Introduction to online learning

Learning without stochastic assumptions

Wojciech Kotłowski

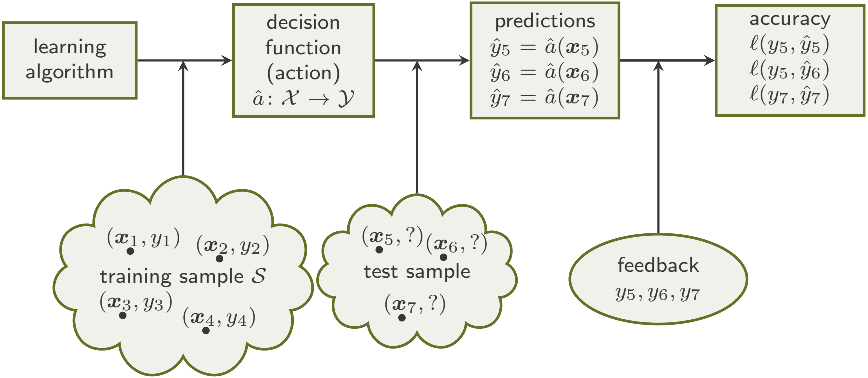
Institute of Computing Science
Poznań University of Technology

Polish Statistical Association Meeting
UAM, 14.03.2018

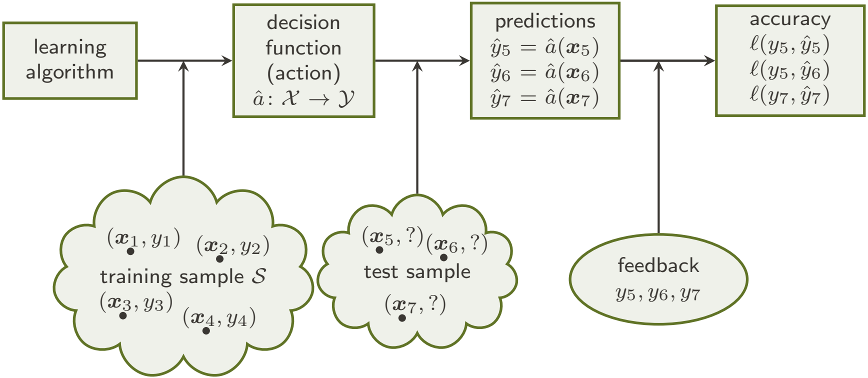
Outline

Outline

A typical scheme in machine learning



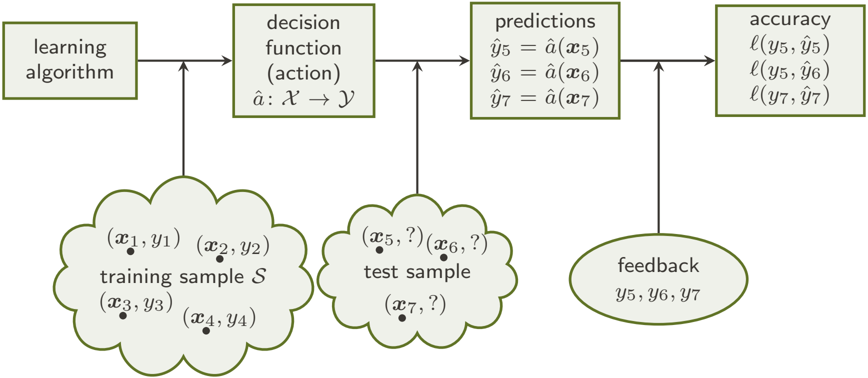
A typical scheme in machine learning



Goal

Given training sample $\mathcal{S}$, learn a decision function $\hat{a}$ so that average loss on a separate test sample is minimized.

A typical scheme in machine learning



Goal

Given training sample $\mathcal{S}$, learn a decision function $\hat{a}$ so that average loss on a separate test sample is minimized.

No reasonable solution without assumptions!

Statistical learning theory

Assumption

Training and test data generated i.i.d. from (unknown) distribution P

- Mean training error of a (empirical risk):

$$L_{\mathcal{S}}(a) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, a(\boldsymbol{x}_i))$$

- Test error $=$ expected error of a (risk):

$$L(a) = \mathbb{E}_{(\boldsymbol{x},y)\sim P} \left[\ell(y, a(\boldsymbol{x})) \right].$$

Given training data $\mathcal{S}$, how to construct $\hat{a}$ to make $L(\hat{a})$ small?

Statistical learning theory

Assumption

Training and test data generated i.i.d. from (unknown) distribution P

- Mean training error of a (empirical risk):

$$L_{\mathcal{S}}(a) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, a(\boldsymbol{x}_i))$$

- Test error $=$ expected error of a (risk):

$$L(a) = \mathbb{E}_{(\boldsymbol{x},y)\sim P} \left[\ell(y, a(\boldsymbol{x}))\right].$$

Given training data $\mathcal{S}$, how to construct $\hat{a}$ to make $L(\hat{a})$ small?
$\Longrightarrow$ Empirical risk minimization.

Generalization bounds

Theorem for finite classes ($0/1$ loss)

Let the class of decision functions $\mathcal{A}$ be finite. If function $\hat{a}$ was selected from $\mathcal{A}$ by minimizing the empirical risk on sample $\mathcal{S}$ of size n:

$$\hat{a} = \arg\min_{a \in \mathcal{A}} L_{\mathcal{S}}(a),$$

then with high probability and on expectation (over $\mathcal{S}$)

$$\underbrace{L(\hat{a}) - \min_{a \in \mathcal{A}} L(a)}_{\text{excess risk}} \; = \; O\left(\sqrt{\frac{\log |\mathcal{A}|}{n}}\right)$$

Generalization bounds

Theorem for VC classes ($0/1$ loss)

Let $\mathcal{A}$ has VC dimension d_{VC}. If function $\hat{a}$ was selected from $\mathcal{A}$ by minimizing the empirical risk on sample $\mathcal{S}$ of size n:

$$\hat{a} = \arg\min_{a \in \mathcal{A}} L_{\mathcal{S}}(a),$$

then with high probability and on expectation (over $\mathcal{S}$)

$$L(\hat{a}) - \min_{a \in \mathcal{A}} L(a) \;=\; O\left(\sqrt{\frac{d_{\mathrm{VC}}}{n}}\right)$$

Outline

Alternative view on learning

- Stochastic (i.i.d.) assumption sometimes unjustified, sometimes clearly invalid (e.g., time series)
- Learning process by its very nature is incremental.
- We do not observe the distributions, we only see the data.

Alternative view on learning

- Stochastic (i.i.d.) assumption sometimes unjustified, sometimes clearly invalid (e.g., time series)
- Learning process by its very nature is incremental.
- We do not observe the distributions, we only see the data.

Motivation

- Can we remove any probabilistic assumptions and treat the data generating process as completely arbitrary?

  Statistics without probabilities???
- Can we obtain performance guarantees solely based on observed quantities?

Alternative view on learning

- Stochastic (i.i.d.) assumption sometimes unjustified, sometimes clearly invalid (e.g., time series)
- Learning process by its very nature is incremental.
- We do not observe the distributions, we only see the data.

Motivation

- Can we remove any probabilistic assumptions and treat the data generating process as completely arbitrary?

  Statistics without probabilities???
- Can we obtain performance guarantees solely based on observed quantities?

we can! $\Longrightarrow$ **online learning theory**

Online learning theory

Also known as universal prediction or sequential prediction

- We do not make any probabilistic assumptions on the data (the data can even be adversarial)
- We consider a sequential setting in which the learning algorithm makes repeated predictions (in rounds) on the data sequence
- As without assumptions on the data, it is clearly impossible to perform well in an absolute sense, we compare our performance to the best predictions that could have been made by some decision function (action) in a restricted class of actions (e.g., linear functions)
- As it is impossible to bound the prediction error at a given round, we focus on cumulative errors over the whole sequence

Example: weather prediction (rain/sunny)

	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$\dots$
					
					

Example: weather prediction (rain/sunny)

	$t=1$	$t=2$	$t=3$	$t=4$	$\ldots$
	50%				
					

Example: weather prediction (rain/sunny)

	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$\ldots$
	50%				
					

Example: weather prediction (rain/sunny)

	$t=1$	$t=2$	$t=3$	$t=4$	$\ldots$
	50%	25%			
					

Example: weather prediction (rain/sunny)

	$t=1$	$t=2$	$t=3$	$t=4$	$\ldots$
	50%	25%			
					

Example: weather prediction (rain/sunny)

	$t=1$	$t=2$	$t=3$	$t=4$	$\dots$
	50%	25%	10%		
					

Example: weather prediction (rain/sunny)

	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$\dots$
	50%	25%	10%		
					

Example: weather prediction (rain/sunny)

	$t=1$	$t=2$	$t=3$	$t=4$	$\dots$
	50%	25%	10%	25%	
					

Example: weather prediction (rain/sunny)

	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$\dots$
	50%	25%	10%	25%	
					

Example: weather prediction (rain/sunny)

	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$\dots$
	50%	25%	10%	25%	$\dots$
					$\dots$

Example: weather prediction (rain/sunny)

expert	$t=1$	$t=2$	$t=3$	$t=4$	...
					
