10. Online learning

GI01/M055: Supervised Learning

Mark Herbster

University College London
Department of Computer Science

8 December 2014

Acknowledgments and References

Thanks

Thanks to Yoav Freund and Claudio Gentile for many of the slides.

A general reference for online learning

- Nicolò Cesa-Bianchi and Gábor Lugosi, *Prediction, learning, and games.*

See final slide for more references.

Batch versus Online learning

Batch

Model: There exists **training** data set (sampled **IID**)
Aim: To build a classifier from the training data that predicts well on new data (*from same distribution*)
Evaluation metric: *Generalization error*

Online

Model: There exists an **online sequence** of data (*usually no distributional assumptions*)
Aim: To sequentially predict and update a classifier to predict well on the sequence (i.e. there is no training and test set distinction)
Evaluation metric: *Cumulative error*

Note

There are a variety of models for online learning. Here we focus on the so-called *worst-case* model. Alternately distributional assumption may be made on the data sequence. Also sometimes the phrase "online learning" is used to refer to "online optimisation" that is to use online learning type algorithms as a *training* method for a batch classifier.

Why online learning?

Pragmatically

- "Often" fast algorithms
- "Often" small memory footprint
- "Often" no "statistical" assumptions required e.g. IID-ness
- As a *training* method for **"BIG DATA"** batch classifiers

Theoretically (learning performance guarantees)

- Non-asymptotic
- No statistical assumptions
- There exist techniques to convert *cumulative error* guarantees to *generalisation error* guarantees

Today

Our focus today is on three foundational online "hypotheses" classes.

- Learning with experts
  1. Halving algorithm
  2. Weighted Majority algorithm
  3. Other loss functions
- Learning with linear combinations of experts
  1. Perceptron
  2. Winnow
  3. Case study: Using perceptron and winnow to learn Disjunctions and `DNF`
- Learning with sequences of experts

Experts

Part I

Learning with Expert Advice

On-Line Learning with expert advice (1) [V90,LW94,HKW98]

Model: There exists an **online sequence** of data

$$S = (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m) \in \{0,1\}^n \times \{0,1\}.$$

Interpretation: The vector $\boldsymbol{x}_t$ is the set of predictions of n experts about an outcome y_t. Where expert i predicts $x_{t,i} \in \{0,1\}$ at time t. Each expert at time t is aiming to predict y_t. What is an "expert"? These may be for example human experts or the predictions of some algorithm.

	experts						
	E_1	E_2	E_3	E_n	prediction	true label	loss
day 1	1	1	0	0	0	1	1
day 2	1	0	1	0	1	0	1
day 3	0	1	1	1	1	1	0
day t	$x_{t,1}$	$x_{t,2}$	$x_{t,3}$	$x_{t,n}$	$\hat{y}_t$	y_t	$\lvert y_t - \hat{y}_t \rvert$

Goal: A *"Master"* algorithm to combine the predictions $\boldsymbol{x}_t$ of the n experts (based on past perf.) to predict $\hat{y}_t$ an estimate of y_t.

On-Line Learning with experts (2)

Protocol of the Master Algorithm

For $t = 1$ To m Do

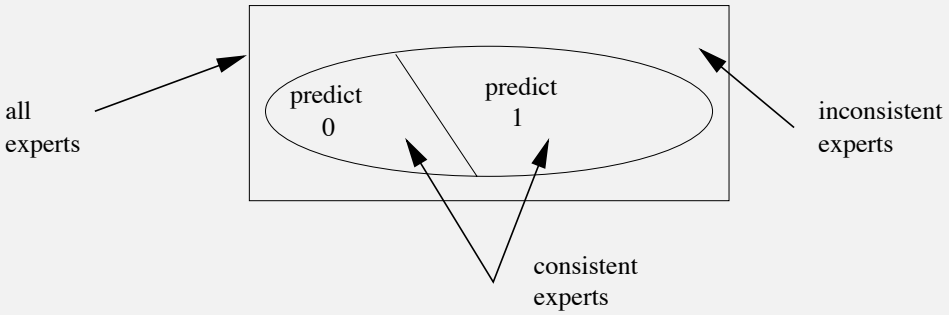
Get instance	$\mathbf{x}_t \in \{0,1\}^n$
Predict	$\hat{y}_t \in \{0,1\}$
Get label	$y_t \in \{0,1\}$
Incur loss (mistakes)	$\lvert y_t - \hat{y}_t \rvert$

Evaluation metric: The loss (mistakes) of Master Algorithm A on sequence S is just

$$L_A(S) := \sum_{t=1}^{m} |y_t - \hat{y}_t|$$

Our Goal: Design master algorithms with "small loss".

A Solution : Halving Algorithm



- Predicts with majority
- If mistake is made then number of consistent experts is (at least) halved

A run of the Halving Algorithm

E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	*majority*	*true label*	loss
1	1	0	0	1	1	0	0	1	0	1
x	x	0	1	x	x	1	1	1	1	0
x	x	x	1	x	x	0	0	0	1	1
x	x	x	↑	x	x	x	x			
			consistent							

Observation: For any sequence with a consistent expert Halving Algorithm makes $\leq \log_2 n$ mistakes.
Exercise: Prove this!

What if no expert is consistent?

Notation

- Recall $L_A(S) := \sum_{t=1}^{m} |y_t - \hat{y}_t|$ is the loss of algorithm A on S
- Let

$$L_i(S) := \sum_{t=1}^{m} |y_t - x_{t,i}|$$

be the loss of i-th expert E_i

Aim

Bounds of the form:

$$\forall S: \quad L_A(S) \leq a \min_i L_i(S) + b \log(n)$$

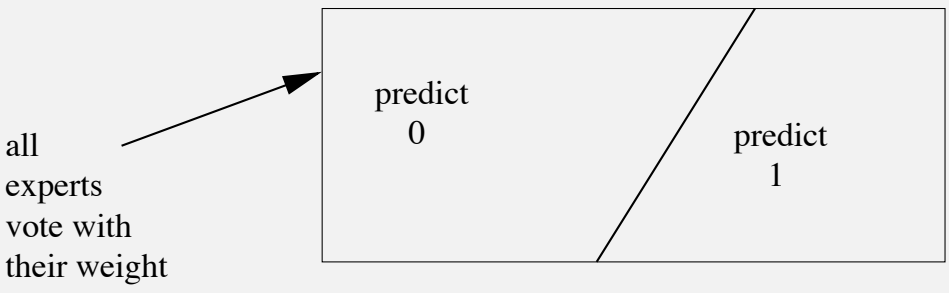
where a, b are "small" constants

Comment: These are known as "Relative" or "Worst-case" loss bounds, i.e., the bounds on the loss of algorithm are "relative to" loss of best expert and they hold even in the "worst-case."

A Solution: Weighted Majority Algorithm [LW94]

Can't wipe out experts!

