5. Online learning I

COMP0078: Supervised Learning

Mark Herbster

28 October 2019

University College London
Department of Computer Science

`SL-onlinelearning19v5`

Acknowledgments and References

Thanks

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

A general reference for online learning

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

A more technical recent reference

- Shai Shalev-Shwartz, *Online Learning and Online Convex Optimization*

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. Refining and generalising the experts model
- Learning linear classifiers
  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

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

Interpretation: The vector $\mathbf{x}_t$ 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 $\mathbf{x}_t$ 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

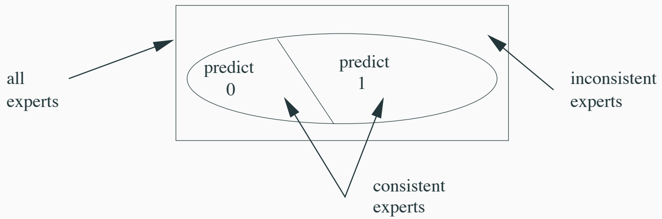
```
For $t = 1$ To $m$ Do
    Get instance             $\mathbf{x}_t \in \{0,1\}^n$
    Predict                  $\hat{y}_t \in \{0,1\}$
    Get label                $y_t \in \{0,1\}$
    Incur loss (mistakes)    $|y_t - \hat{y}_t|$
```

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

								majo	*true*	
E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	*rity*	*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	$\uparrow$	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:

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

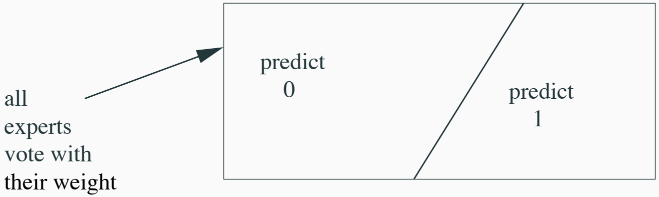
where a, b are "small" constants

Comment: These are known as "Regret", "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

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

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:

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

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

Number of mistakes of the WM algorithm – Continued-1

Hence

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

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

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.

Refining and generalising the experts model – 1

More generally we would like to obtain *regret* bounds for arbitrary loss functions $L : \mathcal{Y} \times \hat{\mathcal{Y}} \to [0, +\infty]$. Making our notion of regret more precise we would like guarantees of the form,

$$L_A(S) - \min_{i \in [n]} L_i(S) \leq o(m)\,,$$

where the right-hand side is termed regret since it is how much we "regret" predicting with the algorithm as opposed to the optimal predictor on the data sequence. Here $o(m)$ denotes some function sublinear in m (the number of examples in S) and possibly dependent on other parameters.

Therefore since

$$\frac{L_A(S) - \min_{i \in [n]} L_i(S)}{m} \leq \frac{o(m)}{m}\,,$$

and thus as $m \to \infty$ we have

$$\frac{L_A(S)}{m} \leq \frac{\min_{i \in [n]} L_i(S)}{m}\,,$$

thus in the limit the mean asymptotic loss of our algorithm is bounded by the mean asymptotic loss of the best expert.

Refining and generalising the experts model – 2

In the following we will show two example of regret bounds generalising the analysis of the weighted majority algorithm.

1. For a loss function $L : \{0, 1\} \times [0, 1] \to [0, +\infty]$ the *entropic loss* $L(y, \hat{y}) := y \ln \frac{y}{\hat{y}} + (1 - y) \ln \frac{1-y}{1-\hat{y}}$
2. For an arbitrary bounded loss function $L : \mathcal{Y} \times \hat{\mathcal{Y}} \to [0, B]$.

For the first the regret will be the small constant $\log(n)$ for the second the regret will be $O(\sqrt{m \log n})$.

A regret bound for the entropic (log) loss

Unlike in the case of the weighted majority we will now allow predictions in $[0,1]$ rather than just $\{0,1\}$ and rather prediction with the "majority" we will predict with the *weighed average*.

One weight per expert:

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

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

Master predicts with the weighted average (dot product)

$$\begin{aligned} v_{t,i} &= \frac{w_{t,i}}{\sum_{i=1}^{n} w_{t,i}} \quad \text{normalized weights} \\ \hat{y}_t &= \sum_{i=1}^{n} v_{t,i}\, x_{t,i} = \mathbf{v}_t \cdot \mathbf{x}_t \end{aligned}$$

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

Set the initial weights $\mathbf{v}_1 := \mathbf{w}_1 := (\frac{1}{n}, \ldots, \frac{1}{n})$.

The WA Algorithm – Summary

WA Algorithm

```
Initialise : $\mathbf{v}_1 := \mathbf{w}_1 := (\frac{1}{n}, \ldots, \frac{1}{n})$, $L_{\text{WA}} := 0$, $\mathbf{L} := \mathbf{0}$,
Select: $\eta \in (0, \infty)$, Loss function $L : (\cdot, \cdot)$.

For $t = 1$ To $m$ Do
    Receive instance  $\mathbf{x}_t \in [0,1]^n$
    Predict           $\hat{y}_t := \mathbf{v}_t \cdot \mathbf{x}_t$
    Receive label     $y_t \in [0,1]$
    Incur loss        $L_{\text{WA}} := L_{\text{WA}} + L(y_t, \hat{y}_t)$, $L_i := L_i + L(y_t, x_{t,i})$ $(i \in [n])$
    Update Weights    $v_{t,i+1} := \frac{v_{t,i} e^{-\eta L(y_t, x_{t,i})}}{\sum_{j=1}^{n} v_{t,j} e^{-\eta L(y_t, x_{t,j})}}$ for $i \in [n]$.
```

Weighted Average Algorithm - Theorem

Theorem

For all sequences of examples

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

the regret of the *weighted average* WA algorithm is

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

with square and entropic loss for $\eta = 1/2$ and $\eta = 1$ respectively.

