Tracking the best Expert

Yoav Freund

February 9, 2006

Outline

Review

mixable loss functions

Outline

Outline

Outline

Outline

Vovk's general prediction game

Γ - prediction space. Ω - outcome space.

Vovk's general prediction game

Γ - prediction space. Ω - outcome space.
On each trial $t = 1, 2, \ldots$

Vovk's general prediction game

Γ - prediction space. Ω - outcome space.
On each trial $t = 1, 2, \ldots$

1. Each expert $i \in \{1 \ldots n\}$ makes a prediction $\gamma_i^t \in \Gamma$

Vovk's general prediction game

Γ - prediction space. Ω - outcome space.
On each trial $t = 1, 2, \ldots$

1. Each expert $i \in \{1 \ldots n\}$ makes a prediction $\gamma_i^t \in \Gamma$
2. The learner, after observing $\langle \gamma_1^t \ldots \gamma_n^t \rangle$, makes its own prediction γ^t

Vovk's general prediction game

Γ - prediction space. Ω - outcome space.
On each trial $t = 1, 2, \ldots$

1. Each expert $i \in \{1 \ldots n\}$ makes a prediction $\gamma_i^t \in \Gamma$
2. The learner, after observing $\langle \gamma_1^t \ldots \gamma_n^t \rangle$, makes its own prediction γ^t
3. Nature chooses an outcome $\omega^t \in \Omega$

Vovk's general prediction game

Γ - prediction space. Ω - outcome space.
On each trial $t = 1, 2, \ldots$

1. Each expert $i \in \{1 \ldots n\}$ makes a prediction $\gamma_i^t \in \Gamma$
2. The learner, after observing $\langle \gamma_1^t \ldots \gamma_n^t \rangle$, makes its own prediction γ^t
3. Nature chooses an outcome $\omega^t \in \Omega$
4. Each expert incurs loss $\ell_i^t = \lambda(\omega^t, \gamma_i^t)$
   The learner incurs loss $\ell_A^t = \lambda(\omega^t, \gamma^t)$

Vovk's algorithm is the the highest achiever [Vovk95]

The pair (a, c) is achieved by some algorithm if and only if it is achieved by Vovk's algorithm.

Vovk's algorithm is the the highest achiever [Vovk95]

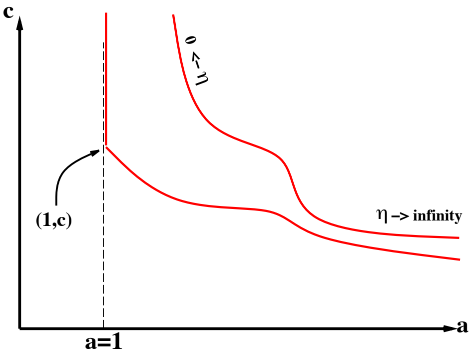
The pair (a, c) is achieved by some algorithm if and only if it is achieved by Vovk's algorithm.

The separation curve is $\left\{ \left(a(\eta), \frac{a(\eta)}{\eta} \right) \middle| \eta \in [0, \infty] \right\}$

Vovk's algorithm is the the highest achiever [Vovk95]

The pair (a, c) is achieved by some algorithm if and only if it is achieved by Vovk's algorithm.

The separation curve is $\left\{ \left(a(\eta), \frac{a(\eta)}{\eta} \right) \middle| \eta \in [0, \infty] \right\}$



Log loss (Entropy loss)

- $$\lambda_{\text{ent}}(\omega,\gamma) = \omega \ln \frac{\omega}{\gamma} + (1-\omega) \ln \frac{1-\omega}{1-\gamma}$$

Log loss (Entropy loss)

- $$\lambda_{\text{ent}}(\omega, \gamma) = \omega \ln \frac{\omega}{\gamma} + (1 - \omega) \ln \frac{1 - \omega}{1 - \gamma}$$
- When $q_t \in \{0, 1\}$ Cumulative log loss $=$ coding length ± 1

Log loss (Entropy loss)

