Biased coins and the Context algorithm

Yoav Freund

January 15, 2014

Outline

Combining experts in the log loss framework

Outline

Outline

Outline

Outline

Outline

Outline

Outline

Outline

Outline

Outline

Outline

Outline

The log-loss framework

- Algorithm A predicts a sequence $c^1, c^2, \ldots, c^T$ over alphabet $\Sigma = \{1, 2, \ldots, k\}$

The log-loss framework

- Algorithm A predicts a sequence $c^1, c^2, \ldots, c^T$ over alphabet $\Sigma = \{1, 2, \ldots, k\}$
- The prediction for the c^tth is a distribution over Σ: $\mathbf{p}_A^t = \langle p_A^t(1), p_A^t(2), \ldots, p_A^t(k) \rangle$

The log-loss framework

- Algorithm A predicts a sequence $c^1, c^2, \ldots, c^T$ over alphabet $\Sigma = \{1, 2, \ldots, k\}$
- The prediction for the c^tth is a distribution over Σ: $\mathbf{p}_A^t = \langle p_A^t(1), p_A^t(2), \ldots, p_A^t(k)\rangle$
- When c^t is revealed, the loss we suffer is $-\log p_A^t(c^t)$

The log-loss framework

- Algorithm A predicts a sequence $c^1, c^2, \ldots, c^T$ over alphabet $\Sigma = \{1, 2, \ldots, k\}$
- The prediction for the c^tth is a distribution over Σ: $\mathbf{p}_A^t = \langle p_A^t(1), p_A^t(2), \ldots, p_A^t(k) \rangle$
- When c^t is revealed, the loss we suffer is $-\log p_A^t(c^t)$
- The cumulative log loss, which we wish to minimize, is $L_A{}^T = -\sum_{t=1}^{T} \log p_A^t(c^t)$

The log-loss framework

- Algorithm A predicts a sequence $c^1, c^2, \ldots, c^T$ over alphabet $\Sigma = \{1, 2, \ldots, k\}$
- The prediction for the c^tth is a distribution over Σ: $\mathbf{p}_A^t = \langle p_A^t(1), p_A^t(2), \ldots, p_A^t(k) \rangle$
- When c^t is revealed, the loss we suffer is $-\log p_A^t(c^t)$
- The cumulative log loss, which we wish to minimize, is $L_A{}^T = -\sum_{t=1}^T \log p_A^t(c^t)$
- $\lceil L_A{}^T \rceil$ is the code length if A is combined with arithmetic coding.

The game

- Prediction algorithm A has access to N experts.

The game

- Prediction algorithm A has access to N experts.
- The following is repeated for $t = 1, \ldots, T$

The game

- Prediction algorithm A has access to N experts.
- The following is repeated for $t = 1, \ldots, T$
  - Experts generate predictive distributions: $\mathbf{p}_1^t, \ldots, \mathbf{p}_N^t$

The game

- Prediction algorithm A has access to N experts.
- The following is repeated for $t = 1, \ldots, T$
  - Experts generate predictive distributions: $\mathbf{p}_1^t, \ldots, \mathbf{p}_N^t$
  - Algorithm generates its own prediction $\mathbf{p}_A^t$

The game

- Prediction algorithm A has access to N experts.
- The following is repeated for $t = 1, \ldots, T$
  - Experts generate predictive distributions: $\mathbf{p}_1^t, \ldots, \mathbf{p}_N^t$
  - Algorithm generates its own prediction $\mathbf{p}_A^t$
  - c^t is revealed.

The game

- Prediction algorithm A has access to N experts.
- The following is repeated for $t = 1, \ldots, T$
  - Experts generate predictive distributions: $\mathbf{p}_1^t, \ldots, \mathbf{p}_N^t$
  - Algorithm generates its own prediction $\mathbf{p}_A^t$
  - c^t is revealed.
- **Goal:** minimize regret:

$$-\sum_{t=1}^{T} \log p_A^t(c^t) + \min_{i=1,\ldots,N} \left(-\sum_{t=1}^{T} \log p_i^t(c^t) \right)$$

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 uniform initial weights $w_i^1 = 1/N$

Simple Bound

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

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

Simple Bound

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

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

Simple Bound

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

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

Simple Bound

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

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

Simple Bound

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

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

Simple Bound

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

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

Simple Bound

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

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

- Dividing by T we get $\frac{L_A^T}{T} = \min_i \frac{L_i^T}{T} + \frac{\log N}{T}$

Bound better than for two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression

Bound better than for two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.

Bound better than for two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.
- Two part code has to point to one of the KN experts
  $L_A \leq \log NK + \min_i L_i^T = \log NK + \min_i L_i^T$

Bound better than for two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.
- Two part code has to point to one of the KN experts
  $L_A \leq \log NK + \min_i L_i^T = \log NK + \min_i L_i^T$
- If we use Bayes predictor + arithmetic coding we get:

$$L_A = -\log W^{T+1} \leq \log K \max_i \frac{1}{NK} e^{-L_i^T} = \log N + \min_i L_i^T$$

Bound better than for two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.
- Two part code has to point to one of the KN experts
  $L_A \leq \log NK + \min_i L_i^T = \log NK + \min_i L_i^T$
- If we use Bayes predictor + arithmetic coding we get:

$$L_A = -\log W^{T+1} \leq \log K \max_i \frac{1}{NK} e^{-L_i^T} = \log N + \min_i L_i^T$$

- We don't pay a penalty for copies.

Bound better than for two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.
- Two part code has to point to one of the KN experts
  $L_A \leq \log NK + \min_i L_i^T = \log NK + \min_i L_i^T$
- If we use Bayes predictor + arithmetic coding we get:

$$L_A = -\log W^{T+1} \leq \log K \max_i \frac{1}{NK} e^{-L_i^T} = \log N + \min_i L_i^T$$

- We don't pay a penalty for copies.
- More generally, the regret is smaller if many of the experts perform well.

Comparison with standard Bayesian statistics

- The weight update rule is the same.

Comparison with standard Bayesian statistics

- The weight update rule is the same.
- Normalized weights $=$ **posterior probability distribution.**

Comparison with standard Bayesian statistics

- The weight update rule is the same.
- Normalized weights $=$ **posterior probability distribution.**
- Bayesian analysis interested in the **final** posterior.

Comparison with standard Bayesian statistics

- The weight update rule is the same.
- Normalized weights $=$ **posterior probability distribution**.
- Bayesian analysis interested in the **final** posterior.
- Bayesian analysis assumes the data is **generated** by a distribution in the support of the prior.