One weight per expert



- Predicts with larger side
- Weights of wrong experts are multiplied by $\beta \in [0, 1)$

Number of mistakes of the WM algorithm

$$
\begin{aligned}
M_{t,i} &= \#\text{ mistakes of expert } E_i \text{ at the start of trial } t \\
M_i &= M_{m+1,i} - \#\text{ of total mistakes of expert } E_i \\
w_{t,i} &= \beta^{M_{t,i}} \text{ weight of } E_i \text{ at beginning of trial } t \\
W_t &= \sum_{i=1}^{n} w_{t,i} \quad \text{total weight at trial } t
\end{aligned}
$$

Minority: $\leq \frac{1}{2} W_t$ Majority $\geq \frac{1}{2} W_t$

If no mistake then minority multiplied by β:
$W_{t+1} \leq 1\ W_t$
Else If mistake then majority multiplied by β:

$$
W_{t+1} \leq \underbrace{1 \quad \frac{1}{2} W_t}_{\text{minority}} + \underbrace{\beta \quad \frac{1}{2} W_t}_{\text{majority}}
$$

$$
= \frac{1+\beta}{2}\ W_t
$$

Number of mistakes of the WM algorithm – Continued-1

Hence

$$
\underset{\substack{\textit{totalfinal} \\ \textit{weight}}}{W_{m+1}} \leq \left(\frac{1+\beta}{2}\right)^{M} W_1
$$

$$
W_{m+1} = \sum_{j=1}^{n} w_{m+1,j} = \sum_{j=1}^{n} \beta^{M_j} \geq \beta^{M_i}
$$

We get:

$$
\left(\frac{1+\beta}{2}\right)^{M} \underbrace{W_1}_{n} \geq \beta^{M_i}
$$

Number of mistakes of the WM algorithm – Continued-2

Solving for M:

$$M \leq \frac{\ln \frac{1}{\beta}}{\ln \frac{2}{1+\beta}} M_i + \frac{1}{\ln \frac{2}{1+\beta}} \ln n$$

$$M \underset{\beta = 1/e}{\leq} \underbrace{2.63}_{a} \min_i M_i + \underbrace{2.63}_{b} \ln n$$

For all sequences, loss of master algorithm is comparable to loss of best expert. Thus the terminology Relative loss bound.

Other Loss Functions

Example loss functions $L : [0,1] \times [0,1] \to [0, +\infty]$

absolute loss: $L(y, \hat{y}) = |y - \hat{y}|$
square loss: $L(y, \hat{y}) = (y - \hat{y})^2$
entropic loss: $L(y, \hat{y}) = y \ln \frac{y}{\hat{y}} + (1 - y) \ln \frac{1-y}{1-\hat{y}}$

One weight per expert:

$$w_{t,i} = \beta^{\,L_{t,i}} = e^{-\eta\, L_{t,i}}$$

where $L_{t,i}$ is total loss of E_i before trial t
and η is a positive learning rate

Master predicts with the weighted average (dot product)

$$v_{t,i} = \frac{w_{t,i}}{\sum_{i=1}^{n} w_{t,i}} \quad \text{normalized weights}$$

$$\hat{y}_t = \sum_{i=1}^{n} v_{t,i}\, x_{t,i} = \boldsymbol{v}_t \cdot \mathbf{x}_t$$

where $x_{t,i}$ is the prediction of E_i in trial t

Other Loss Functions – continued (1)

Theorem

For all sequences of examples

$$S = (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m) \in [0,1]^n \times [0,1]$$

the loss of the *weighted average* WA algorithm is

$$L_{\mathrm{WA}}(S) \leq \min_i \underbrace{1}_{a} L_i(S) + \underbrace{1/\eta}_{b} \ln(n)$$

with square and entropic loss.

Constant b as dependent on loss function

Loss	$b = 1/\eta$
entropic	1
square	2

Note: See [HKW98] for proof.

Other Loss Functions – continued (2)

Comments

- There exist algorithms [V90] with improved (smaller) constants for b with a more complex prediction strategy than weighted average.
- As with the weighted majority algorithm ("mistakes") for the absolute loss it is necessary that $a > 1$.
- There is an extensive theory generalizing these results to a wide range of loss functions beyond our examples of square, entropic, absolute (discrete) loss

Usefulness

- Easy to combine many pretty good experts (algorithms) so that Master is guaranteed to be almost as good as the best
- Bounds logarithmic in number of experts
- Observe updating is multiplicative

Next: Generalizing to linear combinations of experts

Perceptron and Winnow

Part II

Learning with thresholded linear combinations

A more general setting (1)

Instance	**Prediction of alg A**	**Label**	**Loss of alg A**
$\boldsymbol{x}_1$	$\hat{y}_1$	y_1	$L(y_1, \hat{y}_1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\boldsymbol{x}_t$	$\hat{y}_t$	y_t	$L(y_t, \hat{y}_t)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\boldsymbol{x}_T$	$\hat{y}_T$	y_T	$L(y_T, \hat{y}_T)$
		Total Loss	$L_A(S)$

Sequence of examples $S = (\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_T, y_T)$

Comparison class $\mathcal{U} = \{\boldsymbol{u}\}$ (AKA hypothesis space, concept class)

Relative loss

$$L_A(S) - \inf_{\{\boldsymbol{u} \in \mathcal{U}\}} Loss_{\boldsymbol{u}}(S)$$

Goal: Bound relative loss for arbitrary sequence S

A more general setting (2)

Now

- We consider the case where $\boldsymbol{u}$ is a *linear threshold* function.
- For simplicity we will focus on the case where there exists a $\boldsymbol{u}$ s.t. $Loss_{\boldsymbol{u}}(S) = 0$. This is known as *realizable* case. Compare to the previously considered halving algorithm versus weighted majority algorithm.
- We will next see how a linear threshold function may be used to represent a *disjunction* or *conjunction*.

Boolean functions

Claim

A focus of classical AI is on the representation, learning and manipulation of symbolic knowledge. Perhaps the simplest symbolic knowledge representation scheme is *boolean logic*.

- A boolean function f may be represented as a map $f : \{\texttt{true},\texttt{false}\}^n \to \{\texttt{true},\texttt{false}\}$.
- In what follows will we will always associate `true` with the number 1 however we use either -1 or 0 to represent `false` as numerically convenient.

Base boolean functions

(here $\texttt{true} = 1$ and $\texttt{false} = 0$)

1. $x_1 \wedge x_2 := x_1 x_2$ ("and")
2. $x_1 \vee x_2 := \text{sign}(x_1 + x_2)$ ("or")
3. $\bar{x_1} := 1 - x_1$ ("not")

Comparison classes of boolean functions (1)

Terminology:

1. A single (negated) variable is known as a *literal* (e.g., $x_1, x_3, x_7, \bar{x_5}$)
2. A *term* or *conjunction* of literals is an iterated "and" applied to the literals (e.g., $x_1 x_3, \bar{x_4} x_5 x_7$)
3. A *clause* or *disjunction* of literals is an iterated "or" applied to literals (e.g., $x_1 \vee x_3, \bar{x_4} \vee x_5 \vee x_7$)
4. *Monotone* disjunction or conjunction implies no negated literals

Questions : Given n variables denote the comparison class of all possible terms as $\mathcal{U}_\wedge$ and clauses as $\mathcal{U}_\vee$. What are their cardinalities i.e.? $|\mathcal{U}_\wedge|$ and $|\mathcal{U}_\vee|$? What is the cardinality of the set of all n variable boolean functions?

Comparison classes of boolean functions (2)

Recall that a linear threshold $f_{\boldsymbol{u},b} : \mathbf{R}^n \to \{-1, 1\}$ function may be written as,

$$f_{\boldsymbol{u},b}(\boldsymbol{x}) := \text{sign}(\boldsymbol{u} \cdot \boldsymbol{x} + b)\,,$$

i.e. those functions determined by a separating hyperplane. The comparison class of all linear threshold functions is

$$\mathcal{U}_{\text{lt}} := \{f_{\boldsymbol{u},b} : \boldsymbol{u} \in \mathbf{R}^n, b \in \mathbf{R}\}$$

Question: How can we use comparison class of linear-thresholded functions after the feature map to represent monotone disjunctions? and monotone conjunctions?