- $$\lambda_{\text{ent}}(\omega, \gamma) = \omega \ln \frac{\omega}{\gamma} + (1 - \omega) \ln \frac{1 - \omega}{1 - \gamma}$$
- When $q_t \in \{0, 1\}$ Cumulative log loss $=$ coding length ± 1
- If $P[\omega_t = 1] = q$, optimal prediction $\gamma^t = q$

Log loss (Entropy loss)

- $$\lambda_{\text{ent}}(\omega,\gamma) = \omega \ln \frac{\omega}{\gamma} + (1-\omega) \ln \frac{1-\omega}{1-\gamma}$$
- When $q_t \in \{0,1\}$ Cumulative log loss $=$ coding length ± 1
- If $P[\omega_t = 1] = q$, optimal prediction $\gamma^t = q$
- Unbounded loss.

Log loss (Entropy loss)

- $$\lambda_{\text{ent}}(\omega, \gamma) = \omega \ln \frac{\omega}{\gamma} + (1 - \omega) \ln \frac{1 - \omega}{1 - \gamma}$$
- When $q_t \in \{0, 1\}$ Cumulative log loss $=$ coding length ± 1
- If $P[\omega_t = 1] = q$, optimal prediction $\gamma^t = q$
- Unbounded loss.
- Not symmetric $\exists p, q \;\; \lambda(p, q) \neq \lambda(q, p)$.

Log loss (Entropy loss)

- $$\lambda_{\text{ent}}(\omega,\gamma) = \omega \ln \frac{\omega}{\gamma} + (1-\omega) \ln \frac{1-\omega}{1-\gamma}$$
- When $q_t \in \{0,1\}$ Cumulative log loss $=$ coding length ± 1
- If $P[\omega_t = 1] = q$, optimal prediction $\gamma^t = q$
- Unbounded loss.
- Not symmetric $\exists p, q \;\; \lambda(p,q) \neq \lambda(q,p)$.
- No triangle inequality
  $\exists p_1, p_2, p_3 \;\; \lambda(p_1,p_3) > \lambda(p_1,p_2) + \lambda(p_2,p_3)$

Square loss (Breier Loss)

- $$\lambda_{\mathsf{sq}}(\omega, \gamma) = (\omega - \gamma)^2$$

Square loss (Breier Loss)

- $$\lambda_{\text{sq}}(\omega, \gamma) = (\omega - \gamma)^2$$
- $P[\omega^t = 1] = q, \;\; P[\omega^t = 0] = 1 - q$,
  optimal prediction $\gamma^t = q$

Square loss (Breier Loss)

- $$\lambda_{\text{sq}}(\omega, \gamma) = (\omega - \gamma)^2$$
- $P[\omega^t = 1] = q, \quad P[\omega^t = 0] = 1 - q$,
  optimal prediction $\gamma^t = q$
- Bounded loss.

Square loss (Breier Loss)

- $$\lambda_{\text{sq}}(\omega, \gamma) = (\omega - \gamma)^2$$
- $P[\omega^t = 1] = q, \;\; P[\omega^t = 0] = 1 - q$,
  optimal prediction $\gamma^t = q$
- Bounded loss.
- Defines a metric (symmetric and triangle ineq.)

Square loss (Breier Loss)

- $$\lambda_{\mathrm{sq}}(\omega, \gamma) = (\omega - \gamma)^2$$
- $P[\omega^t = 1] = q, \quad P[\omega^t = 0] = 1 - q$,
  optimal prediction $\gamma^t = q$
- Bounded loss.
- Defines a metric (symmetric and triangle ineq.)
- Corresponds to regression.

Absolute loss

- $$\lambda(\omega, \gamma) = |\omega - \gamma|$$

Absolute loss

- $$\lambda(\omega, \gamma) = |\omega - \gamma|$$
- Probability of making a mistake if predicting 0 or 1 using a biased coin

Absolute loss

- $$\lambda(\omega, \gamma) = |\omega - \gamma|$$
- Probability of making a mistake if predicting 0 or 1 using a biased coin
- If $P[\omega^t = 1] = q, \;\; P[\omega^t = 0] = 1 - q$, then the optimal prediction is
$$\gamma^t = \begin{cases} 1 & \text{if } q > 1/2, \\ 0 & \text{otherwise} \end{cases}$$

Mixable Loss Functions

- A Loss function is mixable if a pair of the form $(1, c),\ c < \infty$ is achievable.

$$L_A \leq L_{\min} + c \ln n$$

Mixable Loss Functions

- A Loss function is mixable if a pair of the form $(1, c),\ c < \infty$ is achievable.

$$L_A \leq L_{\min} + c \ln n$$

- Vovk's algorithm with $\eta = 1/c$ achieves this bound.