					
					
					
					
					

Example: weather prediction (rain/sunny)

expert	$t=1$	$t=2$	$t=3$	$t=4$	$\ldots$
	30%				
	10%				
	20%				
	60%				
					
					

Example: weather prediction (rain/sunny)

expert	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$\ldots$
	30%				
	10%				
	20%				
	60%				
	30%				
					

Example: weather prediction (rain/sunny)

expert	$t=1$	$t=2$	$t=3$	$t=4$	$\dots$
	30%				
	10%				
	20%				
	60%				
	30%				
					

Example: weather prediction (rain/sunny)

expert	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$\ldots$
	30%	50%			
	10%	80%			
	20%	70%			
	60%	30%			
	30%				
					

Example: weather prediction (rain/sunny)

expert	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$\ldots$
	30%	50%			
	10%	80%			
	20%	70%			
	60%	30%			
	30%	65%			
					

Example: weather prediction (rain/sunny)

expert	$t=1$	$t=2$	$t=3$	$t=4$	$\ldots$
	30%	50%			
	10%	80%			
	20%	70%			
	60%	30%			
	30%	65%			
					

Example: weather prediction (rain/sunny)

expert	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$\ldots$
	30%	50%	50%		
	10%	80%	50%		
	20%	70%	50%		
	60%	30%	50%		
	30%	65%			
					

Example: weather prediction (rain/sunny)

expert	$t=1$	$t=2$	$t=3$	$t=4$	$\ldots$
	30%	50%	50%		
	10%	80%	50%		
	20%	70%	50%		
	60%	30%	50%		
	30%	65%	50%		
					

Example: weather prediction (rain/sunny)

expert	$t=1$	$t=2$	$t=3$	$t=4$	$\ldots$
	30%	50%	50%		
	10%	80%	50%		
	20%	70%	50%		
	60%	30%	50%		
	30%	65%	50%		
					

Example: weather prediction (rain/sunny)

expert	$t = 1$	$t = 2$	$t = 3$	$t = 4$	...
	30%	50%	50%	10%	
	10%	80%	50%	10%	
	20%	70%	50%	30%	
	60%	30%	50%	80%	
	30%	65%	50%		
					

Example: weather prediction (rain/sunny)

expert	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$\dots$
	30%	50%	50%	10%	
	10%	80%	50%	10%	
	20%	70%	50%	30%	
	60%	30%	50%	80%	
	30%	65%	50%	10%	
					

Example: weather prediction (rain/sunny)

expert	$t=1$	$t=2$	$t=3$	$t=4$	$\ldots$
	30%	50%	50%	10%	
	10%	80%	50%	10%	
	20%	70%	50%	30%	
	60%	30%	50%	80%	
	30%	65%	50%	10%	
					

Example: weather prediction (rain/sunny)

expert	$t = 1$	$t = 2$	$t = 3$	$t = 4$	...
	30%	50%	50%	10%	...
	10%	80%	50%	10%	...
	20%	70%	50%	30%	...
	60%	30%	50%	80%	...
	30%	65%	50%	10%	...
					...

Example: weather prediction (rain/sunny)

- Prediction accuracy evaluated by a loss function, e.g.: $\ell(y_t, \hat{y}_t) = |y_t - \hat{y}_t|$.
- Total performance evaluated by a regret: algorithms's cumulative loss minus cumulative loss of the best expert in hindsight.

Example: weather prediction (rain/sunny)

- Prediction accuracy evaluated by a loss function, e.g.: $\ell(y_t, \hat{y}_t) = |y_t - \hat{y}_t|$.
- Total performance evaluated by a regret: algorithms's cumulative loss minus cumulative loss of the best expert in hindsight.

expert	1	2	3	4	cumulative loss
	30%	50%	50%	10%	
	10%	80%	50%	10%	
	20%	70%	50%	30%	
	60%	30%	50%	80%	
	30%	65%	50%	10%	
					

Example: weather prediction (rain/sunny)

- Prediction accuracy evaluated by a loss function, e.g.: $\ell(y_t, \hat{y}_t) = |y_t - \hat{y}_t|$.
- Total performance evaluated by a regret: algorithms's cumulative loss minus cumulative loss of the best expert in hindsight.

expert	1	2	3	4	cumulative loss
	30%	50%	50%	10%	$0.3 + 0.5 + 0.5 + 0.1 = 1.4$
	10%	80%	50%	10%	$\mathbf{0.1 + 0.2 + 0.5 + 0.1 = 0.9}$
	20%	70%	50%	30%	$0.2 + 0.3 + 0.5 + 0.3 = 1.3$
	60%	30%	50%	80%	$0.6 + 0.7 + 0.5 + 0.8 = 2.6$
	30%	65%	50%	10%	$0.3 + 0.45 + 0.5 + 0.1 = 1.35$
					

Example: weather prediction (rain/sunny)

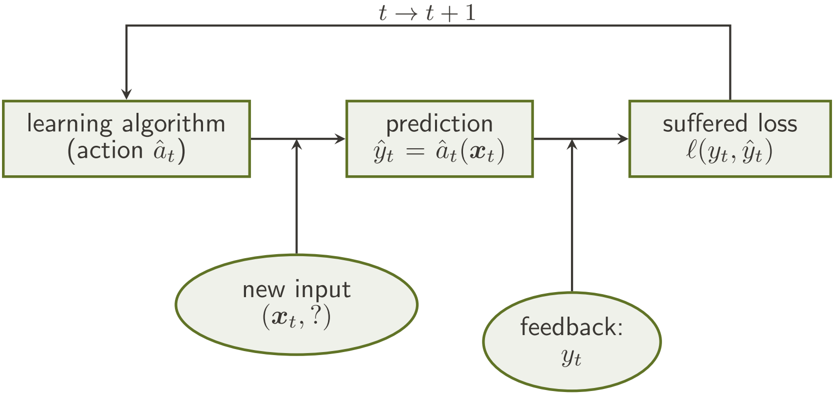
- Prediction accuracy evaluated by a loss function, e.g.: $\ell(y_t, \hat{y}_t) = |y_t - \hat{y}_t|$.
- Total performance evaluated by a regret: algorithms's cumulative loss minus cumulative loss of the best expert in hindsight.

expert	1	2	3	4	cumulative loss
	30%	50%	50%	10%	$0.3 + 0.5 + 0.5 + 0.1 = 1.4$
	10%	80%	50%	10%	$\mathbf{0.1 + 0.2 + 0.5 + 0.1 = 0.9}$
	20%	70%	50%	30%	$0.2 + 0.3 + 0.5 + 0.3 = 1.3$
	60%	30%	50%	80%	$0.6 + 0.7 + 0.5 + 0.8 = 2.6$
	30%	65%	50%	10%	$0.3 + 0.45 + 0.5 + 0.1 = 1.35$
					

Regret of the algorithm: $1.35 - 0.9 = 0.45$.

The goal is to have small regret for any data sequence.

Online learning framework



Online learning framework

Comparator class of actions (decision functions)

The algorithm's performance is compared against a class of actions $\mathcal{A}$.
The goal is to predict (almost) as good as the best action in $\mathcal{A}$

Online learning framework

Comparator class of actions (decision functions)

The algorithm's performance is compared against a class of actions $\mathcal{A}$.
The goal is to predict (almost) as good as the best action in $\mathcal{A}$

Cumulative loss of the algorithm:

$$\hat{L}_n = \sum_{t=1}^{n} \ell(y_t, \hat{y}_t)$$

Cumulative loss of the best action $a \in \mathcal{A}$ in hindsight:

$$L_n^* = \min_{a \in \mathcal{A}} \sum_{t=1}^{n} \ell(y_t, a(\boldsymbol{x}_t))$$

Online learning framework

Comparator class of actions (decision functions)

The algorithm's performance is compared against a class of actions $\mathcal{A}$.
The goal is to predict (almost) as good as the best action in $\mathcal{A}$

Cumulative loss of the algorithm:

$$\hat{L}_n = \sum_{t=1}^{n} \ell(y_t, \hat{y}_t)$$

Cumulative loss of the best action $a \in \mathcal{A}$ in hindsight:

$$L_n^* = \min_{a \in \mathcal{A}} \sum_{t=1}^{n} \ell(y_t, a(\boldsymbol{x}_t))$$

Regret of the algorithm:

$$R_n = \hat{L}_n - L_n^*.$$

Quantifies suboptimality: how much less loss could we have incurred had we played with the optimal action from the beginning?

The goal is to have small (sublinear) regret on all possible data sequences.

Outline

Prediction with expert advice

- N actions ("experts") to follow: $\mathcal{A} = \{a_1, \ldots, a_N\}$
- Input $\boldsymbol{x}_t = (x_{t,1}, \ldots, x_{t,N})$: vector of experts' predictions
- Prediction of k-th expert (action) at round i: $a_k(\boldsymbol{x}_t) = x_{t,k}$
- Algorithm's action $\hat{a}_t$: weight vector $\hat{\boldsymbol{a}}_t = (\hat{a}_{t,1}, \ldots, \hat{a}_{t,N}) \in \Delta^N$, where $\Delta^N = \{\boldsymbol{p} \colon \sum_{k=1}^N p_k = 1, p_k \geq 0\}$

  (current weights assigned to each expert by the algorithm)
- Algorithm's prediction: weighted average of experts' predictions

$$\hat{y}_t = \hat{a}_t(\boldsymbol{x}_t) = \sum_{k=1}^{N} \hat{a}_{t,k} x_{t,k} = \hat{\boldsymbol{a}}_t^\top \boldsymbol{x}_t$$

Prediction with expert advice

Algorithm start with some initial weight vector (action) $\hat{\boldsymbol{a}}_1 \in \Delta^N$ (e.g. uniform distribution $\hat{\boldsymbol{a}}_1 = (\frac{1}{N}, \ldots, \frac{1}{N})$)

For $t = 1, 2, \ldots$:

1. Experts reveals their predictions: $\boldsymbol{x}_t = (x_{t,1}, \ldots, x_{t,N})$
2. Algorithm predicts with $\hat{y}_t = \hat{\boldsymbol{a}}_t^\top \boldsymbol{x}_t$.
3. The environment reveals outcome y_t.
4. Algorithm suffers loss $\hat{\ell}_t = \ell(y_t, \hat{y}_t)$
   and each expert k suffers loss $\ell_t(k) = \ell(y_t, x_{t,k})$, $k = 1, \ldots, N$
5. Algorithm updates its vector $\hat{\boldsymbol{a}}_t \to \hat{\boldsymbol{a}}_{t+1}$.

