The Context Algorithm

Yoav Freund

January 20, 2020

Outline

Outline

Outline

Outline

Review

Fixed Length Markov Models

Variable Length Markov Model (VMM)

Universal coding, an inefficient solution

Outline

Outline

The online Bayes Algorithm

- Total loss of expert i

$$L_i^t = -\sum_{s=1}^{t} \log p_i^s(c^s); \quad L_i^0 = 0$$

The online Bayes Algorithm

- Total loss of expert i

$$L_i^t = -\sum_{s=1}^{t} \log p_i^s(c^s); \quad L_i^0 = 0$$

- Weight of expert i

$$w_i^t = w_i^1 e^{-L_i^{t-1}} = w_i^1 \prod_{s=1}^{t-1} p_i^s(c^s)$$

The online Bayes Algorithm

- Total loss of expert i

$$L_i^t = -\sum_{s=1}^{t} \log p_i^s(c^s); \quad L_i^0 = 0$$

- Weight of expert i

$$w_i^t = w_i^1 e^{-L_i^{t-1}} = w_i^1 \prod_{s=1}^{t-1} p_i^s(c^s)$$

- Freedom to choose initial weights.
  $w_t^1 \geq 0, \sum_{i=1}^{n} w_i^1 = 1$

The online Bayes Algorithm

- Total loss of expert i

$$L_i^t = -\sum_{s=1}^{t} \log p_i^s(c^s); \quad L_i^0 = 0$$

- Weight of expert i

$$w_i^t = w_i^1 e^{-L_i^{t-1}} = w_i^1 \prod_{s=1}^{t-1} p_i^s(c^s)$$

- Freedom to choose initial weights.
  $w_t^1 \geq 0, \sum_{i=1}^{n} w_i^1 = 1$
- Prediction of algorithm A

$$\mathbf{p}_A^t = \frac{\sum_{i=1}^{N} w_i^t \mathbf{p}_i^t}{\sum_{i=1}^{N} w_i^t}$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^{N} w_i^t$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t}$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t}$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t} = p_A^t(c^t)$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t} = p_A^t(c^t)$$

$$-\log \frac{W^{t+1}}{W^t} = -\log p_A^t(c^t)$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t} = p_A^t(c^t)$$

$$-\log \frac{W^{t+1}}{W^t} = -\log p_A^t(c^t)$$

$$-\log \frac{W^{T+1}}{W^1} = -\sum_{t=1}^{T} \log p_A^t(c^t)$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t} = p_A^t(c^t)$$

$$-\log \frac{W^{t+1}}{W^t} = -\log p_A^t(c^t)$$

$$-\log \frac{W^{T+1}}{W^1} = -\sum_{t=1}^T \log p_A^t(c^t) = L_A^T$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t} = p_A^t(c^t)$$

$$-\log \frac{W^{t+1}}{W^t} = -\log p_A^t(c^t)$$

$$-\log W^{T+1} = -\log \frac{W^{T+1}}{W^1} = -\sum_{t=1}^{T} \log p_A^t(c^t) = L_A^T$$

Cumulative loss vs. Final total weight

Total weight: $W^t \doteq \sum_{i=1}^N w_i^t$

$$\frac{W^{t+1}}{W^t} = \frac{\sum_{i=1}^N w_i^t e^{\log p_i^t(c^t)}}{\sum_{i=1}^N w_i^t} = \frac{\sum_{i=1}^N w_i^t p_i^t(c^t)}{\sum_{i=1}^N w_i^t} = p_A^t(c^t)$$

$$-\log \frac{W^{t+1}}{W^t} = -\log p_A^t(c^t)$$

$$-\log W^{T+1} = -\log \frac{W^{T+1}}{W^1} = -\sum_{t=1}^{T} \log p_A^t(c^t) = L_A^T$$

EQUALITY not bound!