Mixable Loss Functions

- A Loss function is mixable if a pair of the form $(1, c),\ c < \infty$ is achievable.

$$L_A \leq L_{\min} + c \ln n$$

- Vovk's algorithm with $\eta = 1/c$ achieves this bound.
- $\lambda_{\text{ent}}, \lambda_{\text{sq}}, \lambda_{\text{hel}}$ are mixable

Mixable Loss Functions

- A Loss function is mixable if a pair of the form $(1, c),\ c < \infty$ is achievable.

$$L_A \leq L_{\min} + c \ln n$$

- Vovk's algorithm with $\eta = 1/c$ achieves this bound.
- $\lambda_{\text{ent}}, \lambda_{\text{sq}}, \lambda_{\text{hel}}$ are mixable
- $\lambda_{\text{abs}}, \lambda_{\text{dot}}$ are not mixable

Summary of bounds for mixable losses

TRACKING THE BEST EXPERT

Loss Functions:	c values: $(\eta = 1/c)$	
	$\mathbf{pred}_{\mathrm{wmean}}(v, x)$	$\mathbf{pred}_{\mathrm{Vovk}}(v, x)$
$L_{\mathrm{sq}}(p, q)$	2	$1/2$
$L_{\mathrm{ent}}(p, q)$	1	1
$L_{\mathrm{hel}}(p, q)$	1	$1/\sqrt{2}$

Figure 2. $(c, 1/c)$-realizability: c values for loss and prediction function pairin

Switching experts setup

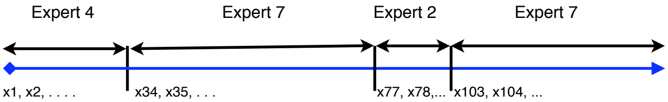
- Usually: compare algorithm's total loss to total loss of the best expert.

Switching experts setup

- Usually: compare algorithm's total loss to total loss of the best expert.
- Switching experts: compare algorithm's total loss to total loss of best expert sequence with k switches.

Switching experts setup

- Usually: compare algorithm's total loss to total loss of the best expert.
- Switching experts: compare algorithm's total loss to total loss of best expert sequence with *k* switches.
-



An inefficient algorithm

- Fix:

An inefficient algorithm

- Fix:
  - l - sequence length

An inefficient algorithm

- Fix:
  - l - sequence length
  - k - number of switches

An inefficient algorithm

- Fix:
  - l - sequence length
  - k - number of switches
  - n - number of experts

An inefficient algorithm

- Fix:
  - l - sequence length
  - k - number of switches
  - n - number of experts
- Consider one partition-expert per sequence of switching experts.

An inefficient algorithm

- Fix:
  - l - sequence length
  - k - number of switches
  - n - number of experts
- Consider one partition-expert per sequence of switching experts.
- No. of partition-experts : $\binom{l}{k-1} n(n-1)^k = O\left(n^{k+1} \left(\frac{el}{k}\right)^k\right)$

An inefficient algorithm

- Fix:
  - l - sequence length
  - k - number of switches
  - n - number of experts
- Consider one partition-expert per sequence of switching experts.
- No. of partition-experts : $\binom{l}{k-1} n(n-1)^k = O\left(n^{k+1} \left(\frac{el}{k}\right)^k\right)$
- The log-loss regret is at most $(k+1)\log n + k \log \frac{l}{k} + k$

An inefficient algorithm

- Fix:
  - l - sequence length
  - k - number of switches
  - n - number of experts
- Consider one partition-expert per sequence of switching experts.
- No. of partition-experts : $\binom{l}{k-1} n(n-1)^k = O\left(n^{k+1}\left(\frac{el}{k}\right)^k\right)$
- The log-loss regret is at most $(k+1)\log n + k \log \frac{l}{k} + k$
- Requires maintaining $O\left(n^{k+1}\left(\frac{el}{k}\right)^k\right)$ weights.