Prediction with expert advice

Algorithm start with some initial weight vector (action) $\hat{\boldsymbol{a}}_1 \in \Delta^N$ (e.g. uniform distribution $\hat{\boldsymbol{a}}_1 = (\frac{1}{N}, \dots, \frac{1}{N})$)

For $t = 1, 2, \dots$:

1. Experts reveals their predictions: $\boldsymbol{x}_t = (x_{t,1}, \dots, x_{t,N})$
2. Algorithm predicts with $\hat{y}_t = \hat{\boldsymbol{a}}_t^\top \boldsymbol{x}_t$.
3. The environment reveals outcome y_t.
4. Algorithm suffers loss $\hat{\ell}_t = \ell(y_t, \hat{y}_t)$
   and each expert k suffers loss $\ell_t(k) = \ell(y_t, x_{t,k})$, $k = 1, \dots, N$
5. Algorithm updates its vector $\hat{\boldsymbol{a}}_t \to \hat{\boldsymbol{a}}_{t+1}$.

Goal: minimize regret with respect to the best expert

$$R_n = \hat{L}_n - \min_{k=1,\dots,N} L_n(k), \qquad \text{where } \hat{L}_n = \sum_{t=1}^n \hat{\ell}_t,\ L_n(k) = \sum_{t=1}^n \ell_t(k)$$

First attempt: Follow the Leader strategy

Follow the Leader (FTL)

At iteration t, follow the expert with the smallest loss so far

$$k_{\min} = \operatorname*{argmin}_{k=1,\dots,N} L_{t-1}(k).$$

Choose $\hat{\boldsymbol{a}}_t$ such that:

$$\hat{a}_{t,k} \ = \ \begin{cases} 1 & \text{if } k = k_{\min} \\ 0 & \text{otherwise} \end{cases}$$

In other words, $\hat{y}_t = x_{t,k_{\min}}$

Failure of FTL

expert	1	2	3	4	5	6	7	loss
								0
								0
								0
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%							0
	25%							0
								0
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%							0
	25%							0
	50%							0
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%							0.75
	25%							0.25
	50%							0.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%						0.75
	25%	0%						0.25
	50%							0.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%						0.75
	25%	0%						0.25
	50%	0%						0.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%						0.75
	25%	0%						1.25
	50%	0%						1.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%					0.75
	25%	0%	0%					1.25
	50%	0%						1.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%					0.75
	25%	0%	0%					1.25
	50%	0%	100%					1.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%					1.75
	25%	0%	0%					1.25
	50%	0%	100%					2.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%				1.75
	25%	0%	0%	0%				1.25
	50%	0%	100%					2.5
								

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%				1.75
	25%	0%	0%	0%				1.25
	50%	0%	100%	0%				2.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%				1.75
	25%	0%	0%	0%				2.25
	50%	0%	100%	0%				3.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%	100%			1.75
	25%	0%	0%	0%	0%			2.25
	50%	0%	100%	0%				3.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%	100%			1.75
	25%	0%	0%	0%	0%			2.25
	50%	0%	100%	0%	100%			3.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%	100%			2.75
	25%	0%	0%	0%	0%			2.25
	50%	0%	100%	0%	100%			4.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%	100%	100%		2.75
	25%	0%	0%	0%	0%	0%		2.25
	50%	0%	100%	0%	100%			4.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%	100%	100%		2.75
	25%	0%	0%	0%	0%	0%		2.25
	50%	0%	100%	0%	100%	0%		4.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%	100%	100%		2.75
	25%	0%	0%	0%	0%	0%		3.25
	50%	0%	100%	0%	100%	0%		5.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%	100%	100%	100%	2.75
	25%	0%	0%	0%	0%	0%	0%	3.25
	50%	0%	100%	0%	100%	0%		5.5
								

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%	100%	100%	100%	2.75
	25%	0%	0%	0%	0%	0%	0%	3.25
	50%	0%	100%	0%	100%	0%	100%	5.5
								

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%	100%	100%	100%	3.75
	25%	0%	0%	0%	0%	0%	0%	3.25
	50%	0%	100%	0%	100%	0%	100%	6.5
								

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%	100%	100%	100%	3.75
	25%	0%	0%	0%	0%	0%	0%	3.25
	50%	0%	100%	0%	100%	0%	100%	6.5
								

$\hat{L}_n \simeq n, \qquad \min_k L_n(k) \simeq \frac{n}{2}, \qquad R_n \simeq \frac{n}{2}$ (regret linear in n)

Failure of FTL

expert	1	2	3	4	5	6	7	loss
	75%	100%	100%	100%	100%	100%	100%	3.75
	25%	0%	0%	0%	0%	0%	0%	3.25
	50%	0%	100%	0%	100%	0%	100%	6.5
								

$$\hat{L}_n \simeq n, \qquad \min_k L_n(k) \simeq \frac{n}{2}, \qquad R_n \simeq \frac{n}{2}$$ (regret linear in n)

Algorithm must hedge its bets on experts!

Exponential weights [Littlestone & Warmuth, 1994]

Algorithm

Each time expert k receives a loss $\ell(k)$, multiply the weight $\hat{a}_k$ associated with that expert by $e^{-\eta\ell(k)}$, where $\eta > 0$.

Exponential weights [Littlestone & Warmuth, 1994]

Algorithm

Each time expert k receives a loss $\ell(k)$, multiply the weight $\hat{a}_k$ associated with that expert by $e^{-\eta\ell(k)}$, where $\eta > 0$.

$$\hat{a}_{t+1,k} = \frac{\hat{a}_{t,k} e^{-\eta\ell_t(k)}}{Z_t}, \qquad \text{where } Z_t = \sum_{k=1}^{N} \hat{a}_{t,k} e^{-\eta\ell_t(k)}$$

Exponential weights [Littlestone & Warmuth, 1994]

Algorithm

Each time expert k receives a loss $\ell(k)$, multiply the weight $\hat{a}_k$ associated with that expert by $e^{-\eta\ell(k)}$, where $\eta > 0$.

$$\hat{a}_{t+1,k} = \frac{\hat{a}_{t,k} e^{-\eta\ell_t(k)}}{Z_t}, \qquad \text{where } Z_t = \sum_{k=1}^{N} \hat{a}_{t,k} e^{-\eta\ell_t(k)}$$

Unwinding this update:

$$\hat{a}_{t+1,k} = \frac{e^{-\eta L_t(k)}}{Z_t}, \qquad \text{where } Z_t = \sum_{k=1}^{N} e^{-\eta L_t(k)}$$

Exponential Weights as Bayesian update

- Prior probability over N alternatives $E_1, \ldots, E_N$.
- Data likelihoods: $P(D_t|E_k)$, $k = 1, \ldots, N$.

$$P(E_k|D_t) \;=\; \frac{P(D_t|E_k) \;\times\; P(E_k)}{\sum_{j=1}^{N} P(D_t|E_j) \times P(E_j)}$$

Exponential Weights as Bayesian update

- Prior probability over N alternatives $E_1, \ldots, E_N$.
- Data likelihoods: $P(D_t|E_k)$, $k = 1, \ldots, N$.

posterior probability $\hat{a}_{t+1,k}$

data likelihood $e^{-\eta \ell_t(k)}$

prior probability $\hat{a}_{t,k}$

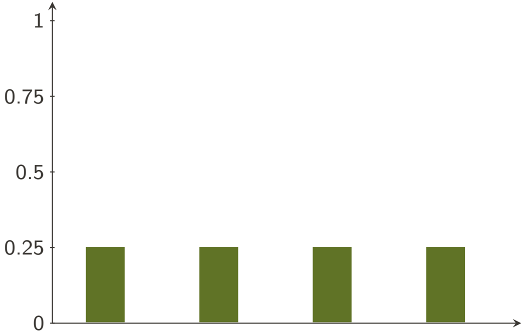
$$\boxed{P(E_k|D_t)} = \frac{\boxed{P(D_t|E_k)} \times \boxed{P(E_k)}}{\boxed{\sum_{j=1}^{N} P(D_t|E_j) \times P(E_j)}}$$

normalization Z_t

Exponential Weights example ($\eta = 2$)

30% 10% 20% 60% 30%



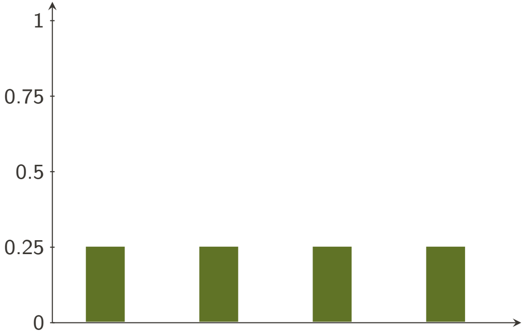
Exponential Weights example ($\eta = 2$)





