Online Learning of Prediction Suffix Trees

Nikos Karampatziakis  Dexter Kozen

Cornell University

April 24, 2009

To appear in ICML 2009
Introduction

Sequential prediction
Introduction

Sequential prediction

Prediction Suffix Trees
Introduction

Sequential prediction

Prediction Suffix Trees

Learning algorithm
Monitoring of applications in a computer system
Motivation

Monitoring of applications in a computer system

System calls
Motivation

Monitoring of applications in a computer system

System calls

Prediction model
Motivation

Monitoring of applications in a computer system

System calls

Prediction model

Some assumptions
Outline

Prediction Suffix Trees

Algorithm and Properties

Results
Outline

Prediction Suffix Trees

Algorithm and Properties

Results
What is the next item in the sequence?

open(), read(), read(), ...
G, A, T, T, A, C, ...
+1, −1, +1, −1, +1, ...
all, your, base, ...
What is the next item in the sequence?

open(), read(), read(), ...
G, A, T, T, A, C, ...
+1, −1, +1, −1, +1, ...
all, your, base, ...

Formally, predict $y_t$ from $y_1, y_2, \ldots, y_{t-2}, y_{t-1}$
What is the next item in the sequence?

open(), read(), read(), ...
G, A, T, T, A, C, ...
+1, −1, +1, −1, +1, ...
all, your, base, ...

Formally, predict $y_t$ from $y_1, y_2, \ldots, y_{t-2}, y_{t-1}$

$$y_t = h(y_{t-1}^{t-1}) \quad y_i^j := y_i, \ldots, y_j$$
open(), read(), read(), ...
G, A, T, T, A, C, ...
+1, −1, +1, −1, +1, ...
all, your, base, ...

Formally, predict \( y_t \) from \( y_1, y_2, \ldots, y_{t-2}, y_{t-1} \)

\[
y_t = h(y_{t-1}^{t-1}) \quad y_i^j := y_i, \ldots, y_j
\]

More general problem: \( p(y_t|y_{1}^{t-1}) \)
open(), read(), read(), ...
G, A, T, T, A, C, ...
+1, −1, +1, −1, +1, ...
all, your, base, ...

Formally, predict $y_t$ from $y_1, y_2, \ldots, y_{t-2}, y_{t-1}$

$$y_t = h(y_{t-1}^{t-1}) \quad y_i^j := y_i, \ldots, y_j$$

More general problem: $p(y_t | y_1^{t-1})$

Markovian Assumption
Markov Models

Zero order: \( p(y_t|y_{1}^{t-1}) = p(y_t) \)
Markov Models

Zero order: \( p(y_t | y_{1:t-1}) = p(y_t) \)

First order:

\[
\begin{align*}
\text{A} & \\
\text{C} & \\
\text{G} & \\
\text{T} & \\
\hline
p(y_t | y_{t-1} = \text{G}')
\end{align*}
\]
Markov Models

Second order:

\[ y_{t-1} \]

\[ y_{t-2} \]

\[ y_{t-2} \]

\[ y_{t-2} \]

\[ y_{t-2} \]

\[ y_{t-2} \]

\[ p(y_t | y_{t-1} = C, y_{t-2} = G) \]
Second order:

\[ p(y_t | y_{t-1} = C, y_{t-2} = G) \]

Third order, \( k \)-th order etc.
How to select the best $k$?

Number of model parameters exponential in $k$.

Poor family of models. What if I want a model with 100 parameters?

What if $y_t$ does not depend on $y_{t-2}$ if $y_{t-1} = A$ but it depends on it if $y_{t-1} = T$?

Prediction Suffix Trees address these problems.

Markov Models — Problems

How to select the best $k$?

Number of model parameters exponential in $k$. 

Prediction Suffix Trees address these problems.

Markov Models — Problems

How to select the best $k$?

Number of model parameters exponential in $k$.

Poor family of models. What if I want a model with 100 parameters?

Prediction Suffix Trees address these problems.

Markov Models — Problems

How to select the best $k$?

Number of model parameters exponential in $k$.

Poor family of models. What if I want a model with 100 parameters?

What if $y_t$ does not depend on $y_{t-2}$ if $y_{t-1} = A$ but it depends on it if $y_{t-1} = T$?
Markov Models — Problems

How to select the best $k$?

Number of model parameters exponential in $k$.

Poor family of models. What if I want a model with 100 parameters?

What if $y_t$ does not depend on $y_{t-2}$ if $y_{t-1} = A$ but it depends on it if $y_{t-1} = T$?

Prediction Suffix Trees address these problems.

Prediction Suffix Trees

Assumptions:

\[ p(y_t | y_{t-1} = T, y_{t-2}) = p(y_t | y_{t-1} = T) \],
\[ p(y_t | y_{t-1} = A, y_{t-2} = \neg A) = p(y_t | y_{t-1} = A) \].
Prediction Suffix Trees

Keep a model for each $k$. Use a weighted sum.

$\mathbf{A} C \ G \ T$

Assumptions:

$p(y_t | y_{t-1} = T, y_{t-2}) = p(y_t | y_{t-1} = T)$,

$p(y_t | y_{t-1} = A, y_{t-2} = \neg A) = p(y_t | y_{t-1} = A)$.
Prediction Suffix Trees

Keep a model for each $k$. Use a weighted sum. **Prune** useless branches and subtrees.

$\Pr(y_t)$

- $p(y_t|A)$
  - $p(y_t|y_{t-1}=T, y_{t-2})$

  - $p(y_t|A)$
    - $p(y_t|C)$
      - $p(y_t|G)$
        - $p(y_t|T)$

- $p(y_t|C)$
- $p(y_t|G)$
- $p(y_t|T)$
Prediction Suffix Trees

Keep a model for each $k$. Use a weighted sum. Prune useless branches and subtrees.