generalization to mixable losses

- In this lecture we assume loss function is mixable.

generalization to mixable losses

- In this lecture we assume loss function is mixable.
- There is an exponential weights algorithm with learning rate η that achieves (in the non-switching case) a bound

$$L_A \leq \min_i L_i + \frac{1}{\eta} \log n$$

generalization to mixable losses

- In this lecture we assume loss function is mixable.
- There is an exponential weights algorithm with learning rate η that achieves (in the non-switching case) a bound

$$L_A \leq \min_i L_i + \frac{1}{\eta} \log n$$

- Then using the partition-expert algorithm for the switching-experts case we get a bound on the regret $\frac{1}{\eta}\left((k+1)\log n + k \log \frac{l}{k} + k\right)$

Weight sharing algorithms

- Update weights in two stages: loss update then share update.

Weight sharing algorithms

- Update weights in two stages: loss update then share update.
- Prediction uses the normalized s weights $w^s_{t,i} / \sum_j w^s_{t,j}$

Weight sharing algorithms

- Update weights in two stages: loss update then share update.
- Prediction uses the normalized s weights $w^s_{t,i} / \sum_j w^s_{t,j}$
- Loss update is the same as always, but defines intermediate m weights:

$$w^m_{t,i} = w^s_{t,i} e^{-\eta L(y_t, x_{t,i})}$$

Weight sharing algorithms

- Update weights in two stages: loss update then share update.
- Prediction uses the normalized s weights $w^s_{t,i} / \sum_j w^s_{t,j}$
- Loss update is the same as always, but defines intermediate m weights:

$$w^m_{t,i} = w^s_{t,i} e^{-\eta L(y_t, x_{t,i})}$$

- Share update: redistribute the weights

Weight sharing algorithms

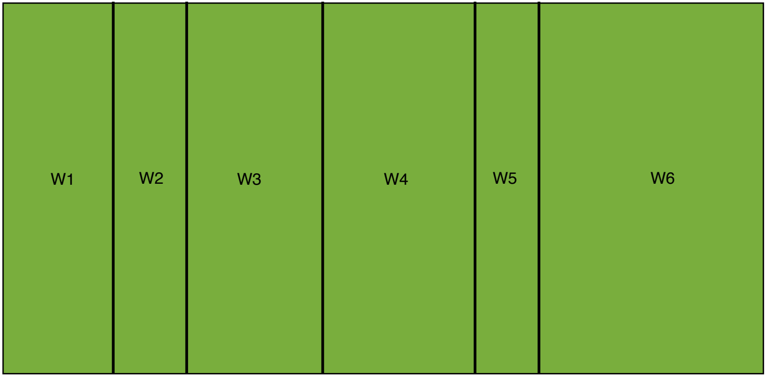
- Update weights in two stages: loss update then share update.
- Prediction uses the normalized s weights $w^s_{t,i} / \sum_j w^s_{t,j}$
- Loss update is the same as always, but defines intermediate m weights:

$$w^m_{t,i} = w^s_{t,i} e^{-\eta L(y_t, x_{t,i})}$$

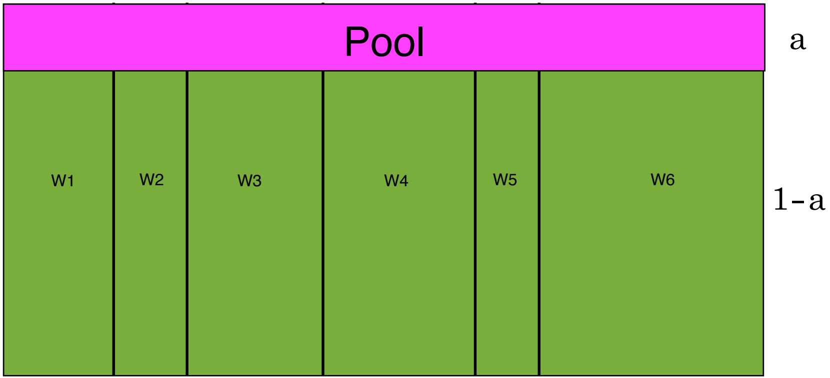
- Share update: redistribute the weights
- Fixed-share:

$$\begin{aligned} pool &= \alpha \sum_{i=1}^{n} w^m_{t,i} \\ w^s_{t+1,i} &= (1-\alpha) w^m_{t,i} + \frac{1}{n-1}\left(pool - \alpha w^m_{t,i}\right) \end{aligned}$$