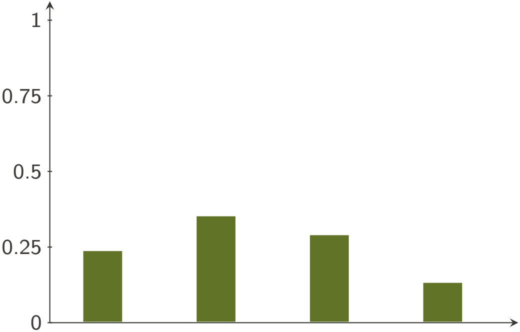

30% 10% 20% 60% 30%

0.3 **0.1** **0.2** **0.6** **0.3**



Exponential Weights example ($\eta = 2$)

					
30%	10%	20%	60%	30%	
0.3	**0.1**	**0.2**	**0.6**	**0.3**	

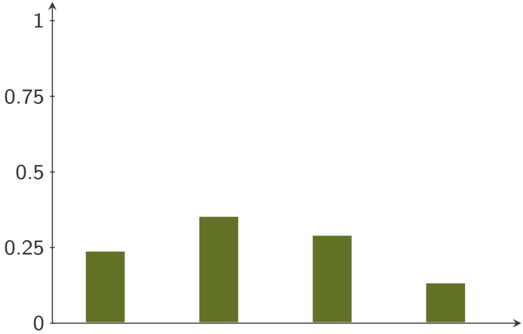


Exponential Weights example ($\eta = 2$)



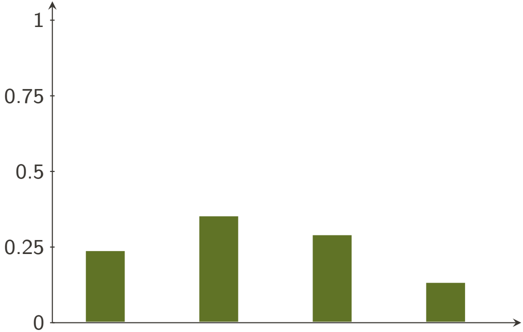



50% 80% 70% 30% 64%



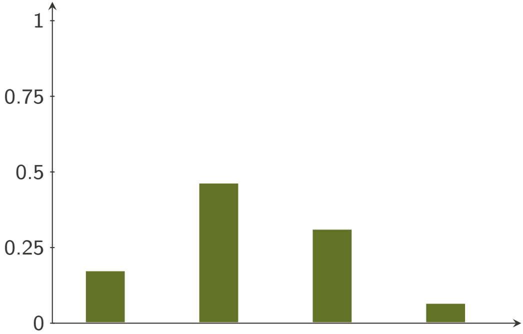
Exponential Weights example ($\eta = 2$)

					
50%	80%	70%	30%	64%	
0.5	**0.2**	**0.3**	**0.7**	**0.36**	



Exponential Weights example ($\eta = 2$)

					
50%	80%	70%	30%	64%	
0.5	**0.2**	**0.3**	**0.7**	**0.36**	



Exponential Weights example ($\eta = 2$)

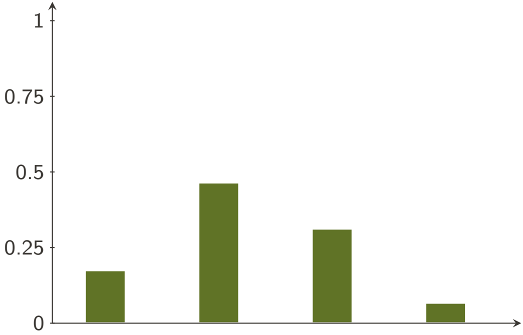
50%

50%

50%

50%

50%

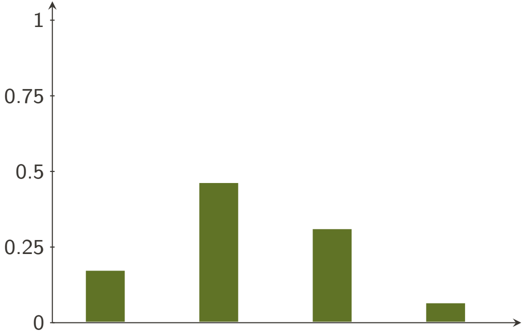


Exponential Weights example ($\eta = 2$)





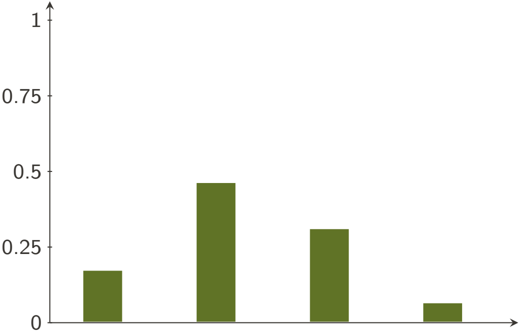

50%	50%	50%	50%	50%	
0.5	**0.5**	**0.5**	**0.5**	**0.5**	



Exponential Weights example ($\eta = 2$)

50%	50%	50%	50%	50%
0.5	**0.5**	**0.5**	**0.5**	**0.5**



Exponential Weights example ($\eta = 2$)

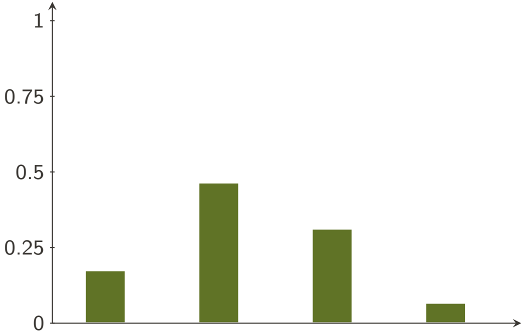
10%

10%

30%

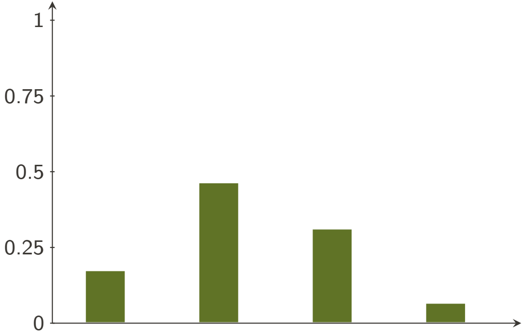
80%

21%



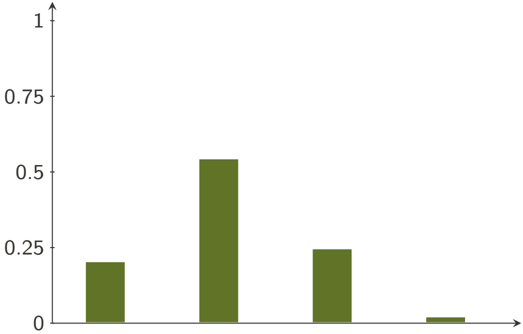
Exponential Weights example ($\eta = 2$)

					
10%	10%	30%	80%	21%	
0.1	**0.1**	**0.3**	**0.8**	**0.21**	



Exponential Weights example ($\eta = 2$)

					
10%	10%	30%	80%	21%	
0.1	**0.1**	**0.3**	**0.8**	**0.21**	



Exponential weights analysis: convex losses

Let $\ell(y, \hat{y})$ be bounded (e.g. in $[0, 1]$) and convex with respect to $\hat{y}$

Exponential weights analysis: convex losses

Let $\ell(y, \hat{y})$ be bounded (e.g. in $[0, 1]$) and convex with respect to $\hat{y}$

Regret bound

For any data sequence, when $\eta = \sqrt{\frac{8 \log N}{n}}$,

$$R_n \leq \sqrt{\frac{n \log N}{2}}$$

Sublinear regret: regret per trial $\frac{R_n}{n}$ converges to 0 as $\frac{1}{\sqrt{n}}$

Exponential weights analysis: convex losses

Let $\ell(y, \hat{y})$ be bounded (e.g. in $[0,1]$) and convex with respect to $\hat{y}$

Regret bound

For any data sequence, when $\eta = \sqrt{\frac{8 \log N}{n}}$,

$$R_n \leq \sqrt{\frac{n \log N}{2}}$$

Sublinear regret: regret per trial $\frac{R_n}{n}$ converges to 0 as $\frac{1}{\sqrt{n}}$

Regret bound

For any data sequence, let $L_n^* = \min_k L_n(k)$. When $\eta = \sqrt{\frac{2 \ln N}{L_n^*}}$,

$$R_n \leq \sqrt{2 L_n^* \ln N} + \ln N$$

Both bounds are tight.

