\theta}_t^- + \alpha y_t (-\boldsymbol{x}_t^+)$. Using the induction hypothesis the latter update is $\boldsymbol{\theta}_{t+1}^- = -\boldsymbol{\theta}_t^+ - \alpha y_t \boldsymbol{x}_t^+ = -\boldsymbol{\theta}_{t+1}^+$ □

By lemma 1, the decision at time t can be written as:

$$\hat{y}_t = \langle \boldsymbol{w}_t^+ - \boldsymbol{w}_t^-, \boldsymbol{x}_t^+ \rangle = \sum_{i=1}^{d} \frac{\sinh(\theta_{t,i}^+)}{\sum_{j=1}^{d} \cosh(\theta_{t,j}^+)} x_{t,i}^+$$

where we have used the definitions of hyperbolic sine and cosine[2]. For the purposes of decisions, the quantity $\sum_{j=1}^{d} \cosh(\theta_{t,j}^+)$ is a positive constant so it will not affect the sign of $\hat{y}_t$. Let $g_{t,i} = \sinh(\theta_{t,i}^+)$. $x_{t,i}^+$ is ignored when $g_{t,i}$ is zero. Since the only root of the hyperbolic sine is at zero, $g_{t,i} = 0$ if and only if $\theta_{t,i}^+ = 0$.

The strategy to derive a multiplicative update algorithm whose hypothesis is sparse is to make sure that some of the parameters remain to zero by introducing some noise that cancels the update for these specific parameters. When analyzing the algorithm, we will have to show that the progress made with each mistake overshadows the effect of noise. The algorithm will be phrased in terms of the normalized weights to maintain the correspondence with Balanced Winnow. In a practical implementation, one should update the values of the parameters $\boldsymbol{\theta}_t$ and use the unnormalized weights $g_{t,i}$ to make predictions.

[2] $\sinh(x) = \frac{e^x - e^{-x}}{2}$ and $\cosh(x) = \frac{e^x + e^{-x}}{2}$

3.2. Winnow for Prediction Suffix Trees

Our formal setup is as follows. We wish to predict the items of a sequence $y_1, y_2, \ldots, y_T$, $y_i \in \Sigma$, one at a time. For now, we assume that $\Sigma = \{-1, 1\}$ and we discuss extensions to larger alphabets in section 4. Let $|y|$ be the length of a sequence $y \in \Sigma^*$ and let y_i^j denote the subsequence $y_i, \ldots, y_j$. The prediction for y_t will be based on a suffix of y_1^{t-1}, typically one that contains only the last few symbols in y_1^{t-1}. We start with the preconception that symbols near t should be more important for the prediction task than symbols away from t (Helmbold & Schapire, 1997; Willems et al., 1995). This can be encoded in the features which will be of the form $\boldsymbol{x}_t = [1, -1] \otimes \boldsymbol{x}_t^+$ where:

$$x_{t,s}^+ = \begin{cases} (1-\epsilon)^{|s|} & \text{if} \quad s = y_{t-i}^{t-1} \qquad i = 1, \ldots, t-1 \\ 0 & \text{otherwise} \end{cases}$$

Therefore $\boldsymbol{x}_t$ will conceptually have at most $T(T-1)$ dimensions, twice as many as the number of distinct substrings of y_1^T. Here and below the vector $\boldsymbol{x}^+$ is indexed by strings from the language $\Sigma^{\leq T}$ and the notation $x_{t,s}^+$ simply means $x_{t,\ell(s)}^+$ where $\ell(s)$ is the position of s in the lexicographic order of $\mathcal{Y}_T$, the set of all substrings of y_1^T. Another useful function is the inverse of $\ell(\cdot)$: $s(i)$ is the string at position i in the aforementioned lexicographic order. Furthermore, we extend $s(\cdot)$ to be a periodic function (with period $|\mathcal{Y}_T|$). This allows to handle indices from the extended features $\boldsymbol{x}_t$ so that we can write for example $|x_{t,i}| = x_{t,s(i)}^+$.