Feature map: We can use the feature map

$$\phi(\boldsymbol{x}) := (x_1, 1 - x_1, \ldots, x_n, 1 - x_n)$$

in order to represent *non-monotone* disjunctions and conjunctions. (Why?)

Example : Representing a monotone disjunction as a linear threshold function

variables/experts

E_1	E_2	E_3	E_4	*true* *label*	$E_1 \vee E_3$	$E_3 \vee E_4$
1	1	0	0	0	1	0
1	0	1	0	1	1	1
0	1	1	1	0	1	1
0	1	0	0	1	0	0
$x_{t,1}$	$x_{t,2}$	$x_{t,3}$	$x_{t,4}$		↑	↑
					3	2
						mistakes

$E_1 \vee E_3$ becomes $\boldsymbol{u} = (1, 0, 1, 0)$ and $b = -1/2$

$E_1 \vee E_3$ is one on $\mathbf{x}_t \in \{0, 1\}^n$ iff $\boldsymbol{u} \cdot \mathbf{x}_t \geq 1$

A suboptimal solution

Problem

Goal: An algorithm to predict as well as best k out of n literal (monotone) disjunction.

Solution

Use weighted majority where each disjunction is an expert maintain one weight per disjunction: thus $\binom{n}{k}$ weights.

$$\text{Mistakes of WM} \leq \quad 2.63\, M \;+\; 2.63\, k \ln \frac{n\, e}{k}$$

Where M is number of mistakes of best disjunction.

Note

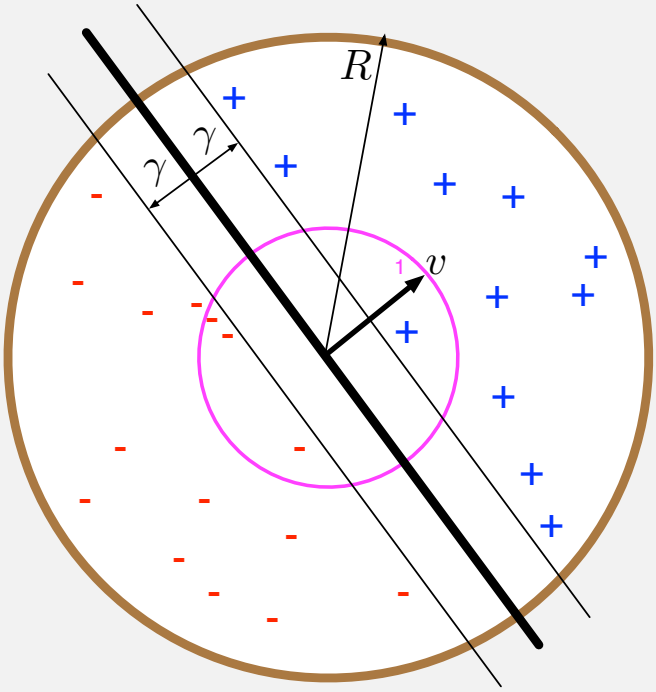
- We used the inequality $\binom{n}{k} \leq (\frac{n\, e}{k})^k$
- Time (and space) exponential in k
- **Our goal:** efficient algs: one weight per literal

Preview : Solutions!

- We will now develop two efficient algorithms for learning disjunctions, more generally linearly threshold functions
- The algorithms will be efficient with only one weight per literal maintained.
- Each update will will be $O(1)$ time per literal.
- Our solutions will be the PERCEPTRON and the WINNOW algorithm
- The WINNOW algorithm will have the advantage of a better performance guarantee at least on disjunctions
- The PERCEPTRON algorithm will have the advantage of compatibility with the "kernel trick"

The Perceptron set-up

Assumption: Data is linearly separable by some margin γ. Hence exists a hyperplane with normal vector $\boldsymbol{v}$ such that



1. $\|\boldsymbol{v}\| = 1$
2. All examples $(\boldsymbol{x}_t, y_t)$
   - $\forall y_t \;\; y_t \in \{-1, +1\}$.
   - $\forall \boldsymbol{x}_t, \;\; \|\boldsymbol{x}_t\| \leq R$.
3. $\forall (\boldsymbol{x}_t, y_t), y_t(\boldsymbol{x}_t \cdot \boldsymbol{v}) \geq \gamma$

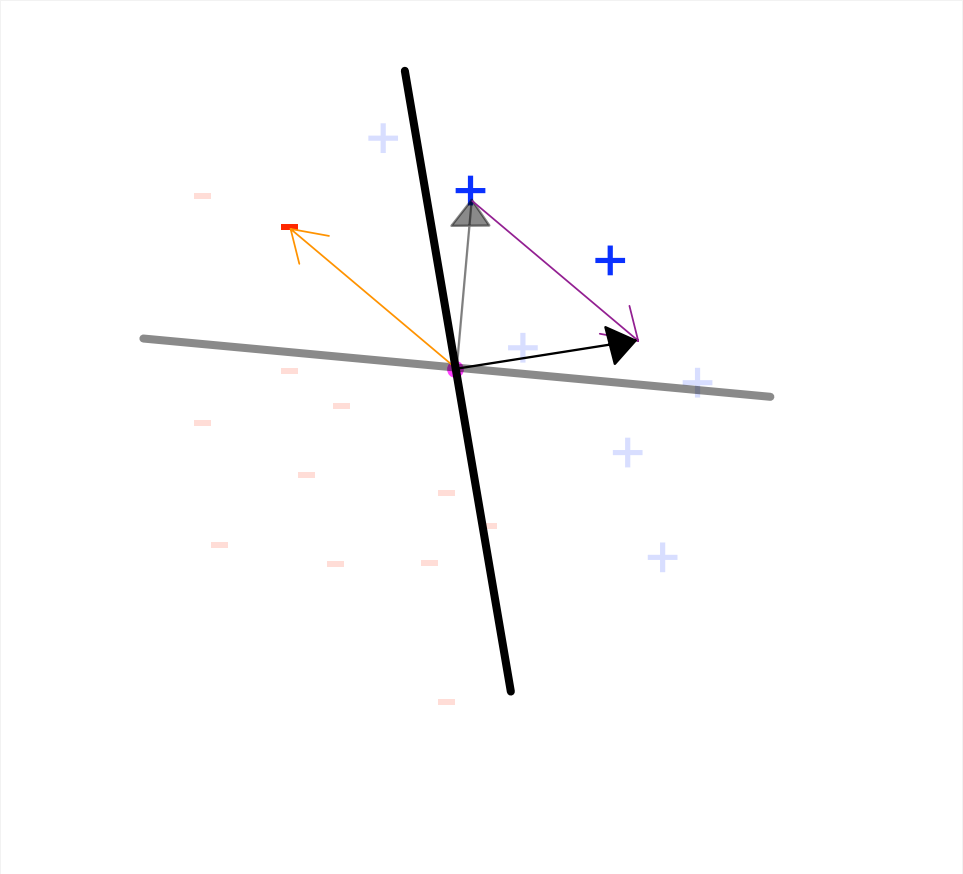
The Perceptron learning algorithm

Perceptron Algorithm

Input: $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m) \in \mathbf{R}^n \times \{-1, 1\}$

1. Initialise $\boldsymbol{w}_1 = \vec{0}$; $M_1 = 0$.
2. For $t = 1$ to m do
3. Receive pattern: $\boldsymbol{x}_t \in \mathbf{R}^n$
4. Predict: $\hat{y}_t = \text{sign}(\boldsymbol{w}_t \cdot \boldsymbol{x}_t)$
5. Receive label: y_t
6. If mistake $(\hat{y}_t y_t \leq 0)$
   - Then Update $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + y_t \boldsymbol{x}_t$; $M_{t+1} = M_t + 1$ else
7. Else $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t$; $M_{t+1} = M_t$.
8. End For

Example: trace for the PERCEPTRON algorithm



Bound on number of mistakes

- The number of mistakes that the perceptron algorithm can make is at most $\left(\frac{R}{\gamma}\right)^2$.
- Proof by combining upper and lower bounds on $\|\boldsymbol{w}\|$.