Comparison with standard Bayesian statistics

- The weight update rule is the same.
- Normalized weights $=$ **posterior probability distribution.**
- Bayesian analysis interested in the **final** posterior.
- Bayesian analysis assumes the data is **generated** by a distribution in the support of the prior.
- Goal of Bayesian is to **estimate true distribution**, goal of online learning is to **minimize regret**.

Comparison with standard Bayesian statistics

- The weight update rule is the same.
- Normalized weights $=$ **posterior probability distribution.**
- Bayesian analysis interested in the **final** posterior.
- Bayesian analysis assumes the data is **generated** by a distribution in the support of the prior.
- Goal of Bayesian is to **estimate true distribution**, goal of online learning is to **minimize regret**.
- Optimality of algorithm is **axiom** of Bayesian statistics.

Comparison with standard Bayesian statistics

- The weight update rule is the same.
- Normalized weights $=$ **posterior probability distribution.**
- Bayesian analysis interested in the **final** posterior.
- Bayesian analysis assumes the data is **generated** by a distribution in the support of the prior.
- Goal of Bayesian is to **estimate true distribution**, goal of online learning is to **minimize regret**.
- Optimality of algorithm is **axiom** of Bayesian statistics.
- Bayesian methods perform poorly when the loss is not log loss and the data not generated by a distribution in the support.

Comparison with standard Bayesian statistics

- The weight update rule is the same.
- Normalized weights $=$ **posterior probability distribution.**
- Bayesian analysis interested in the **final** posterior.
- Bayesian analysis assumes the data is **generated** by a distribution in the support of the prior.
- Goal of Bayesian is to **estimate true distribution**, goal of online learning is to **minimize regret.**
- Optimality of algorithm is **axiom** of Bayesian statistics.
- Bayesian methods perform poorly when the loss is not log loss and the data not generated by a distribution in the support.
    - Loss can sometimes be defined through the noise distribution: square loss is equivalent to assuming guassian noise.

Comparison with standard Bayesian statistics

- The weight update rule is the same.
- Normalized weights $=$ **posterior probability distribution**.
- Bayesian analysis interested in the **final** posterior.
- Bayesian analysis assumes the data is **generated** by a distribution in the support of the prior.
- Goal of Bayesian is to **estimate true distribution**, goal of online learning is to **minimize regret**.
- Optimality of algorithm is **axiom** of Bayesian statistics.
- Bayesian methods perform poorly when the loss is not log loss and the data not generated by a distribution in the support.
  - Loss can sometimes be defined through the noise distribution: square loss is equivalent to assuming guassian noise.
  - For number of mistakes - Bayesian method cannot be “fixed”. Requires variable learning rate.

Computational Issues

- Naive implementation: calculate the prediction of each of the N experts.

Computational Issues

- Naive implementation: calculate the prediction of each of the N experts.
- Puts severe limit on number of experts.

Computational Issues

- Naive implementation: calculate the prediction of each of the N experts.
- Puts severe limit on number of experts.
- What if set of experts is uncountably infinite.

Computational Issues

- Naive implementation: calculate the prediction of each of the N experts.
- Puts severe limit on number of experts.
- What if set of experts is uncountably infinite.
- Bayesian tricks:

Computational Issues

- Naive implementation: calculate the prediction of each of the N experts.
- Puts severe limit on number of experts.
- What if set of experts is uncountably infinite.
- Bayesian tricks:
  - **Conjugate priors**: A prior over a continuous domain whose functional form does not change with when updated.

Computational Issues

- Naive implementation: calculate the prediction of each of the N experts.
- Puts severe limit on number of experts.
- What if set of experts is uncountably infinite.
- Bayesian tricks:
  - **Conjugate priors**: A prior over a continuous domain whose functional form does not change with when updated.

Computational Issues

- Naive implementation: calculate the prediction of each of the N experts.
- Puts severe limit on number of experts.
- What if set of experts is uncountably infinite.
- Bayesian tricks:
  - **Conjugate priors**: A prior over a continuous domain whose functional form does not change with when updated. Number of parameters defining posterior is constant.

Computational Issues

- Naive implementation: calculate the prediction of each of the N experts.
- Puts severe limit on number of experts.
- What if set of experts is uncountably infinite.
- Bayesian tricks:
  - **Conjugate priors**: A prior over a continuous domain whose functional form does not change with when updated. Number of parameters defining posterior is constant. Update rule translates into update of parameters.

Computational Issues

- Naive implementation: calculate the prediction of each of the N experts.
- Puts severe limit on number of experts.
- What if set of experts is uncountably infinite.
- Bayesian tricks:
  - **Conjugate priors**: A prior over a continuous domain whose functional form does not change with when updated. Number of parameters defining posterior is constant. Update rule translates into update of parameters. parameters correspond to “sufficient statistics”.

Computational Issues

- Naive implementation: calculate the prediction of each of the N experts.
- Puts severe limit on number of experts.
- What if set of experts is uncountably infinite.
- Bayesian tricks:
  - **Conjugate priors**: A prior over a continuous domain whose functional form does not change with when updated. Number of parameters defining posterior is constant. Update rule translates into update of parameters. parameters correspond to “sufficient statistics”. exists for the familty of exponential distributions.
  - **Markov Chain Monte Carlo** Sample the posterior.

Computational Issues

- Naive implementation: calculate the prediction of each of the N experts.
- Puts severe limit on number of experts.
- What if set of experts is uncountably infinite.
- Bayesian tricks:
  - **Conjugate priors**: A prior over a continuous domain whose functional form does not change with when updated. Number of parameters defining posterior is constant. Update rule translates into update of parameters. parameters correspond to “sufficient statistics”. exists for the familty of exponential distributions.
  - **Markov Chain Monte Carlo** Sample the posterior.

Computational Issues

- Naive implementation: calculate the prediction of each of the N experts.
- Puts severe limit on number of experts.
- What if set of experts is uncountably infinite.
- Bayesian tricks:
  - **Conjugate priors**: A prior over a continuous domain whose functional form does not change with when updated. Number of parameters defining posterior is constant. Update rule translates into update of parameters. parameters correspond to “sufficient statistics”. exists for the familty of exponential distributions.
  - **Markov Chain Monte Carlo** Sample the posterior. Can sometimes be done efficiently.