As before, the algorithm will have to learn a vector of parameters $\boldsymbol{\theta}_t = [\boldsymbol{\theta}_t^+, \boldsymbol{\theta}_t^-]$ while keeping its *support*, the strings for which the corresponding entries are non zero, small. The decisions of the algorithm will be

$$\hat{y}_t = \sum_i g_{t,i} x_{t,i}^+$$

where $g_{t,i} = \sinh(\theta_{t,i}^+)$ and i ranges over the support of $\boldsymbol{\theta}_t$. The set A that contains the support of $\boldsymbol{\theta}$, every suffix of every element in the support, and the empty string can be viewed as a tree. This tree is constructed in the following way. Every element of A is a node in the tree. The root corresponds to the empty string. Let $y \in \Sigma$ and $u \in \Sigma^*$. If $v = yu \in A$ then $u \in A$ by the definition of A and v is a direct child of u in the tree with the link from u to v being labeled with y. In this view, $\boldsymbol{\theta}^+$, and consequently $\boldsymbol{g}$ assigns weights to nodes in the tree and the predictions of the online algorithm amount to taking a weighted sum of the values in the

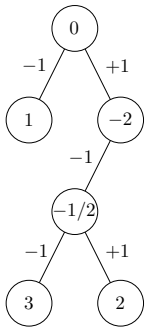

Figure 1. A PST whose support is the union of $-1,-1-1+1,+1-1+1$ and all their suffixes. If $\epsilon = 1/4$ and $y_{t-2}^t = -1-1+1$ then the prediction will be $\hat{y}_{t+1} = \text{sign}(\frac{3}{4}\sinh(-2) + \frac{9}{16}\sinh(-\frac{1}{2}) + \frac{27}{64}\sinh(3)) = +1$. The algorithm is able to overcome the prior belief that a longer suffix is less relevant by assigning its node a large value.

nodes of a path in the tree. This path starts from the root and follows the links that correspond to the symbols y_{t-1},y_{t-2},..., until we reach a leaf or the next child does not exist. Figure 1 shows a PST.

Balanced Winnow starts with $\boldsymbol{\theta}_1 = \mathbf{0}$ therefore initially the support of $\boldsymbol{\theta}$ is the empty string and the corresponding tree has only one node. As learning progresses, each mistake the algorithm makes will cause some of the parameters to be updated to non zero values. Hence we expect the support of $\boldsymbol{\theta}$ to increase. In fact we will require that the support of $\boldsymbol{\theta}_t$ is a subset of the support of $\boldsymbol{\theta}_{t+1}$. Thus, the algorithm can be thought as growing the tree that corresponds to A.[3]

If we just apply Balanced Winnow to the task of learning such a PST we will need memory that is $O(T^2)$. This happens because every distinct substring of y_1^T will have an associated parameter. Hence, our strategy will be to have an adaptive bound d_t on the length of the suffixes that we will be considering, which is the same as the depth up to which we will grow the tree.

The proposed algorithm modifies Balanced Winnow so that the parameter update is

$$\theta_{t+1,i} = \begin{cases} \theta_{t,i} + \alpha y_t x_{t,i} & \text{if } |s(i)| \le d_t \\ \theta_{t,i} & \text{otherwise} \end{cases}$$

This can be equivalently written as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha y_t \boldsymbol{x}_t + \alpha \boldsymbol{n}_t$$

where $\boldsymbol{n}_t$ is a noise vector that cancels some of the components of the update:

$$n_{t,i} = \begin{cases} -y_t x_{t,i} & \text{if } |s(i)| > d_t \\ 0 & \text{otherwise} \end{cases}$$

Note that $||\boldsymbol{n}_t||_\infty = (1-\epsilon)^{(d_t+1)}$, a fact that we will use later in the derivation of the mistake bound. Let h_t be the length of the maximal downward path from the root using the symbols $y_{t-1}, y_{t-2}, \ldots$. Clearly, if at any point our way of setting d_t suggests that it should be less than h_t there is no reason to let this happen because we already have grown the tree beyond this point. Letting $d_t < h_t$ will only make the norm of $\boldsymbol{n}_t$ larger without any computational benefit. Therefore $d_t \ge h_t$ and then it is easy to prove the following

Lemma 2. *For all t and i either $\theta_{t,i} = 0$ or $n_{t,i} = 0$.*

Proof. There are two cases: either $|s(i)| \le d_t$ or $|s(i)| > d_t$. If $|s(i)| \le d_t$ then $n_{t,i} = 0$ by definition. Now suppose $\theta_{t,i} \neq 0$ and $|s(i)| > d_t$. That means there was a point $t' < t$ when $d_{t'} > d_t$. That would imply $d_{t'} > h_t$ but since the tree never shrinks this cannot happen. Therefore $\theta_{t,i} = 0$ if $|s(i)| > d_t$. □