Exp-concave losses

Exp-concave function

Function $f(x)$ is α-exp-concave, if $e^{-\alpha f(x)}$ is concave

Loss $\ell(y, \hat{y})$ is α-exp-concave if for any y, $f(\hat{y}) = \ell(y, \hat{y})$ is α-exp-concave

Exp-concavity implies convexity, but not vice versa

Exp-concave losses

Exp-concave function

Function $f(x)$ is α-exp-concave, if $e^{-\alpha f(x)}$ is concave

Loss $\ell(y, \hat{y})$ is α-exp-concave if for any y, $f(\hat{y}) = \ell(y, \hat{y})$ is α-exp-concave

Exp-concavity implies convexity, but not vice versa

- Squared loss
$$\ell(y, \hat{y}) = (y - \hat{y})^2$$
is $\frac{1}{2}$-exp-concave for $y, \hat{y} \in [0, 1]$

Exp-concave losses

Exp-concave function

Function $f(x)$ is α-exp-concave, if $e^{-\alpha f(x)}$ is concave

Loss $\ell(y, \hat{y})$ is α-exp-concave if for any y, $f(\hat{y}) = \ell(y, \hat{y})$ is α-exp-concave

Exp-concavity implies convexity, but not vice versa

- Squared loss
$$\ell(y, \hat{y}) = (y - \hat{y})^2$$
is $\frac{1}{2}$-exp-concave for $y, \hat{y} \in [0, 1]$
- Cross-entropy loss
$$\ell(y, \hat{y}) = -y \ln \hat{y} - (1 - y) \ln(1 - \hat{y})$$
for $y, \hat{y} \in [0, 1]$ is 1-exp-concave

Exp-concave losses

Exp-concave function

Function $f(x)$ is α-exp-concave, if $e^{-\alpha f(x)}$ is concave

Loss $\ell(y, \hat{y})$ is α-exp-concave if for any y, $f(\hat{y}) = \ell(y, \hat{y})$ is α-exp-concave

Exp-concavity implies convexity, but not vice versa

- Squared loss
$$\ell(y, \hat{y}) = (y - \hat{y})^2$$
is $\frac{1}{2}$-exp-concave for $y, \hat{y} \in [0, 1]$
- Cross-entropy loss
$$\ell(y, \hat{y}) = -y \ln \hat{y} - (1 - y) \ln(1 - \hat{y})$$
for $y, \hat{y} \in [0, 1]$ is 1-exp-concave
- Absolute loss
$$\ell(y, \hat{y}) = |y - \hat{y}|$$
is not exp-concave for any α

Exponential weights analysis: exp-concave losses

Let loss function $\ell(y, \hat{y})$ be α-exp-concave

Exponential weights analysis: exp-concave losses

Let loss function $\ell(y, \hat{y})$ be α-exp-concave

Regret bound

For any data sequence, when $\eta = \alpha$,

$$R_n \leq \frac{\ln N}{\alpha}$$

Constant regret: regret per trial $\frac{R_n}{n}$ converges to 0 as $\frac{1}{n}$

General losses

Can we still achieve sublinear regret for non-convex loss $\ell(y, \hat{y})$?

General losses

Can we still achieve sublinear regret for non-convex loss $\ell(y, \hat{y})$?

Yes, but the algorithm needs to randomize:

- Update weights using Exponential Weights algorithm
- At trial t, predict as expert k (i.e., $\hat{y}_t = x_{t,k}$) with probability $\hat{a}_{t,k}$

General losses

Can we still achieve sublinear regret for non-convex loss $\ell(y, \hat{y})$?

Yes, but the algorithm needs to randomize:

- Update weights using Exponential Weights algorithm
- At trial t, predict as expert k (i.e., $\hat{y}_t = x_{t,k}$) with probability $\hat{a}_{t,k}$

Expected (with respect to internal randomization) loss of the algorithm:

$$\mathbb{E}[\hat{\ell}_t] = \sum_{i=1}^{n} \hat{a}_{t,k}\, \ell(y_t, x_{t,k})$$

This loss is effectively linear (hence convex) as a function of $\hat{\boldsymbol{a}}_t$

Exponential weights analysis: general losses

Let the loss function $\ell(y, \hat{y})$ be bounded (e.g. in $[0, 1]$)

Regret bound

For any data sequence, when $\eta = \sqrt{\frac{8 \log N}{n}}$, it holds on expectation and with high probability (with respect to internal randomization of the algorithm):

$$R_n \leq \sqrt{\frac{n \log N}{2}}$$

The same bound as for convex losses

Statistical learning theory vs. online learning theory

Let $\mathcal{A}$ be a *finite* class of decision functions/actions

Statistical learning theory

Theorem

Function $\hat{a}$ trained by empirical risk minimization achieves:

$$\underbrace{L(\hat{a}) - \min_{a \in \mathcal{A}} L(a)}_{\text{excess risk}} = O\left(\sqrt{\frac{\ln |\mathcal{A}|}{n}}\right)$$

Online learning theory

Theorem

Exponential Weights algorithm achieves:

$$\frac{1}{n}\Big(\underbrace{\hat{L}_n - \min_{a \in \mathcal{A}} L_n(a)}_{\text{regret}}\Big) = O\left(\sqrt{\frac{\ln |\mathcal{A}|}{n}}\right)$$

Statistical learning theory vs. online learning theory

Let $\mathcal{A}$ be a *finite* class of decision functions/actions

Statistical learning theory

Theorem

Function $\hat{a}$ trained by empirical risk minimization achieves:

$$\underbrace{L(\hat{a}) - \min_{a \in \mathcal{A}} L(a)}_{\text{excess risk}} = O\left(\sqrt{\frac{\ln |\mathcal{A}|}{n}}\right)$$

Online learning theory

Theorem

Exponential Weights algorithm achieves:

$$\frac{1}{n}\Big(\underbrace{\hat{L}_n - \min_{a \in \mathcal{A}} L_n(a)}_{\text{regret}}\Big) = O\left(\sqrt{\frac{\ln |\mathcal{A}|}{n}}\right)$$

Essentially the same performance (with excess risk replaced by regret per trial) without i.i.d. assumption!

Online to batch conversion

Theorem

Let $\ell(y, \hat{y})$ be a convex loss function.

Let $\hat{\boldsymbol{a}}_1, \hat{\boldsymbol{a}}_2, \ldots, \hat{\boldsymbol{a}}_n$ be a sequence of actions produced by an online learning algorithm, which guarantees the regret to be bounded by $R_n \leq g(n)$ for any data sequence.

Let $\bar{\boldsymbol{a}}_n = \frac{1}{n}\sum_{t=1}^{n} \hat{\boldsymbol{a}}_t$. Then, the excess risk of $\bar{\boldsymbol{a}}_n$ is bounded by:

$$L(\bar{\boldsymbol{a}}_n) - \min_{a \in \mathcal{A}} L(a) \leq \frac{g(n)}{n}$$

Online to batch conversion

Theorem

Let $\ell(y, \hat{y})$ be a convex loss function.

Let $\hat{\boldsymbol{a}}_1, \hat{\boldsymbol{a}}_2, \ldots, \hat{\boldsymbol{a}}_n$ be a sequence of actions produced by an online learning algorithm, which guarantees the regret to be bounded by $R_n \leq g(n)$ for any data sequence.

Let $\bar{\boldsymbol{a}}_n = \frac{1}{n} \sum_{t=1}^{n} \hat{\boldsymbol{a}}_t$. Then, the excess risk of $\bar{\boldsymbol{a}}_n$ is bounded by:

$$L(\bar{\boldsymbol{a}}_n) - \min_{a \in \mathcal{A}} L(a) \leq \frac{g(n)}{n}$$

Similar conversion for non-convex losses (requires randomization)

Finite action spaces: extensions

- Large (or countably infinite) classes of actions.
- Concept drift: competing with the best sequence of actions.
- Competing with the best small set of recurring actions.
- Ranking: competing with the best permutation.
- Partial feedback: multi-armed bandits.
- ...

Outline

Online convex optimization

Let $\mathcal{A} \subseteq \mathbb{R}^d$ be a convex set of actions

For $t = 1, 2, \ldots$

1. Algorithm picks action $\hat{\boldsymbol{a}}_t \in \mathcal{A}$
2. The environment reveals convex loss $\ell_t \colon \mathcal{A} \to \mathbb{R}$
3. The algorithm suffers loss $\ell_t(\hat{\boldsymbol{a}}_t)$

(information about inputs $\boldsymbol{x}_t$ and outputs y_t hidden inside ℓ_t)

The goal is to minimize regret:

$$R_n = \sum_{t=1}^{n} \ell_t(\hat{\boldsymbol{a}}_t) - \min_{\boldsymbol{a} \in \mathcal{A}} \sum_{t=1}^{n} \ell_t(\boldsymbol{a})$$

Example: linear classification and regression

Action $\boldsymbol{a} \in \mathcal{A}$: parameter vector of a linear classifier/regression function:

$$\boldsymbol{a} = (a_1, \ldots, a_d) \in \mathbb{R}^d$$

$\mathcal{A}$ can be $\mathbb{R}^d$ or can be a regularization ball $\mathcal{A} = \{\boldsymbol{a} \colon \|\boldsymbol{a}\|_p \leq B\}$.