Simple Bound

- Use non-uniform initial weights $\sum_i w_i^1 = 1$

Simple Bound

- Use non-uniform initial weights $\sum_i w_i^1 = 1$
- Total Weight is at least the weight of the best expert.

$$L_A^T = -\log W^{T+1}$$

Simple Bound

- Use non-uniform initial weights $\sum_i w_i^1 = 1$
- Total Weight is at least the weight of the best expert.

$$L_A^T = -\log W^{T+1}$$

Simple Bound

- Use non-uniform initial weights $\sum_i w_i^1 = 1$
- Total Weight is at least the weight of the best expert.

$$L_A^T \quad = \quad -\log W^{T+1} = -\log \sum_{i=1}^{N} w_i^{T+1}$$

Simple Bound

- Use non-uniform initial weights $\sum_i w_i^1 = 1$
- Total Weight is at least the weight of the best expert.

$$
\begin{aligned}
L_A^T &= -\log W^{T+1} = -\log \sum_{i=1}^{N} w_i^{T+1} \\
&= -\log \sum_{i=1}^{N} w_i^1 e^{-L_i^T}
\end{aligned}
$$

Simple Bound

- Use non-uniform initial weights $\sum_i w_i^1 = 1$
- Total Weight is at least the weight of the best expert.

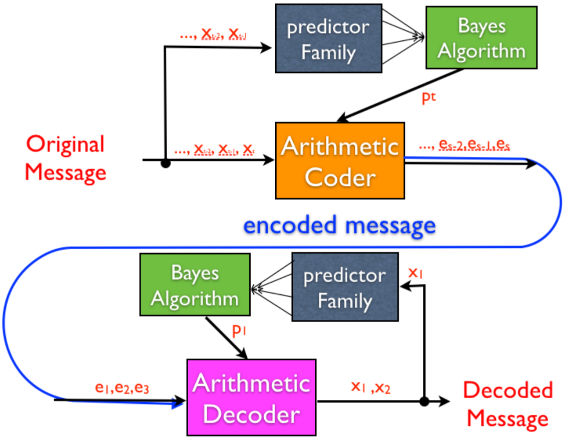
$$
\begin{aligned}
L_A^T &= -\log W^{T+1} = -\log \sum_{i=1}^{N} w_i^{T+1} \\
&= -\log \sum_{i=1}^{N} w_i^1 e^{-L_i^T} \leq -\log \max_i \left(w_i^1 e^{-L_i^T} \right)
\end{aligned}
$$

Simple Bound

- Use non-uniform initial weights $\sum_i w_i^1 = 1$
- Total Weight is at least the weight of the best expert.

$$
\begin{aligned}
L_A^T &= -\log W^{T+1} = -\log \sum_{i=1}^{N} w_i^{T+1} \\
&= -\log \sum_{i=1}^{N} w_i^1 e^{-L_i^T} \leq -\log \max_i \left(w_i^1 e^{-L_i^T} \right) \\
&= \min_i \left(L_i^T - \log w_i^1 \right)
\end{aligned}
$$

Universal Online coding



Combining large predictor families

- Log loss is **mixable** = each predictor in the family can use a Bayesian combination of a family of sub-predictors, with no additional loss.

Combining large predictor families

- Log loss is **mixable** = each predictor in the family can use a Bayesian combination of a family of sub-predictors, with no additional loss.
- We talked about the KT preictor.

Combining large predictor families

- Log loss is **mixable** = each predictor in the family can use a Bayesian combination of a family of sub-predictors, with no additional loss.
- We talked about the KT preictor.
- Today we consider the much richer set of variable length markov models.

Combining large predictor families

- Log loss is **mixable** = each predictor in the family can use a Bayesian combination of a family of sub-predictors, with no additional loss.
- We talked about the KT preictor.
- Today we consider the much richer set of variable length markov models.
- The set of predictors is of exponential size, but the algorithm is efficient.

A fixed length Markov Model

A fixed length Markov Model

A fixed length Markov Model

- Observe a binary sequence.

A fixed length Markov Model

- Observe a binary sequence.
- $x_1, \ldots, x_{t-1}$

A fixed length Markov Model

- Observe a binary sequence.
- $x_1, \ldots, x_{t-1}$
- Predict next bit from past

A fixed length Markov Model

- Observe a binary sequence.
- $x_1, \ldots, x_{t-1}$
- Predict next bit from past
- $P(x_t = 1 | x_{t-1}, x_{t-2}, \ldots, x_1)$

A fixed length Markov Model

- Observe a binary sequence.
- $x_1, \ldots, x_{t-1}$
- Predict next bit from past
- $P(x_t = 1 | x_{t-1}, x_{t-2}, \ldots, x_1)$
- Use only last k bits