Constant b as dependent on loss function

Loss		$b = 1/\eta$
entropic	$L_{\mathrm{en}}(y, \hat{y}) = y \ln \frac{y}{\hat{y}} + (1-y) \ln \frac{1-y}{1-\hat{y}}$	1
square	$L_{\mathrm{sq}}(y, \hat{y}) = (y - \hat{y})^2$	2

- For simplicity, we will prove only for entropic loss when $\mathcal{Y} := \{0,1\}$ and $\hat{\mathcal{Y}} := [0,1]$. The result holds for many loss function (sufficient smoothness and convexity with different η and b). See [KV99] for proof.

Weighted Average Algorithm - Proof [Not Examined 19/20]

Notation: $\Delta_n := \{\mathbf{x} \in [0,1]^n : \sum_{i=1}^n x_i = 1\}$. Let $d : \Delta_n \times \Delta_n \to [0,\infty]$ be the rel. entropy $d(\mathbf{u},\mathbf{v}) := \sum_{i=1}^n u_i \ln \frac{u_i}{v_i}$. Note: $L_{\text{en}}(y,\hat{y}) = d((y,1-y),(\hat{y},1-\hat{y}))$.

Proof – 1

We first prove the following "progress versus regret" equality.

$$L_{\text{en}}(y_t,\hat{y}_t) - \sum_{i=1}^n u_i L_{\text{en}}(y_t,x_{t,i}) = d(\mathbf{u},\mathbf{v}_t) - d(\mathbf{u},\mathbf{v}_{t+1}) \text{ for all } \mathbf{u} \in \Delta_n\,. \qquad (1)$$

Observe that

$$d(\mathbf{u},\mathbf{v}_t) - d(\mathbf{u},\mathbf{v}_{t+1}) = \sum_{i=1}^n u_i \ln \frac{v_{t+1,i}}{v_{t,i}}$$

If $y_t = 1$ then (using $L_{\text{en}}(1,x) = -\ln x$)

$$\sum_{i=1}^n u_i \ln \frac{v_{t+1,i}}{v_{t,i}} = \sum_{i=1}^n u_i \ln \frac{\frac{v_{t,i}e^{-L_{\text{en}}(1,x_{t,i})}}{\sum_{i=1}^n v_{t,i}e^{-L_{\text{en}}(1,x_{t,i})}}}{v_{t,i}} = \sum_{i=1}^n u_i \ln \frac{\frac{v_{t,i}x_{t,i}}{\sum_{i=1}^n v_{t,i}x_{t,i}}}{v_{t,i}} = \sum_{i=1}^n u_i \ln \frac{x_{t,i}}{\hat{y}_t}$$

$$= \left(\sum_{i=1}^n u_i \ln x_{t,i}\right) - \ln(\hat{y}_t) = L_{\text{en}}(y_t,\hat{y}_t) - \sum_{i=1}^n u_i L_{\text{en}}(y_t,x_{t,i})$$

by symmetry we also have the case $y = 0$ demonstrating (1).

Weighted Average Algorithm - Proof continued

Proof – 2

Now observe that (1) is a telescoping equality and we have

$$\sum_{t=1}^{m} L_{\text{en}}(y_t, \hat{y}_t) - \sum_{t=1}^{m}\sum_{i=1}^{n} u_i L_{\text{en}}(x_{t,i}, \hat{y}_t) = d(\mathbf{u}, \mathbf{v}_1) - d(\mathbf{u}, \mathbf{v}_{m+1})$$

Now since since the above holds for any $\mathbf{u} \in \Delta_n$ in particular the unit vectors $(1, 0, \dots, 0), (0, 1, \dots, 0), \dots, (0, 0, \dots, 1)$ and then if we upper bound by noting that $d(\mathbf{u}, \mathbf{v}_1) \leq \ln n$ and $-d(\mathbf{u}, \mathbf{v}_{m+1}) \leq 0$ we have proved theorem. $\square$

Weighted Average Algorithm - Hedge

Hedge was introduced in [FS97], generalising the weighted majority analysis to the *allocation* setting.

Allocation setting

On each trial the learner plays an allocation $\mathbf{v}_t \in \Delta_n$, then nature returns a loss vector $\boldsymbol{\ell}_t$. I.e., the loss of expert i is $\ell_{t,i}$.

Two models for the learner's play:

1. We simply incur loss so that $L_A(t) := \mathbf{v}_t \cdot \boldsymbol{\ell}_t$.
2. Alternately learner randomly selects an action $\hat{y} \in [n]$ according to the discrete distribution over $[n]$ so that $\mathrm{Prob}(j) := v_{t,j}$. Thus

$$E[L_A(t)] = E[L_{t,\hat{y}}] = \mathbf{v}_t \cdot \boldsymbol{\ell}_t \,.$$

- Observe that this setting can *simulate* the setting where we receive side-information $\mathbf{x}_t$ and have a fixed loss function.
- For the randomised setting the "mechanism" generating the loss vectors $\boldsymbol{\ell}_t$ must be oblivious to the learner's selection $\hat{y}$ until trial $t + 1$.