Assumptions: $p(y_t|y_{t-1} = T, y_{t-2}) = p(y_t|y_{t-1} = T), \quad p(y_t|y_{t-1} = A, y_{t-2} = \neg A) = p(y_t|y_{t-1} = A).$
We just want to predict $y_t$ from $y_{t-1}$
We just want to predict $y_t$ from $y_{t-1}^{t-1}$

Assume $y_t \in \{-1, +1\}$
We just want to predict \( y_t \) from \( y_{1}^{t-1} \)

Assume \( y_t \in \{-1, +1\} \)

Store a weight instead of a probability
Moving Away from Probabilistic Modeling

We just want to predict $y_t$ from $y_1^{t-1}$

Assume $y_t \in \{-1, +1\}$

Store a weight instead of a probability

Use the sign of a weighted sum of the values in the appropriate path
Example

True Sequence: $-1, -1, +1, -1, -1, +1, \ldots$
Discounting: A node at depth $d$ is discounted by $2^{-d}$
Tree:

Input Sequence:
Decision:
Example

True Sequence: $-1, -1, +1, -1, -1, +1, \ldots$
Discounting: A node at depth $d$ is discounted by $2^{-d}$
Tree:

```
          0
         / \  /
        /   +1
       -1   -2
      / \   /
     /   -1
    5
```

Input Sequence: $\ldots, -1, -1$
Decision: 0
Example

True Sequence: $-1, -1, +1, -1, -1, +1, \ldots$

Discounting: A node at depth $d$ is discounted by $2^{-d}$

Tree:

Input Sequence: $\ldots, -1, -1$

Decision: $0 - \frac{1}{2} \cdot 1$
Example

True Sequence: $-1, -1, +1, -1, -1, +1, \ldots$

Discounting: A node at depth $d$ is discounted by $2^{-d}$

Tree:

```
   0
  /   \
-1    +1
 / \
-1 -2
 / \
-1 5
```

Input Sequence: $\ldots, -1, -1$

Decision: $0 - \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 5$
Example

True Sequence: \(-1, -1, 1, -1, -1, 1, \ldots\)

Discounting: A node at depth \(d\) is discounted by \(2^{-d}\)

Tree:

```
   0
  / \  \
-1   +1
 /    \
-2    
```

Input Sequence: \(\ldots, -1, -1\)

Decision: 
\[
0 - \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 5 = \frac{3}{4} \xrightarrow{\text{sign}} +1
\]
Example

True Sequence: $-1, -1, +1, -1, -1, +1, \ldots$

Discounting: A node at depth $d$ is discounted by $2^{-d}$

Tree:

```
  0
 / \    \
-1   +1
   / \   \
-1   -2
  /   / \
-1   5   \
```

Input Sequence: $\ldots, +1, -1$

Decision: $0$
Example

True Sequence: $-1, -1, +1, -1, -1, +1, \ldots$
Discounting: A node at depth $d$ is discounted by $2^{-d}$

Tree:

```
0
  /   \
-1    +1
  /  \
-1  -2
  /  \
5  -1
```

Input Sequence: $\ldots, +1, -1$
Decision: $0 - \frac{1}{2} \cdot 1$
Example

True Sequence: \(-1, -1, +1, -1, -1, +1, \ldots\)

Discounting: A node at depth \(d\) is discounted by \(2^{-d}\)

Tree:

```
    0
   /\  \
  -1 +1
 /\    \
-1   -2
 /\    \\  \
5   -1   
```

Input Sequence: \(\ldots, +1, -1\)

Decision: \(0 - \frac{1}{2} \cdot 1 = -\frac{1}{2} \xrightleftharpoons{\text{sign}} -1\)
Example

True Sequence: \(-1, -1, +1, -1, -1, +1, \ldots\)

Discounting: A node at depth \(d\) is discounted by \(2^{-d}\)

Tree:

```
  0
 / \  /  \
-1 +1
 /   \
-1   -2
  /   \
-1   -2
 /   \
 5
```

Input Sequence: \(\ldots, +1\)

Decision: 0
Example

True Sequence: \(-1, -1, +1, -1, -1, +1, \ldots\)
Discounting: A node at depth \(d\) is discounted by \(2^{-d}\)
Tree:

Input Sequence: \(\ldots, +1\)
Decision: \(0 - \frac{1}{2} \cdot 2\)
Example

True Sequence: $-1, -1, +1, -1, -1, +1, \ldots$
Discounting: A node at depth $d$ is discounted by $2^{-d}$

Tree:

Decision: $0 - \frac{1}{2} \cdot 2 = -1 \Rightarrow -1$
Notation for the node values: $g_{t,s}$
Notation for the node values: $g_{t,s}$

Discounting scheme in this work:

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

The best value of $\epsilon$ will be determined (much) later.
Notation for the node values: $g_{t,s}$

Discounting scheme in this work:

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

The best value of $\epsilon$ will be determined (much) later.

Decision at time $t$: \(\text{sign} \left( \sum_s g_{t,s} x_{t,s}^+ \right) = \text{sign} \langle g_t, x_t^+ \rangle\)
Outline

Prediction Suffix Trees

Algorithm and Properties

Results
Outline

Prediction Suffix Trees

Algorithm and Properties

Results
SVMs, logistic regression, ...
Algorithms for Linear Prediction

SVMs, logistic regression, ... 

Focus on online setting
Algorithms for Linear Prediction

SVMs, logistic regression, ...

Focus on online setting

Algorithm maintains hypothesis $w_t$. In each round:

The algorithm receives information $x_t$. It outputs $\hat{y}_t = \text{sign}(\langle w_t, x_t \rangle)$. It receives the correct output $y_t$. It updates: $w_{t+1} \leftarrow f(w_t)$. A mistake is made when $y_t \langle w_t, x_t \rangle \leq 0$. 