Pythagorean Lemma

On trials where "mistakes" occur we have the following inequality,

Lemma: If $(\boldsymbol{w}_t \cdot \boldsymbol{x}_t) y_t < 0$ then $\|\boldsymbol{w}_{t+1}\|^2 \leq \|\boldsymbol{w}_t\|^2 + \|\boldsymbol{x}_t\|^2$

Proof:

$$\begin{aligned} \|\boldsymbol{w}_{t+1}\|^2 &= \|\boldsymbol{w}_t + y_t \boldsymbol{x}_t\|^2 \\ &= \|\boldsymbol{w}_t\|^2 + 2(\boldsymbol{w}_t \cdot \boldsymbol{x}_t) y_t + \|\boldsymbol{x}_t\|^2 \\ &\leq \|\boldsymbol{w}_t\|^2 + \|\boldsymbol{x}_t\|^2 \end{aligned}$$

Upper bound on $\|\boldsymbol{w}_t\|$

Lemma: $\|\boldsymbol{w}_t\|^2 \leq M_t R^2$

Proof: By induction

- Claim: $\|\boldsymbol{w}_t\|^2 \leq M_t R^2$
- Base: $M_1 = 0$, $\|\boldsymbol{w}_1\|^2 = 0$
- Induction step (assume for t and prove for $t+1$) when we have a mistake on trial t:

$$\|\boldsymbol{w}_{t+1}\|^2 \leq \|\boldsymbol{w}_t\|^2 + \|\boldsymbol{x}_t\|^2 \leq \|\boldsymbol{w}_t\|^2 + R^2 \leq (M_{t+1}) R^2$$

Here we used the Pythagorean lemma. If there is no mistake, then trivially $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t$ and $M_{t+1} = M_t$.

Lower bound on $\|\boldsymbol{w}_t\|$

Lemma: $M_t\gamma \leq \|\boldsymbol{w}_t\|$

Observe: $\|\boldsymbol{w}_t\| \geq \boldsymbol{w}_t \cdot \boldsymbol{v}$ because $\|\boldsymbol{v}\| = 1$. (Cauchy-Schwarz)
We prove a lower bound on $\boldsymbol{w}_t \cdot \boldsymbol{v}$ using induction over t

- Claim: $\boldsymbol{w}_t \cdot \boldsymbol{v} \geq M_t\gamma$
- Base: $t = 1$, $\boldsymbol{w}_1 \cdot \boldsymbol{v} = 0$
- Induction step (assume for t and prove for $t+1$):
  If mistake then

$$\begin{aligned}\boldsymbol{w}_{t+1} \cdot \boldsymbol{v} &= (\boldsymbol{w}_t + \boldsymbol{x}_t y_t) \cdot \boldsymbol{v} \\ &= \boldsymbol{w}_t \cdot \boldsymbol{v} + y_t \boldsymbol{x}_t \cdot \boldsymbol{v} \\ &\geq M_t\gamma + \gamma \\ &= (M_t + 1)\gamma\end{aligned}$$

Combining the upper and lower bounds

Let $M := M_{m+1}$ denote the total number of updates ("mistakes") then

$$(M\gamma)^2 \leq \|\boldsymbol{w}_{m+1}\|^2 \leq MR^2$$

Thus simplifying we have the famous ...

Theorem (Perceptron Bound [Novikoff])

For all sequences of examples

$$S = (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m) \in \mathbf{R}^n \times \{-1, 1\}$$

the mistakes of the PERCEPTRON algorithm is bounded by

$$M \leq \left(\frac{R}{\gamma}\right)^2,$$

with $R := \max_t \|\boldsymbol{x}_t\|$. If there exists a vector $\boldsymbol{v}$ with $\|\boldsymbol{v}\| = 1$ and constant γ such that $(\boldsymbol{v} \cdot \boldsymbol{x}_t)y_t \geq \gamma$.

Comments

Comments

- It is often convenient to express the bound in the following form. Here define $\boldsymbol{u} := \frac{\boldsymbol{v}}{\gamma}$ then

$$M \leq R^2 \|\boldsymbol{u}\|^2 \quad (\forall \boldsymbol{u} : (\boldsymbol{u} \cdot \boldsymbol{x}_t) y_t \geq 1)$$

- Suppose we have *linearly separable* data set S. Questions:
  1. Observe that $\boldsymbol{w}_{m+1}$ does not necessarily linearly separate S. Why?
  2. How can we use the PERCEPTRON to find a vector $\boldsymbol{w}$ that separates S?
  3. How long will this computation take?
- There are variants on the PERCEPTRON that operate on a single example at a time that converge to the "SVM" max-margin linear separator.

Perceptron algorithm for disjunctions

We may use the previous bound to prove the following (**Exercise**):

- Perceptron bound as applied to monotone disjunction learning

$$M \leq O(kn)$$

- Here k is the number of literals out the n possible literals

Note: There exists a generic lower bound for rotation invariant algs (perceptron and svm are examples):

$$M = \Omega(n)$$

See : *The Perceptron algorithm vs. Winnow: linear vs. logarithmic mistake bounds when few input variables are relevant* (Kivinen and Warmuth, 1995).

WINNOW Algorithm for disjunctions [L88]

WINNOW algorithm

Input: $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m) \in \{0,1\}^n \times \{0,1\}$

1. Initialise $\boldsymbol{w}_1 = \vec{1}$.
2. For $t = 1$ to m do
3. Receive pattern: $\boldsymbol{x}_t \in \{0,1\}^n$
4. Predict:
$$\hat{y}_t = \begin{cases} 0 & \boldsymbol{w}_t \cdot \boldsymbol{x}_t < n \\ 1 & \boldsymbol{w}_t \cdot \boldsymbol{x}_t \geq n \end{cases}$$
5. Receive label: $y_t \in \{0,1\}$
6. If mistake $(\hat{y}_t \neq y_t)$
   - Update: $w_{t+1,i} = w_{t,i}\, 2^{(y_t - \hat{y}_t)\, x_{t,i}} \qquad (1 \leq i \leq n)$.
7. End For

Mistake bound of WINNOW

Theorem: Mistake bound of WINNOW (Littlestone)

The mistakes of WINNOW may be bounded by

$$M \leq 3k(\log n + 1) + 2$$

If there exists a *consistent* k-literal monotone disjunction.

Observation

There is an exponential improvement in bound (over the perceptron) with respect to the dimension n in the upper bound for disjunction learning. Although we will not give the bound for linear threshold learning with WINNOW in that case the bounds are incomparable (wrt PERCEPTRON) where WINNOW (like lasso) prefers *sparse* hypotheses.

Proof of WINNOW Bound

Bound on "mistakes" on positive examples. ($y_t = 1$)

1. On a mistake : at least one *relevant* weight is doubled.
2. Relevant weights never decrease.
3. Once a relevant weight $w_{t,i} \geq n$ it will no longer change

Conclusion: Mistakes on positive examples $M_p \leq k(\log n + 1)$

Bound on "mistakes" on negative examples. ($y_t = 0$)

Let $W_t = \sum_{i=1}^{n} w_{t,i}$. Denote M_f as mistakes on negatives examples.

