The Context Algorithm

Yoav Freund

January 26, 2006

Outline

Review

Outline

Outline

Outline

- Review
- Fixed Length Markov Models
- Variable Length Markov Model (VMM)
- Assigning weights to trees

Outline

Review

Fixed Length Markov Models

Variable Length Markov Model (VMM)

Assigning weights to trees

Learning the structure, an inefficient solution

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.

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

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

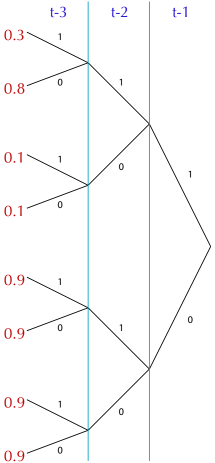
- Observe a binary sequence.
- $x_1, \ldots, x_{t-1}$
- Predict next bit from past
- $P(x_t = 1 | x_{t-1}, x_{t-2}, \ldots, x_1)$
- Use only last k bits
- $P(x_t = 1 | x_{t-1}, \ldots, x_{t-k})$

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

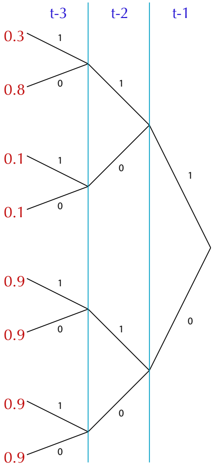
- Each tree leaf is associated with a binary sequence $y_1, \ldots, y_k$
- For each leaf keep two counters:
  - $a_{y_1,\ldots,y_k}$ = number of times $x_{t-1} = y_1, \ldots, x_{t-k} = y_k$ and $x_t = 0$
  - $b_{y_1,\ldots,y_k}$ = number of times $x_{t-1} = y_1, \ldots, x_{t-k} = y_k$ and $x_t = 1$
-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

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

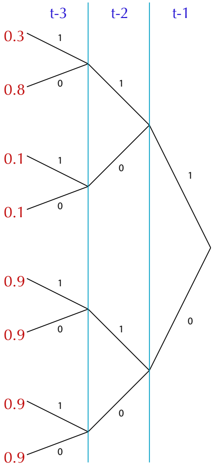
- Total regret is at most $2^{k-1} \log T$

How variable length markov can reduce regret

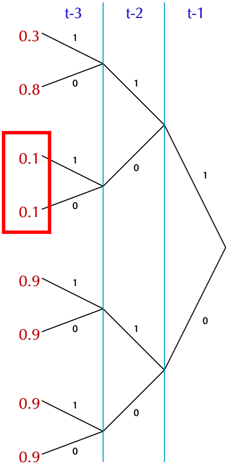
How variable length markov can reduce regret



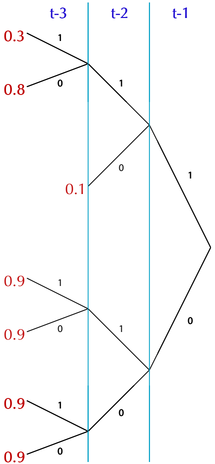
How variable length markov can reduce regret



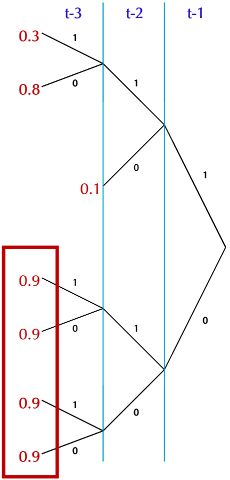
How variable length markov can reduce regret



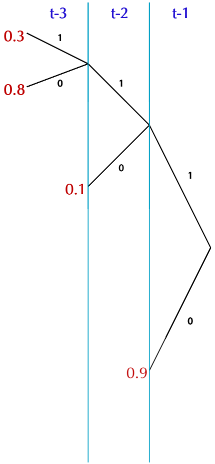
How variable length markov can reduce regret



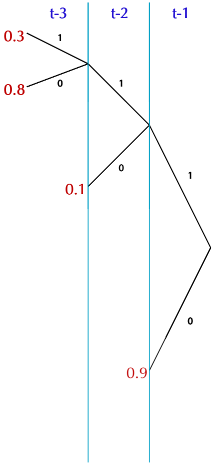
How variable length markov can reduce regret



How variable length markov can reduce regret

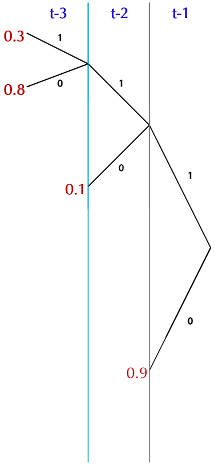


How variable length markov can reduce regret



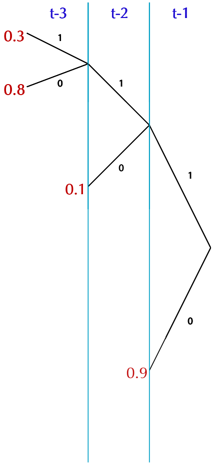
- Reducing number of leaves from 8 to 4 means

How variable length markov can reduce regret



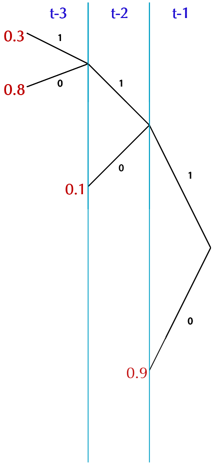
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$