Hedge - Theorem

Theorem Hedge (Bound) [LW94,FS97]

For all sequence of loss vectors

$$S = \boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_m \in [0,1]^n$$

the regret of the *weighted average* WA algorithm with $\eta = \sqrt{2 \ln n / m}$ is

$$E[L_{\mathrm{WA}}(S)] - \min_i L_i(S) \leq \sqrt{2m \ln n}\,.$$

HEDGE Theorem - Proof (1) [Not Examined 19/20]

Proof – 1

We first prove the following "progress versus regret" inequality.

$$\mathbf{v}_t \cdot \boldsymbol{\ell}_t - \mathbf{u}_t \cdot \boldsymbol{\ell}_t \leq \frac{1}{\eta}\left(d(\mathbf{u}, \mathbf{v}_t) - d(\mathbf{u}, \mathbf{v}_{t+1})\right) + \frac{\eta}{2}\sum_{i=1}^{n} v_{t,i}\ell_{t,i}^2 \text{ for all } \mathbf{u} \in \Delta_n\,. \tag{2}$$

Let $Z_t := \sum_{i=1}^{n} v_{t,i} \exp(-\eta\ell_{t,i})$. Observe that

$$\begin{aligned}
d(\mathbf{u}, \mathbf{v}_t) - d(\mathbf{u}, \mathbf{v}_{t+1}) &= \sum_{i=1}^{n} u_i \ln \frac{v_{t+1,i}}{v_{t,i}} \\
&= -\eta \sum_{i=1}^{n} u_i \ell_{t,i} - \sum_{i=1}^{n} u_{t,i} \ln Z_t \\
&= -\eta \mathbf{u} \cdot \boldsymbol{\ell}_t - \ln \sum_{i=1}^{n} v_{t,i} \exp(-\eta\ell_{t,i}) \\
&\geq -\eta \mathbf{u} \cdot \boldsymbol{\ell}_t - \ln \sum_{i=1}^{n} v_{t,i}(1 - \eta\ell_{t,i} + \frac{1}{2}\eta^2\ell_{t,i}^2) & (3) \\
&= -\eta \mathbf{u} \cdot \boldsymbol{\ell}_t - \ln(1 - \eta\mathbf{v}_t \cdot \boldsymbol{\ell}_t + \frac{1}{2}\eta^2 \sum_{i=1}^{n} v_{t,i}\ell_{t,i}^2) \\
&\geq \eta(\mathbf{v}_t \cdot \boldsymbol{\ell}_t - \mathbf{u} \cdot \boldsymbol{\ell}_t) - \frac{1}{2}\eta^2 \sum_{i=1}^{n} v_{t,i}\ell_{t,i}^2 & (4)
\end{aligned}$$

Using inequalities $e^{-x} \leq 1 - x + \frac{x^2}{2}$ for $x \geq 0$ and $\ln(1+x) \leq x$ for (3) and (4) demonstrating (2).

HEDGE Theorem - Proof (2) [Not Examined 19/20]

Proof – 2

Summing over t and rearranging we have

$$\sum_{t=1}^{m} (\mathbf{v}_t \cdot \boldsymbol{\ell}_t - \mathbf{u}_t \cdot \boldsymbol{\ell}_t) \leq \frac{1}{\eta} (d(\mathbf{u}, \mathbf{v}_1) - d(\mathbf{u}, \mathbf{v}_{m+1})) + \frac{\eta}{2} \sum_{t=1}^{m} \sum_{i=1}^{n} v_{t,i} \ell_{t,i}^2$$
$$\leq \frac{\ln n}{\eta} + \frac{\eta}{2} \sum_{t=1}^{m} \sum_{i=1}^{n} v_{t,i} \ell_{t,i}^2 \qquad (5)$$

Now since since the above holds for any $\mathbf{u} \in \Delta_n$ it then holds in particular for the unit vectors $(1, 0, \ldots, 0), (0, 1, \ldots, 0), \ldots, (0, 0, \ldots, 1)$ and then we upper bound by noting that $d(\mathbf{u}, \mathbf{v}_1) \leq \ln n$, $-d(\mathbf{u}, \mathbf{v}_{m+1}) \leq 0$, and $\sum_{t=1}^{m} \sum_{i=1}^{n} v_{t,i} \ell_{t,i}^2 \leq m$. Finally we "tune" by choosing $\eta = \sqrt{2 \ln n / m}$ and we have proved theorem. $\square$

Question: how can we the above to prove a theorem if the loss is now in the range $[0, B]$?

Comments

- 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. Use many experts! Limits only in computational resources.
- Observe updating is multiplicative
- There exist algorithms [V90] with improved (smaller) constants for b with a more complex prediction strategies than weighted average.

Next: So far we have given bounds which grow slowly in the number of experts. The only significant drawback is potentially computational if we wish to work with large classes of experts. With this is mind we may wish to work with *structured* sets of experts for either a computational advantages or advantages in bound. We consider now linear combinations of experts (linear classifiers).