1. $W_1 = n$
2. On a positive mistake ($y_t = 1$) $W_{t+1} \leq W_t + n$
3. On a negative mistake ($y_t = 0$) $W_{t+1} \leq W_t - \frac{n}{2}$
4. Combining $W_{m+1} \leq n + M_p n - M_f \frac{n}{2}$
5. Thus $M_f \leq 2k(\log n + 1) + 2$.

Theorem: $M \leq M_p + M_f \leq 2 + 3k(\log n + 1)$

Case study : DNF with PERCEPTRON and WINNOW

In the following we will ...

1. We note that *surprisingly* the previous mistake bounds of PERCEPTRON and WINNOW are stated in terms of the minimally consistent disjunction. However finding such disjunctions is NP-complete
2. Then we will compare the time complexity and mistake bounds for learning boolean DNF functions which are a natural and complete class of boolean functions. (Whether DNF is PAC Learnable under the uniform distribution has been an open problem for 25+ years!!)

Feature selection is hard

Observation (finding **minimally** sparse linear classifiers):

A simple instance instance of feature selection would be to find the minimal k-literal disjunction consistent with a data set. However we will see that the decision is problem is **NP-complete**.

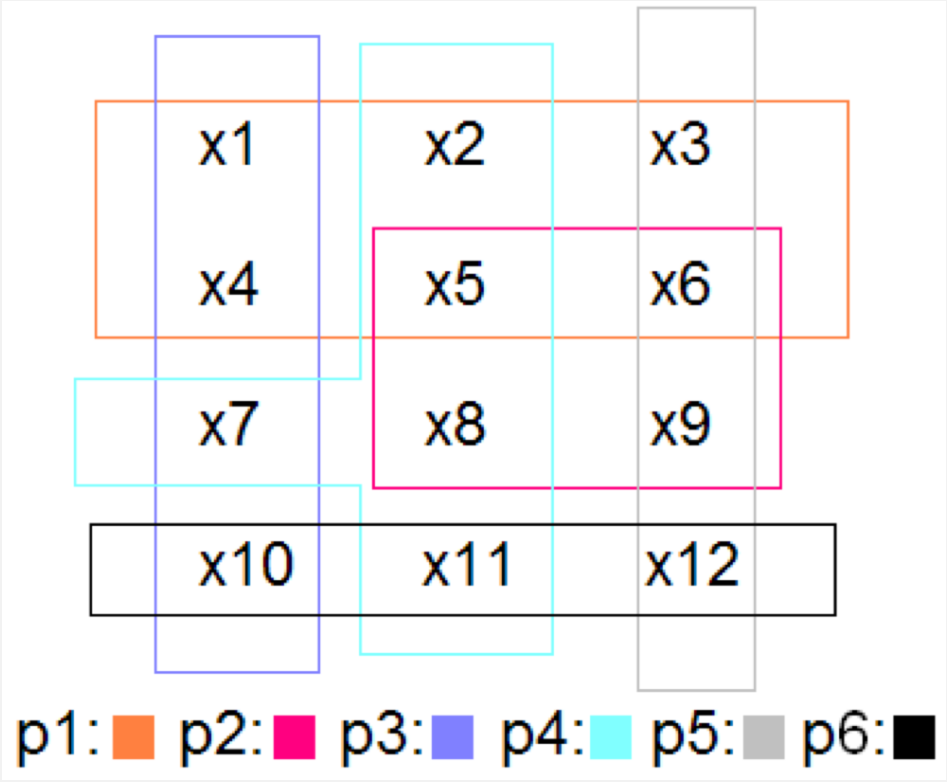
Theorem: set-cover is NP-complete

Given a set of m elements $\mathbb{N}_m := \{1, 2, \ldots, m\}$, a collection of n subsets of elements $S_1, \ldots, S_n \subseteq \mathbb{N}_m$ and a positive integer k the problem of deciding if there exist k subsets from $S_{i_1}, \ldots, S_{i_k} \in \{S_1, \ldots, S_n\}$ that cover, $\bigcup_{j=1}^{k} S_{i_j} = \mathbb{N}_m$, the m elements is NP-complete.

Theorem: consistent k-disjunction is NP-complete

Proof sketch. The problem is of finding k-set cover reduces to the problem of finding a k-literal disjunction consistent with a set of examples. *Sketching.* Let X be an $m \times n$ matrix of m, $n-$ *dimensional* "positive" examples then the n "columns" are the "sets" and a consistent disjunction is a set cover.

Illustration: Set Cover and Consistent Disjunction – 1



Set cover illustration from: ''Lecture Comp 260: Advanced Algorithms, Lenore Cowen Tufts University, Spring 2011''

Illustration: Set Cover and Consistent Disjunction – 2

As a data matrix (these are 'positive' examples thus $y_1 = \cdots y_{12} = 1$)

	P_1	P_2	P_3	P_4	P_5	P_6
x_1	1	0	1	0	0	0
x_2	1	0	0	1	0	0
x_3	1	0	0	0	1	0
x_4	1	0	1	1	0	0
x_5	1	1	0	1	0	0
x_6	1	1	0	0	1	0
x_7	0	0	1	1	0	0
x_8	0	1	0	1	0	0
x_9	0	1	0	0	1	0
x_{10}	0	0	1	0	0	1
x_{11}	0	0	0	1	0	1
x_{12}	0	0	0	0	1	1

Observe: $P_1 \vee P_2 \vee P_4 \vee P_6$ covers while $P_3 \vee P_4 \vee P_5$ is minimal.
Interestingly. The prediction bounds of winnow and the perceptron are in terms of minimum consistent disjunction but neither find/learn or a minimal consistent disjunction.

Comparison classes of boolean functions (3)

DNF (Disjunctive Normal Form): is a disjunction of terms. For example

$$x_1 x_4 x_7 \vee x_1 \bar{x_2} \vee x_2 x_5$$

- DNF is then the set of all disjunctions of terms.
- DNF is a natural form for knowledge representation for example: the concept "cat" (from Lisa Hellerstein)

$$(\text{IsHousePet} \wedge \text{Purrs} \wedge \text{HasTail}) \vee (\overline{\text{HasTail}} \wedge \text{Furry} \wedge \text{TaperedEars} \wedge \text{RoundEyes}) \vee (\text{NamedSylvester} \wedge \text{ChasesTweetyBird})$$

- Note all boolean functions may be represented as a DNF (Why?)
- Many unsolved problem in machine learning regarding DNF learning

k-term (monotone) DNF via Feature Expansion

$$\boldsymbol{x} = \begin{matrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{matrix} \quad \Rightarrow \quad \Phi(\boldsymbol{x}) = \begin{matrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \\ x_1\,x_2 \\ \vdots \\ \vdots \\ x_1\,x_2\,\ldots\,x_n \end{matrix}$$

n inputs $\qquad\qquad$ 2^n features

k-term DNF in input space is k-literal disjunction in feature space

$$\underbrace{\Phi(\boldsymbol{x})}_{2^n} \cdot \underbrace{\Phi(\boldsymbol{y})}_{2^n} = \underbrace{\prod_{i=1}^{n}(1 + x_i\,y_i)}_{O(n)\text{ time}} = K_{\text{anova}}(\underbrace{\boldsymbol{x}}_{n}, \underbrace{\boldsymbol{y}}_{n}) \text{ (Simple ANOVA kernel)}$$

Winnow versus Perceptron for k-term DNF

Perceptron: $\boldsymbol{w}_t = \sum_{q \text{ mistake}} \alpha_q \Phi(\boldsymbol{x}_i)$

Prediction:

$$\boldsymbol{w}_t \cdot \Phi(\boldsymbol{x}_t) = \left(\sum_{q \text{ mistake}} \alpha_q \Phi(\boldsymbol{x}_q)\right) \cdot \Phi(\boldsymbol{x}_t) = \sum_{q \text{ mistake}} \alpha_q \Phi(\boldsymbol{x}_q) \cdot \Phi(\boldsymbol{x}_t)$$

$$= \underbrace{\sum_{q \text{ mistake}} \alpha_q K(\boldsymbol{x}_q, \boldsymbol{x}_t)}$$

Prediction time: $O(n \cdot \#\text{ mistakes}) \leq O(nm)$

Mistake bound: $O(k\,2^n)$ (Why?)