Computational Issues

- Naive implementation: calculate the prediction of each of the N experts.
- Puts severe limit on number of experts.
- What if set of experts is uncountably infinite.
- Bayesian tricks:
  - **Conjugate priors**: A prior over a continuous domain whose functional form does not change with when updated. Number of parameters defining posterior is constant. Update rule translates into update of parameters. parameters correspond to “sufficient statistics”. exists for the familty of exponential distributions.
  - **Markov Chain Monte Carlo** Sample the posterior. Can sometimes be done efficiently. Efficient sampling relates to mixing rate of markov chain whose limit dist is the posterior dist.

Standardizing online prediction algorithms

- Fix a universal Turing machine U.

Standardizing online prediction algorithms

- Fix a universal Turing machine U.
- An online prediction algorithm E is a program that

Standardizing online prediction algorithms

- Fix a universal Turing machine U.
- An online prediction algorithm E is a program that
  - given as input The past $\vec{X} \in \{0,1\}^t$

Standardizing online prediction algorithms

- Fix a universal Turing machine U.
- An online prediction algorithm E is a program that
  - given as input The past $\vec{X} \in \{0,1\}^t$
  - runs finite time and outputs

Standardizing online prediction algorithms

- Fix a universal Turing machine U.
- An online prediction algorithm E is a program that
  - given as input The past $\vec{X} \in \{0,1\}^t$
  - runs finite time and outputs
  - A prediction for the next bit $p(\vec{X}) \in [0,1]$.

Standardizing online prediction algorithms

- Fix a universal Turing machine U.
- An online prediction algorithm E is a program that
  - given as input The past $\vec{X} \in \{0,1\}^t$
  - runs finite time and outputs
  - A prediction for the next bit $p(\vec{X}) \in [0,1]$.
  - To ensure p has a finite description. Restrict to rational numbers n/m

Standardizing online prediction algorithms

- Fix a universal Turing machine U.
- An online prediction algorithm E is a program that
  - given as input The past $\vec{X} \in \{0,1\}^t$
  - runs finite time and outputs
  - A prediction for the next bit $p(\vec{X}) \in [0,1]$.
  - To ensure p has a finite description. Restrict to rational numbers n/m
- Any online prediction algorithm can be represented as code $\vec{b}(E)$ for U. The code length is $|\vec{b}(E)|$.

Standardizing online prediction algorithms

- Fix a universal Turing machine U.
- An online prediction algorithm E is a program that
  - given as input The past $\vec{X} \in \{0,1\}^t$
  - runs finite time and outputs
  - A prediction for the next bit $p(\vec{X}) \in [0,1]$.
  - To ensure p has a finite description. Restrict to rational numbers n/m
- Any online prediction algorithm can be represented as code $\vec{b}(E)$ for U. The code length is $|\vec{b}(E)|$.
- Most sequences do not correspond to valid prediction algorithms.

Standardizing online prediction algorithms

- Fix a universal Turing machine U.
- An online prediction algorithm E is a program that
  - given as input The past $\vec{X} \in \{0,1\}^t$
  - runs finite time and outputs
  - A prediction for the next bit $p(\vec{X}) \in [0,1]$.
  - To ensure p has a finite description. Restrict to rational numbers n/m
- Any online prediction algorithm can be represented as code $\vec{b}(E)$ for U. The code length is $|\vec{b}(E)|$.
- Most sequences do not correspond to valid prediction algorithms.
- $V(\vec{b}, \vec{X}, t) = 1$ if the program $\vec{b}$, given $\vec{X}$ as input, halts within t steps and outputs a well-formed prediction. Otherwise $V(\vec{b}, \vec{X}, t) = 0$

Standardizing online prediction algorithms

- Fix a universal Turing machine U.
- An online prediction algorithm E is a program that
  - given as input The past $\vec{X} \in \{0,1\}^t$
  - runs finite time and outputs
  - A prediction for the next bit $p(\vec{X}) \in [0,1]$.
  - To ensure p has a finite description. Restrict to rational numbers n/m
- Any online prediction algorithm can be represented as code $\vec{b}(E)$ for U. The code length is $|\vec{b}(E)|$.
- Most sequences do not correspond to valid prediction algorithms.
- $V(\vec{b}, \vec{X}, t) = 1$ if the program $\vec{b}$, given $\vec{X}$ as input, halts within t steps and outputs a well-formed prediction. Otherwise $V(\vec{b}, \vec{X}, t) = 0$
- $V(\vec{b}, \vec{X}, t)$ is computable (recursively enumerable).

A universal prediction machine

- Assign to the code $\vec{b}$ the initial weight $w_{\vec{b}}^{1} = 2^{-|\vec{b}|-\log_2 |\vec{b}|}$.

A universal prediction machine

- Assign to the code $\vec{b}$ the initial weight $w^1_{\vec{b}} = 2^{-|\vec{b}|-\log_2|\vec{b}|}$.
- The total initial weight over all finite binary sequences is one.

A universal prediction machine

- Assign to the code $\vec{b}$ the initial weight $w^1_{\vec{b}} = 2^{-|\vec{b}|-\log_2 |\vec{b}|}$.
- The total initial weight over all finite binary sequences is one.
- Run the Bayes algorithm over “all” prediction algorithms.

A universal prediction machine

- Assign to the code $\vec{b}$ the initial weight $w^1_{\vec{b}} = 2^{-|\vec{b}|-\log_2|\vec{b}|}$.
- The total initial weight over all finite binary sequences is one.
- Run the Bayes algorithm over "all" prediction algorithms.
- technical details: On iteration t, $|\vec{X}| = t$. Use the predictions of programs $\vec{b}$ such that $|\vec{b}| \leq t$ and for which $V(\vec{b}, \vec{X}, 2^t) = 1$. Assing the remaining mass the prediction $1/2$ (insuring a loss of 1)

Performance of the universal prediction algorithm

- Using $L_A \leq \min_i \left(L_i - \log w_i^1 \right)$

Performance of the universal prediction algorithm

- Using $L_A \leq \min_i \left(L_i - \log w_i^1\right)$
- Assume E is a prediction algorithm which generates the tth prediction in time smaller than 2^t

Performance of the universal prediction algorithm

- Using $L_A \leq \min_i \left(L_i - \log w_i^1 \right)$
- Assume E is a prediction algorithm which generates the tth prediction in time smaller than 2^t
- When $t \leq |\vec{b}(E)|$ the algorithm is not used and thus it's loss is 1

Performance of the universal prediction algorithm

- Using $L_A \leq \min_i \left(L_i - \log w_i^1\right)$
- Assume E is a prediction algorithm which generates the tth prediction in time smaller than 2^t
- When $t \leq |\vec{b}(E)|$ the algorithm is not used and thus it's loss is 1
- We get that the loss of the Universal algorithm is at most $2|\vec{b}(E)| + \log_2 |\vec{b}(E)| + L_E$