Nikos Karampatziakis, Dexter Kozen

Online Learning of Prediction Suffix Trees
SVMs, logistic regression, . . .

Focus on online setting

Algorithm maintains hypothesis $w_t$. In each round:
  The algorithm receives information $x_t$. 
  It outputs $\hat{y}_t = \text{sign}(\langle w_t, x_t \rangle)$.
  It receives the correct output $y_t$.
  It updates: $w_{t+1} \leftarrow f(w_t)$. 
  A mistake is made when $y_t \langle w_t, x_t \rangle \leq 0$. 

Nikos Karampatziakis, Dexter Kozen

Online Learning of Prediction Suffix Trees
SVMs, logistic regression, . . .

Focus on online setting

Algorithm maintains hypothesis $w_t$. In each round:
  - The algorithm receives information $x_t$.
  - It outputs $\hat{y}_t = \text{sign}(\langle w_t, x_t \rangle)$.
SVMs, logistic regression, ... 

Focus on online setting 

Algorithm maintains hypothesis $w_t$. In each round:

- The algorithm receives information $x_t$. 
- It outputs $\hat{y}_t = \text{sign}(\langle w_t, x_t \rangle)$. 
- It receives the correct output $y_t$. 
Algorithms for Linear Prediction

SVMs, logistic regression, ... 

Focus on online setting

Algorithm maintains hypothesis $w_t$. In each round:

The algorithm receives information $x_t$.
It outputs $\hat{y}_t = \text{sign}(\langle w_t, x_t \rangle)$.
It receives the correct output $y_t$.
It updates: $w_{t+1} \leftarrow f(w_t)$. 
SVMs, logistic regression,...

Focus on online setting

Algorithm maintains hypothesis $w_t$. In each round:

- The algorithm receives information $x_t$.
- It outputs $\hat{y}_t = \text{sign}(\langle w_t, x_t \rangle)$.
- It receives the correct output $y_t$.
- It updates: $w_{t+1} \leftarrow f(w_t)$.

A mistake is made when $y_t \langle w_t, x_t \rangle \leq 0$. 
If a mistake is made at round $t$: $w_{t+1} = w_t + \alpha y_t x_t$
If a mistake is made at round $t$: $w_{t+1} = w_t + \alpha y_t x_t$

$\alpha$ is a learning rate.
If a mistake is made at round $t$: $w_{t+1} = w_t + \alpha y_t x_t$

$\alpha$ is a learning rate.

It is selected so that it optimizes various tradeoffs.
Balanced Winnow [Littlestone, 1989]
\[
\theta_1 \leftarrow 0
\]
\[
\text{for } t = 1, 2, \ldots, T \text{ do}
\]
\[
\begin{align*}
    w_{t,i} & \leftarrow \frac{e^{\theta_{t,i}}}{\sum_j e^{\theta_{t,j}}} \\
    \hat{y}_t & \leftarrow \langle w_t, x_t \rangle \\
    \text{if } y_t \hat{y}_t \leq 0 & \\
        \theta_{t+1} & \leftarrow \theta_t + \alpha y_t x_t \\
    \text{else } & \\
        \theta_{t+1} & \leftarrow \theta_t
\end{align*}
\]

Perceptron [Rosenblatt, 1958]
\[
\theta_1 \leftarrow 0
\]
\[
\text{for } t = 1, 2, \ldots, T \text{ do}
\]
\[
\begin{align*}
    w_t & \leftarrow \theta_t \\
    \hat{y}_t & \leftarrow \langle w_t, x_t \rangle \\
    \text{if } y_t \hat{y}_t \leq 0 & \\
        \theta_{t+1} & \leftarrow \theta_t + \alpha y_t x_t \\
    \text{else } & \\
        \theta_{t+1} & \leftarrow \theta_t
\end{align*}
\]
Balanced Winnow  \[ \text{[Littlestone, 1989]} \]
\[
\begin{align*}
\theta_1 & \leftarrow 0 \\
\text{for } t = 1, 2, \ldots, T & \text{ do} \\
\theta_{t+1} & \leftarrow \theta_t + \alpha y_t x_t \\
\end{align*}
\]

Perceptron  \[ \text{[Rosenblatt, 1958]} \]
\[
\begin{align*}
\theta_1 & \leftarrow 0 \\
\text{for } t = 1, 2, \ldots, T & \text{ do} \\
\theta_{t+1} & \leftarrow \theta_t + \alpha y_t x_t \\
\end{align*}
\]

Important for Balanced Winnow:
\[
x_t = [x_t^+, -x_t^+] = [x_{t,1}^+, \ldots, x_{t,d}^+, -x_{t,1}^+, \ldots, -x_{t,d}^+] \\
\]
\[
x_{t,s}^+ = \begin{cases} 
(1 - \epsilon)^{|s|} & \text{if } s = y_{t-i}^{t-1} \\
0 & \text{otherwise} 
\end{cases} \\
i = 1, \ldots, t - 1 \]
Advantages

Multiplicative updates — fast convergence.
Advantages

Multiplicative updates — fast convergence.

Can cope with many features when few of them are relevant.
Advantages

Multiplicative updates — fast convergence.

Can cope with many features when few of them are relevant.

For our application it can track changes better.
Where’s the catch?

Perceptron and Winnow don’t care about the sparsity of their hypothesis.
Where’s the catch?

Perceptron and Winnow don’t care about the sparsity of their hypothesis.

Eventually, we have a feature for every substring of $y_1^T$. 
Where’s the catch?

Perceptron and Winnow don’t care about the sparsity of their hypothesis.