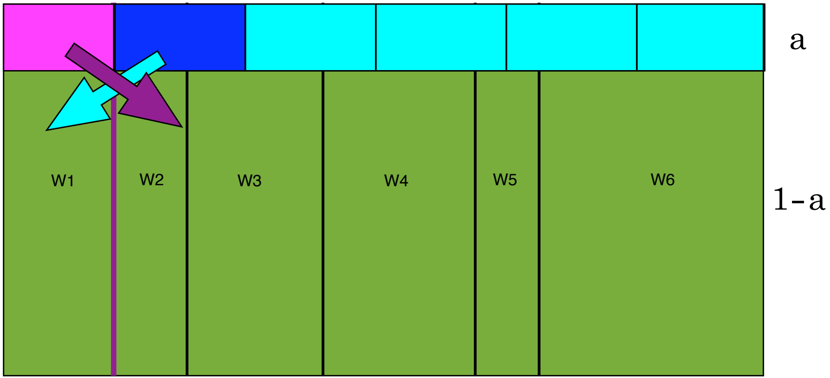
The fixed-share algorithm



The fixed-share algorithm



The fixed-share algorithm



Proving a bound on the fixed-share

- The relation between algorithm loss and total weight does not change because share update does not change the total weight.

Proving a bound on the fixed-share

- The relation between algorithm loss and total weight does not change because share update does not change the total weight.
- Thus we still have

$$L_A \leq \frac{1}{\eta} \sum_{i=1}^{n} w^s_{l+1,i}$$

Proving a bound on the fixed-share

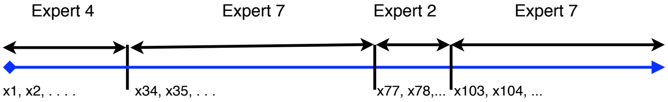
- The relation between algorithm loss and total weight does not change because share update does not change the total weight.
- Thus we still have

$$L_A \leq \frac{1}{\eta} \sum_{i=1}^{n} w^s_{l+1,i}$$

- The harder question is how to lower bound $\sum_{i=1}^{n} w^s_{l+1,i}$

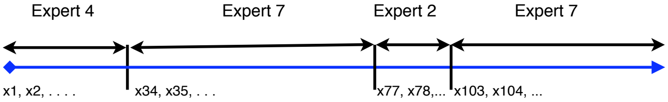
Lower bounding the final total weight

- Fix some switching experts sequence:



Lower bounding the final total weight

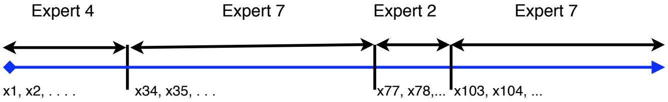
- Fix some switching experts sequence:



- "follow" the weight of the chosen expert i_t.

Lower bounding the final total weight

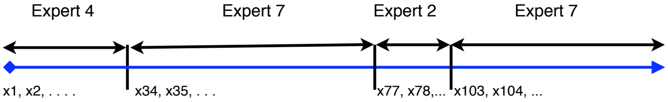
- Fix some switching experts sequence:



- "follow" the weight of the chosen expert i_t.
- The loss update reduces the weight by a factor of $e^{-\eta \ell_{t,i_t}}$.