A fixed length Markov Model

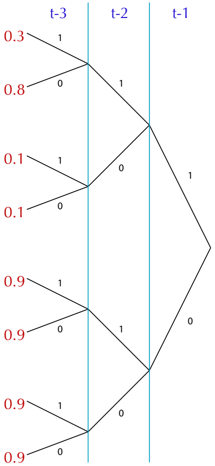
- Observe a binary sequence.
- $x_1, \ldots, x_{t-1}$
- Predict next bit from past
- $P(x_t = 1 | x_{t-1}, x_{t-2}, \ldots, x_1)$
- Use only last k bits
- $P(x_t = 1 | x_{t-1}, \ldots, x_{t-k})$

A fixed length Markov Model

- Observe a binary sequence.
- $x_1, \ldots, x_{t-1}$
- Predict next bit from past
- $P(x_t = 1 | x_{t-1}, x_{t-2}, \ldots, x_1)$
- Use only last k bits
- $P(x_t = 1 | x_{t-1}, \ldots, x_{t-k})$
- Markov model of order k

A fixed length Markov Model

- Observe a binary sequence.
- $x_1, \ldots, x_{t-1}$
- Predict next bit from past
- $P(x_t = 1 | x_{t-1}, x_{t-2}, \ldots, x_1)$
- Use only last k bits
- $P(x_t = 1 | x_{t-1}, \ldots, x_{t-k})$
- Markov model of order k



Learning a markov distribution

- Each tree leaf is associated with a binary sequence $y_1, \ldots, y_k$

Learning a markov distribution

- Each tree leaf is associated with a binary sequence $y_1, \ldots, y_k$
- For each leaf keep two counters:

Learning a markov distribution

- Each tree leaf is associated with a binary sequence $y_1, \ldots, y_k$
- For each leaf keep two counters:
  - $a_{y_1,\ldots,y_k}$ = number of times $x_{t-1} = y_1, \ldots, x_{t-k} = y_k$ and $x_t = 0$

Learning a markov distribution

- Each tree leaf is associated with a binary sequence $y_1, \ldots, y_k$
- For each leaf keep two counters:
  - $a_{y_1,\ldots,y_k}$ = number of times $x_{t-1} = y_1, \ldots, x_{t-k} = y_k$ and $x_t = 0$
  - $b_{y_1,\ldots,y_k}$ = number of times $x_{t-1} = y_1, \ldots, x_{t-k} = y_k$ and $x_t = 1$

Learning a markov distribution

- Each tree leaf is associated with a binary sequence $y_1, \ldots, y_k$
- For each leaf keep two counters:
  - $a_{y_1,\ldots,y_k}$ = number of times $x_{t-1} = y_1, \ldots, x_{t-k} = y_k$ and $x_t = 0$
  - $b_{y_1,\ldots,y_k}$ = number of times $x_{t-1} = y_1, \ldots, x_{t-k} = y_k$ and $x_t = 1$
- Prediction (using Kritchevski Trofimov)

$$p(x_t = 1 | x_{t-1} = y_1, \ldots, x_{t-k} = y_k) = \frac{b_{y_1,\ldots,y_k} + 1/2}{a_{y_1,\ldots,y_k} + b_{y_1,\ldots,y_k} + 1}$$

Learning a markov distribution

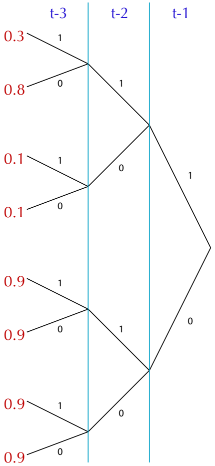
- Each tree leaf is associated with a binary sequence $y_1, \ldots, y_k$
- For each leaf keep two counters:
  - $a_{y_1,\ldots,y_k}$ = number of times $x_{t-1} = y_1, \ldots, x_{t-k} = y_k$ and $x_t = 0$
  - $b_{y_1,\ldots,y_k}$ = number of times $x_{t-1} = y_1, \ldots, x_{t-k} = y_k$ and $x_t = 1$
- Prediction (using Kritchevski Trofimov)

$$p(x_t = 1 | x_{t-1} = y_1, \ldots, x_{t-k} = y_k) = \frac{b_{y_1,\ldots,y_k} + 1/2}{a_{y_1,\ldots,y_k} + b_{y_1,\ldots,y_k} + 1}$$