Furthermore the sum of the norms of the noise vectors will turn up in the proof of the mistake bound, therefore we would like to keep it bounded by a function of the number of mistakes. Let J_t be the subset of rounds $1,\ldots,t$ in which a mistake was made and let $M_t = |J_t|$. Also let

$$P_t = \sum_{i \in J_t} (1-\epsilon)^{(d_i+1)}$$

and $M_0 = P_0 = 0$. We will require that d_t is the smallest integer such that[4]

$$P_t \le M_t^{2/3}$$

subject to $d_t \ge h_t$. We can then prove the following

Lemma 3. *Setting $d_t = \max\{h_t, b(P_{t-1})\}$ ensures that for all t, $P_t \le M_t^{2/3}$ where*

$$b(p) = \left\lceil \log_{1-\epsilon}(\sqrt[3]{p^3 + 2p^{3/2} + 1} - p) - 1 \right\rceil \qquad (1)$$

Proof. We proceed by induction. For $t = 1$ the claim clearly holds. If there is no mistake on round t then $P_t = P_{t-1}$ and $M_t = M_{t-1}$ so the claim holds. If there is a mistake then $P_t = P_{t-1} + ||\boldsymbol{n}_t||_\infty$, $M_t = M_{t-1} + 1$ and it suffices to show that $P_t^3 \le P_{t-1}^3 + 2P_{t-1}^{3/2} + 1$ since by induction $P_{t-1}^3 + 2P_{t-1}^{3/2} + 1 \le (M_{t-1}+1)^2$. So

[3] Even if an update causes a parameter to become zero after it has taken a non zero value, we will still keep the corresponding node in the tree.

[4] It would also be possible to design the algorithm using the invariant $P_t \le \sqrt{M_t}$ as in (Dekel et al., 2004). Our choice of a greater upper bound on the noise seems to allow for superior performance in practice (cf. section 4)

Algorithm 2 Balanced Winnow for PSTs

1: $P_0 \leftarrow 0$
2: $A_1 \leftarrow \{\lambda\}$ {the empty string}
3: $\boldsymbol{\theta}_1 \leftarrow \mathbf{0}$
4: **for** $t = 1, 2, \ldots, T$ **do**
5: $\quad h_t \leftarrow \max\{j : y_{t-j}^{t-1} \in A_t\}$
6: $\quad d_t \leftarrow \max\{h_t, b(P_{t-1})\}$ {Lemma 3}
7: $\quad$ Compute $\boldsymbol{x}_t$ from y_1^{t-1}
8: $\quad w_{t,i} \leftarrow e^{\theta_{t,i}} / \sum_j e^{\theta_{t,j}}$
9: $\quad \hat{y}_t \leftarrow \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle$
10: $\quad$ **if** $y_t \hat{y}_t \leq 0$ **then**
11: $\quad\quad P_t \leftarrow P_{t-1} + (1-\epsilon)^{(d_t+1)}$
12: $\quad\quad A_{t+1} \leftarrow A_t \cup \{y_{t-i}^{t-1} : 1 \leq i \leq d_t\}$
13: $\quad\quad \boldsymbol{\theta}_{t+1,i} \leftarrow \begin{cases} \boldsymbol{\theta}_{t,i} + \alpha y_t \boldsymbol{x}_{t,i} & \text{if } |s(i)| \leq d_t \\ \boldsymbol{\theta}_{t,i} & \text{otherwise} \end{cases}$
14: $\quad$ **else**
15: $\quad\quad P_t \leftarrow P_{t-1}$
16: $\quad\quad A_{t+1} \leftarrow A_t$
17: $\quad\quad \boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t$
18: $\quad$ **end if**
19: **end for**

$$(P_{t-1} + ||\boldsymbol{n}_t||_\infty)^3 \leq P_{t-1}^3 + 2P_{t-1}^{3/2} + 1$$