Winnow: $w_{t,i} = \exp\left(-\eta \sum_{q \text{ mistake}} \alpha_q\, \Phi(\boldsymbol{x}_q)_i\right)$

log of weights is linear combination of past examples

Mistake bound: $O(k \ln 2^n) = O(k\,n)$ Prediction time: $\Omega(2^n \#\text{ mistakes})$

No kernel trick with purely mult. updates!, i.e., no obvious fast way to compute $\boldsymbol{w}_t \cdot \Phi(\boldsymbol{x}_t)$ for WINNOW.

Summary: PERCEPTRON: (fast!, poor bound)

WINNOW: (slow, good bound!)

Summary so far

- Learning relative to best expert and best disjunction
- Learning relative to *linear combinations* of experts
- Linear combinations may represent Boolean functions
- PERCEPTRON versus WINNOW – similar algorithms very different performance when we consider a feature space expansion as applied to DNF
- We focused on boolean function learning for PERCEPTRON and WINNOW, however, they both learn linear threshold functions directly and have strictly non-comparable bounds. WINNOW like LASSO is better for "sparse" linear threshold functions

Share Algorithm

Part III

Learning with sequences of experts

Overview

- An online learning algorithm
- Tracks concepts which change over time
- Designed to combine other algorithms
- Online performance guarantees are given
  - Nonasymptotic
  - No statistical assumptions
  - Tight lower bounds
- Time complexity: "linear"
- Practical Applications

On-line Learning (Review)

time t	1	2	3	4	$\cdots$	t
expert 1	.5	3	2	1	$\cdots$	$x_{t,1}$
expert 2	.75	-1.5	-1	-2	$\cdots$	$x_{t,2}$
expert 3	-1.5	3	2	-4	$\cdots$	$x_{t,3}$
expert 4	.75	-1.3	1.5	1.5	$\cdots$	$x_{t,4}$
alg. preds	0	1.5	1.5	.75	$\cdots$	$\hat{y}_t$
label	.75	-1.5	2	1	$\cdots$	y_t
alg. loss	0.56	9	.25	.06	$\cdots$	$(\hat{y}_t - y_t)^2$

For a sequence of examples $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_\ell, y_\ell)\}$

$$\text{Loss}_A(S) = \sum_{t=1}^{\ell} (\hat{y}_t - y_t)^2$$

Aim: to bound $\text{Loss}_A(S)$ in terms of the loss of the best expert.

$$\text{Loss}_i(S) = \sum_{t=1}^{\ell} (x_{t,i} - y_t)^2$$

Formal Model

In each trial $t = 1, \ldots, \ell$ the algorithm receives an a vector of predictions

$$\boldsymbol{x}_t = (x_{t,1}, \ldots, x_{t,n}) \in [0,1]^n$$

and predicts

$$\hat{y}_t \in [0,1]$$

then it observes the *outcome*

$$y_t \in [0,1]$$

The performance of the algorithm A is measured by its loss

$$\text{Loss}_A = \sum_{t=1}^{\ell} L(\hat{y}_t, y_t)$$

where L is a loss function such as the square or the Hellenger loss.

The Experts Algorithm (Review)

The algorithms maintain one weight $w_{t,i}$ per expert/component. The weight vector $\boldsymbol{w}_t$ is used for forming the prediction. For example

$$\hat{y}_t = \boldsymbol{w}_t \cdot \mathbf{x}_t$$

The weight vector $\boldsymbol{w}_t$ is updated after the label y_t is received. For example

$$\text{Expert update: } w_{t+1,i} = \frac{w_{t,i} e^{-\eta(x_{t,i}-y_t)^2}}{\sum_{j=1}^{n} w_{t,j} e^{-\eta(x_{t,j}-y_t)^2}} \quad i = 1, \ldots, n$$

Static Relative Loss Bounds

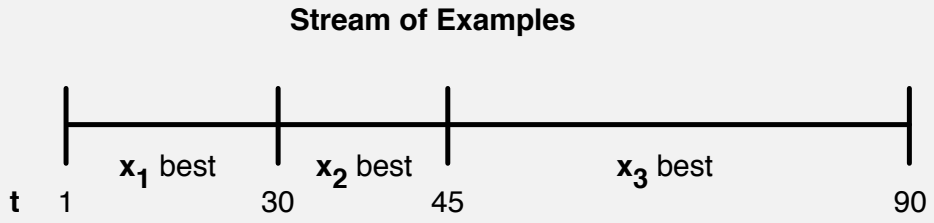
For all sequences of examples S

$$\text{Loss}_A(S) \leq \text{Loss}_i(S) + c \ln n \quad [V90, LW94, HKW98]$$

- The loss of the algorithm is bounded in terms of the best single expert.

So far the comparator is "static"

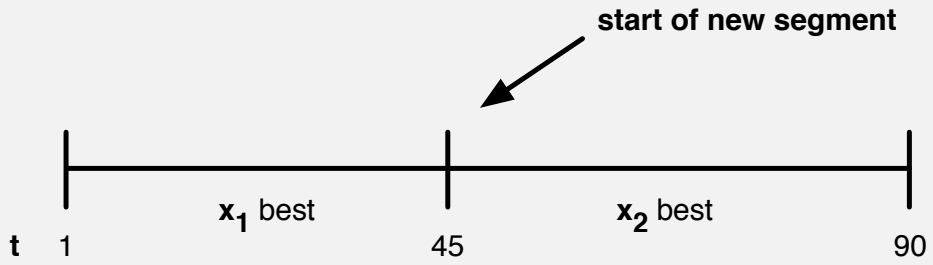
Shifting Expert



- The loss of the algorithm is compared against the loss of the best sequence of experts.
- The loss of a sequence of experts is the sum of the loss of the expert for each segment.
- The algorithms do not know when a new segment begins and which expert are best in each segment.

Problem for Algorithms

The crucial weights may be too small or too large. It might take the algorithm too long to recover.



- Lower bounds on weights [LW,AW,BB]
- Weights as "probabilities" [HWa,V]
- Keep the weights in a bounded region defined in terms of a "divergence function": [HWb]

Shifting Experts – Details

- In the *static* case the loss bound is given relative to a single *expert*.
- In the *shifting* case the loss bound is given relative to a *sequence* of experts.

The loss defined relative to a sequence of experts

$$\text{Loss}_{\{i_1,i_2,\ldots,i_\ell\}}(S) = \sum_{t=1}^{\ell}(x_{t,i_t} - y_t)^2$$

Algorithm Design

Given the number of distinct sequences of experts is

$$O([n\frac{\ell}{k}]^k)$$

Why?