Perceptron and Winnow

Part II

Online learning of linear classifiers

A more general setting (1)

Instance	Prediction of alg A	Label	Loss of alg A	
$\mathbf{x}_1$	$\hat{y}_1$	y_1	$L(y_1, \hat{y}_1)$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$\mathbf{x}_t$	$\hat{y}_t$	y_t	$L(y_t, \hat{y}_t)$	Sequence of examples
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$\mathbf{x}_m$	$\hat{y}_m$	y_m	$L(y_m, \hat{y}_m)$	
		Total Loss	$L_A(S)$	

$S = (\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)$

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

Relative loss (Regret)

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

Goal: Bound relative loss for arbitrary sequence S

A more general setting (2)

Now

- We consider the case where $\mathcal{U}$ is a set of *linear threshold* function.
- For simplicity we will focus on the case where there exists a $\mathbf{u} \in \mathcal{U}$ 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 $f_{\mathbf{u},b} : \mathbf{R}^n \to \{-1, 1\}$ function may be written as,

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

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

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

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

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

in order to represent *non-monotone* disjunctions and conjunctions. (How does this work?)

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
$x_{t,1}$	$x_{t,2}$	$x_{t,3}$	$x_{t,4}$		$\uparrow$	$\uparrow$
					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.

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

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

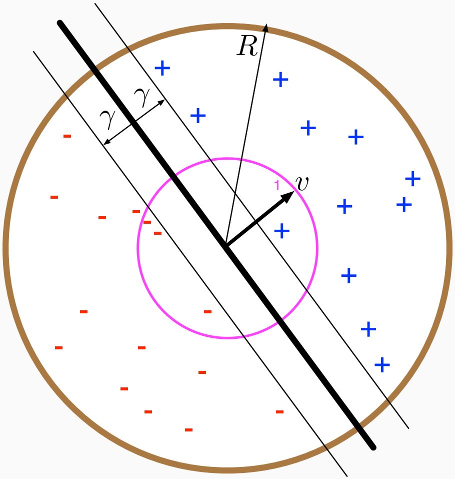
Preview : Solutions!

- 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"

Perceptron

The `Perceptron` set-up

Assumption: Data is linearly separable by some margin γ. Hence exists a hyperplane with normal vector **v** such that



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

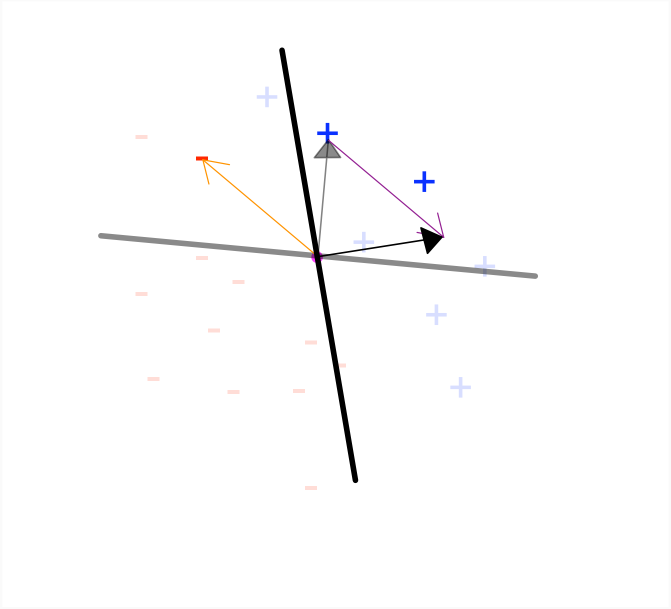
The PERCEPTRON learning algorithm

PERCEPTRON ALGORITHM

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

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

Pythagorean Lemma

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

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

Proof:

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

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

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

Proof: By induction

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

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

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

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

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

Observe: $\|\mathbf{w}_t\| \geq \mathbf{w}_t \cdot \mathbf{v}$ because $\|\mathbf{v}\| = 1$. (Cauchy-Schwarz)

We prove a lower bound on $\mathbf{w}_t \cdot \mathbf{v}$ using induction over t

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

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

Combining the upper and lower bounds

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

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

Thus simplifying we have the famous ...

Theorem (Perceptron Bound [Novikoff])

For all sequences of examples

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

the mistakes of the PERCEPTRON algorithm is bounded by

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

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

Comments

Comments

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

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

- Suppose we have *linearly separable* data set S. Questions:
  1. Observe that $\mathbf{w}_{m+1}$ does not necessarily linearly separate S. Why?
  2. How can we use the PERCEPTRON to find a vector $\mathbf{w}$ 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 Bound in Noisy case [NOT EXAMINED 19/20]

Define hinge loss $h(y, \hat{y}) := \max(0, -y\hat{y})$.

Theorem ([G03])

For all sequences of examples

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

the mistakes of the Perceptron algorithm is bounded by

$$\sum_{t=1}^{m} [y_t \neq \text{sign}(\mathbf{w}_t \cdot \mathbf{x}_t)] \leq \sum_{t=1}^{m} h(y_t, \mathbf{u} \cdot \mathbf{x}_t) - \|\mathbf{u}\| R \sqrt{\sum_{t=1}^{m} h(y_t, \mathbf{u} \cdot \mathbf{x}_t) + \|\mathbf{u}\|^2 R^2}$$

with $R := \max_t \|\mathbf{x}_t\|$, for all $\mathbf{u} \in \mathbf{R}^n$.