Performance of the universal prediction algorithm

- Using $L_A \leq \min_i \left(L_i - \log w_i^1\right)$
- Assume E is a prediction algorithm which generates the tth prediction in time smaller than 2^t
- When $t \leq |\vec{b}(E)|$ the algorithm is not used and thus it's loss is 1
- We get that the loss of the Universal algorithm is at most $2|\vec{b}(E)| + \log_2 |\vec{b}(E)| + L_E$
- More careful analysis can reduce $2|\vec{b}(E)| + \log_2 |\vec{b}(E)|$ to $|\vec{b}(E)|$

Performance of the universal prediction algorithm

- Using $L_A \leq \min_i \left(L_i - \log w_i^1\right)$
- Assume E is a prediction algorithm which generates the tth prediction in time smaller than 2^t
- When $t \leq |\vec{b}(E)|$ the algorithm is not used and thus it's loss is 1
- We get that the loss of the Universal algorithm is at most $2|\vec{b}(E)| + \log_2 |\vec{b}(E)| + L_E$
- More careful analysis can reduce $2|\vec{b}(E)| + \log_2 |\vec{b}(E)|$ to $|\vec{b}(E)|$
- Code length is arbitrarily close to the Kolmogorov Complexity of the sequence.

Performance of the universal prediction algorithm

- Using $L_A \leq \min_i \left(L_i - \log w_i^1\right)$
- Assume E is a prediction algorithm which generates the tth prediction in time smaller than 2^t
- When $t \leq |\vec{b}(E)|$ the algorithm is not used and thus it's loss is 1
- We get that the loss of the Universal algorithm is at most $2|\vec{b}(E)| + \log_2 |\vec{b}(E)| + L_E$
- More careful analysis can reduce $2|\vec{b}(E)| + \log_2 |\vec{b}(E)|$ to $|\vec{b}(E)|$
- Code length is arbitrarily close to the Kolmogorov Complexity of the sequence.
- Ridiculously bad running time.

Bayes coding is better than two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression

Bayes coding is better than two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.

Bayes coding is better than two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.
- Two part code has to point to one of the KN experts
  $L_A \leq \log NK + \min_i L_i^T = \log NK + \min_i L_i^T$

Bayes coding is better than two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.
- Two part code has to point to one of the KN experts $L_A \leq \log NK + \min_i L_i^T = \log NK + \min_i L_i^T$
- If we use Bayes predictor + arithmetic coding we get:

$$L_A = -\log W^{T+1} \leq \log K \max_i \frac{1}{NK} e^{-L_i^T} = \log N + \min_i L_i^T$$

Bayes coding is better than two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.
- Two part code has to point to one of the KN experts
  $L_A \leq \log NK + \min_i L_i^T = \log NK + \min_i L_i^T$
- If we use Bayes predictor + arithmetic coding we get:

$$L_A = -\log W^{T+1} \leq \log K \max_i \frac{1}{NK} e^{-L_i^T} = \log N + \min_i L_i^T$$

- We don't pay a penalty for copies.

Bayes coding is better than two part codes

- Simple bound as good as bound for two part codes (MDL) but enables online compression
- Suppose we have K copies of each expert.
- Two part code has to point to one of the KN experts
  $L_A \leq \log NK + \min_i L_i^T = \log NK + \min_i L_i^T$
- If we use Bayes predictor + arithmetic coding we get:

$$L_A = -\log W^{T+1} \leq \log K \max_i \frac{1}{NK} e^{-L_i^T} = \log N + \min_i L_i^T$$

- We don't pay a penalty for copies.
- More generally, the regret is smaller if many of the experts perform well.

The biased coins set of experts

- Each expert corresponds to a biased coin, predicts with a fixed $\theta \in [0, 1]$.

The biased coins set of experts

- Each expert corresponds to a biased coin, predicts with a fixed $\theta \in [0, 1]$.
- Set of experts is uncountably infinite.

The biased coins set of experts

- Each expert corresponds to a biased coin, predicts with a fixed $\theta \in [0, 1]$.
- Set of experts is uncountably infinite.
- Only countably many experts can be assigned non-zero weight.

The biased coins set of experts

- Each expert corresponds to a biased coin, predicts with a fixed $\theta \in [0, 1]$.
- Set of experts is uncountably infinite.
- Only countably many experts can be assigned non-zero weight.
- Instead, we assign the experts a Density Measure.

The biased coins set of experts

- Each expert corresponds to a biased coin, predicts with a fixed $\theta \in [0, 1]$.
- Set of experts is uncountably infinite.
- Only countably many experts can be assigned non-zero weight.
- Instead, we assign the experts a Density Measure.
- $L_A \leq \min_i \left(L_i - \log w_i^1\right)$ is meaningless.

The biased coins set of experts

- Each expert corresponds to a biased coin, predicts with a fixed $\theta \in [0, 1]$.
- Set of experts is uncountably infinite.
- Only countably many experts can be assigned non-zero weight.
- Instead, we assign the experts a Density Measure.
- $L_A \leq \min_i \left(L_i - \log w_i^1 \right)$ is meaningless.
- Can we still get a meaningful bound?

Bayes Algorithm for biased coins

- Replace the initial weight by a density measure $w(\theta) = w^1(\theta)$, $\int_0^1 w(\theta) d\theta = 1$

Bayes Algorithm for biased coins

- Replace the initial weight by a density measure $w(\theta) = w^1(\theta)$, $\int_0^1 w(\theta)d\theta = 1$
- Relationship between final total weight and total log loss remains unchanged:

$$L_A = \ln \int_0^1 w(\theta) e^{-L_\theta^{T+1}} d\theta$$

Bayes Algorithm for biased coins

- Replace the initial weight by a density measure $w(\theta) = w^1(\theta)$, $\int_0^1 w(\theta) d\theta = 1$
- Relationship between final total weight and total log loss remains unchanged:

$$L_A = \ln \int_0^1 w(\theta) e^{-L_\theta^{T+1}} d\theta$$

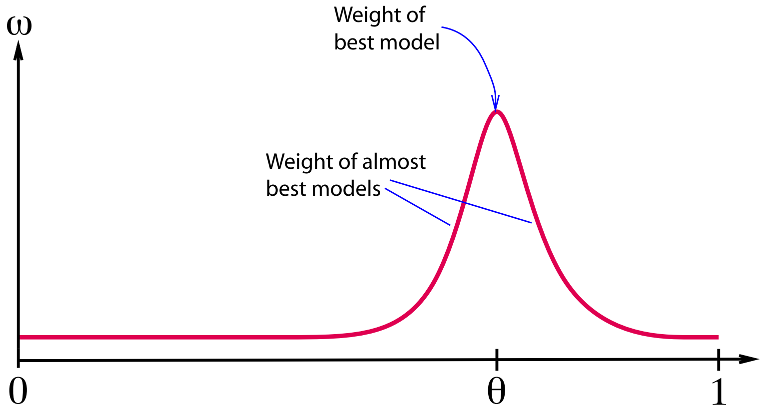
- We need a new lower bound on the final total weight

Main Idea

If $w^t(\theta)$ is large then $w^t(\theta + \epsilon)$ is also large.