Lower bounding the final total weight

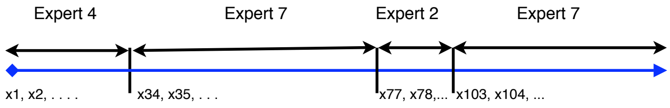
- Fix some switching experts sequence:



- "follow" the weight of the chosen expert i_t.
- The loss update reduces the weight by a factor of $e^{-\eta \ell_{t,i_t}}$.
- The share update reduces the weight by a factor larger than:

Lower bounding the final total weight

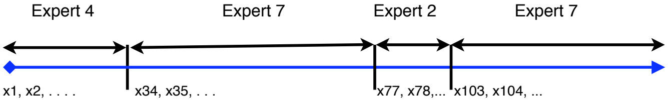
- Fix some switching experts sequence:



- "follow" the weight of the chosen expert i_t.
- The loss update reduces the weight by a factor of $e^{-\eta \ell_{t,i_t}}$.
- The share update reduces the weight by a factor larger than:
  - $1 - \alpha$ on iterations with no switch.

Lower bounding the final total weight

- Fix some switching experts sequence:



- "follow" the weight of the chosen expert i_t.
- The loss update reduces the weight by a factor of $e^{-\eta \ell_{t,i_t}}$.
- The share update reduces the weight by a factor larger than:
  - $1 - \alpha$ on iterations with no switch.
  - $\frac{\alpha}{n-1}$ on iterations where a switch occurs.

Bound for arbitrary α

- Combining we lower bound the final weight of the last expert in the sequence

$$w^s_{l+1,e_k} \geq \frac{1}{n} e^{-\eta L_*} (1-\alpha)^{l-k-1} \left(\frac{\alpha}{n-1} \right)^k$$

Where L_* is the cumulative loss of the switching sequence of experts.

Bound for arbitrary α

- Combining we lower bound the final weight of the last expert in the sequence

$$w^{s}_{l+1,e_k} \geq \frac{1}{n} e^{-\eta L_*} (1-\alpha)^{l-k-1} \left(\frac{\alpha}{n-1} \right)^{k}$$

Where L_* is the cumulative loss of the switching sequence of experts.

- Combining the upper and lower bounds we get that for any sequence

$$L_A \leq L_* + \frac{1}{\eta} \left(\ln n + (l-k-1) \ln \frac{1}{1-\alpha} + k \left(\ln \frac{1}{\alpha} + \ln(n-1) \right) \right)$$

Tuning α

- let k^* be the best number of switches (in hind sight) and $\alpha^* = k^*/l$

Tuning α

- let k^* be the best number of switches (in hind sight) and $\alpha^* = k^*/l$
- Suppose we use $\alpha \approx \alpha^*$ then the bound that we get is

$$L_A \leq L_* + \frac{1}{\eta}((k+1)\ln n + (l-1)(H(\alpha^*) + D_{\mathrm{KL}}(\alpha^* || \alpha)))$$

Where

$$H(\alpha^*) = -\alpha^* \ln \alpha^* - (1-\alpha^*)\ln(1-\alpha^*)$$

$$D_{\mathrm{KL}}(\alpha^* || \alpha) = \alpha^* \ln \frac{\alpha^*}{\alpha}(1-\alpha^*)\ln \frac{1-\alpha^*}{1-\alpha}$$

Tuning α

- let k^* be the best number of switches (in hind sight) and $\alpha^* = k^*/l$
- Suppose we use $\alpha \approx \alpha^*$ then the bound that we get is

$$L_A \leq L_* + \frac{1}{\eta}((k+1)\ln n + (l-1)(H(\alpha^*) + D_{\mathrm{KL}}(\alpha^*||\alpha)))$$

Where

$$H(\alpha^*) = -\alpha^* \ln \alpha^* - (1-\alpha^*)\ln(1-\alpha^*)$$

$$D_{\mathrm{KL}}(\alpha^*||\alpha) = \alpha^* \ln \frac{\alpha^*}{\alpha}(1-\alpha^*)\ln \frac{1-\alpha^*}{1-\alpha}$$

- This is very close to the loss of the computationally inefficient algorithm.