Perceptron algorithm for disjunctions – 1

Corollary

With the feature map $\phi(\mathbf{x}) := (\mathbf{x}, 1)$ we may use the perceptron to learn monotone disjunctions so that

$$M \leq (4k+1)(n+1)$$

Where 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).

Perceptron algorithm for disjunctions – 2

Proof

We will use the following form of the perceptron bound,

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

Assume $\mathbf{x} \in \{0,1\}^n$ then with the feature map $\phi(\mathbf{x}) := (\mathbf{x}, 1)$. We claim the following $\mathbf{u}^* \in \mathbf{R}^{n+1}$ separates with a margin of 1,

$$u_i^* := \begin{cases} 2 & i \text{ is a literal} \\ 0 & i \text{ is not a literal} \\ -1 & i \text{ is the bias weight} \end{cases} .$$

I.e.

1. $\mathbf{u}^* \cdot \phi(\mathbf{x}) \geq 1$ (positive examples, $y_t = 1$)
2. $\mathbf{u}^* \cdot \phi(\mathbf{x}) = -1$ (negative examples, $y_t = -1$)

Note that since $\mathbf{x} \in \{0,1\}^n$ then $\|\phi(\mathbf{x})\|^2 \leq (n+1)$ and $\|\mathbf{u}^*\|^2 = 4k + 1$. Thus $M \leq (4k+1)(n+1)$. $\square$

WINNOW Algorithm for disjunctions [L88]