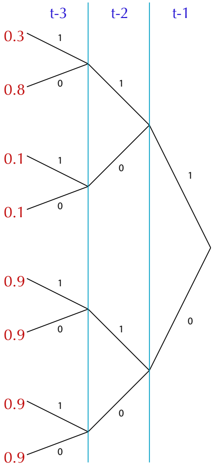
- Total regret is at most $2^{k-1} \log T$

How variable length markov can reduce regret

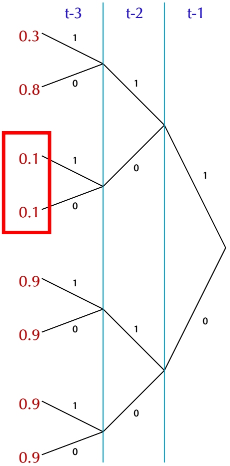
How variable length markov can reduce regret



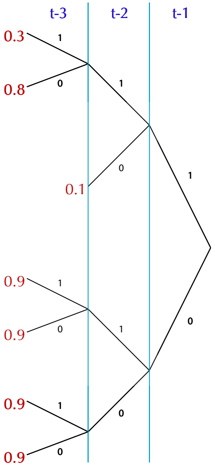
How variable length markov can reduce regret



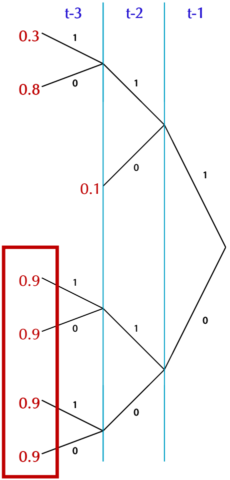
How variable length markov can reduce regret



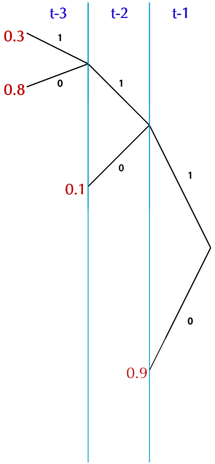
How variable length markov can reduce regret



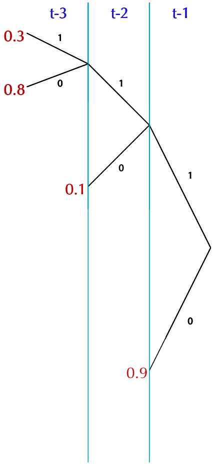
How variable length markov can reduce regret



How variable length markov can reduce regret

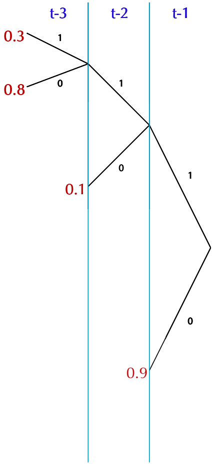


How variable length markov can reduce regret



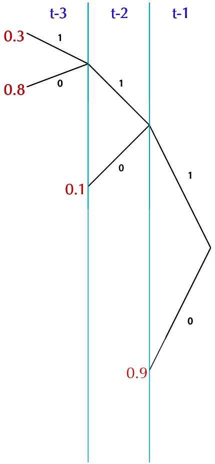
- Reducing number of leaves from 8 to 4 means

How variable length markov can reduce regret



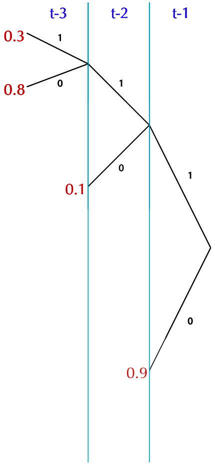
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$

How variable length markov can reduce regret



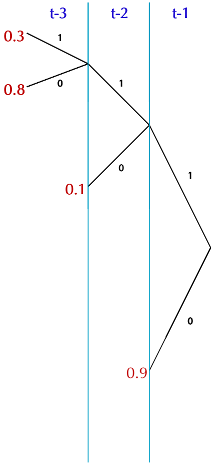
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B

How variable length markov can reduce regret



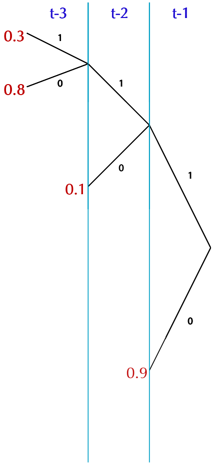
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B