Eventually, we have a feature for every substring of $y_1^T$.

Implications:

Need to learn $O(T^2)$ weights in $T$ rounds 😞
Where’s the catch?

Perceptron and Winnow don’t care about the sparsity of their hypothesis.

Eventually, we have a feature for every substring of $y_1^T$.

Implications:

Need to learn $O(T^2)$ weights in $T$ rounds 😞

Need to store $O(T^2)$ numbers 😞

Naively, all weights affect the decision and must be stored.
Does the decision depend on every feature?

Recall the specific form of features: \( x_t = [x_t^+, x_t^-] = [x_t^+, -x_t^+] \).
Does the decision depend on every feature?

Recall the specific form of features: 
\[ x_t = [x_t^+, x_t^-] = [x_t^+, -x_t^+]. \]

Notation: 
\[ w_t = [w_t^+, w_t^-] \quad \theta_t = [\theta_t^+, \theta_t^-]. \]

Recall 
\[ w_{t,i} = \frac{e^{\theta_{t,i}}}{\sum_j e^{\theta_{t,j}}} \]

\[ \sinh(x) = \frac{e^x - e^{-x}}{2} \quad \text{and} \quad \cosh(x) = \frac{e^x + e^{-x}}{2} \]
Does the decision depend on every feature?

Recall the specific form of features: \( x_t = [x_t^+, x_t^-] = [x_t^+, -x_t^+] \).

Notation: \( w_t = [w_t^+, w_t^-] \quad \theta_t = [\theta_t^+, \theta_t^-] \). Recall \( w_{t,i} = \frac{e^{\theta_{t,i}}}{\sum_j e^{\theta_{t,j}}} \)

Easy inductive argument can show that \( \theta_t^- = -\theta_t^+ \).
Does the decision depend on every feature?

Recall the specific form of features: \( x_t = [x^+_t, x^-_t] = [x^+_t, -x^+_t] \).

Notation: \( w_t = [w^+_t, w^-_t] \quad \theta_t = [\theta^+_t, \theta^-_t] \). Recall \( w_{t,i} = \frac{e^{\theta_{t,i}}}{\sum_j e^{\theta_{t,j}}} \).

Easy inductive argument can show that \( \theta^-_t = -\theta^+_t \).

Decision \( \hat{y}_t = \langle w_t, x_t \rangle = \langle w^+_t - w^-_t, x^+_t \rangle = 1 \)

\[
= \sum_{i=1}^{d} \frac{\sinh(\theta^+_t)}{\sum_{j=1}^{d} \cosh(\theta^+_t)} x^+_t, i \propto \sum_{i=1}^{d} \sinh(\theta^+_t) x^+_t, i
\]

\( \sinh(x) = \frac{e^x - e^{-x}}{2} \) and \( \cosh(x) = \frac{e^x + e^{-x}}{2} \).
Does the decision depend on every feature?

Recall the specific form of features: \( x_t = [x_t^+, x_t^-] = [x_t^+, -x_t^+] \).

Notation: \( w_t = [w_t^+, w_t^-] \), \( \theta_t = [\theta_t^+, \theta_t^-] \). Recall \( w_{t,i} = \frac{e^{\theta_{t,i}}}{\sum_j e^{\theta_{t,j}}} \)

Easy inductive argument can show that \( \theta_t^- = -\theta_t^+ \)

Decision \( \hat{y}_t = \langle w_t, x_t \rangle = \langle w_t^+ - w_t^-, x_t^+ \rangle = 1 \)

\[
= \sum_{i=1}^{d} \frac{\sinh(\theta_{t,i}^+)}{\sum_{j=1}^{d} \cosh(\theta_{t,j}^+)} x_{t,i}^+ \propto \sum_{i=1}^{d} \sinh(\theta_{t,i}^+) x_{t,i}^+
\]

Iff \( \theta_{t,i}^+ = 0 \) the decision does not depend on feature \( i \).

\[
\sinh(x) = \frac{e^x - e^{-x}}{2} \text{ and } \cosh(x) = \frac{e^x + e^{-x}}{2}
\]
If $\theta_{t,s} = 0$ we don’t have to store it.
Observations

If $\theta_{t,s} = 0$ we don’t have to store it.

Initially $\theta_1 = 0$. The tree has only one node.
If $\theta_{t,s} = 0$ we don’t have to store it.

Initially $\theta_1 = 0$. The tree has only one node.

As mistakes are made, the tree grows.
If $\theta_{t,s} = 0$ we don’t have to store it.

Initially $\theta_1 = 0$. The tree has only one node.

As mistakes are made, the tree grows.

Classic Winnow/Perceptron update quickly destroys sparsity.
Input Sequence: \ldots, -1, +1, -1, +1, -1, ?

\[ x_{t,s}^+ = \begin{cases} 
2^{-|s|} & \text{if } s = y_{t-i}^{t-1} \quad i = 1, \ldots, t - 1 \\
0 & \text{otherwise}
\end{cases} \]

Tree:

Decision:
Input Sequence: \ldots, -1, +1, -1, +1, -1, ?

\[ x_{t,s}^+ = \begin{cases} 
2^{-|s|} & \text{if } s = y_{t-i}^{t-1}, \quad i = 1, \ldots, t - 1 \\
0 & \text{otherwise}
\end{cases} \]

Tree:

Decision: \[ \frac{1}{2} \cdot \sinh(1) \]
Input Sequence: \(\ldots, -1, +1, -1, +1, -1, ?\)

\[ x_{t,s}^+ = \begin{cases} 
  2^{-|s|} & \text{if } s = y_{t-i}^{t-1} \\
  0 & \text{otherwise}
\end{cases} \quad i = 1, \ldots, t - 1 \]

Tree:

Decision: \( \frac{1}{2} \cdot \sinh(1) > 0 \)
Input Sequence: \( \ldots, -1, +1, -1, +1, -1, ? \)

\[
x^+_t,s = \begin{cases} 
2^{-|s|} & \text{if } s = y^{t-1}_{t-i} \quad i = 1, \ldots, t - 1 \\
0 & \text{otherwise}
\end{cases}
\]

Tree:

```
         0
        / \  \
      -1   1
     / \   /  \\
  -1   +1 -2
 /     /   \
-5
```

Decision: \( \frac{1}{2} \cdot \sinh(1) > 0 \overset{\text{sign}}{\rightarrow} +1 \)
Illustration

Input Sequence: \ldots, -1, +1, -1, +1, -1, -1

\[ x_t^+ = \begin{cases} 
2^{-|s|} & \text{if } s = y_{t-i}^{t-1} \\
0 & \text{otherwise}
\end{cases} \quad i = 1, \ldots, t - 1 \]

Tree:

Decision: \[ \frac{1}{2} \cdot \sinh(1) > 0 \quad \text{sign} \rightarrow +1 \]
Input Sequence: \( \ldots, -1, +1, -1, +1, -1, -1 \)

\[
x_{t,s}^+ = \begin{cases} 
2^{-|s|} & \text{if } s = y_{t-i}^{t-1} \quad i = 1, \ldots, t - 1 \\
0 & \text{otherwise}
\end{cases}
\]

Tree:

Decision: \( \frac{1}{2} \cdot \sinh(1) > 0 \overset{\text{sign}}{\Rightarrow} +1 \)
Input Sequence: \ldots, -1, +1, -1, +1, -1, -1

\[ x_{t,s}^+ = \begin{cases} 
2^{-|s|} & \text{if } s = y_{t-i}^{t-1} \\
0 & \text{otherwise}
\end{cases} \]

Tree:

Decision: \( \frac{1}{2} \cdot \sinh(1) > 0 \) \( \Rightarrow +1 \)
Mistake at time $t$: $O(t)$ nodes are inserted in the tree.
Summary of the Problem

Mistake at time $t$: $O(t)$ nodes are inserted in the tree.

$x^+_{t,s}$ is non-zero even when $s$ is very long.
Summary of the Problem

Mistake at time $t$: $O(t)$ nodes are inserted in the tree.

$x^+_{t,s}$ is non-zero even when $s$ is very long.

**Bad idea:** change the definition of $x^+_{t,s}$ to avoid this.
Mistake at time $t$: $O(t)$ nodes are inserted in the tree.

$x_{t,s}^+$ is non-zero even when $s$ is very long.

Bad idea: change the definition of $x_{t,s}^+$ to avoid this.

Need to learn a good $\theta$ while keeping it sparse.
Mistake at time $t$: $O(t)$ nodes are inserted in the tree.

$x^+_{t,s}$ is non-zero even when $s$ is very long.

Bad idea: change the definition of $x^+_{t,s}$ to avoid this.

Need to learn a good $\theta$ while keeping it sparse.

Not all sparse vectors are equal.
Better Update Rule

Adaptive bound $d_t$ on the length of the suffixes.
Better Update Rule

Adaptive bound $d_t$ on the length of the suffixes.
Same as the depth up to which the tree will grow on round $t$. 

New update:

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

Equivalently:

$$\theta_{t+1} = \theta_t + \alpha y_t x_t + \alpha n_t$$

where $n_t$ is a noise vector that cancels part of the update:

$$n_{t, s} = \begin{cases} 
-y_t x_{t, s} & \text{if } |s| > d_t \\
0 & \text{otherwise}
\end{cases}$$
Better Update Rule

Adaptive bound $d_t$ on the length of the suffixes.
Same as the depth up to which the tree will grow on round $t$.

$d_t$ will be growing slowly if necessary.
Better Update Rule

Adaptive bound $d_t$ on the length of the suffixes.
Same as the depth up to which the tree will grow on round $t$.

$d_t$ will be growing slowly if necessary.

New update:

$$\theta_{t+1,s} = \begin{cases} 
\theta_{t,s} + \alpha y_t x_{t,s} & \text{if } |s| \leq d_t \\
\theta_{t,s} & \text{otherwise}
\end{cases}$$
Better Update Rule

Adaptive bound $d_t$ on the length of the suffixes.
   Same as the depth up to which the tree will grow on round $t$.

$d_t$ will be growing slowly if necessary.

New update:

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

Equivalently:

$$\theta_{t+1} = \theta_t + \alpha y_t x_t + \alpha n_t$$

where $n_t$ is a noise vector that cancels part of the update:

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

$h_t = \text{the length of the path from the root using } y_{t-1}, y_{t-2}, \ldots$
Setting $d_t$

$h_t = \text{the length of the path from the root using } y_{t-1}, y_{t-2}, \ldots$

$J_t$ is the subset of rounds $1, \ldots, t$ in which a mistake was made.
Setting $d_t$

$$h_t = \text{the length of the path from the root using } y_{t-1}, y_{t-2}, \ldots$$

$J_t$ is the subset of rounds $1, \ldots, t$ in which a mistake was made.

$$P_t = \sum_{i \in J_t} \|n_i\|_\infty = \sum_{i \in J_t} (1 - \epsilon)^{(d_i+1)}.$$
Setting $d_t$

$h_t = \text{the length of the path from the root using } y_{t-1}, y_{t-2}, \ldots$.

$J_t$ is the subset of rounds $1, \ldots, t$ in which a mistake was made.

$P_t = \sum_{i \in J_t} ||n_i||_\infty = \sum_{i \in J_t} (1 - \epsilon)^{(d_i+1)}$.