For $t = 1, 2, \ldots$:

1. Algorithm picks action $\hat{\boldsymbol{a}}_t \in \mathcal{A}$
2. The environment reveals feature vector $\boldsymbol{x}_t$
3. The algorithm predicts class label/real output $\hat{y}_t = \hat{\boldsymbol{a}}_t^\top \boldsymbol{x}_t$
4. The environment reveals y_t
5. The algorithm suffers loss $\ell_t(\hat{\boldsymbol{a}}_t) = \ell(y_t, \hat{y}_t)$ convex in $\hat{y}_t$ (and thus convex in $\hat{\boldsymbol{a}}_t$)

Goal: minimize regret to the best linear function in $\mathcal{A}$:

$$R_n = \sum_{t=1}^{n} \ell_t(\hat{\boldsymbol{a}}_t) - \min_{\boldsymbol{a} \in \mathcal{A}} \sum_{t=1}^{n} \ell_t(\boldsymbol{a}) = \sum_{t=1}^{n} \ell(y_t, \hat{y}_t) - \min_{\boldsymbol{a} \in \mathcal{A}} \sum_{t=1}^{n} \ell(y_t, \boldsymbol{a}^\top \boldsymbol{x}_t)$$

Examples

- Linear regression: $y \in \mathbb{R}$ and

$$\ell(y, \hat{y}) = (y - \hat{y})^2.$$

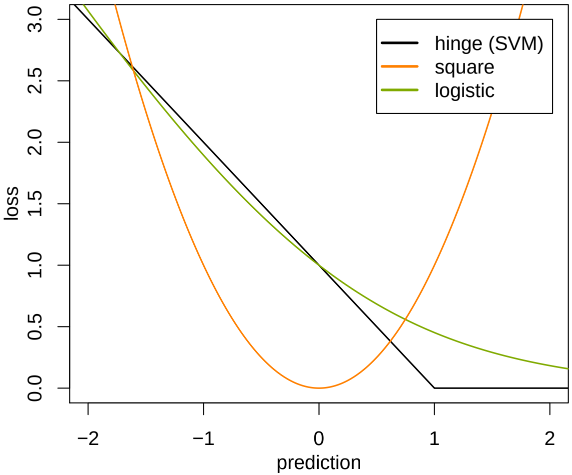
- Logistic regression: $y \in \{0, 1\}$ and

$$\ell(y, \hat{y}) = \ln\left(1 + e^{-y\hat{y}}\right).$$

- Support vector machines: $y \in \{0, 1\}$ and

$$\ell(y, \hat{y}) = (1 - y\hat{y})_+$$

Examples



Logistic and hinge losses plotted for $y = 1$.
Squared error loss plotted for $y = 0$.

Exponential Weights algorithm

Using Bayesian interpretation of Exponential Weights, we can extend it to continuous spaces of actions:

Algorithm starts with a prior distribution P_1 over $\mathcal{A}$

For $t = 1, 2, \ldots$:

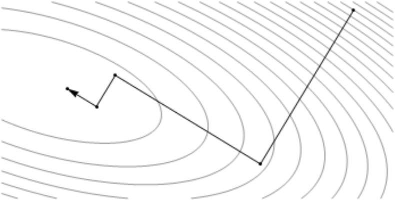
1. Algorithm chooses action $\hat{\boldsymbol{a}}_t = \mathbb{E}_{\boldsymbol{a} \sim P_t}[\boldsymbol{a}]$
2. Loss function $\ell_t \colon \mathcal{A} \to \mathbb{R}$ is revealed and algorithm suffers loss $\ell_t(\hat{\boldsymbol{a}}_t)$
3. Algorithm updates its distribution:

$$P_t(\boldsymbol{a}) \;=\; \frac{1}{Z_t} e^{-\eta \ell_t(\boldsymbol{a})} P_t(\boldsymbol{a}), \qquad \text{where} \;\; Z_t \;=\; \int_{\mathcal{A}} e^{-\eta \ell_t(\boldsymbol{a})} \, \mathrm{d}P_t(\boldsymbol{a})$$

Works very well and has good regret bounds, but computationally inefficient in most cases

Gradient descent method

Minimize a function $f(\boldsymbol{a})$ over $\boldsymbol{a} \in \mathbb{R}^d$.



Gradient descent method:

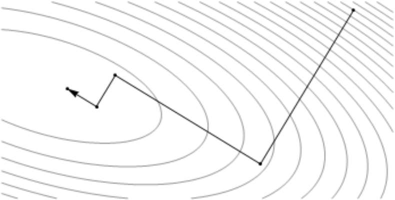
$$\hat{\boldsymbol{a}}_{t+1} = \hat{\boldsymbol{a}}_t - \eta_t \nabla f(\hat{\boldsymbol{a}}_t).$$

where η_t is a step size.

Source: `http://www-bcf.usc.edu/~larry/`, `http://takisword.files.wordpress.com`

Gradient descent method

Minimize a function $f(\boldsymbol{a})$ over $\boldsymbol{a} \in \mathbb{R}^d$.



Gradient descent method:

$$\hat{\boldsymbol{a}}_{t+1} = \hat{\boldsymbol{a}}_t - \eta_t \nabla f(\hat{\boldsymbol{a}}_t).$$

where η_t is a step size.

If we have a set of constraints $\boldsymbol{a} \in \mathcal{A}$, after each step we need to project back to $\mathcal{A}$:

$$\hat{\boldsymbol{a}}_{t+1} \leftarrow \arg\min_{\boldsymbol{a} \in \mathcal{A}} \|\hat{\boldsymbol{a}}_{t+1} - \boldsymbol{a}\|^2.$$

Source: http://www-bcf.usc.edu/~larry/, http://takisword.files.wordpress.com

Online (stochastic) gradient descent

Algorithm

Start with any initial vector $\hat{\boldsymbol{a}}_1 \in \mathcal{A}$.

For $t = 1, 2, \ldots$:

1. Algorithm picks an action $\hat{\boldsymbol{a}}_t$
2. Loss function $\ell_t \colon \mathcal{A} \to \mathbb{R}$ is revealed and algorithm suffers loss $\ell_t(\hat{\boldsymbol{a}}_t)$
3. Algorithm updates its action:
$$\hat{\boldsymbol{a}}_{t+1} = \hat{\boldsymbol{a}}_t - \eta_t \nabla \ell_t(\hat{\boldsymbol{a}}_t).$$
4. If $\hat{\boldsymbol{a}}_{t+1} \notin \mathcal{A}$, project it back to $\mathcal{A}$:
$$\hat{\boldsymbol{a}}_{t+1} \leftarrow \min_{\boldsymbol{a} \in \mathcal{A}} \|\hat{\boldsymbol{a}}_{t+1} - \boldsymbol{a}\|^2.$$

Online vs. standard gradient descent

The function we want to minimize:

$$f(\boldsymbol{a}) = \sum_{t=1}^{n} \ell_t(\boldsymbol{a}).$$

Standard = batch GD

$$\begin{aligned}\hat{\boldsymbol{a}}_{t+1} &:= \hat{\boldsymbol{a}}_t - \eta_t \nabla f(\hat{\boldsymbol{a}}_t) \\ &= \hat{\boldsymbol{a}}_t - \eta_t \sum_{j=1}^{n} \nabla \ell_j(\hat{\boldsymbol{a}}_t)\end{aligned}$$

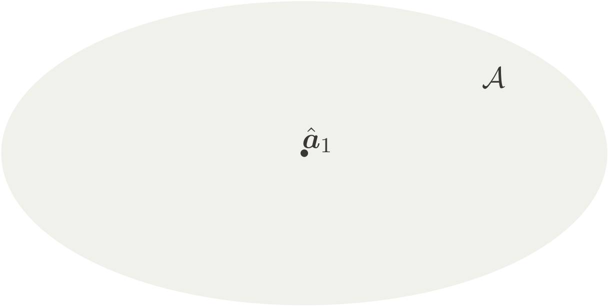
$O(n)$ per iteration, need to see all data.

Online GD

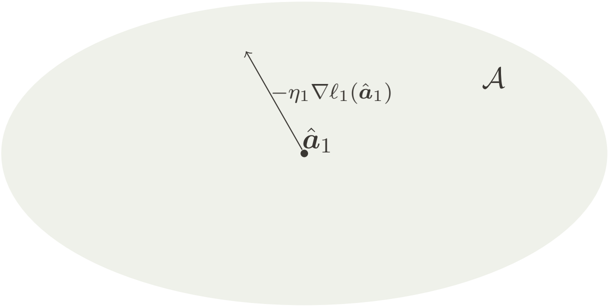
$$\hat{\boldsymbol{a}}_{t+1} := \hat{\boldsymbol{a}}_t - \eta_t \nabla \ell_t(\hat{\boldsymbol{a}}_t).$$

$O(1)$ per iteration, need to see a single data point.