How variable length markov can reduce regret



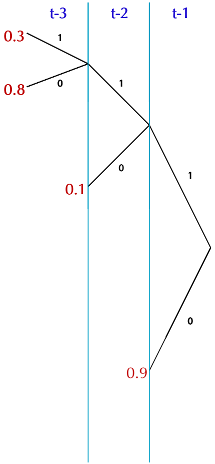
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A

How variable length markov can reduce regret



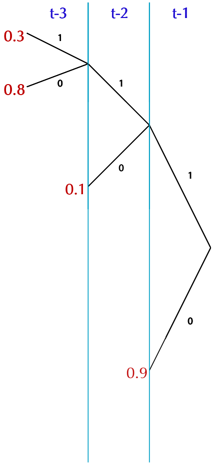
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A R

How variable length markov can reduce regret



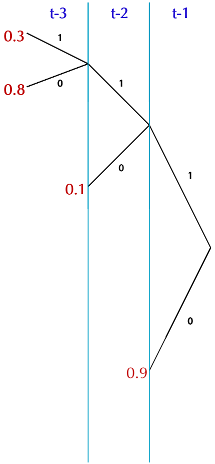
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A R O

How variable length markov can reduce regret



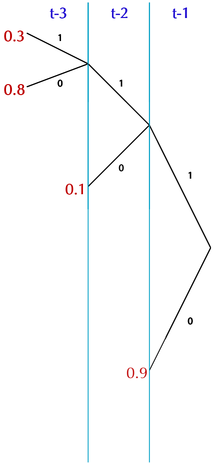
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A R O Q

How variable length markov can reduce regret



- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A R O Q U

How variable length markov can reduce regret



- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A R O Q U E
- When we have little data, we can get better prediction even if the children are not Exactly the same

Prefix trees / Tries

- In a prefix binary tree each node has either 0 or 2 children.

Prefix trees / Tries

- In a prefix binary tree each node has either 0 or 2 children.
- A variable length markov model corresponds to a prefix tree.

Prefix trees / Tries

- In a prefix binary tree each node has either 0 or 2 children.
- A variable length markov model corresponds to a prefix tree.
- You can think of a prefix trees as different prunings of a maximal tree.

Prefix trees / Tries

- In a prefix binary tree each node has either 0 or 2 children.
- A variable length markov model corresponds to a prefix tree.
- You can think of a prefix trees as different prunings of a maximal tree.
- We don't know a-priori which pruning to use!

Prefix trees / Tries

- In a prefix binary tree each node has either 0 or 2 children.
- A variable length markov model corresponds to a prefix tree.
- You can think of a prefix trees as different prunings of a maximal tree.
- We don't know a-priori which pruning to use!
- The number of prunings trees increases exponentially with the number of nodes in the maximal tree.

Prefix trees / Tries

- In a prefix binary tree each node has either 0 or 2 children.
- A variable length markov model corresponds to a prefix tree.
- You can think of a prefix trees as different prunings of a maximal tree.
- We don't know a-priori which pruning to use!
- The number of prunings trees increases exponentially with the number of nodes in the maximal tree.
- We will use the Online Bayes to predict almost as well as the best prefix tree in hind-sight.

Prefix trees / Tries

- In a prefix binary tree each node has either 0 or 2 children.
- A variable length markov model corresponds to a prefix tree.
- You can think of a prefix trees as different prunings of a maximal tree.
- We don't know a-priori which pruning to use!
- The number of prunings trees increases exponentially with the number of nodes in the maximal tree.
- We will use the Online Bayes to predict almost as well as the best prefix tree in hind-sight.
- First - simple but inefficient algorithm, Second - efficient algorithms.

Using online Bayes to learn the structure

- We assign to each tree an initial weight of 2^{-n} where n is the number of nodes in the pruned tree.

Using online Bayes to learn the structure

- We assign to each tree an initial weight of 2^{-n} where n is the number of nodes in the pruned tree.
- We combine the predictions of the trees using online Bayes.

Using online Bayes to learn the structure

- We assign to each tree an initial weight of 2^{-n} where n is the number of nodes in the pruned tree.
- We combine the predictions of the trees using online Bayes.
- The total regret would be $\frac{l}{2}\log T + n$ where l is the number of leaves in the prefix tree.