WINNOW algorithm

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

```
1. Initialise $\mathbf{w}_1 = \vec{1}$.
2. For $t = 1$ to $m$ do
3. Receive pattern: $\mathbf{x}_t \in \{0,1\}^n$
4. Predict:
       $$\hat{y}_t = \begin{cases} 0 & \mathbf{w}_t \cdot \mathbf{x}_t < n \\ 1 & \mathbf{w}_t \cdot \mathbf{x}_t \geq n \end{cases}$$
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 negative 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: $M \leq M_p + M_f \leq 2 + 3k(\log n + 1)$

Online bounds give Batch Bounds

- We can use online bounds to give bounds in a batch setting.
- Suppose the data is drawn from a distribution P and a mistake bound for algorithm $\mathcal{A}$ for any such set is B.

Theorem

Given m, then let t be drawn uniformly at random from $\{1, \ldots, m\}$. Let $\mathcal{S}$ be consist of t examples sampled iid from P, let $(\mathbf{x}', y')$ be an additional example sampled from P then

$$\mathrm{Prob}(\mathcal{A}_{\mathcal{S}}(\mathbf{x}') \neq y') \leq \frac{B}{m}$$

with respect to the draw of t, $\mathcal{S}$, and $(\mathbf{x}', y')$.

Proof

There are no more than B "trials" with mistakes therefore since t is drawn uniformly from $\{1, \ldots, m\}$ there is no more than a B/m probability of hitting a trial with a mistake.

Note: More sophisticated online to batch conversions exist with stronger properties.

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!!)

Finding maximally sparse classifiers is hard (1)

Sparsity is important concept in machine learning. Here the idea is to find a classifier which depends on the minimal number of features. Such a classifier has the two-fold advantage.

1. In general it will generalize well to new data
2. It may be useful to discover which features are particularly predictive

Finding maximally sparse classifiers is hard (2)

A common method to encourage sparsity (L1 Objective) is to minimise

$$\arg\min_{\mathbf{w}} \sum_{t=1}^{m} [1 - y_t(\mathbf{x}_t \cdot \mathbf{w})]_+ + \lambda \|\mathbf{w}\|_1$$

where this is a proxy for the nonconvex objective

$$\arg\min_{\mathbf{w}} \sum_{t=1}^{m} [1 - y_t(\mathbf{x}_t \cdot \mathbf{w})]_+ + \lambda \|\mathbf{w}\|_p \text{ as } p \to 0$$

We will argue the above is hard in the limit $\lambda \to 0$.
Note $[a]_+ := a\mathcal{I}[a > 0]$.

Observation (finding maximally 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**.

Finding maximally sparse classifiers is hard (3)

Decision problem (*k*-consistent disjunction): Given examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m) \in \{0,1\}^n \times \{0,1\}$ does there exist a consistent k-literal disjunction? i.e, does there exist $\mathbf{u} \in \{0,1\}^n$ s.t.

$$(\mathbf{u} \cdot \mathbf{x} \geq 1 \iff y_t = 1) \text{ and } (\mathbf{u} \cdot \mathbf{x} = 0 \iff y_t = 0)$$

and $\|\mathbf{u}\|^2 \leq k$.

Theorem: consistent *k*-disjunction is NP-complete

The proof works by the following sequence of reductions.

$$\texttt{CNFSAT} \leq_p k\texttt{-set cover} \leq_p k\texttt{-consistent disjunction}$$

Cook – Levin Theorem

CNF (Conjunctive Normal Form): is a conjunction of clauses. For example

$$(x_1 \vee x_4 \vee x_7) \wedge x_1 \wedge (\bar{x_2} \vee x_2 \vee x_5)$$

CNFSAT – decision problem

Given a boolean function in conjuctive normal form does there exist an assignment to the variables so that the function is true.

Theorem

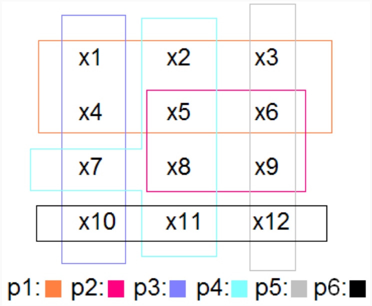
CNFSAT is **NP**-complete.

Observation: if $A \in \mathbf{NP}$ and CNFSAT $\leq_p A$ then A is **NP**-complete.

NP-complete (set cover) – 1

Theorem: k-set-cover is NP-complete

Given a set of m elements $X := \{x_1, x_2, \ldots, x_m\}$, a collection of n subsets of elements $S_1, \ldots, S_n \subseteq X$ 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, $\bigcup_{j=1}^{k} S_{i_j} = X$, the m elements is NP-complete.



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

NP-complete (set cover) – 2

Proof.

1. `set cover` $\in$ **NP**. The verifier simply checks if the proposed cover is a cover.
2. We proceed to prove that `CNFSAT` $\leq_p$ `set cover`. Any instance of `CNFSAT` may be written as follows. Let

   $$L = \{x_1, \ldots, x_m, \bar{x}_1, \ldots, \bar{x}_m\}$$

   be a set of literals and then

   $$\{C_1, \ldots, C_n\} \subseteq L$$

   are n subsets of literals then a CNF formula may be written as,

   $$\bigwedge_{i=1}^{n} \Big(\bigvee_{x \in C_i} x \Big)$$

NP-complete (set cover) – 3