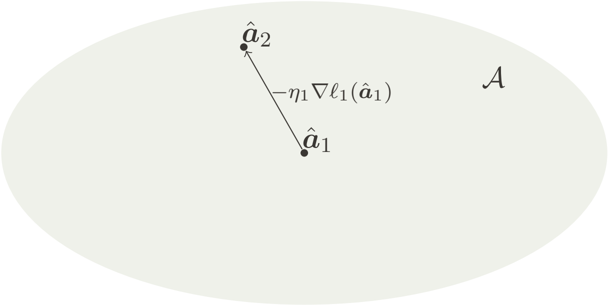
Online (stochastic) gradient descent



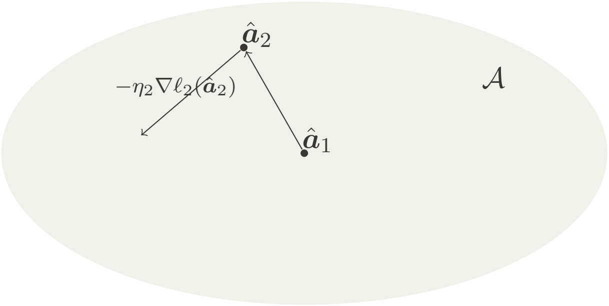
Online (stochastic) gradient descent



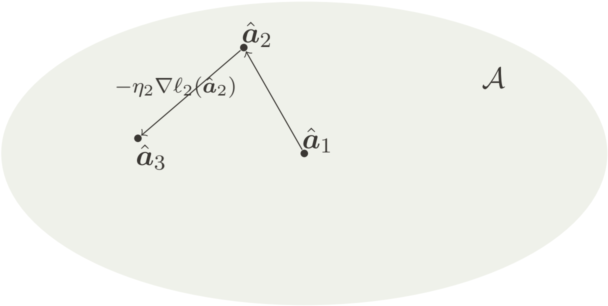
Online (stochastic) gradient descent



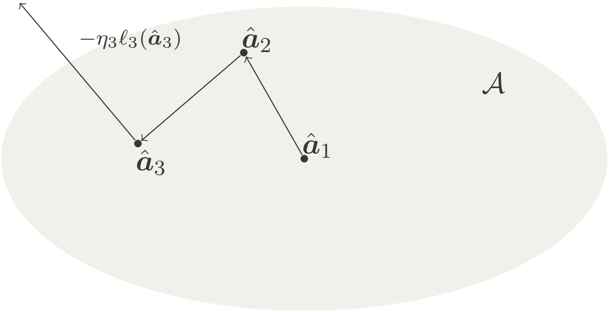
Online (stochastic) gradient descent



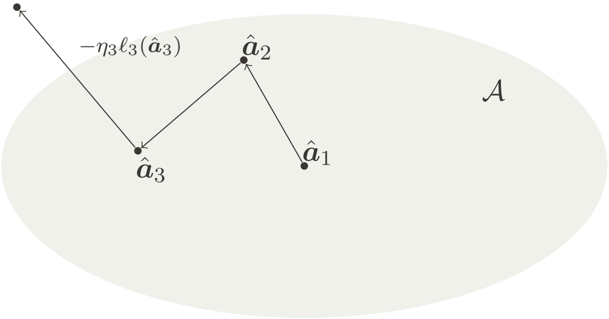
Online (stochastic) gradient descent



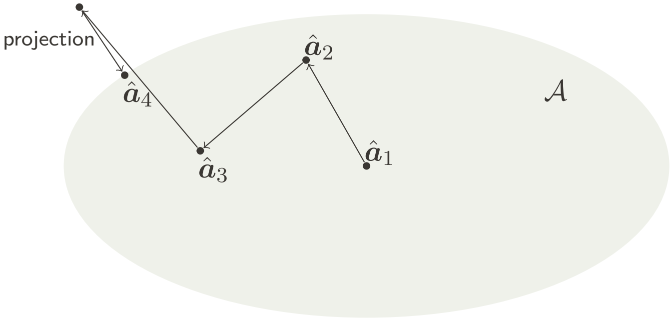
Online (stochastic) gradient descent



Online (stochastic) gradient descent



Online (stochastic) gradient descent



Calculating the gradient

- The gradient $\nabla \ell_t(\boldsymbol{a})$ can be obtained by applying a chain rule to $\ell_t(\boldsymbol{a}) = \ell(y_t, \hat{y})$ with $\hat{y} = \boldsymbol{a}^\top \boldsymbol{x}_t$:

$$\begin{aligned}\nabla \ell_t(\boldsymbol{a}) &= \frac{\partial \ell(y_t, \hat{y})}{\partial \hat{y}} \nabla(\boldsymbol{a}^\top \boldsymbol{x}_t) \\ &= \frac{\partial \ell(y_t, \hat{y})}{\partial \hat{y}} \boldsymbol{x}_t.\end{aligned}$$

Update rules for specific losses

- Linear regression:

$$\ell(y, \hat{y}) = (y - \hat{y})^2 \qquad \frac{\partial \ell(y, \hat{y})}{\partial \hat{y}} = -2(y - \hat{y})$$

Update:

$$\hat{\boldsymbol{a}}_{t+1} = \hat{\boldsymbol{a}}_t + 2\eta_t(y_t - \hat{y}_t)\boldsymbol{x}_t.$$

Update rules for specific losses

- Linear regression:

$$\ell(y, \hat{y}) = (y - \hat{y})^2 \qquad \frac{\partial \ell(y, \hat{y})}{\partial \hat{y}} = -2(y - \hat{y})$$

Update:

$$\hat{\boldsymbol{a}}_{t+1} = \hat{\boldsymbol{a}}_t + 2\eta_t (y_t - \hat{y}_t)\boldsymbol{x}_t.$$

- Logistic regression:

$$\ell(y, \hat{y}) = \ln\left(1 + e^{-y\hat{y}}\right) \qquad \frac{\partial \ell(y, \hat{y})}{\partial \hat{y}} = -\frac{y}{1 + e^{y\hat{y}}}$$

Update:

$$\hat{\boldsymbol{a}}_{t+1} = \hat{\boldsymbol{a}}_t + \eta_t \frac{y_t \boldsymbol{x}_t}{1 + e^{y_t \hat{y}_t}}.$$

Update rules for specific losses

- Linear regression:

$$\ell(y,\hat{y}) = (y-\hat{y})^2 \qquad \frac{\partial \ell(y,\hat{y})}{\partial \hat{y}} = -2(y-\hat{y})$$

Update:

$$\hat{\boldsymbol{a}}_{t+1} = \hat{\boldsymbol{a}}_t + 2\eta_t(y_t - \hat{y}_t)\boldsymbol{x}_t.$$

- Logistic regression:

$$\ell(y,\hat{y}) = \ln\left(1+e^{-y\hat{y}}\right) \qquad \frac{\partial \ell(y,\hat{y})}{\partial \hat{y}} = -\frac{y}{1+e^{y\hat{y}}}$$

Update:

$$\hat{\boldsymbol{a}}_{t+1} = \hat{\boldsymbol{a}}_t + \eta_t \frac{y_t \boldsymbol{x}_t}{1+e^{y_t \hat{y}_t}}.$$

- Support vector machines:

$$\ell(y,\hat{y}) = (1-y\hat{y})_+ \qquad \frac{\partial \ell(y,\hat{y})}{\partial \hat{y}} = \begin{cases} 0 & \text{if} \quad y\hat{y} > 1 \\ -y & \text{if} \quad y\hat{y} \le 1 \end{cases}$$

Update:

$$\hat{\boldsymbol{a}}_{t+1} = \hat{\boldsymbol{a}}_t + \eta_t \mathbb{1}[y_t\hat{y}_t \le 1] y_t \boldsymbol{x}_t$$

Update rules for specific losses

- Linear regression:

$$\ell(y,\hat{y}) = (y-\hat{y})^2 \qquad \frac{\partial \ell(y,\hat{y})}{\partial \hat{y}} = -2(y-\hat{y})$$

Update:

$$\hat{\boldsymbol{a}}_{t+1} = \hat{\boldsymbol{a}}_t + 2\eta_t(y_t - \hat{y}_t)\boldsymbol{x}_t.$$

- Logistic regression:

$$\ell(y,\hat{y}) = \ln\left(1+e^{-y\hat{y}}\right) \qquad \frac{\partial \ell(y,\hat{y})}{\partial \hat{y}} = -\frac{y}{1+e^{y\hat{y}}}$$

Update:

$$\hat{\boldsymbol{a}}_{t+1} = \hat{\boldsymbol{a}}_t + \eta_t \frac{y_t \boldsymbol{x}_t}{1+e^{y_t\hat{y}_t}}.$$

- Support vector machines:

$$\ell(y,\hat{y}) = (1-y\hat{y})_+ \qquad \frac{\partial \ell(y,\hat{y})}{\partial \hat{y}} = \begin{cases} 0 & \text{if} \quad y\hat{y} > 1 \\ -y & \text{if} \quad y\hat{y} \le 1 \end{cases}$$

Update:

$$\hat{\boldsymbol{a}}_{t+1} = \hat{\boldsymbol{a}}_t + \eta_t \mathbb{1}[y_t\hat{y}_t \le 1] y_t \boldsymbol{x}_t \qquad \Leftarrow \text{perceptron!}$$

Projection

$$\hat{\boldsymbol{a}}_t \leftarrow \arg\min_{\boldsymbol{a} \in \mathcal{A}} \|\hat{\boldsymbol{a}}_t - \boldsymbol{a}\|^2.$$

Projection

$$\hat{\boldsymbol{a}}_t \leftarrow \arg\min_{\boldsymbol{a}\in\mathcal{A}} \|\hat{\boldsymbol{a}}_t - \boldsymbol{a}\|^2.$$

- When $\mathcal{A} = \mathbb{R}^d \Rightarrow$ no projection step.

Projection

$$\hat{\boldsymbol{a}}_t \leftarrow \arg\min_{\boldsymbol{a} \in \mathcal{A}} \|\hat{\boldsymbol{a}}_t - \boldsymbol{a}\|^2.$$

- When $\mathcal{A} = \mathbb{R}^d \Rightarrow$ no projection step.
- When $\mathcal{A} = \{\boldsymbol{a} \colon \|\boldsymbol{a}\| \leq B\}$ is an L_2-ball, projection corresponds to renormalization of the weight vector:

$$\text{if} \;\; \|\hat{\boldsymbol{a}}_t\| > B \quad \Longrightarrow \quad \hat{\boldsymbol{a}}_t \leftarrow \frac{B\hat{\boldsymbol{a}}_t}{\|\hat{\boldsymbol{a}}_t\|}.$$

Equivalent to L_2 regularization.