Tuning α

- let k^* be the best number of switches (in hind sight) and $\alpha^* = k^*/l$
- Suppose we use $\alpha \approx \alpha^*$ then the bound that we get is

$$L_A \leq L_* + \frac{1}{\eta}((k+1)\ln n + (l-1)(H(\alpha^*) + D_{\mathrm{KL}}(\alpha^*||\alpha)))$$

Where

$$H(\alpha^*) = -\alpha^* \ln \alpha^* - (1-\alpha^*)\ln(1-\alpha^*)$$

$$D_{\mathrm{KL}}(\alpha^*||\alpha) = \alpha^* \ln \frac{\alpha^*}{\alpha}(1-\alpha^*)\ln \frac{1-\alpha^*}{1-\alpha}$$

- This is very close to the loss of the computationally inefficient algorithm.
- For the log loss case this is essentially optimal.

Tuning α

- let k^* be the best number of switches (in hind sight) and $\alpha^* = k^*/l$
- Suppose we use $\alpha \approx \alpha^*$ then the bound that we get is

$$L_A \leq L_* + \frac{1}{\eta}((k+1)\ln n + (l-1)(H(\alpha^*) + D_{\text{KL}}(\alpha^*||\alpha)))$$

Where

$$H(\alpha^*) = -\alpha^* \ln \alpha^* - (1-\alpha^*)\ln(1-\alpha^*)$$

$$D_{\text{KL}}(\alpha^*||\alpha) = \alpha^* \ln \frac{\alpha^*}{\alpha}(1-\alpha^*)\ln \frac{1-\alpha^*}{1-\alpha}$$

- This is very close to the loss of the computationally inefficient algorithm.
- For the log loss case this is essentially optimal.
- Not so for square loss!

What can we hope to improve?

- In the fixed-share algorithm, the weight of a suboptimal expert never decreases below α/n.

What can we hope to improve?

- In the fixed-share algorithm, the weight of a suboptimal expert never decreases below α/n.
- The algorithm does not concentrate only on the best expert, even if the last switch is in the distant past.

What can we hope to improve?

- In the fixed-share algorithm, the weight of a suboptimal expert never decreases below α/n.
- The algorithm does not concentrate only on the best expert, even if the last switch is in the distant past.
- The regret depends on the length of the sequence.

The idea of variable-share

- Let the fraction of the total weight given to the best expert get arbitrarily close to 1.

The idea of variable-share

- Let the fraction of the total weight given to the best expert get arbitrarily close to 1.
- we can get a regret bound that depends only on the number of switches, not on the lenght of the sequence.

The idea of variable-share

- Let the fraction of the total weight given to the best expert get arbitrarily close to 1.
- we can get a regret bound that depends only on the number of switches, not on the lenght of the sequence.
- Requires that the loss be bounded.

The idea of variable-share

- Let the fraction of the total weight given to the best expert get arbitrarily close to 1.
- we can get a regret bound that depends only on the number of switches, not on the lenght of the sequence.
- Requires that the loss be bounded.
- Works for square loss, but not for log loss!

Variable-share

$$
\begin{aligned}
pool &= \sum_{i=1}^{n} \left(1 - (1-\alpha)^{\ell_{t,i}}\right) w^{m}_{t,i} \\
w^{s}_{t+1,i} &= (1-\alpha)^{\ell_{t,i}} w^{m}_{t,i} + \frac{1}{n-1}\left(pool - \left(1 - (1-\alpha)^{\ell_{t,i}}\right) w^{m}_{t,i}\right)
\end{aligned}
$$

Variable-share

$$\begin{aligned} pool &= \sum_{i=1}^{n} \left(1 - (1-\alpha)^{\ell_{t,i}}\right) w^m_{t,i} \\ w^s_{t+1,i} &= (1-\alpha)^{\ell_{t,i}} w^m_{t,i} + \frac{1}{n-1}\left(pool - \left(1 - (1-\alpha)^{\ell_{t,i}}\right) w^m_{t,i}\right) \end{aligned}$$

If $\ell_{t,i} = 0$, then expert i does not contribute to the pool.

Variable-share

$$\begin{aligned} pool &= \sum_{i=1}^{n} \left(1 - (1-\alpha)^{\ell_{t,i}}\right) w_{t,i}^{m} \\ w_{t+1,i}^{s} &= (1-\alpha)^{\ell_{t,i}} w_{t,i}^{m} + \frac{1}{n-1}\left(pool - \left(1 - (1-\alpha)^{\ell_{t,i}}\right) w_{t,i}^{m}\right) \end{aligned}$$

If $\ell_{t,i} = 0$, then expert i does not contribute to the pool.
Expert can get fraction of the total weight arbitrarily close to 1.