Main Idea

If $w^t(\theta)$ is large then $w^t(\theta + \epsilon)$ is also large.



Expanding the exponent around the peak

- For log loss the best θ is empirical distribution of the seq.

$$\hat{\theta} = \frac{\#\{x^t = 1;\;\; 1 \leq t \leq T\}}{T}$$

Expanding the exponent around the peak

- For log loss the best θ is empirical distribution of the seq.

$$\hat{\theta} = \frac{\#\{x^t = 1; \;\; 1 \le t \le T\}}{T}$$

- The total loss scales with T

$$L_\theta = T \cdot (\hat{\theta}\ell(\theta, 1) + (1 - \hat{\theta})\ell(\theta, 0)) \doteq T \cdot g(\hat{\theta}, \theta)$$

Expanding the exponent around the peak

- For log loss the best θ is empirical distribution of the seq.

$$\hat{\theta} = \frac{\#\{x^t = 1; \;\; 1 \leq t \leq T\}}{T}$$

- The total loss scales with T

$$L_\theta = T \cdot (\hat{\theta}\ell(\theta, 1) + (1 - \hat{\theta})\ell(\theta, 0)) \doteq T \cdot g(\hat{\theta}, \theta)$$

Expanding the exponent around the peak

- For log loss the best θ is empirical distribution of the seq.

$$\hat{\theta} = \frac{\#\{x^t = 1; \ \ 1 \leq t \leq T\}}{T}$$

- The total loss scales with T

$$L_\theta = T \cdot (\hat{\theta}\ell(\theta, 1) + (1 - \hat{\theta})\ell(\theta, 0)) \doteq T \cdot g(\hat{\theta}, \theta)$$

$$L_A - L_{\min} \quad \leq \quad \ln \int_0^1 w(\theta) e^{-L_\theta} d\theta - \ln e^{L_{\min}}$$

Expanding the exponent around the peak

- For log loss the best θ is empirical distribution of the seq.

$$\hat{\theta} = \frac{\#\{x^t = 1; \ \ 1 \leq t \leq T\}}{T}$$

- The total loss scales with T

$$L_\theta = T \cdot (\hat{\theta}\ell(\theta, 1) + (1 - \hat{\theta})\ell(\theta, 0)) \doteq T \cdot g(\hat{\theta}, \theta)$$

$$\begin{aligned} L_A - L_{\min} &\leq \ln \int_0^1 w(\theta) e^{-L_\theta} d\theta - \ln e^{L_{\min}} \\ &= \ln \int_0^1 w(\theta) e^{-(L_\theta - L_{\min})} d\theta \end{aligned}$$

Expanding the exponent around the peak

- For log loss the best θ is empirical distribution of the seq.

$$\hat{\theta} = \frac{\#\{x^t = 1; \ \ 1 \leq t \leq T\}}{T}$$

- The total loss scales with T

$$L_\theta = T \cdot (\hat{\theta}\ell(\theta, 1) + (1 - \hat{\theta})\ell(\theta, 0)) \doteq T \cdot g(\hat{\theta}, \theta)$$

$$\begin{aligned} L_A - L_{\min} &\leq \ln \int_0^1 w(\theta) e^{-L_\theta} d\theta - \ln e^{L_{\min}} \\ &= \ln \int_0^1 w(\theta) e^{-(L_\theta - L_{\min})} d\theta \\ &= \ln \int_0^1 w(\theta) e^{T(g(\hat{\theta},\theta) - g(\hat{\theta},\hat{\theta}))} d\theta \end{aligned}$$

Laplace approximation (idea)

- Taylor expansion of $g(\hat{\theta}, \theta) - g(\hat{\theta}, \hat{\theta})$ around $\theta = \hat{\theta}$.

Laplace approximation (idea)

- Taylor expansion of $g(\hat{\theta}, \theta) - g(\hat{\theta}, \hat{\theta})$ around $\theta = \hat{\theta}$.
- First and second terms in the expansion are zero.

Laplace approximation (idea)

- Taylor expansion of $g(\hat{\theta}, \theta) - g(\hat{\theta}, \hat{\theta})$ around $\theta = \hat{\theta}$.
- First and second terms in the expansion are zero.
- Third term gives a quadratic expression in the exponent

Laplace approximation (idea)

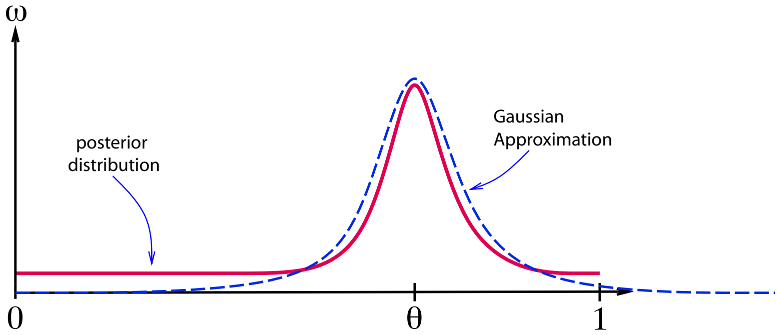
- Taylor expansion of $g(\hat{\theta}, \theta) - g(\hat{\theta}, \hat{\theta})$ around $\theta = \hat{\theta}$.
- First and second terms in the expansion are zero.
- Third term gives a quadratic expression in the exponent
- $\Rightarrow$ a gaussian approximation of the posterior.

Laplace approximation (idea)

- Taylor expansion of $g(\hat{\theta},\theta) - g(\hat{\theta},\hat{\theta})$ around $\theta = \hat{\theta}$.
- First and second terms in the expansion are zero.
- Third term gives a quadratic expression in the exponent
- $\Rightarrow$ a gaussian approximation of the posterior.