Projection

$$\hat{\boldsymbol{a}}_t \leftarrow \arg\min_{\boldsymbol{a} \in \mathcal{A}} \|\hat{\boldsymbol{a}}_t - \boldsymbol{a}\|^2.$$

- When $\mathcal{A} = \mathbb{R}^d \Rightarrow$ no projection step.
- When $\mathcal{A} = \{\boldsymbol{a} \colon \|\boldsymbol{a}\| \leq B\}$ is an L_2-ball, projection corresponds to renormalization of the weight vector:

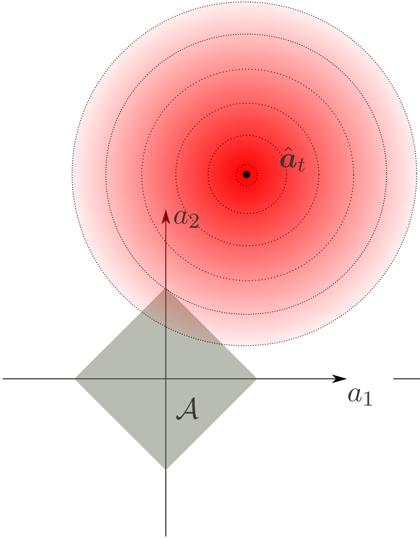
  $$\text{if } \|\hat{\boldsymbol{a}}_t\| > B \quad \Longrightarrow \quad \hat{\boldsymbol{a}}_t \leftarrow \frac{B\hat{\boldsymbol{a}}_t}{\|\hat{\boldsymbol{a}}_t\|}.$$

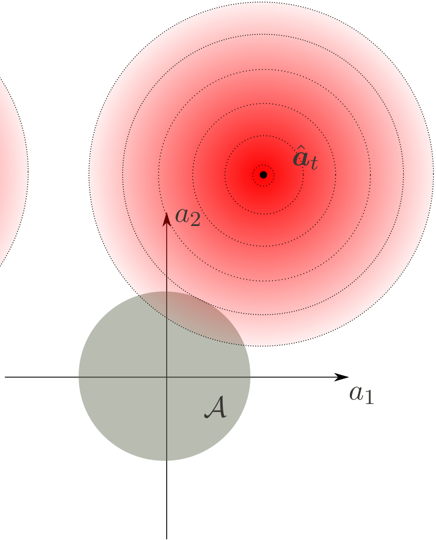
  Equivalent to L_2 regularization.
- When $\mathcal{A} = \{\boldsymbol{a} \colon \sum_{k=1}^d |a_k| \leq B\}$ is L_1-ball, projection corresponds to an additive shift of absolute values and clipping smaller weights to 0.

  Equivalent to L_1 regularization, results in sparse solutions.

L_1 vs. L_2 projection

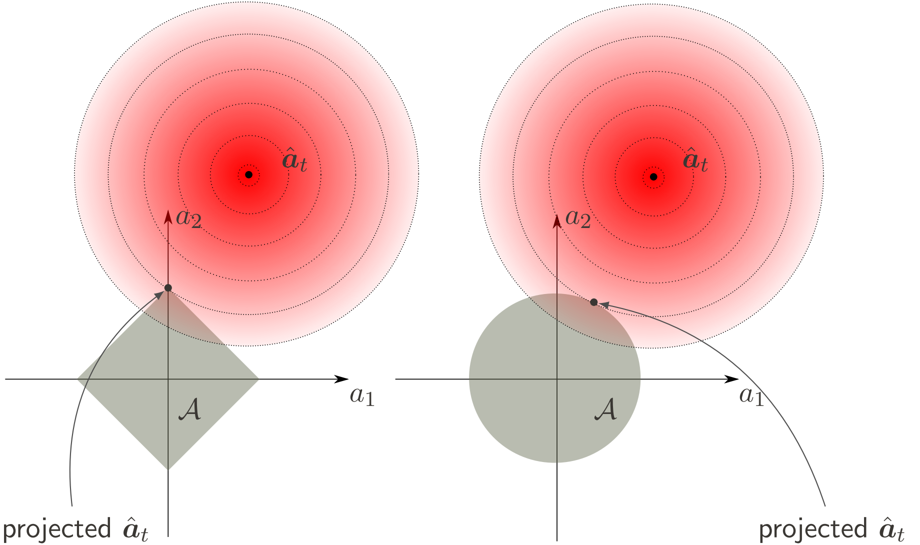
$$\hat{\boldsymbol{a}}_t := \arg\min_{\boldsymbol{a}\in\mathcal{A}} \|\hat{\boldsymbol{a}}_t - \boldsymbol{a}\|^2.$$





L_1 vs. L_2 projection

$$\hat{\boldsymbol{a}}_t := \arg\min_{\boldsymbol{a}\in\mathcal{A}} \|\hat{\boldsymbol{a}}_t - \boldsymbol{a}\|^2.$$



Convergence of online gradient descent

Theorem

Assume $\|\nabla \ell_t(\boldsymbol{a})\| \leq L$ for all t, and let $\|\mathcal{A}\| = \max_{\boldsymbol{a},\boldsymbol{a}' \in \mathcal{A}} \|\boldsymbol{a} - \boldsymbol{a}'\|$.

Then when $\eta_t = \frac{1}{\sqrt{t}} \frac{\|\mathcal{A}\|}{L}$ the regret is bounded by:

$$R_n \leq \frac{3}{2} \|\mathcal{A}\| L \sqrt{n},$$

Convergence of online gradient descent

Theorem

Assume $\|\nabla \ell_t(\boldsymbol{a})\| \leq L$ for all t, and let $\|\mathcal{A}\| = \max_{\boldsymbol{a},\boldsymbol{a}' \in \mathcal{A}} \|\boldsymbol{a} - \boldsymbol{a}'\|$.

Then when $\eta_t = \frac{1}{\sqrt{t}} \frac{\|\mathcal{A}\|}{L}$ the regret is bounded by:

$$R_n \leq \frac{3}{2} \|\mathcal{A}\| L \sqrt{n},$$

The algorithm can be generalized to online mirror descent, which will work for any pair of dual norms

Exponentiated Gradient [Kivinen & Warmuth 1997]

A special version of online mirror descent

$$\hat{a}_{t+1,k} := \frac{1}{Z_t} \hat{a}_{t,k} e^{-\eta_t (\nabla \ell_t(\hat{\boldsymbol{a}}_t))_k}, \qquad k = 1, \ldots, d$$

- Requires positive weights, but can be applied in a general setting by doubling features.
- Works much better than online gradient descent when:
  - d is very large (many features)
  - only a small number of features is relevant.

Theorem

Let $\mathcal{A} = \Delta^d$. Assume $\|\nabla \ell_t(\boldsymbol{a})\|_\infty \leq L$ for all t.

Then when $\hat{\boldsymbol{a}}_1 = (\frac{1}{d}, \ldots, \frac{1}{d})$, $\eta_t = \frac{1}{\sqrt{t}} \frac{\sqrt{2 \ln d}}{L}$, the regret is bounded by:

$$R_n \leq L\sqrt{2n \ln d}$$

Extensions

- Concept drift: competing with drifting parameter vectors.
- Partial feedback: contextual multi-armed bandit problems.
- Improvements for some (strongly convex, exp-concave) loss functions.
- Infinite-dimensional feature spaces via kernel trick.
- Learning matrix parameters (matrix norm regularization, positive definiteness, permutation matrices).
- . . .

Outline

Conclusions

- A theoretical framework for learning without stochastic assumption.
- Performance bounds match those in the stochastic setting, but often simpler to prove.
- Easy to generalize to changing environments (concept drift), partial information (multiarmed bandits), etc.
- Results in online algorithms directly applicable to large-scale learning problems.
- Most of currently used offline learning algorithms employ online learning as an optimization routine.

References

- Hazan, E.: Introduction to Online Convex Optimization. Foundations and Trends® in Optimization, vol. 2, no. 3-4, pp. 157–325, 2016.
- Shalev-Schwartz, S.: Online Learning and and Online Convex Optimization. Foundations and Trends® in Machine Learning, vol. 4, no. 2, pp. 107–194, 2011.
- Cesa-Bianchi, N., Lugosi, G.: Prediction, Learning and Games. Cambridge University Press, 2006.
- Beck, A., Teboulle, M.: Mirror Descent and Nonlinear Projected Subgradient Methods for Convex Optimization. Operations Research Letters, vol. 31, no. 3, pp. 167–185, 2003.
- Zinkevich, M.: Online Convex Programming and Generalized Infinitesimal Gradient Ascent. International Conference on Machine Learning (ICML), pp. 928-936, AAAI Press, 2003.
- Kivinen, J., Warmuth, M. K.: Exponentiated Gradient Versus Gradient Descent for Linear Predictors. Information and Computation, vol. 132, no. 1, pp. 1–63, 1997.
- Littlestone, N., Warmuth, M. K.: The Weighted Majority Algorithm. Information and Computation, vol. 108, no. 2, pp. 212–261, 1994.
- Blumer A., Ehrenfeucht, A., Haussler, D., Warmuth, M.: Occam's Razor. Information Processing Letters, vol. 24, no. 6, pp. 377–380, 1987.
- Nemirovski, A. S., Yudin, D. B.: Problem Complexity and Efficiency in Optimization. John Wiley and Sons, 1983.
- Vapnik, V. N., Chervonenkis, A. Ya.: On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. Theory of Probability and its Applications, vol. 16, no. 2, pp. 264–280, 1971.