Using online Bayes to learn the structure

- We assign to each tree an initial weight of 2^{-n} where n is the number of nodes in the pruned tree.
- We combine the predictions of the trees using online Bayes.
- The total regret would be $\frac{l}{2} \log T + n$ where l is the number of leaves in the prefix tree.
- This algorithm maintains a weight for each prefix tree.

Using online Bayes to learn the structure

- We assign to each tree an initial weight of 2^{-n} where n is the number of nodes in the pruned tree.
- We combine the predictions of the trees using online Bayes.
- The total regret would be $\frac{l}{2}\log T + n$ where l is the number of leaves in the prefix tree.
- This algorithm maintains a weight for each prefix tree.
- The number of prunings of a full tree of depth k is $O(2^{2^k})$ while maintaining all of the counts requires $O(2^k)$.

Efficient generation of prior

- Prior distribution is generated by a stochastic recursion.

Efficient generation of prior

- Prior distribution is generated by a stochastic recursion.
- Start with root node (always exists)

Efficient generation of prior

- Prior distribution is generated by a stochastic recursion.
- Start with root node (always exists)
- For each node flip a fair coin.

Efficient generation of prior

- Prior distribution is generated by a stochastic recursion.
- Start with root node (always exists)
- For each node flip a fair coin.
  - Heads Set node to be a leaf (0 children)

Efficient generation of prior

- Prior distribution is generated by a stochastic recursion.
- Start with root node (always exists)
- For each node flip a fair coin.
  - Heads Set node to be a leaf (0 children)
  - Tails Create 2 children nodes to the node.

Efficient generation of prior

- Prior distribution is generated by a stochastic recursion.
- Start with root node (always exists)
- For each node flip a fair coin.
  - Heads Set node to be a leaf (0 children)
  - Tails Create 2 children nodes to the node.
- Defines a distribution over all prefix trees.

Efficient generation of prior

- Prior distribution is generated by a stochastic recursion.
- Start with root node (always exists)
- For each node flip a fair coin.
  - Heads Set node to be a leaf (0 children)
  - Tails Create 2 children nodes to the node.
- Defines a distribution over all prefix trees.
- Probability of a tree with n nodes is 2^{-n}

Efficient averaging over the prior (observations)

- Maintain a KT estimator at each node of the tree.

Efficient averaging over the prior (observations)

- Maintain a KT estimator at each node of the tree.
- Allocate counters only for nodes that have been visited.

Efficient averaging over the prior (observations)

- Maintain a KT estimator at each node of the tree.
- Allocate counters only for nodes that have been visited.
- At iteration t only t counters need to be updated.

Efficient averaging over the prior (observations)

- Maintain a KT estimator at each node of the tree.
- Allocate counters only for nodes that have been visited.
- At iteration t only t counters need to be updated.
- Only k counters if depth of tree is bounded.

Efficient averaging over the prior (observations)

- Maintain a KT estimator at each node of the tree.
- Allocate counters only for nodes that have been visited.
- At iteration t only t counters need to be updated.
- Only k counters if depth of tree is bounded.
- Each node is visited on a subset of the iterations.

Efficient averaging over the prior (observations)

- Maintain a KT estimator at each node of the tree.
- Allocate counters only for nodes that have been visited.
- At iteration t only t counters need to be updated.
- Only k counters if depth of tree is bounded.
- Each node is visited on a subset of the iterations.
- Subset corresponding to node is contained in subset corresponding to node's parent.

Efficient averaging over the prior (procedure)

- This is not the method used in the original paper, it appears in a later paper by *Willems, Tjalkens and Ignatenko. Available on github.*

Definitions

- s is a bit sequence corresponding to a node in the tree. The chidren of this node are $0s$ and $1s$.

Definitions

- s is a bit sequence corresponding to a node in the tree. The chidren of this node are $0s$ and $1s$.
- The sequence of past bits up to time t is denoted x_1^{t-1}, the t'th bit is denoted X_t

Definitions

- s is a bit sequence corresponding to a node in the tree. The chidren of this node are $0s$ and $1s$.
- The sequence of past bits up to time t is denoted x_1^{t-1}, the t'th bit is denoted X_t
- s determines a subsequence of x_1^{t-1}: the locations preceded by the reverse of s.