Laplace approximation (idea)

- Taylor expansion of $g(\hat{\theta},\theta) - g(\hat{\theta},\hat{\theta})$ around $\theta = \hat{\theta}$.
- First and second terms in the expansion are zero.
- Third term gives a quadratic expression in the exponent
- $\Rightarrow$ a gaussian approximation of the posterior.



Laplace Approximation (details)

$$\int_0^1 w(\theta) e^{T(g(\hat{\theta},\theta)-g(\hat{\theta},\hat{\theta}))} d\theta$$

Laplace Approximation (details)

$$
\begin{aligned}
&\int_0^1 w(\theta) e^{T(g(\hat{\theta},\theta)-g(\hat{\theta},\hat{\theta}))} d\theta \\
&= \quad w(\hat{\theta}) \sqrt{\frac{-2\pi}{T \left. \frac{d^2}{d\theta^2} \right|_{\theta=\hat{\theta}} (g(\hat{\theta},\theta) - g(\hat{\theta},\hat{\theta}))}} + O(T^{-3/2})
\end{aligned}
$$

Choosing the optimal prior

- Choose $w(\theta)$ to maximize the worst-case final total weight

$$\min_{\hat{\theta}} w(\hat{\theta}) \sqrt{\frac{-2\pi}{T \left. \frac{d^2}{d\theta^2} \right|_{\theta=\hat{\theta}} (g(\hat{\theta}, \theta) - g(\hat{\theta}, \hat{\theta}))}}$$

Choosing the optimal prior

- Choose $w(\theta)$ to maximize the worst-case final total weight

$$\min_{\hat{\theta}} w(\hat{\theta}) \sqrt{\frac{-2\pi}{T \left.\frac{d^2}{d\theta^2}\right|_{\theta=\hat{\theta}} (g(\hat{\theta},\theta) - g(\hat{\theta},\hat{\theta}))}}$$

- Make bound equal for all $\hat{\theta} \in [0,1]$ by choosing

$$w^*(\hat{\theta}) = \frac{1}{Z}\sqrt{\frac{\left.\frac{d^2}{d\theta^2}\right|_{\theta=\hat{\theta}} (g(\hat{\theta},\theta) - g(\hat{\theta},\hat{\theta}))}{-2\pi}},$$

where Z is the normalization factor:

$$Z = \sqrt{\frac{1}{2\pi}} \int_0^1 \sqrt{\left.\frac{d^2}{d\theta^2}\right|_{\theta=\hat{\theta}} (g(\hat{\theta},\hat{\theta}) - g(\hat{\theta},\theta))} \; d\hat{\theta}$$

The bound for the optimal prior

- Plugging in we get

$$
\begin{aligned}
L_A - L_{\min} &\leq \ln \int_0^1 w^*(\theta) e^{T(g(\hat{\theta},\theta) - g(\hat{\theta},\hat{\theta}))} d\theta \\
&= \ln \left(\sqrt{\frac{2\pi Z}{T}} + O(T^{-3/2}) \right) \\
&= \frac{1}{2} \ln \frac{T}{2\pi} - \frac{1}{2} \ln Z + O(1/T) \, .
\end{aligned}
$$

Solving for log-loss

- The exponent in the integral is

$$g(\hat{\theta},\theta) - g(\hat{\theta},\hat{\theta}) = \hat{\theta} \ln \frac{\hat{\theta}}{\theta} + (1-\hat{\theta}) \ln \frac{1-\hat{\theta}}{1-\theta} = D_{KL}(\hat{\theta}||\theta)$$

Solving for log-loss

- The exponent in the integral is

$$g(\hat{\theta},\theta) - g(\hat{\theta},\hat{\theta}) = \hat{\theta}\ln\frac{\hat{\theta}}{\theta} + (1-\hat{\theta})\ln\frac{1-\hat{\theta}}{1-\theta} = D_{KL}(\hat{\theta}||\theta)$$

- The second derivative

$$\left.\frac{d^2}{d\theta^2}\right|_{\theta=\hat{\theta}} D_{KL}(\hat{\theta}||\theta) = \hat{\theta}(1-\hat{\theta})$$

Is called the empirical Fisher information

Solving for log-loss

- The exponent in the integral is

$$g(\hat{\theta},\theta) - g(\hat{\theta},\hat{\theta}) = \hat{\theta}\ln\frac{\hat{\theta}}{\theta} + (1-\hat{\theta})\ln\frac{1-\hat{\theta}}{1-\theta} = D_{KL}(\hat{\theta}||\theta)$$

- The second derivative

$$\left.\frac{d^2}{d\theta^2}\right|_{\theta=\hat{\theta}} D_{KL}(\hat{\theta}||\theta) = \hat{\theta}(1-\hat{\theta})$$

  Is called the empirical Fisher information
- The optimal prior:

$$w^*(\hat{\theta}) = \frac{1}{\pi\sqrt{\hat{\theta}(1-\hat{\theta})}}$$

  Known in general as Jeffrey's prior. And, in this case, the Dirichlet-$(1/2, 1/2)$ prior.

The cumulative log loss of Bayes using Jeffrey's prior

- $$L_A - L_{\min} \leq \frac{1}{2}\ln(T+1) + \frac{1}{2}\ln\frac{\pi}{2} + O(1/T)$$

But what is the prediction rule?

- As luck would have it the Dirichlet prior is the conjugate prior for the Binomial distribution.

But what is the prediction rule?

- As luck would have it the Dirichlet prior is the conjugate prior for the Binomial distribution.
- Observed t bits, n of which were 1. The posterior is:

$$\frac{1}{Z\sqrt{\theta(1-\theta)}}\theta^{n}(1-\theta)^{t-n} = \frac{1}{Z}\theta^{n-1/2}(1-\theta)^{t-n-1/2}$$

But what is the prediction rule?

- As luck would have it the Dirichlet prior is the conjugate prior for the Binomial distribution.
- Observed t bits, n of which were 1. The posterior is:

$$\frac{1}{Z\sqrt{\theta(1-\theta)}}\theta^{n}(1-\theta)^{t-n}=\frac{1}{Z}\theta^{n-1/2}(1-\theta)^{t-n-1/2}$$

- The posterior average is:

$$\frac{\int_0^1 \theta^{n+1/2}(1-\theta)^{t-n-1/2}d\theta}{\int_0^1 \theta^{n-1/2}(1-\theta)^{t-n-1/2}d\theta}=\frac{n+1/2}{t+1}$$

But what is the prediction rule?