$d_t$ will be the smallest integer such that $P_t \leq |J_t|^{2/3}$. 
Setting $d_t$

$h_t = \text{the length of the path from the root using } y_{t-1}, y_{t-2}, \ldots$

$J_t$ is the subset of rounds $1, \ldots, t$ in which a mistake was made.

$$P_t = \sum_{i \in J_t} ||n_i||_{\infty} = \sum_{i \in J_t} (1 - \epsilon)^{(d_i+1)}.$$  

$d_t$ will be the smallest integer such that $P_t \leq |J_t|^{2/3}$.

To guarantee that we can set

$$d_t = \max \left\{ h_t, \left\lfloor \log_{1-\epsilon} \left( \sqrt[3]{P_{t-1}^3} + 2P_{t-1}^{3/2} + 1 - P_{t-1} \right) - 1 \right\rfloor \right\}$$
Mistake Bound

If there exists a tree which over the input sequence $y_1, y_2, \ldots, y_T$ correctly predicts all items with confidence $\geq \delta$, our algorithm’s mistakes will be at most

$$\max \left\{ \frac{8 \log T}{\delta^2}, \frac{64}{\delta^3} \right\}$$
Mistake Bound

If there exists a tree which over the input sequence $y_1, y_2, \ldots, y_T$ correctly predicts all items with confidence $\geq \delta$, our algorithm’s mistakes will be at most

$$\max \left\{ \frac{8 \log T}{\delta^2}, \frac{64}{\delta^3} \right\}$$

Instead of assuming a perfect tree, assume a tree $u$ that attains a cumulative $\delta$-hinge loss $L > 0$. The mistake bound becomes:

$$\max \left\{ \frac{2L}{\delta} + \frac{8 \log T}{\delta^2}, \frac{64}{\delta^3} \right\}$$
Mistake Bound

If there exists a tree which over the input sequence $y_1, y_2, \ldots, y_T$ correctly predicts all items with confidence $\geq \delta$, our algorithm’s mistakes will be at most

$$\max \left\{ \frac{8 \log T}{\delta^2}, \frac{64}{\delta^3} \right\}$$

Instead of assuming a perfect tree, assume a tree $u$ that attains a cumulative $\delta$-hinge loss $L > 0$. The mistake bound becomes:

$$\max \left\{ \frac{2L}{\delta} + \frac{8 \log T}{\delta^2}, \frac{64}{\delta^3} \right\}$$

$$L = \sum_{t=1}^{T} \ell_t(u) \text{ where } \ell_t(u) = \max(0, \delta - y_t(u, x_t))$$
Let $M_t$ be the number of mistakes up to round $t$.
Let $M_t$ be the number of mistakes up to round $t$

Set $\epsilon$ so that

$$x_{t,s}^+ = \begin{cases} 
2^{-|s|/3} & \text{if } s = y_{t-i}^{t-1} \quad i = 1, \ldots, t - 1 \\
0 & \text{otherwise}
\end{cases}$$
Let $M_t$ be the number of mistakes up to round $t$

Set $\epsilon$ so that

$$x_{t,s}^+ = \begin{cases} 
2^{-|s|/3} & \text{if } s = y_{t-i}^{t-1} \quad i = 1, \ldots, t-1 \\
0 & \text{otherwise}
\end{cases}$$

Growth Bound

Our algorithm will not grow a tree deeper than $\log_2 M_{T-1} + 4$
How to go about proving the properties

Bounding how fast the tree grows is straightforward.
How to go about proving the properties

Bounding how fast the tree grows is straightforward.

Mistake bound: show that each update contributes towards a goal.
How to go about proving the properties

Bounding how fast the tree grows is straightforward.

Mistake bound: show that each update contributes towards a goal.

We need a goal and measure of progress.
Goal: A fixed tree with good performance.
Measuring Progress

Goal: A fixed tree with good performance.
For example, a vector $u$ such that $y_t \langle u, x_t \rangle \geq \delta$. 
Measuring Progress

Goal: A fixed tree with good performance.

For example, a vector $u$ such that $y_t \langle u, x_t \rangle \geq \delta$.

Our adaptive tree is represented by $w_t$. Remember $\sum_i w_{t,i} = 1$. 
Goal: A fixed tree with good performance.

For example, a vector $u$ such that $y_t \langle u, x_t \rangle \geq \delta$.

Our adaptive tree is represented by $w_t$. Remember $\sum_i w_{t,i} = 1$.

To be fair assume $u_i \geq 0$ and $\sum_i u_i = 1$.
Goal: A fixed tree with good performance.

For example, a vector $u$ such that $y_t \langle u, x_t \rangle \geq \delta$.

Our adaptive tree is represented by $w_t$. Remember $\sum_i w_{t,i} = 1$.

To be fair assume $u_i \geq 0$ and $\sum_i u_i = 1$

Measure of progress: Relative entropy between $u$ and $w_t$

$$D(u||w_t) = \sum_i u_i \log \frac{u_i}{w_{t,i}}$$
Measuring Progress

Goal: A fixed tree with good performance.
    For example, a vector $u$ such that $y_t\langle u, x_t \rangle \geq \delta$.

Our adaptive tree is represented by $w_t$. Remember $\sum_i w_{t,i} = 1$.

To be fair assume $u_i \geq 0$ and $\sum_i u_i = 1$

Measure of progress: Relative entropy between $u$ and $w_t$

$$D(u||w_t) = \sum_i u_i \log \frac{u_i}{w_{t,i}}$$

Potential function: $\Phi(w_t) = D(u||w_t) \geq 0$
Proof Technique

Upper bound the initial potential.
Proof Technique

Upper bound the initial potential.

Lower bound the change in the potential with each update.
Proof Technique

Upper bound the initial potential.

Lower bound the change in the potential with each update.