Let $z = ||\boldsymbol{n}_t||_\infty$, $z > 0$. The above can be written as

$$z^3 + 3P_{t-1}z^2 + 3P_{t-1}^2 z - 2P_{t-1}^{3/2} - 1 \leq 0$$

and is valid when z is less than its only real root:

$$z \leq \sqrt[3]{P_{t-1}^3 + 2P_{t-1}^{3/2} + 1} - P_{t-1}$$

Since $z = ||\boldsymbol{n}_t||_\infty = (1-\epsilon)^{(d_t+1)}$ we finish the proof

$$d_t \geq \log_{1-\epsilon}\left(\sqrt[3]{P_{t-1}^3 + 2P_{t-1}^{3/2} + 1} - P_{t-1}\right) - 1$$

The statement of the lemma simply enforces that d_t is an integer and that $d_t \geq h_t$. □

We can now state how everything is put together in Algorithm 2. The set of suffixes A_t can be implemented as a tree, as was described above. The algorithm starts with an empty tree and in each round it predicts the next symbol using the parameters in the tree. If a mistake is made and if $d_t > h_t$ we grow the tree in line 12. Finally, we update the parameter values for the nodes in the tree and repeat.

In the following theorem we show how the number of mistakes of this algorithm compares against the performance of the optimal tree in hindsight. This is measured by the cumulative δ-hinge loss of the optimal vector $\boldsymbol{u}$. The δ-hinge loss that $\boldsymbol{u}$ attains at round t is

$$L_t = \max\{0, \delta - y_t \langle \boldsymbol{u}, \boldsymbol{x}_t \rangle\}.$$

Before we state our result, we prove two useful lemmas.

Lemma 4. *For all $x \in [-1, 1]$ and $\alpha > 0$ the following holds $e^{\alpha x} \leq \cosh(\alpha) + \sinh(\alpha) x$.*

Proof. First note that $e^{-\alpha} = \cosh(\alpha) - \sinh(\alpha)$ and $e^{\alpha} = \cosh(\alpha) + \sinh(\alpha)$. The lemma follows by the convexity of $e^{\alpha x}$. □

Lemma 5. *For all $\alpha > 0$, $\log\cosh(\alpha) \leq \frac{\alpha^2}{2}$.*

Proof. The bound follows by noticing that it is equivalent to $\int_0^\alpha \int_0^x 1 - \tanh^2(u) du dx \leq \int_0^\alpha \int_0^x 1 du dx$ □

Our main result is the following theorem:

Theorem 2. *Let $y_1, y_2, \ldots, y_T$ be a sequence of symbols $\in \{-1, +1\}$. Let $\boldsymbol{u}$ be a vector such that for all i $u_i \geq 0$, $\sum_i u_i = 1$ and $\sum_t L_t = L$. Then Algorithm 2 will make at most $\max\left\{\frac{2L}{\delta} + \frac{8 \log T}{\delta^2}, \frac{64}{\delta^3}\right\}$ mistakes.*

Proof. We will use the relative entropy of $\boldsymbol{u}$ and $\boldsymbol{w}$ as our measure of progress. Recall that we are working with vectors of dimension $d \leq T(T-1)$ and let

$$\Phi(\boldsymbol{w}_t) = D(\boldsymbol{u}||\boldsymbol{w}_t) = \sum_{i=1}^d u_i \log \frac{u_i}{w_{t,i}}$$

Let $\Delta_t = \Phi(\boldsymbol{w}_t) - \Phi(\boldsymbol{w}_{t+1})$. The telescoping sum

$$\sum_{t=1}^T \Delta_t = \Phi(\boldsymbol{w}_1) - \Phi(\boldsymbol{w}_{T+1})$$

is not greater than $\Phi(\boldsymbol{w}_1)$ because of the non-negativity of $\Phi(\boldsymbol{w}_{T+1})$. Furthermore

$$\Phi(\boldsymbol{w}_1) \leq \sum_{i=1}^d u_i \log u_i + \sum_{i=1}^d u_i \log(d) \leq \log d \leq 2 \log T$$

where we have used the non-negativity of the entropy of $\boldsymbol{u}$ and that $\boldsymbol{w}_1$ starts as a uniform distribution. Putting all these together, we have the upper bound

$$\sum_{t=1}^T \Delta_t \leq 2 \log T \tag{2}$$

If on round t there was no mistake then $\Delta_t = 0$. Now assume that a mistake was made on round t. To lower bound Δ_t, we first write $\boldsymbol{w}_{t+1}$ in terms of $\boldsymbol{w}_t$:

$$w_{t+1,i} = \frac{e^{\theta_{t+1,i}}}{Z_t} = \frac{e^{\theta_{t,i} + \alpha y_t x_{t,i} + \alpha n_{t,i}}}{Z_t} = \frac{w_{t,i} e^{\alpha y_t x_{t,i} + \alpha n_{t,i}}}{Z_t}$$

where Z_t is the normalization constant:

$$Z_t = \sum_{j=1}^d w_{t,j} e^{\alpha y_t x_{t,j} + \alpha n_{t,j}} \tag{3}$$

Replacing the above in $\Delta_t = \sum_{i=1}^d u_i \log \frac{w_{t+1,i}}{w_{t,i}}$, which is just another way to define Δ_t, leads to

$$\begin{aligned} \Delta_t &= \sum_{i=1}^d u_i \log \frac{w_{t,i} e^{\alpha y_t x_{t,i} + \alpha n_{t,i}}}{Z_t w_{t,i}} \\ &= \alpha y_t \langle \boldsymbol{u}, \boldsymbol{x}_t \rangle + \alpha \langle \boldsymbol{u}, \boldsymbol{n}_t \rangle - \log Z_t \end{aligned} \tag{4}$$

We bound the three terms from below. We use Lemma 4 to bound the exponential in (3) by a linear function:

$$\begin{aligned} Z_t &= \sum_{i=1}^{d} w_{t,i} e^{\alpha(y_t x_{t,i} + n_{t,i})} \\ &\leq \sum_{i=1}^{d} w_{t,i}(\sinh(\alpha)(y_t x_{t,i} + n_{t,i}) + \cosh(\alpha)) \\ &= \sinh(\alpha) y_t \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle + \langle \boldsymbol{w}_t, \boldsymbol{n}_t \rangle + \cosh(\alpha) \end{aligned}$$

We have assumed that the algorithm made a mistake on round t. This means that $\sinh(\alpha) y_t \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle \leq 0$. Furthermore in lemma 2 we showed that either $\theta_{t,i} = 0$ or $n_{t,i} = 0$ and we have previously argued that when $\theta_{t,i}$ is zero then $\langle \boldsymbol{w}_t, \boldsymbol{v} \rangle$ does not depend on $w_{t,i}$. Therefore $\langle \boldsymbol{w}_t, \boldsymbol{n}_t \rangle = 0$. These observations lead to

$$Z_t \leq \cosh(\alpha).$$

Furthermore, from Hölder's inequality we have that

$$\langle \boldsymbol{u}, \boldsymbol{n}_t \rangle \geq -||\boldsymbol{u}||_1 ||\boldsymbol{n}_t||_\infty = -||\boldsymbol{n}_t||_\infty$$

Moreover, $y_t \langle \boldsymbol{u}, \boldsymbol{x}_t \rangle \geq \delta - L_t$ by the definition of L_t, so, by combining the three bounds, (4) becomes

$$\Delta_t \geq \alpha\delta - \alpha L_t - \alpha ||\boldsymbol{n}_t||_\infty - \log(\cosh(\alpha)). \quad (5)$$

Summing up the individual lower bounds for all t and using the definitions of M_t, P_t and L we get

$$\sum_{t=1}^{T} \Delta_t \geq M_T(\alpha\delta - \log(\cosh(\alpha))) - \alpha P_T - \alpha L$$

Substituting the invariant $P_t \leq M_t^{2/3}$ and combining this with the upper bound (2) leads to

$$M_T(\alpha\delta - \log\cosh(\alpha)) - \alpha {M_T}^{2/3} - \alpha L \leq 2\log T$$

Let $z = {M_T}^{1/3}$. The inequality

$$(\alpha\delta - \log\cosh(\alpha)) z^3 - \alpha z^2 - \alpha L - 2\log T \leq 0$$

is valid when z is less than the only real root[5] of the polynomial in the left hand size. Let ρ be this root. Then the above inequality is equivalent to $z \leq \rho$. Even though there is an exact analytical expression for ρ, much intuition can be gained from a simple upper bound (Marden, 1966) theorem (30,2): For any $\lambda \geq 1$

$$\rho \leq \max\left\{ \left(\frac{\lambda(\alpha L + 2\log T)}{\alpha\delta - \log\cosh(\alpha)} \right)^{\frac{1}{3}}, \frac{\frac{\lambda}{\lambda-1}\alpha}{\alpha\delta - \log\cosh(\alpha)} \right\}$$

Using this bound we get that

$$M_T \leq \max\left\{ \frac{2L}{\delta} + \frac{8\log T}{\delta^2}, \frac{64}{\delta^3} \right\} \quad (6)$$

by setting $\alpha = \delta/2$, $\lambda = 3/2$, and using lemma 5. □

We further discuss this bound in the light of the theoretical and empirical results in the next two sections.