Using the "experts" algorithm we can design an algorithm with a loss bound of

$$L_A(S) \leq L^* + ck(\ln n + \ln \frac{\ell}{k} + \text{“}const\text{”})$$

How?

Caveats:

- Algorithm is exponential in k
- Loss bound dependent on ℓ

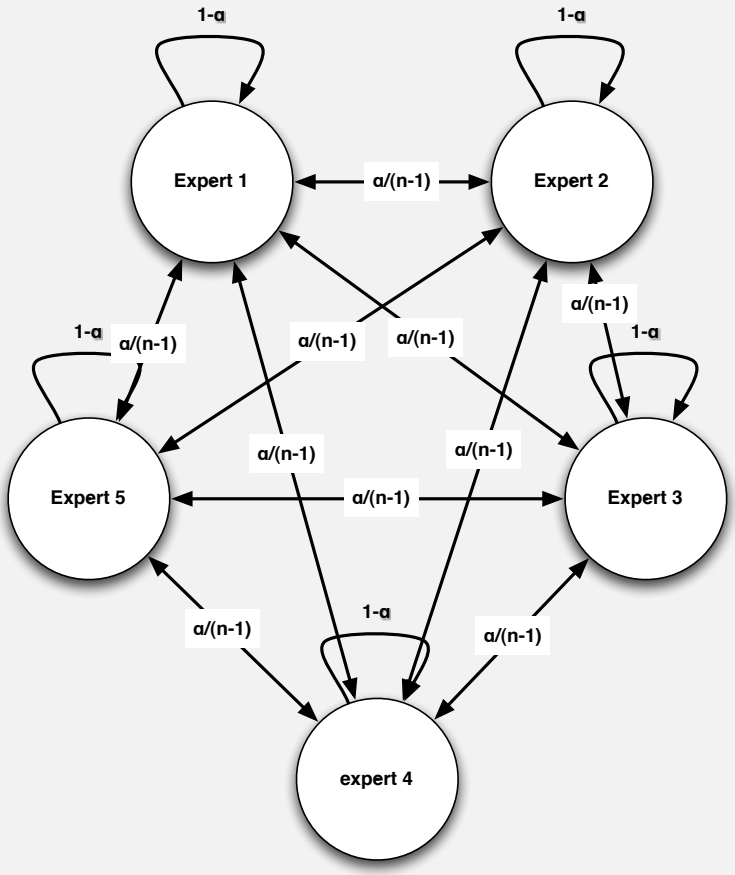
Fixed and Variable Share Algorithms

- **Parameters:** $0 \leq \eta$ and $0 \leq \alpha \leq 1$.
- **Initialization:** Set the weights to $w^s_{1,1} = \ldots = w^s_{1,n} = \frac{1}{n}$.
- **Prediction:** Let $v_{t,i} = \frac{w^s_{t,i}}{W_t}$, where $W_t = \sum_{i=1}^n w^s_{t,i}$. Predict with $\hat{y}_t = \boldsymbol{v}_t \cdot \boldsymbol{x}_t$
- **Loss Update:** After receiving the tth outcome y_t,

$$\forall i : 1, \ldots, n : w^m_{t,i} = w^s_{t,i} e^{-\eta L(y_t, x_{t,i})}$$

- **Fixed Share Update:**
  - $\text{pool} = \sum_{i=1}^n \alpha w^m_{t,i}$
  - $\forall i : 1, \ldots, n : w^s_{t+1,i} = (1-\alpha) w^m_{t,i} + \frac{1}{n-1} w^s_{t,i}(\text{pool} - \alpha w^m_{t,i})$
- **Variable Share Update:**
  - $\text{pool} = \sum_{i=1}^n (1 - (1-\alpha)^{L(y_t, x_{t,i})}) w^m_{t,i}$
  - $\forall i : 1, \ldots, n : w^s_{t+1,i} =$
    $(1-\alpha)^{L(y_t, x_{t,i})} w^m_{t,i} + \frac{1}{n-1} w^s_{t,i}(\text{pool} - (1 - (1-\alpha)^{L(y_t, x_{t,i})}))$

Fixed Share – Visualization



Shifting Loss Bounds

Shifting Experts:

- Let $k := \text{size}(\{i_1, i_2, \ldots, i_\ell\})$
- $L^* := \text{Loss}_{\{i_1, i_2, \ldots, i_\ell\}}(S)$
- Choose $\alpha \in [0, 1]$,

We then have the bound:

$$L_A(S) \leq L^* + c[(\ell - 1)(H(\alpha^*) + D(\alpha^* \| \alpha)) + k \ln(n-1) + \ln n],$$

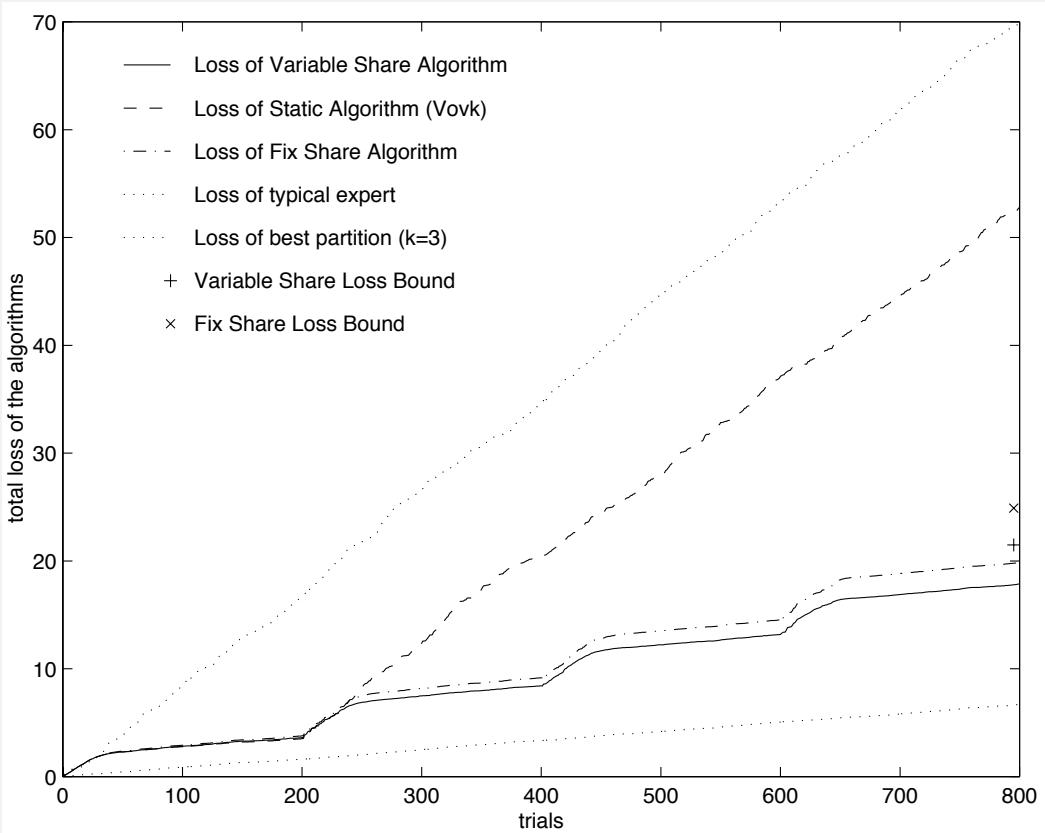
where $\alpha^* = \frac{k}{\ell - 1}$, $H(p) = \frac{1}{p} \ln \frac{1}{p} + \frac{1}{1-p} \ln \frac{1}{1-p}$ and
$D(p^* \| p) = p^* \ln \frac{p^*}{p} + (1 - p^*) \ln \frac{1-p^*}{1-p}$
When $\alpha = \frac{k}{\ell - 1}$, then the above is upper bounded by