Keep the total effect of noise bounded. [Dekel et al., 2004]
Potential: \( \Phi(w_1) \)

Noise \( P_t \):

Example Correct Prediction

\( x_1 \)

Progress due to classic update

Effect of noise

Net progress
Pictorially

Potential: $\Phi(w_1)$

Noise $P_t$:

Example Correct Prediction

$x_1$ $\times$

Progress due to classic update
Effect of noise
Net progress
Potential: $\Phi(w_1)$

Noise $P_t$:

Example Correct Prediction $x_1$
Pictorially

Potential: \( \Phi(w_1) \)

Noise \( P_t \):

Example Correct Prediction
\[ x_1 \]
\[ x_2 \]

Progress due to classic update
Effect of noise
Net progress
Pictorially

Potential: $\Phi(w_1)$

Noise $P_t$:

Example Correct Prediction

$\times$

$x_1$

$x_2$

Progress due to classic update
Effect of noise
Net progress

Nikos Karampatziakis, Dexter Kozen
Online Learning of Prediction Suffix Trees
Pictorially

Potential: $\Phi(w_1)$

Noise $P_t$:

Example Correct Prediction
$x_1 \times$
$x_2 \times$

Progress due to classic update
Effect of noise
Net progress
Pictorially

Potential: \( \Phi(w_1) \)

Noise \( P_t \):

Example Correct Prediction
\( x_1 \) 
\( x_2 \) 
\( x_3 \)

Progress due to classic update
Effect of noise
Net progress

Nikos Karampatziakis, Dexter Kozen
Online Learning of Prediction Suffix Trees
Pictorially

Potential: $\Phi(w_1)$

Noise $P_t$:

Example: $x_1$, $x_2$, $x_3$
Correct Prediction: $\times$, $\times$, $\checkmark$

Progress due to classic update
Effect of noise
Net progress
Potential: $\Phi(w_1)$

Noise $P_t$:

Example  Correct Prediction

$x_1$  
$x_2$  
$x_3$  
$x_4$  

Progress due to classic update
Effect of noise
Net progress
Pictorially

Potential: \[ \Phi(w_1) \]

Noise \( P_t \):

Example Correct Prediction
\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ x_4 \]

Progress due to classic update
Effect of noise
Net progress
Potential: $\Phi(w_1)$

Noise $P_t$:

Example  Correct Prediction

$x_1$  
$x_2$  
$x_3$  ✔
$x_4$  ✗

Progress due to classic update
Effect of noise
Net progress
Potential: $\Phi(w_1)$

Noise $P_t$:

Example Correct Prediction

$x_1$ ×
$x_2$ ×
$x_3$ ✓
$x_4$ ×
$x_5$ ×

Progress due to classic update
Effect of noise
Net progress
Pictorially

Potential: \[ \Phi(w_1) \]

Noise \( P_t \):

<table>
<thead>
<tr>
<th>Example</th>
<th>Correct Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>( x_4 )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( x_5 )</td>
<td>( \checkmark )</td>
</tr>
</tbody>
</table>

Progress due to classic update
Effect of noise
Net progress
Potential: $\Phi(w_1)$

Noise $P_t$:

Example | Correct Prediction
---|---
$x_1$ | $\times$
$x_2$ | $\times$
$x_3$ | $\checkmark$
$x_4$ | $\times$
$x_5$ | $\checkmark$
$x_6$ | $\checkmark$

Progress due to classic update
Effect of noise
Net progress
Pictorially

Potential: \( \Phi(w_1) \)

Noise \( P_t \):

Example | Correct Prediction
---|---
\( x_1 \) | \xmark
\( x_2 \) | \xmark
\( x_3 \) | \cmark
\( x_4 \) | \xmark
\( x_5 \) | \cmark
\( x_6 \) | \xmark

Progress due to classic update
Effect of noise
Net progress
Pictorially

Potential: \( \Phi(w_1) \)

Noise \( P_t \):

Example  Correct Prediction
\( x_1 \)  \( \times \)
\( x_2 \)  \( \times \)
\( x_3 \)  \( \checkmark \)
\( x_4 \)  \( \times \)
\( x_5 \)  \( \checkmark \)
\( x_6 \)  \( \times \)

Progress due to classic update
Effect of noise
Net progress
Pictorially

Potential: \( \Phi(w_1) \)

Noise \( P_t \):

Example | Correct Prediction
--- | ---
\( x_1 \) | \( \times \)
\( x_2 \) | \( \times \)
\( x_3 \) | \( \checkmark \)
\( x_4 \) | \( \times \)
\( x_5 \) | \( \checkmark \)
\( x_6 \) | \( \times \)

Progress due to classic update
Effect of noise
Net progress

Nikos Karampatziakis, Dexter Kozen
Online Learning of Prediction Suffix Trees
length of \leq \Phi(w_1)
Proof

\[
\text{length of } \mathbf{\textcolor{blue}{\text{block}}} \leq \Phi(w_1)
\]

\[
\text{length of } \mathbf{\textcolor{green}{\text{block}}} - \text{length of } \mathbf{\textcolor{red}{\text{block}}} \leq \Phi(w_1)
\]
Proof

\[
\text{length of } \leq \Phi(w_1)
\]

\[
\text{length of } - \text{length of } \leq \Phi(w_1)
\]

\[
\geq \text{min size} \times \text{mistakes}
\]
Proof

\[ \text{length of } \begin{array}{c} \text{blue} \\ \geq \text{min size} \times \text{mistakes} \end{array} \leq \Phi(w_1) \]

\[ - \text{length of } \begin{array}{c} \text{green} \\ \leq \text{mistakes}^{2/3} \end{array} \leq \Phi(w_1) \]
Proof

\[ \text{length of } \Phi(w_1) \leq \Phi(w_1) \]

\[ \text{length of } \geq \text{min size } \times \text{mistakes} \quad - \quad \text{length of } \leq \text{mistakes}^{2/3} \leq \Phi(w_1) \]

\[ \text{min size } \cdot \text{mistakes} - \text{mistakes}^{2/3} \leq \Phi(w_1) \]
Multiclass extension