[5] This is true assuming $\alpha L + 2\log T > 0$, which is safe

3.3. Growth Bound

We also bound how quickly Algorithm 2 grows the tree. In fact, we can show that the depth of the tree grows logarithmically with the number of mistakes:

Theorem 3. *Let $A_1, A_2, \ldots$ be the sequence of trees generated by Algorithm 2 with $\epsilon = 1 - 2^{-1/3}$. Then, for all $T \geq 2$ the depth of A_{T+1} is upper bounded by* $\log_2 M_{T-1} + 3\log_2 \frac{5}{2}$

Proof. The depth of A_{T+1} is the maximum of the d_t values, given by lemma 3, over all rounds t. However, if at round t there was no mistake or $d_t \leq h_t$ then the tree is not grown. So, it suffices to show that, for all rounds t in which a mistake was made, the quantity $b(P_{t-1})$ in equation (1) is not greater than the bound. That expression is upper bounded by $\log_{1-\epsilon}(\sqrt[3]{P_{t-1}^3 + 2P_{t-1}^{3/2} + 1} - P_{t-1})$. Now we switch to base 2 because we want to relate the bound on the depth of the tree with the actual size of the tree, we substitute $\epsilon = 1 - 2^{-1/3}$ and get

$$3\log_2 \frac{1}{\sqrt[3]{P_{t-1}^3 + 2P_{t-1}^{3/2} + 1} - P_{t-1}}$$

It is easy to show that the expression inside the logarithm can be upper bounded by $\frac{3\sqrt{P_{t-1}}+2}{2}$ because such an inequality is equivalent to showing that the polynomial $36y^5 + 48y^4 + 7y^3 + 30y^2 + 36y$ is non negative for $y \geq 0$ where $y = \sqrt{P_{t-1}}$. Therefore the above quantity is upper bounded by

$$3\log_2 \frac{3\sqrt{P_{t-1}} + 2}{2} \leq 3\log_2 \frac{3M_{t-1}^{1/3} + 2}{2}$$

where we used the invariant $P_t \leq {M_t}^{2/3}$. M_t is a non decreasing sequence and $M_{T-1} \geq 1$ since $T \geq 2$ and the first prediction is always a zero which counts as a mistake. So we upper bound the above quantity by

$$3\log_2 \frac{3M_{T-1}^{1/3} + 2}{2} \leq \log_2 M_{T-1} + 3\log_2 \frac{5}{2}$$

where we have used the inequality $\log_2(3\mu + 2) \leq \log_2(5\mu)$, which is true for $\mu \geq 1$. □

Theorem 3 says that the amount of memory needed by the tree grows only linearly with the number of mistakes. The respective growth bound given in (Dekel et al., 2004), also scales linearly but with a larger constant. This is reflected in the experiments, too.

However, the comparison of their mistake bound and ours is complex and in general multiplicative and additive update algorithms have different merits (Kivinen & Warmuth, 1997). Our case is even more intricate because $\boldsymbol{x}_t$ conceptually has twice as many dimensions as that in (Dekel et al., 2004) and the feature values are different. Therefore, the optimal weight vector in hindsight and the value of the margin will be different