Definitions

- s is a bit sequence corresponding to a node in the tree. The chidren of this node are $0s$ and $1s$.
- The sequence of past bits up to time t is denoted x_1^{t-1}, the t'th bit is denoted X_t
- s determines a subsequence of x_1^{t-1}: the locations preceded by the reverse of s.
- $a_s(x_1^{t-1}), b_s(x_1^{t-1})$ count the number of 0's and 1's in the subsequence corresponding to s

Definitions

- s is a bit sequence corresponding to a node in the tree. The chidren of this node are $0s$ and $1s$.
- The sequence of past bits up to time t is denoted x_1^{t-1}, the t'th bit is denoted X_t
- s determines a subsequence of x_1^{t-1}: the locations preceded by the reverse of s.
- $a_s(x_1^{t-1}), b_s(x_1^{t-1})$ count the number of 0's and 1's in the subsequence corresponding to s
- The KT estimate associated with node s.

$$P_e^s\left(X_t = 1 | x_1^{t-1}\right) = \frac{b_s(x_1^{t-1}) + 1/2}{a_s(x_1^{t-1}) + b_s(x_1^{t-1}) + 1}$$

Assigning probabilities to complete sequences

- Using the chain rule, we can use a prediction rule to assign probabilities to a complete sequence.

$$P(x_1 = y_1, \ldots, x_T = y_T) = p(x_1 = y_1)p(x_2 = y_2 | x_1 = y_1) \ldots$$

Assigning probabilities to complete sequences

- Using the chain rule, we can use a prediction rule to assign probabilities to a complete sequence.

$$P(x_1 = y_1, \dots, x_T = y_T) = p(x_1 = y_1)p(x_2 = y_2|x_1 = y_1)\dots$$

- We can translate probabilities for complete sequences back into predictions.

$$p(x_t = 1|x_1 = y_1, \dots, x_{t-1} = y_{t-1}) = \frac{p(x_1 = y_1, \dots, x_{t-1} = y_{t-1}, x_t = 1)}{p(x_1 = y_1, \dots, x_{t-1} = y_{t-1})}$$

Mixing Factors

- $P_w^s\left(X_t = 1|x_1^{t-1}\right)$ The posterior average of the predictions associated with the descendants of the node s.

Mixing Factors

- $P_w^s\left(X_t = 1 | x_1^{t-1}\right)$ The posterior average of the predictions associated with the descendants of the node s.
- Using the chain rule we can also define unconditional probabilities $P_w^s\left(x_1^{t-1}\right)$

Mixing Factors

- $P_w^s\left(X_t = 1 | x_1^{t-1}\right)$ The posterior average of the predictions associated with the descendants of the node s.
- Using the chain rule we can also define unconditional probabilities $P_w^s\left(x_1^{t-1}\right)$
- The mixing factors according to the Prior distribution is 0.5: stop, 0.5: continue

Mixing Factors

- $P_w^s\left(X_t = 1 | x_1^{t-1}\right)$ The posterior average of the predictions associated with the descendants of the node s.
- Using the chain rule we can also define unconditional probabilities $P_w^s\left(x_1^{t-1}\right)$
- The mixing factors according to the Prior distribution is 0.5: stop, 0.5: continue
- After observing x_1^{t-1} the odds change. The updated odds are represented by $\beta^s(x_1^{t-1})$:

Mixing Factors

- $P_w^s\left(X_t = 1|x_1^{t-1}\right)$ The posterior average of the predictions associated with the descendants of the node s.
- Using the chain rule we can also define unconditional probabilities $P_w^s\left(x_1^{t-1}\right)$
- The mixing factors according to the Prior distribution is 0.5: stop, 0.5: continue
- After observing x_1^{t-1} the odds change. The updated odds are represented by $\beta^s(x_1^{t-1})$:
- $P_w^s\left(X_t = 1|x_1^{t-1}\right)$ The posterior average of the predictions associated with the descendants of the node s.

Outline of algorithm

- **Forward**: Traverse the tree from root to leaf.

Outline of algorithm

- **Forward**: Traverse the tree from root to leaf.
- **extend**: Add two children to the leaf. Initialized counts to 0,1.