How variable length markov can reduce regret



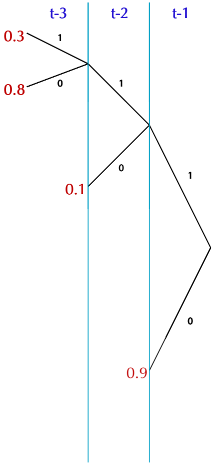
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B

How variable length markov can reduce regret



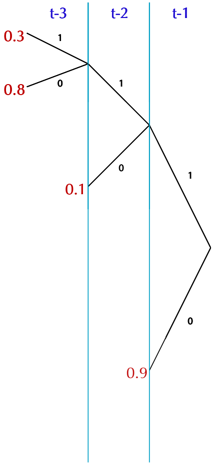
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B

How variable length markov can reduce regret



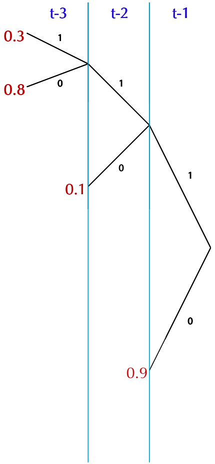
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A

How variable length markov can reduce regret



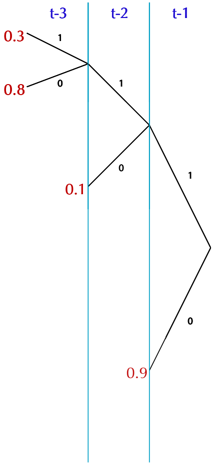
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A R

How variable length markov can reduce regret



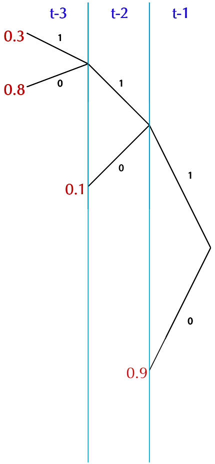
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A R O

How variable length markov can reduce regret



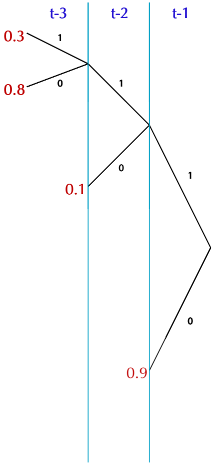
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A R O Q

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

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

Prefix trees

- In a prefix binary tree each node has either 0 or 2 children.
- A node with 1 child means that some past histories are not covered.

Prefix trees

- In a prefix binary tree each node has either 0 or 2 children.
- A node with 1 child means that some past histories are not covered.
- A variable length markov model corresponds to a prefix tree.

Prefix trees

- In a prefix binary tree each node has either 0 or 2 children.
- A node with 1 child means that some past histories are not covered.
- A variable length markov model corresponds to a prefix tree.
- But we don't know which prefix tree to use!

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.

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

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

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

Assigning probabilities to complete sequences

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

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

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

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

- Will come in handy soon!

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 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 prefix 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 prefix 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 prefix 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.
- The papers do things slightly differently because they bound the depth of the tree by k.

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 prefix 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.
- The papers do things slightly differently because they bound the depth of the tree by k.
- This algorithm maintains a weight for each 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 prefix 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.
- The papers do things slightly differently because they bound the depth of the tree by k.
- This algorithm maintains a weight for each tree.
- Requires maintaining $O(2^l)$ weights!

Efficient implementation

- First idea: Estimate probabilities of complete sequences and use conditional to generate predictions.

Efficient implementation

- First idea: Estimate probabilities of complete sequences and use conditional to generate predictions.
- The prior weights are used for averaging the complete sequence probabilities - they don't need to be updated.

Efficient implementation

- First idea: Estimate probabilities of complete sequences and use conditional to generate predictions.
- The prior weights are used for averaging the complete sequence probabilities - they don't need to be updated.
- Second idea: Compute the average over the prior efficiently.

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 once node is visited.

Efficient averaging over the prior (observations)

- Maintain a KT estimator at each node of the tree.
- Allocate counters only once node is 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 once node is 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 once node is 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 once node is 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)

- $P_e(a_s, b_s)$ the probability that the KT estimator at node $s = \langle y_1, .., y_k \rangle$ assigns to it's subsequence of the past.

Efficient averaging over the prior (procedure)

- $P_e(a_s, b_s)$ the probability that the KT estimator at node $s = \langle y_1, .., y_k \rangle$ assigns to it's subsequence of the past.
- P_w^s the average probability assigned to the subsequence over all VMM rooted at the node s

Efficient averaging over the prior (procedure)

- $P_e(a_s, b_s)$ the probability that the KT estimator at node $s = \langle y_1, .., y_k \rangle$ assigns to it's subsequence of the past.
- P_w^s the average probability assigned to the subsequence over all VMM rooted at the node s
- $$P_w^s \doteq \frac{P_e(a_s, b_s) + P_w^{0s} P_w^{1s}}{2}$$

Efficient averaging over the prior (procedure)

- $P_e(a_s, b_s)$ the probability that the KT estimator at node $s = \langle y_1, .., y_k \rangle$ assigns to it's subsequence of the past.
- P_w^s the average probability assigned to the subsequence over all VMM rooted at the node s
- $$P_w^s \doteq \frac{P_e(a_s, b_s) + P_w^{0s} P_w^{1s}}{2}$$
- Average probability assigned by the complete tree is P_w^λ where λ is the root node.