Table 1. Summary of experimental results

Dataset	Ulysses Bit		Ulysses A-Z		Excel		Outlook		Firefox	
Update	Perc.	Winn.	Perc.	Winn.	Perc.	Winn.	Perc.	Winn.	Perc.	Winn.
% Error	24.32	20.49	67.58	65.58	22.68	20.59	5.1	4.43	14.86	13.88
PST Size	675K	270K	13.2M	10.3M	24402	15338	41239	25679	21081	12662

for the two algorithms. The latter is likely to be larger in our case because our feature values are larger. In this view our mistake bound dependence on $\frac{1}{\delta^3}$ may not be important since δ may be large. Notice that for the multiplicative algorithm we can afford to push the features as close to 1 as we wish without changing the mistake bound which depends on just $||\boldsymbol{x}_t||_\infty \leq 1$. The algorithm in (Dekel et al., 2004) however depends on $||\boldsymbol{x}_t||_2 \leq 1$ and their features cannot be made larger without affecting their mistake or growth bound. In theorem 3 we set ϵ so that the features are as large as possible while keeping the PST size linear with the number of mistakes. Finally our bound also depends on $\log T$ and even though this dependence also exists in a lower bound (Kivinen & Warmuth, 1997), this assumes that an adversary selects the feature values which is not true in this case. Nevertheless, as the next section shows the actual number of mistakes in practice may differ markedly from these worst case bounds.

4. Experiments

We have conducted some experiments to demonstrate the effectiveness of the proposed algorithm. We are interested in two quantities, the online error rate and the size of the PST. Since the data from our motivating application are not publicly available, we first describe two reproducible experiments. We used the text from James Joyce's Ulysses to define two problems: predicting the next bit and predicting the next alphabetic character in the text (non alphabetic characters were discarded in this case).

The first two columns of table 1 show the results of these experiments. For both tasks, our Winnow variant makes fewer mistakes and grows a smaller tree than the perceptron of (Dekel et al., 2004). In all experiments, α was set to 0.1 and ϵ was set as in theorem 3. The baseline error rates are 50% and 88% respectively.

Predicting the next letter as well as the problems from our motivating application are multiclass problems. To handle them, we adapt ideas from (Crammer & Singer, 2002) and maintain weights $\boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(k)}$ one for each class. The decision at time t is $\hat{y}_t = \arg\max_i \langle \boldsymbol{w}^{(i)}, \boldsymbol{x}_t \rangle$ and if $\hat{y}_t \neq y_t$ we update $\boldsymbol{w}^{(\hat{y}_t)}$ and $\boldsymbol{w}^{(y_t)}$ by updating their parameters: $\theta^{(\hat{y}_t)}_{t+1,i} = \theta^{(\hat{y}_t)}_{t,i} - \alpha x_{t,i}$ and $\theta^{(y_t)}_{t+1,i} = \theta^{(y_t)}_{t,i} + \alpha x_{t,i}$ (if $|s(i)| \leq d_t$ as usual). A problem with this formulation is that we need the normalization constant $Z^{(j)}_t$ for each class j. Strictly speaking, $Z^{(j)}_t$ cannot be computed without a priori knowledge of the length T of the sequence, because we conceptually have a weight for each substring of the whole sequence. However, the following approximation

$$\tilde{Z}^{(j)}_t = \sum_{i \in A_t} e^{\theta^{(j)}_{t,i}}$$

works very well in practice, can be quickly computed from $\tilde{Z}^{(j)}_{t-1}$, and was used in all our experiments.

We now turn to experiments from our motivating application. The task is to predict the next system call a program will execute based on the previous system calls. Our data consists of three applications, Firefox and Microsoft's Excel and Outlook, for each of which we have 40 sequences of system calls. Our monitoring application was recording 23 different system calls. The last three columns in table 1 summarize the results of all experiments. For brevity, we report for each application the average online prediction error and tree size over the 40 sequences. However, table 1 understates the difference of the two algorithms in this problem since the multiplicative algorithm is always making less mistakes and growing smaller trees than the additive one for all three applications and all tested sequences. All paired sample two sided t-tests showed that the differences of the reported quantities are significant ($p < 10^{-5}$ for all tests). However, there exist sequences that can force Winnow to make more mistakes than perceptron. In two out of the 120 sequences Winnow was initially making more mistakes and started outperforming perceptron only after the first half of the sequence.

Finally, it is interesting to note that if instead of $P_t \leq M_t^{2/3}$ we had enforced $P_t \leq M_t^{1/2}$ we would have derived a mistake bound that always scales with $\frac{1}{\delta^2}$ and a similar growth bound for $\epsilon = 1 - 2^{-1/2}$. Surprisingly, this variant has empirically no clear advantage over the additive algorithm. Our analysis allows selecting a smaller ϵ i.e. larger features which in turn allow the margin δ to be larger and this improves the mistake bound. At the same time larger features mean that the norms of the noise vectors will be larger. Hence we need to decide how much noise we should tolerate. Too much noise will hurt the mistake bound and too

little will cause the tree to grow very fast. A good balance can be achieved by requiring the size of the tree to scale linearly with the number of mistakes.