Proof – continued.
From the CNF we construct a `set cover` instance as follows. We will construct a collection of sets $\mathcal{S} := \{S_1, \ldots, S_m, \bar{S}_1, \ldots, \bar{S}_m\}$ from the set of $n + m$ elements $E = \{c_1, \ldots, c_n, z_1, \ldots, z_m\}$ from the CNF formula $\bigwedge_{i=1}^{n}(\bigvee_{x \in C_i} x)$. Construct

$$S_i := \{c_j : x_i \in C_j\} \cup \{z_i\} \quad (1 \leq i \leq m)$$
$$\bar{S}_i := \{c_j : \bar{x}_i \in C_j\} \cup \{z_i\} \quad (1 \leq i \leq m)$$

and thus the setcover problem is

Does there exist m sets in $\mathcal{S}$ that cover E? We

now argue that for the proposed reduction that

yes-instance of k-`set cover` $\Longleftrightarrow$ yes-instance of `CNFSAT`

NP-complete (set cover) – 4

Proof – continued.

(`set cover` $\Longleftarrow$ `CNFSAT`) Observe that every **satisfying** assignment to the variables $x_1, \ldots, x_m$ corresponds to a m element set cover of $\mathcal{S}$ in particular if $\mathbf{x}^* \in \{t, f\}^m$ is a satisfying assignment then

$$\bigcup_{i=1}^{m} \{S_i : x_i^* = t\} \cup \{\bar{S}_i : x_i^* = f\}$$

is a cover of size m of E (why?).

(`set cover` $\Longrightarrow$ `CNFSAT`) Observe that every cover $\mathcal{C}$ of size m corresponds to satisfying truth assignment, as we may construct the truth assignment,

$$x_i^* := \begin{cases} t & S_i \in \mathcal{C} \\ f & \bar{S}_i \in \mathcal{C} \end{cases} \qquad (1 \leq i \leq m)$$

Note that S_i or $\bar{S}_i$ is in $\mathcal{C}$ otherwise element z_i is not covered; likewise both S_i or $\bar{S}_i$ cannot be in $\mathcal{C}$ because if they were some element z_j would not be covered (why?). The assignment $\mathbf{x}^*$ must satisfy since every c_j is covered. $\square$

Set Cover $\leq_p$ consistent disjunction

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. Let X be an $m \times n$ matrix of m, $n - \textit{dimensional}$ "positive" examples then the n "columns" are the "sets" and a consistent disjunction is a set cover. 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 the minimum consistent disjunction but neither find/learn or a minimal consistent disjunction.

Learning DNF with Winnow and the Perceptron – 1

DNF (Disjunctive Normal Form) corresponds to a simple boolean network with a single layer as such it may be learned by a neural network with a single hidden layer.

In the following we show that these boolean functions may be learned efficiently with the perceptron but with poor prediction performance. And inefficiently by winnow but with good prediction performance.

Learning DNF with Winnow and the Perceptron – 2

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

$$\mathbf{x} = \begin{array}{c} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{array} \Rightarrow \Phi(\mathbf{x}) = \begin{array}{c} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \\ x_1\, x_2 \\ \vdots \\ \vdots \\ x_1\, x_2 \ldots x_n \end{array}$$

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

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

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

Winnow versus Perceptron for k-term DNF

Perceptron: $\mathbf{w}_t = \sum_{q \in \text{mistakes}} \alpha_q \Phi(\mathbf{x}_q)$

Prediction:

$$\mathbf{w}_t \cdot \Phi(\mathbf{x}_t) = \left(\sum_{q \in \text{mistakes}} \alpha_q \Phi(\mathbf{x}_q) \right) \cdot \Phi(\mathbf{x}_t) = \sum_{q \in \text{mistakes}} \alpha_q \Phi(\mathbf{x}_q) \cdot \Phi(\mathbf{x}_t)$$

$$= \underbrace{\sum_{q \in \text{mistakes}} \alpha_q K(\mathbf{x}_q, \mathbf{x}_t)}_{\text{Prediction time: } O(n \cdot \#\text{ mistakes}) \le O(nm)}$$

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

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

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 $\mathbf{w}_t \cdot \Phi(\mathbf{x}_t)$ 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 $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$

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

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

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

Static Relative Loss Bounds

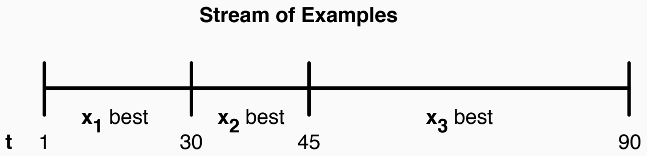
For all sequences of examples S

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

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

So far the comparator is “static”

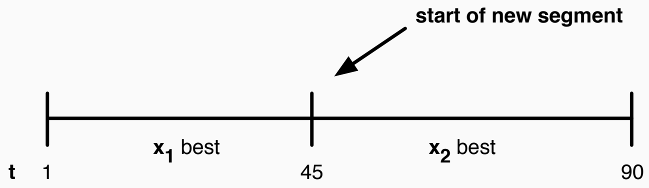
Shifting Expert