Variable-share

$$\begin{aligned} pool &= \sum_{i=1}^{n} \left(1 - (1-\alpha)^{\ell_{t,i}}\right) w^m_{t,i} \\ w^s_{t+1,i} &= (1-\alpha)^{\ell_{t,i}} w^m_{t,i} + \frac{1}{n-1}\left(pool - \left(1 - (1-\alpha)^{\ell_{t,i}}\right) w^m_{t,i}\right) \end{aligned}$$

If $\ell_{t,i} = 0$, then expert i does not contribute to the pool.
Expert can get fraction of the total weight arbitrarily close to 1.
Shares the weight quickly if $\ell_{t,i} > 0$

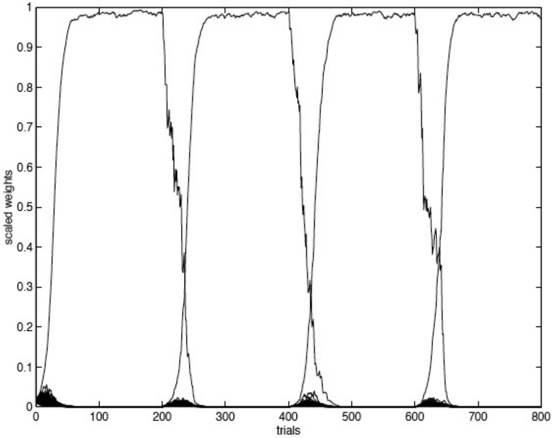
Bound for variable share

- $$\frac{1}{\eta}\ln n + \left(1 + \frac{1}{(1-\alpha)\eta}\right)L_* + k\left(1 + \frac{1}{\eta}\left(\ln n - 1 + \ln\frac{1}{\alpha} + \ln\frac{1}{1-\alpha}\right)\right)$$

Bound for variable share

- $$\frac{1}{\eta}\ln n + \left(1 + \frac{1}{(1-\alpha)\eta}\right)L_* + k\left(1 + \frac{1}{\eta}\left(\ln n - 1 + \ln\frac{1}{\alpha} + \ln\frac{1}{1-\alpha}\right)\right)$$
- α should be tuned so that it is (close to) $\frac{k}{2k+L_*}$

An experiment using variable share



Next Class

- Suppose the best switching sequence is repeatedly switching among a small subset of the experts $n' \ll n$

Next Class

- Suppose the best switching sequence is repeatedly switching among a small subset of the experts $n' \ll n$
- In the context of speech recognition - the speaker repeatedly uses a small number of phonemes.

Next Class

- Suppose the best switching sequence is repeatedly switching among a small subset of the experts $n' \ll n$
- In the context of speech recognition - the speaker repeatedly uses a small number of phonemes.
- If we know the subset, we can pay $\ln n'$ per switch rather than $\ln n$

Next Class

- Suppose the best switching sequence is repeatedly switching among a small subset of the experts $n' \ll n$
- In the context of speech recognition - the speaker repeatedly uses a small number of phonemes.
- If we know the subset, we can pay $\ln n'$ per switch rather than $\ln n$
- Can track switches much more closely.

Next Class

- Suppose the best switching sequence is repeatedly switching among a small subset of the experts $n' \ll n$
- In the context of speech recognition - the speaker repeatedly uses a small number of phonemes.
- If we know the subset, we can pay $\ln n'$ per switch rather than $\ln n$
- Can track switches much more closely.
- Easy to describe an inefficient algorithm (consider all $\binom{n}{n'}$ subsets.)

Next Class

- Suppose the best switching sequence is repeatedly switching among a small subset of the experts $n' \ll n$
- In the context of speech recognition - the speaker repeatedly uses a small number of phonemes.
- If we know the subset, we can pay $\ln n'$ per switch rather than $\ln n$
- Can track switches much more closely.
- Easy to describe an inefficient algorithm (consider all $\binom{n}{n'}$ subsets.)
- Next class - how to do as well with just one weight per expert.