- As luck would have it the Dirichlet prior is the conjugate prior for the Binomial distribution.
- Observed t bits, n of which were 1. The posterior is:

$$\frac{1}{Z\sqrt{\theta(1-\theta)}}\theta^n(1-\theta)^{t-n} = \frac{1}{Z}\theta^{n-1/2}(1-\theta)^{t-n-1/2}$$

- The posterior average is:

$$\frac{\int_0^1 \theta^{n+1/2}(1-\theta)^{t-n-1/2}d\theta}{\int_0^1 \theta^{n-1/2}(1-\theta)^{t-n-1/2}d\theta} = \frac{n+1/2}{t+1}$$

- This is called the Trichevsky Trofimov prediction rule.

Laplace Rule of Succession

- Laplace suggested using the uniform prior, which is also a conjugate prior.

Laplace Rule of Succession

- Laplace suggested using the uniform prior, which is also a conjugate prior.
- In this case the posterior average is:

$$\frac{\int_0^1 \theta^{n+1}(1-\theta)^{t-n}d\theta}{\int_0^1 \theta^{n}(1-\theta)^{t-n}d\theta} = \frac{n+1}{t+2}$$

Laplace Rule of Succession

- Laplace suggested using the uniform prior, which is also a conjugate prior.
- In this case the posterior average is:

$$\frac{\int_0^1 \theta^{n+1}(1-\theta)^{t-n}d\theta}{\int_0^1 \theta^{n}(1-\theta)^{t-n}d\theta} = \frac{n+1}{t+2}$$

- The bound on the cumulative log loss is worse:

$$L_A - L_{\min} = \ln T + O(1)$$

Laplace Rule of Succession

- Laplace suggested using the uniform prior, which is also a conjugate prior.
- In this case the posterior average is:

$$\frac{\int_0^1 \theta^{n+1}(1-\theta)^{t-n}d\theta}{\int_0^1 \theta^{n}(1-\theta)^{t-n}d\theta} = \frac{n+1}{t+2}$$

- The bound on the cumulative log loss is worse:

$$L_A - L_{\min} = \ln T + O(1)$$

- Suffers larger regret when $\hat{\theta}$ is far from $1/2$

Shtarkov Lower bound

- What is the optimal prediction when T is know in advance?

Shtarkov Lower bound

- What is the optimal prediction when T is know in advance?
- $$L_*^T - \min_\theta L_\theta^T \geq \frac{1}{2}\ln(T+1) + \frac{1}{2}\ln\frac{\pi}{2} - O(\frac{1}{\sqrt{T}})$$

Multinomial Distributions

- For a distribution over k elements (Multinomial) [Xie and Barron]

Multinomial Distributions

- For a distribution over k elements (Multinomial) [Xie and Barron]
- Use the add 1/2 rule (KT).

$$p(i) = \frac{n_i + 1/2}{t + k/2}$$

Multinomial Distributions

- For a distribution over k elements (Multinomial) [Xie and Barron]
- Use the add 1/2 rule (KT).

$$p(i) = \frac{n_i + 1/2}{t + k/2}$$

- Bound is

$$L_A - L_{\min} \leq \frac{k-1}{2} \ln T + C + o(1)$$

Multinomial Distributions

- For a distribution over k elements (Multinomial) [Xie and Barron]
- Use the add 1/2 rule (KT).

$$p(i) = \frac{n_i + 1/2}{t + k/2}$$

- Bound is

$$L_A - L_{\min} \leq \frac{k-1}{2} \ln T + C + o(1)$$

- The constant C is optimal.

Exponential Distributions

- For any set of distributions from the exponential family defined by k parameters (constituting an open set) [Rissanen96]

Exponential Distributions

- For any set of distributions from the exponential family defined by k parameters (constituting an open set) [Rissanen96]
- Use Bayes Algorithm with Jeffrey's prior:

$$w^*(\hat{\theta}) = \frac{1}{Z} \frac{1}{\sqrt{\left| \mathbf{H}\left(D_{KL}(\hat{\theta}||\theta) \right)\Big|_{\theta=\hat{\theta}} \right|}}$$

H denotes the Hessian.

Exponential Distributions

- For any set of distributions from the exponential family defined by k parameters (constituting an open set) [Rissanen96]
- Use Bayes Algorithm with Jeffrey's prior:

$$w^*(\hat{\theta}) = \frac{1}{Z}\frac{1}{\sqrt{\left|\mathbf{H}\left(D_{KL}(\hat{\theta}||\theta)\right)\Big|_{\theta=\hat{\theta}}\right|}}$$

H denotes the Hessian.

- $$L_A - L_{\min} \leq \frac{k-1}{2}\ln T - \ln Z + o(1)$$

General Distributions

- Characterize distribution family by metric entropy.

General Distributions

- Characterize distribution family by metric entropy.
- Fixed parameter set usually corresponds to polynomial metrix entropy

$$N(1/\epsilon) = O\left(\frac{1}{\epsilon^d}\right)$$

d is the number of parameters.

General Distributions

- Characterize distribution family by metric entropy.
- Fixed parameter set usually corresponds to polynomial metrix entropy
$$N(1/\epsilon) = O\left(\frac{1}{\epsilon^d}\right)$$
d is the number of parameters.
- [Haussler and Opper] show that the coefficient in front of $\ln T$ is optimal for distribution families where the metric entropy is up to
$$N(1/\epsilon) = O\left(e^{\epsilon^{-\alpha}}\right)$$
For all $\alpha \leq 5/2$.

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

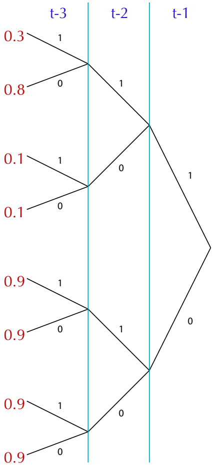
- Observe a binary sequence.
- $x_1, \ldots, x_{t-1}$
- Predict next bit from past
- $P(x_t = 1 | x_{t-1}, x_{t-2}, \ldots, x_1)$
- Use only last k bits
- $P(x_t = 1 | x_{t-1}, \ldots, x_{t-k})$

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

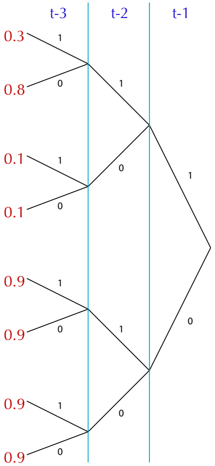
- Each tree leaf is associated with a binary sequence $y_1, \ldots, y_k$
- For each leaf keep two counters:
  - $a_{y_1,\ldots,y_k}$ = number of times $x_{t-1} = y_1, \ldots, x_{t-k} = y_k$ and $x_t = 0$
  - $b_{y_1,\ldots,y_k}$ = number of times $x_{t-1} = y_1, \ldots, x_{t-k} = y_k$ and $x_t = 1$
-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