- 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 (square) loss defined relative to a sequence of experts

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

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

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

Where $L^* := \text{Loss}_{\{i_1, i_2, \ldots, i_\ell\}}(S)$

How?

Caveats:

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

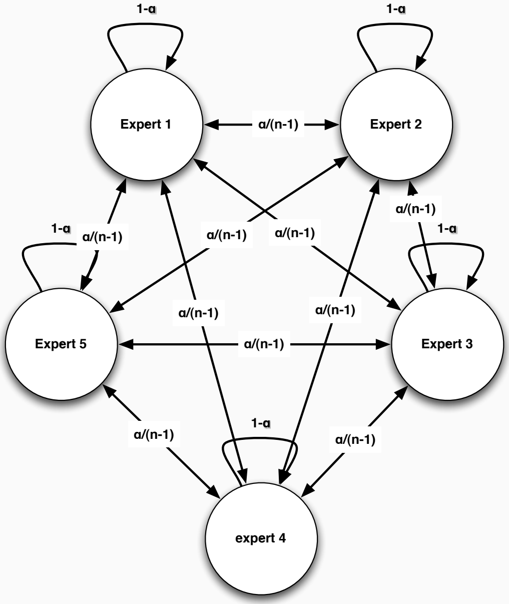
Fixed and Variable Share Algorithms

- **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 $\hat{y}_t = \mathbf{v}_t \cdot \mathbf{x}_t$
- **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:**
  - 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:**
  - 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

$$k := \text{size}(\{i_1, i_2, \ldots, i_\ell\}) := \sum_{k=1}^{\ell-1} [i_k \neq i_{k+1}]$$

- $L^* := \text{Loss}_{\{i_1, i_2, \ldots, i_\ell\}}(S)$
- 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

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

Shifting Loss Bounds (Proof Sketch – 1)

Proof Sketch – 1

1. Observe that the bound $\mathrm{Loss}_A(S) \leq \mathrm{Loss}_i(S) + c \ln n$ for the expert setting trivially generalises to

$$\mathrm{Loss}_A(S) \leq \mathrm{Loss}_i(S) + c \ln \frac{1}{w_{1,i}} \quad (i \in [n])$$

   i.e., rather than assign all experts uniform probability initially we can choose an arbitrary prior weights (probability).

2. Consider the following set of ℓ^n initial weights:

$$w_{1,\{i_1,i_2,\ldots,i_\ell\}} := \frac{1}{n}(1-\alpha)^{\ell-1-k}\left(\frac{\alpha}{n-1}\right)^k \quad (\{i_1, i_2, \ldots, i_\ell\} \in [n]^\ell)$$

3. Observe that the sum of these weights is 1.
4. These are just “meta-experts” that describe all possible expert sequences over ℓ trials.

Shifting Loss Bounds (Proof Sketch – 2)

Proof Sketch – 2

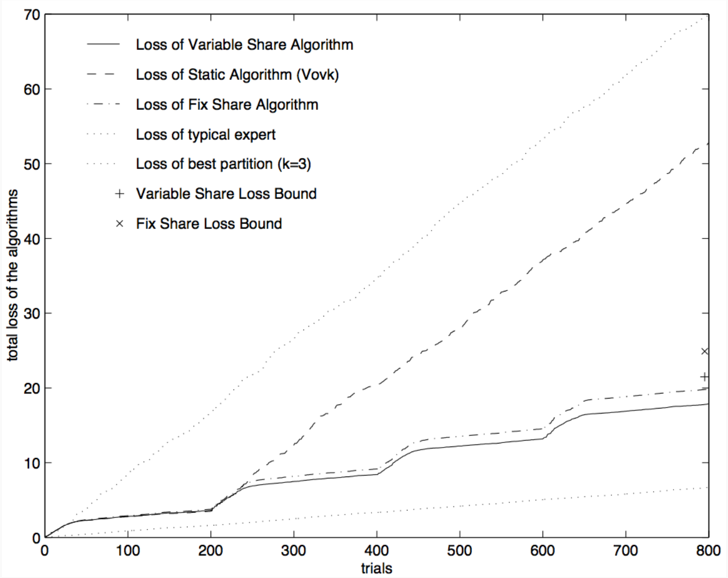
1. Observe if we apply the WA algorithm with these meta-experts then the previous bound follows (with some algebra) by just simplifying $\ln \frac{1}{w_{1,\{i_1,i_2,\ldots,i_\ell\}}}$.
2. With a more detailed analysis one can then show FIXED-SHARE algorithm perfectly simulates the prediction of the algorithm with ℓ^n meta-experts in $O(n)$ time.

□

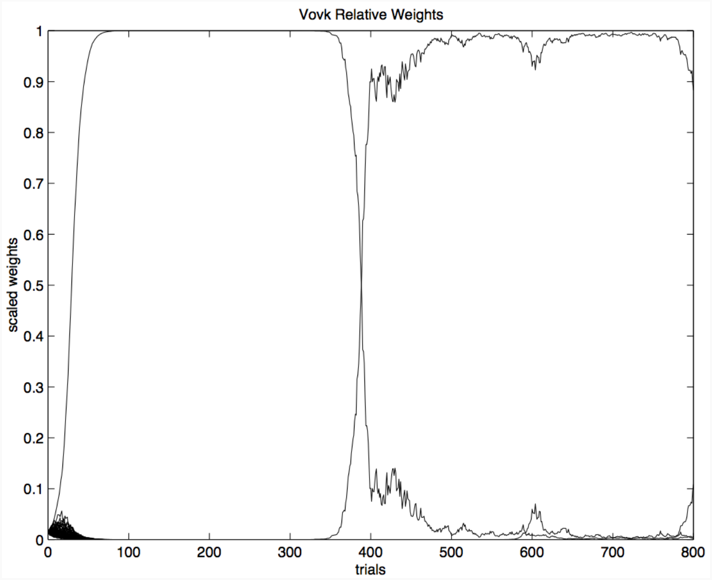
Experiment (simulation)

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

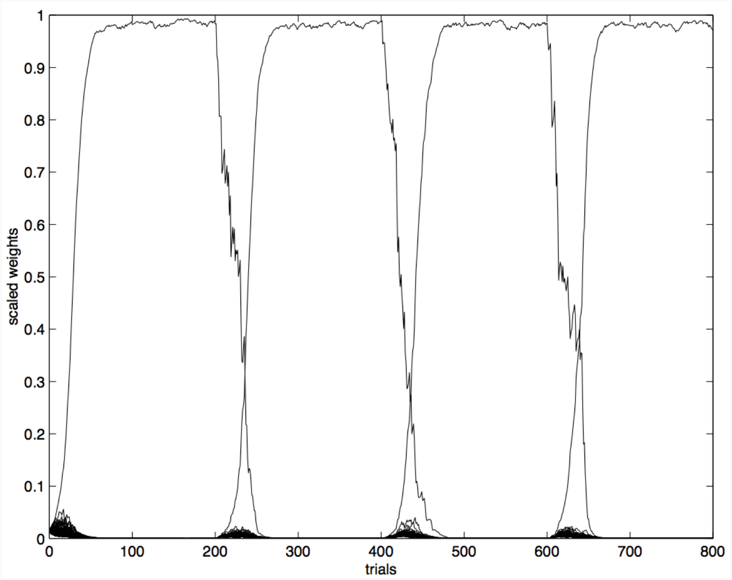
Share Algorithms – Performance



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

Problems – 1

1. Suppose $\mathcal{X} = \{\textsc{True}, \textsc{False}\}^n$. Give a polynomial time algorithm $\mathcal{A}$ with a mistake bound $M(\mathcal{A}) \leq O(n^2)$ for any example sequence which is consistent with a k-literal conjunction. Your answer should contain an argument that $M(\mathcal{A}) \leq O(n^2)$.
2. State the perceptron convergence theorem [Novikoff] explaining the relation with the hard margin support vector machine solution.
3. **[Hard]:** Define the c-regret of learning algorithm as

$$c\text{−regret}(m) = L_A(S) - c \min_{i \in [n]} L_i(S)$$

   thus the usual regret is the 1-regret.

   3.1 Argue for the weighted majority set-up argue that without randomised prediction it is impossible for all training sequences to obtain sublinear c-regret for $c < 2$.

   3.2 Show how to select β to obtain sublinear 2-regret.

4. Consider binary prediction with expert advice, with a perfect expert. Prove that any algorithm makes at least $\Omega(\min(m, \log_2 n))$ mistakes in the worst case.

Problems – 2

1. Recall that by tuning the weighted majority we achieved a bound

$$M \leq 2.63 \min_i M_i + 2.63 \ln n$$

Now by using randomisation in the prediction, design an algorithm with a bound that has the property

$$M \leq \min_{i \in [n]} M_i \text{ as } m \to \infty\,,$$

for the weighted majority setting. Recalling that m is the number of examples (and the "tuning" of the algorithm may depend on m). For contrast compare this to problem 3.1 above.

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. Y. Freund and R. Schapire, *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*, (1997)
6. Haussler, D., Kivinen, J. and Warmuth, M.K. *Sequential Prediction of Individual Sequences Under General Loss Functions,* (1998)
7. M. Herbster and M. Warmuth, *Tracking the Best Expert*, (1998)
8. J. Kivinen and M. Warmuth, *Averaging Expert predictions*, (1999)
9. S. Shalev-Schwartz, *Online Learning and Online Convex Optimization*, (2011)