Outline of algorithm

- **Forward**: Traverse the tree from root to leaf.
- **extend**: Add two children to the leaf. Initialized counts to 0,1.
- **Backward traversal**: Traverse back to root.
  For each node s

Outline of algorithm

- **Forward**: Traverse the tree from root to leaf.
- **extend**: Add two children to the leaf. Initialized counts to 0,1.
- **Backward traversal**: Traverse back to root.
  For each node s
  - compute $P_e^s\left(X_t = 1 | x_1^{t-1}\right)$ and $P_w^s\left(X_t = 1 | x_1^{t-1}\right)$

Outline of algorithm

- **Forward**: Traverse the tree from root to leaf.
- **extend**: Add two children to the leaf. Initialized counts to 0,1.
- **Backward traversal**: Traverse back to root.
  For each node s
  - compute $P_e^s\left(X_t = 1|x_1^{t-1}\right)$ and $P_w^s\left(X_t = 1|x_1^{t-1}\right)$
  - update counts: a^s, b^s.

Outline of algorithm

- **Forward**: Traverse the tree from root to leaf.
- **extend**: Add two children to the leaf. Initialized counts to 0,1.
- **Backward traversal**: Traverse back to root.
  For each node s
  - compute $P_e^s\left(X_t = 1|x_1^{t-1}\right)$ and $P_w^s\left(X_t = 1|x_1^{t-1}\right)$
  - update counts: a^s, b^s.
  - update β^s

Slides from Frans Willems

IX. Betas: Introduction

Consider an internal node s in the context tree $\mathcal{T}_{\mathcal{D}}$ and the corresponding *conditional* weighted probability $P_w^s(X_t = 1|x_1^{t-1})$. Assuming that $0s$ (and not $1s$) is a suffix of the context x_{1-D}^0, x_1^{t-1} of x_t, we obtain for this probability that

$$
\begin{aligned}
P_w^s(X_t = 1|x_1^{t-1}) &= \frac{P_e^s(x_1^{t-1}, X_t = 1) + P_w^{0s}(x_1^{t-1}, X_t = 1) P_w^{1s}(x_1^{t-1})}{P_e^s(x_1^{t-1}) + P_w^{0s}(x_1^{t-1}) P_w^{1s}(x_1^{t-1})} \\
&= \frac{\beta^s(x_1^{t-1}) P_e^s(X_t = 1|x_1^{t-1}) + P_w^{0s}(X_t = 1|x_1^{t-1})}{\beta^s(x_1^{t-1}) + 1} \qquad (1)
\end{aligned}
$$

where

$$
\beta^s(x_1^{t-1}) \triangleq \frac{P_e^s(x_1^{t-1})}{P_w^{0s}(x_1^{t-1}) P_w^{1s}(x_1^{t-1})}. \qquad (2)
$$

If we start in the context-leaf and work our way down to the root, we finally find $P_w^\lambda(X_t = 1|x_1^{t-1})$.

Slides from Frans Willems

Implementation

Assume that in node s the counts $a_s(x_1^{t-1})$ and $b_s(x_1^{t-1})$ are stored, as well as $\beta^s(x_1^{t-1})$. We then get the following sequence of operations:

1. Node $0s$ delivers cond. wei. probability $P_w^{0s}(X_t = 1|x_1^{t-1})$ to node s.
2. Cond. est. probability $P_e^s(X_t = 1|x_1^{t-1})$ is determined as follows:
$$P_e^s(X_t = 1|x_1^{t-1}) = \frac{b_s(x_1^{t-1}) + 1/2}{a_s(x_1^{t-1}) + b_s(x_1^{t-1}) + 1}. \quad (3)$$
3. Now $P_w^s(X_t = 1|x_1^{t-1})$ can be computed as in (1).
4. The ratio $\beta^s(\cdot)$ is then updated with symbol x_t as follows:
$$\beta^s(x_1^{t-1}, x_t) = \beta^s(x_1^{t-1}) \cdot \frac{P_e^s(X_t = x_t|x_1^{t-1})}{P_w^{0s}(X_t = x_t|x_1^{t-1})}. \quad (4)$$
5. Finally, depending on the value x_t, either count $a_s(x_1^{t-1})$ or $b_s(x_1^{t-1})$ is incremented.

Binary Tree

Sequence: 0,0,1,1,0,1,0,1,0,1,1,0,1,1,1