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

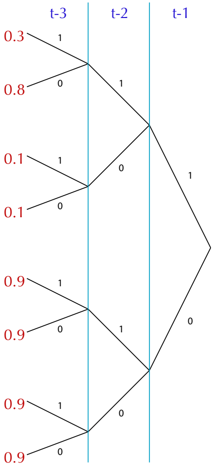
- Total regret is at most $2^{k-1} \log T$

How variable length markov can reduce regret

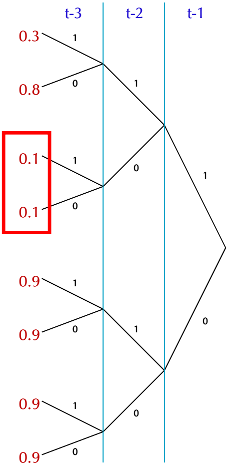
How variable length markov can reduce regret



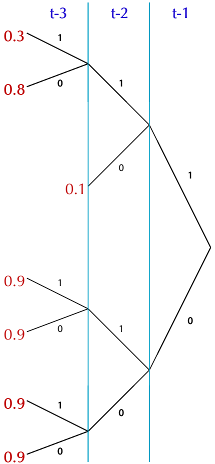
How variable length markov can reduce regret



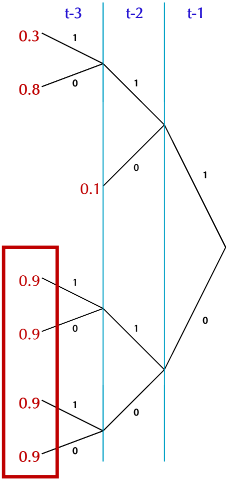
How variable length markov can reduce regret



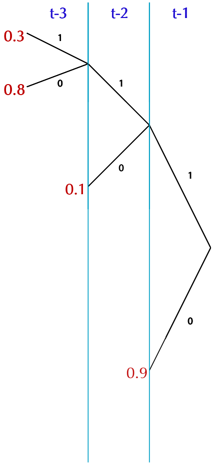
How variable length markov can reduce regret



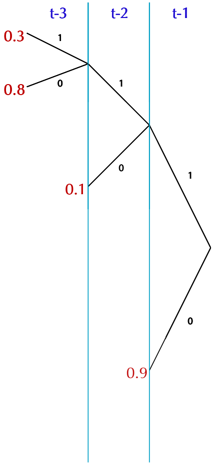
How variable length markov can reduce regret



How variable length markov can reduce regret

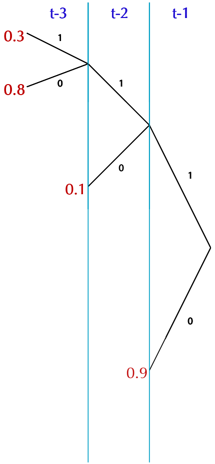


How variable length markov can reduce regret



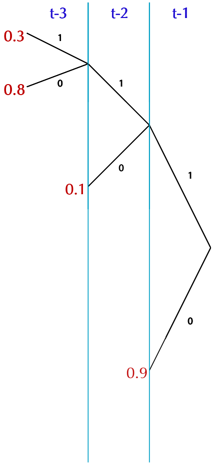
- Reducing number of leaves from 8 to 4 means

How variable length markov can reduce regret



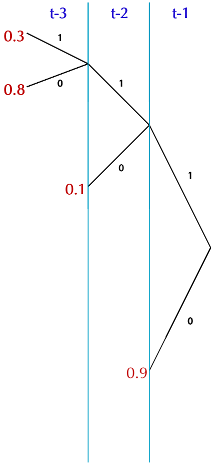
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$

How variable length markov can reduce regret



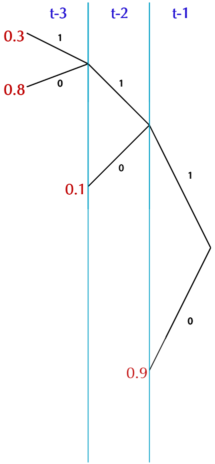
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B

How variable length markov can reduce regret



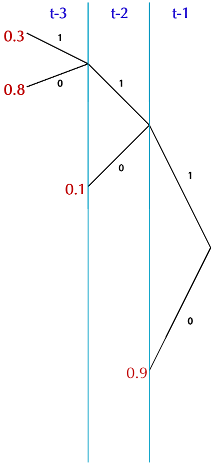
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B

How variable length markov can reduce regret



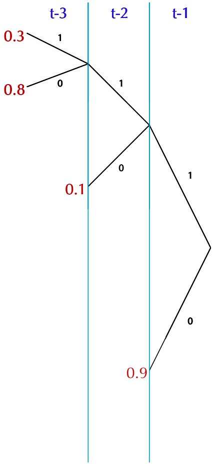
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A

How variable length markov can reduce regret



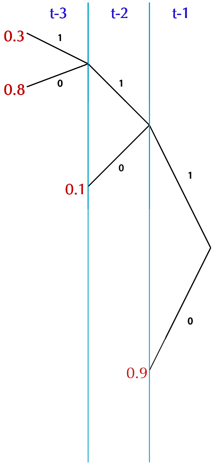
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A R

How variable length markov can reduce regret



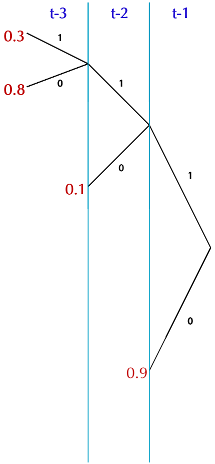
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A R O

How variable length markov can reduce regret



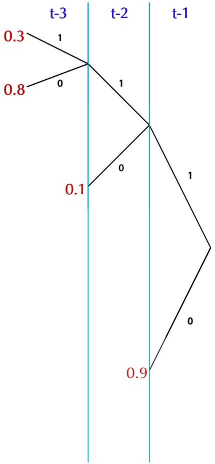
- Reducing number of leaves from 8 to 4 means
- reducing regret from $4 \log T$ to $2 \log T$
- English example:
  B A R O Q

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.

Context-Tree Weighting and Maximizing: Processing Betas

Frans Willems, Tjalling Tjalkens, and Tanya Ignatenko,
Eindhoven University of Technology,
Eindhoven, The Netherlands
Switch to second set of slides.