$$L_A(S) \leq L^* + c[k \ln \frac{\ell - 1}{k} + k \ln(n-1) + \ln n + k],$$

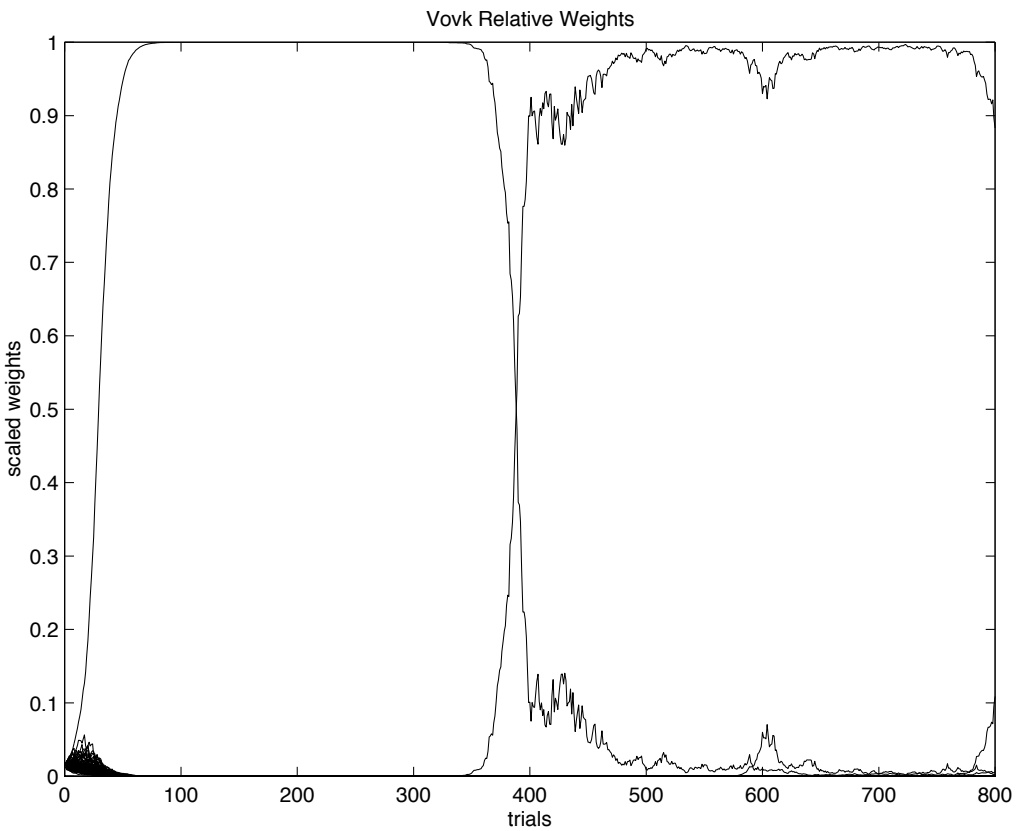
Experiment (simulation)

1. Trials (ℓ) : 800
2. Shifts(k): 3
3. Number of Experts (n) : 64
4. Loss function (L): $L(y, \hat{y}) = (y - \hat{y})^2$ ($c = 1/2, \eta = 2$)
5. Share Parameter(α) : 0.024
6. Typical expert expected Loss/Trial : $1/12$
7. "Best" expert expected Loss/Trial : $1/120$

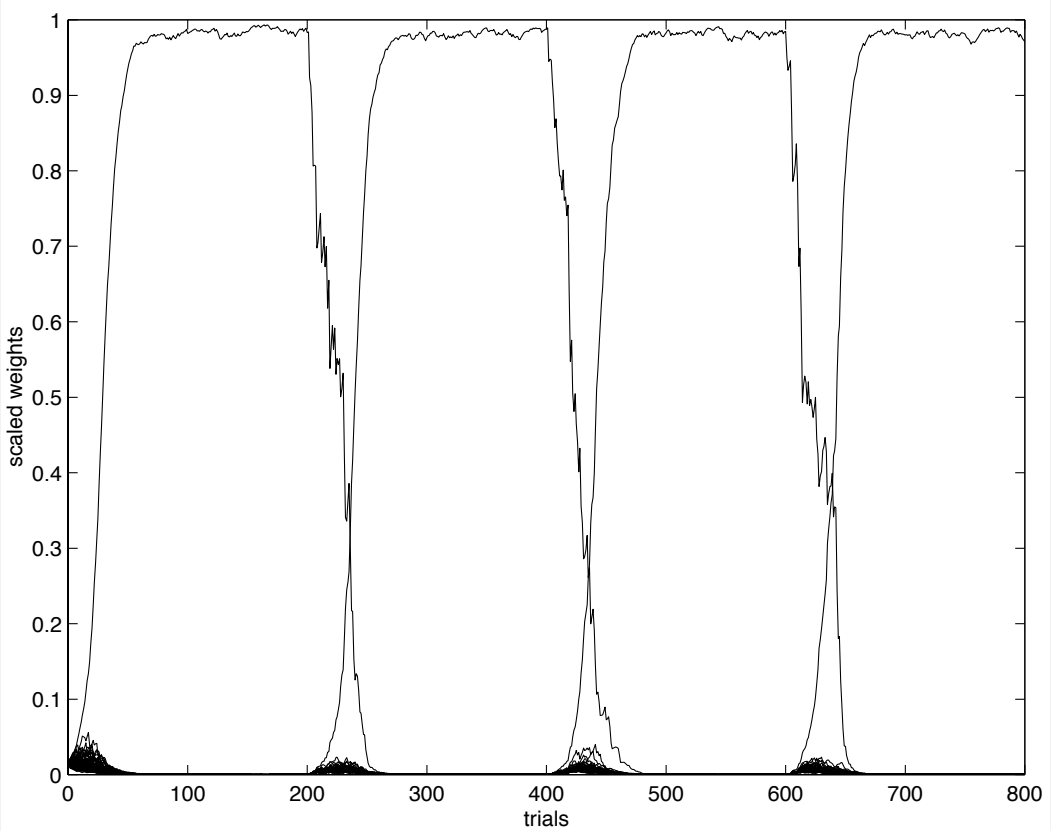
Share Algorithms – Performance



Expert Algorithm's Weights



Variable Share Weights



Some Applications

1. Predicting disk idle times
2. Online load balancing (process migration determination)
3. Predicting TCP packet inter-arrival times
4. Financial prediction by combining portfolios
5. Combining language for domain and topic adaptation

Useful references

1. Nicolò Cesa-Bianchi and Gábor Lugosi, *Prediction, learning, and games.*, (2006), Note this is a book.
2. N. Littlestone, *Learning quickly when irrelevant attributes abound: a new linear threshold algorithm*, (1988).
3. N. Littlestone and M. K. Warmuth. *The weighted majority algorithm*, (1994)
4. V. Vovk, *Aggregating strategies*, (1990).
5. Haussler, D., Kivinen, J. and Warmuth, M.K. *Sequential Prediction of Individual Sequences Under General Loss Functions,* (1998)
6. M. Herbster and M. Warmuth, *Tracking the Best Expert*, (1998)
7. S. Shalev-Schwartz, *Online Learning and Online Convex Optimization*, (2011)