We maintain weights $w^{(1)}, w^{(2)}, \ldots, w^{(k)}$

Predict $\hat{y}_t = \arg\max_i \langle w^{(i)}, x_t \rangle$

In case of a mistake

$$
\theta^{(\hat{y}_t)}_{t+1,s} = \theta^{(\hat{y}_t)}_{t,s} - \alpha x_{t,s} \\
\theta^{(y_t)}_{t+1,s} = \theta^{(y_t)}_{t,s} + \alpha x_{t,s}
$$

for all $s$ such that $|s| \leq d_t$. 
Outline

Prediction Suffix Trees

Algorithm and Properties

Results
Outline

Prediction Suffix Trees

Algorithm and Properties

Results
120 sequences of system calls from Outlook, Excel and Firefox (40 each).

The monitoring program records 23 different system calls.

A typical sequence contains hundreds of thousands of system calls.
## Results

Averages over the 40 sequences

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Perceptron</th>
<th>Winnow</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Error</td>
<td>5.1</td>
<td>4.43</td>
</tr>
<tr>
<td>PST Size</td>
<td>41239</td>
<td>25679</td>
</tr>
</tbody>
</table>

Moreover, Winnow made less mistakes and grew smaller trees for all 120 sequences.

"Perceptron" is the PST learning algorithm of [Dekel et al., 2004].
## Results

### Averages over the 40 sequences

<table>
<thead>
<tr>
<th></th>
<th>Perceptron</th>
<th>Winnow</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outlook</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Error</td>
<td>5.1</td>
<td>4.43</td>
</tr>
<tr>
<td>PST Size</td>
<td>41239</td>
<td>25679</td>
</tr>
<tr>
<td><strong>Firefox</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Error</td>
<td>14.86</td>
<td>13.88</td>
</tr>
<tr>
<td>PST Size</td>
<td>21081</td>
<td>12662</td>
</tr>
</tbody>
</table>

“Perceptron” is the PST learning algorithm of [Dekel et al., 2004].
## Results

### Averages over the 40 sequences

<table>
<thead>
<tr>
<th></th>
<th>Perceptron</th>
<th>Winnow</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outlook</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Error</td>
<td>5.1</td>
<td>4.43</td>
</tr>
<tr>
<td>PST Size</td>
<td>41239</td>
<td>25679</td>
</tr>
<tr>
<td><strong>Firefox</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Error</td>
<td>14.86</td>
<td>13.88</td>
</tr>
<tr>
<td>PST Size</td>
<td>21081</td>
<td>12662</td>
</tr>
<tr>
<td><strong>Excel</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Error</td>
<td>22.68</td>
<td>20.59</td>
</tr>
<tr>
<td>PST Size</td>
<td>24402</td>
<td>15338</td>
</tr>
</tbody>
</table>

Moreover, Winnow made less mistakes and grew smaller trees for all 120 sequences. “Perceptron” is the PST learning algorithm of [Dekel et al., 2004].
## Results

Averages over the 40 sequences

<table>
<thead>
<tr>
<th></th>
<th>Perceptron</th>
<th>Winnow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlook</td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Error</td>
<td>5.1</td>
<td>4.43</td>
</tr>
<tr>
<td>PST Size</td>
<td>41239</td>
<td>25679</td>
</tr>
<tr>
<td>Firefox</td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Error</td>
<td>14.86</td>
<td>13.88</td>
</tr>
<tr>
<td>PST Size</td>
<td>21081</td>
<td>12662</td>
</tr>
<tr>
<td>Excel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Error</td>
<td>22.68</td>
<td>20.59</td>
</tr>
<tr>
<td>PST Size</td>
<td>24402</td>
<td>15338</td>
</tr>
</tbody>
</table>

Moreover, Winnow made less mistakes and grew smaller trees for all 120 sequences.

“Perceptron” is the PST learning algorithm of [Dekel et al., 2004].
Results

Nikos Karampatziakis, Dexter Kozen

Online Learning of Prediction Suffix Trees
Results

Nikos Karampatziakis, Dexter Kozen

Online Learning of Prediction Suffix Trees
Results

Nikos Karampatziakis, Dexter Kozen  Online Learning of Prediction Suffix Trees
Related Work

Many learning algorithms assume an *a priori* bound on the tree’s depth [Willems et al., 1995], [Ron et al., 1996], [Pereira & Singer, 1999]...

[Dekel et al., 2004] present a perceptron algorithm similar to ours.

[Kivinen & Warmuth, 1997] show how to compete against vectors $u$ with $\|u\|_1 \leq U$

Sparsifying Winnow is popular (e.g. [Blum, 1997]) but no guarantees.
Outlined the benefits of prediction suffix trees.
Outline the benefits of prediction suffix trees.

Introduced an online learning algorithm to learn PSTs.
Outlined the benefits of prediction suffix trees.

Introduced an online learning algorithm to learn PSTs. It is competitive with the best fixed PST in hindsight.
Summary

Outlined the benefits of prediction suffix trees.

Introduced an online learning algorithm to learn PSTs.
   It is competitive with the best fixed PST in hindsight.
   The resulting trees grow slowly if necessary.
Outlined the benefits of prediction suffix trees.

Introduced an online learning algorithm to learn PSTs.
   It is competitive with the best fixed PST in hindsight.
   The resulting trees grow slowly if necessary

On our task, it made less mistakes and grew smaller trees than other state-of-the-art algorithms.