5. Related Work

Apart from (Dekel et al., 2004), with which we compared in the experiments, there are many other methods for learning PSTs which are based on a wide range of principles such as PAC learning (Ron et al., 1996), structural risk minimization (Kearns & Mansour, 1998) and online Bayesian mixtures (Willems et al., 1995) and their generalizations (Helmbold & Schapire, 1997; Pereira & Singer, 1999). All these methods assume that we have a bound on the maximum depth of the tree. For many applications this is not known and the algorithm should be allowed to estimate a good value of this parameter from the data.

(Kivinen & Warmuth, 1997) contains many results and insights about multiplicative algorithms including variants that are competitive with any vector u such that $||u||_1 \leq U$. Extending our algorithm to be competitive with any vector in this ball is also possible. Additionally, many authors, starting with (Blum, 1997), have noticed that Winnow's weight vector can be sparsified by zeroing the entries which have small weights compared to the largest weight in the hypothesis. This procedure is effective in practice but comes with no theoretical guarantees. In the case of learning PSTs however, the tree implicitly defines a partial order for the costs of setting the parameters to non zero values. This allows us to have a sparse hypothesis and still be able to characterize the number of mistakes.

6. Conclusions

We presented a modification of Balanced Winnow for learning PSTs in order to predict the next item in a sequence. The algorithm presented here does not rely on any assumptions about an underlying PST that generates the data. Our algorithm comes with theoretical guarantees about the number of mistakes it will make relative to the best PST determined in hindsight and about the amount of memory it will use to store its hypothesis. In all our experiments we found that it makes fewer mistakes and uses less memory than an additive algorithm with similar guarantees (Dekel et al., 2004).

Acknowledgments

We thank Robert Kleinberg for many valuable discussions, Carla Marceau and ATC-NY for providing the data, and the anonymous reviewers for their helpful comments. This work was supported by the Air Force Office of Sponsored Research under contract FA9550-07-C-0127, Application Monitors for Not-Yet-Trusted Software. The Small Business Technology Transfer effort was carried out jointly by Cornell and ATC-NY.

References

Blum, A. (1997). Empirical Support for Winnow and Weighted-Majority Algorithms: Results on a Calendar Scheduling Domain. *Machine Learning*, *26*, 5–23.

Buhlmann, P., & Wyner, A. (1999). Variable length Markov chains. *Annals of Statistics*, *27*, 480–513.

Crammer, K., & Singer, Y. (2002). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, *2*, 265–292.

Dekel, O., Shalev-Shwartz, S., & Singer, Y. (2004). The power of selective memory: Self-bounded learning of prediction suffix trees. *Neural Information Processing Systems*, *17*, 345–352.

Helmbold, D., & Schapire, R. (1997). Predicting Nearly As Well As the Best Pruning of a Decision Tree. *Machine Learning*, *27*, 51–68.

Kearns, M., & Mansour, Y. (1998). A fast, bottom-up decision tree pruning algorithm with near-optimal generalization. *International Conference on Machine Learning*, 269–277.

Kivinen, J., & Warmuth, M. (1997). Exponentiated Gradient versus Gradient Descent for Linear Predictors. *Information and Computation*, *132*, 1–63.

Littlestone, N. (1989). *Mistake bounds and logarithmic linear-threshold learning algorithms*. Doctoral dissertation, Santa Cruz, CA, USA.

Marden, M. (1966). *Geometry of Polynomials*. AMS.

Pereira, F., & Singer, Y. (1999). An Efficient Extension to Mixture Techniques for Prediction and Decision Trees. *Machine Learning*, *36*, 183–199.

Ron, D., Singer, Y., & Tishby, N. (1996). The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length. *Machine Learning*, *25*, 117–149.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*, 386–408.

Shalev-Shwartz, S. (2007). *Online learning: Theory, algorithms, and applications*. Doctoral dissertation, The Hebrew University of Jerusalem.

Willems, F., Shtarkov, Y., & Tjalkens, T. (1995). The context-tree weighting method: basic properties. *IEEE Transactions on Information Theory*, *41*, 653–664.