Today’s Class

- Algorithms that predict, learn and optimize in an online manner suited for streams:
  - Act and adapt upon every example
  - Keep very small state
  - Computation per example is small
- The algorithms are all simple and somewhat intuitive
  - Their analysis – less so
- Just a sample from a huge field
Online Learning - Definition

- A model of induction that learns one instance at a time

Typical setting: the algorithm faces a stream of trials

- World presents input to the algorithm
  - Input can be stochastic, adversarial, conforming to some concept in a concept space, etc.
  - Algorithm takes an action/decision with respect to the input
  - World reveals some outcome or label to the algorithm
  - Algorithm adapts internal model based on <input,outcome> pair

- Goal: minimize some error/loss/regret
  - With provable bounds

Learning from Experts - Setting

- We are faced with a binary prediction problem
  - E.g. “will it rain today?”

- To our help come n experts, each providing their own opinions x₁,...,xₙ

- By applying some function f(x₁,...,xₙ), we reach a decision (“I bet it will rain”)

- In time, we learn whether our prediction was correct

- This situation happens repeatedly with the same set of experts

- Our goal: learn a “good” f()
Learning from Experts: Target Function

- Our goal: learn a “good” $f$
  - How do we measure whether $f$ is “good”?  
- This class: worst-case competitive analysis
- At all times $t$, the number of wrong predictions we’ve made should be competitive with the best expert so far
  - The expert that made the fewest prediction mistakes till time $t$
- No assumptions are made about the quality of the experts
  - They may all be making random predictions
  - They may be adversarial
  - The quality of the experts may change arbitrarily over time
- The following algorithms and proofs are due to Littlestone and Warmuth, from 1994

Learning from Experts: Weighted Majority Algorithm

1. $w_1 = w_2 = ... = w_n \leftarrow 1$  // expert weights
2. Foreach prediction problem:
   1. Collect predictions $x_1, ..., x_n$ from the experts
   2. Apply the weighted majority rule:
      - Output 1 iff $\sum_{i:x_i=1} w_i \geq \sum_{j:x_j=0} w_j$
   3. Receive the correct answer $b$
4. Foreach $j$ s.t. $x_j \neq b$:  // foreach wrong expert
   1. $w_j \leftarrow w_j/2$  // reduce weight by half
Weighted Majority Algorithm: Analysis

- Notations:
  - \( M \): # of mistakes made by the weighted majority algorithm
  - \( m \): # of mistakes made by the best expert so far
  - Let \( W \) denote the sum of all weights, \( W = \sum w_i \); initially, \( W = n \)
- Theorem: \( M \leq (\log 4/3)^{-1}(m + \log n) \approx 2.41 (m + \log n) \)
- Proof:
  - Each prediction mistake reduces the weights by at least \( \frac{1}{4} \)
  - Therefore, \( W \leq n (3/4)^m \)
  - At the same time, \( W \geq 2^{-m} \) (the weight of the best expert so far)
  - So \( 2^{-m} \leq n (3/4)^m \), hence \( M \leq (\log 4/3)^{-1}[m+ \log n] \)

Weighted Selection Algorithm: Better, Randomized Variant

1. \( w_1 = w_2 = \ldots = w_n \leftarrow 1 \) // expert weights
2. Foreach prediction problem:
   1. Collect predictions \( x_1, \ldots, x_n \) from the experts
   2. Output \( x_i \) with probability \( w_i/\sum w_j \)
   3. Receive the correct answer \( b \)
   4. Foreach \( j \) s.t. \( x_j \neq b \): // foreach wrong expert
      1. \( w_j \leftarrow w_j \ast \beta \) // multiply weight by \( \beta \)

- \( \beta \in (0,1) \) is a parameter (and the subject of optimization)
Weighted Selection Algorithm: Analysis

- Slightly modified notations:
  - \( M \): expected number of mistakes made by the algorithm
  - \( m \): number of mistakes made by the best expert so far
  - \( W_t \): the sum of weights before \( t \)th prediction (\( W_1 = n \))
  - \( F_i \): fraction of weight on the wrong experts at time \( i \)
- Note that \( M = \sum_i F_i \)
- Theorem: \( M \leq \left[ \ln n - m \ln \beta \right] / (1-\beta) \)

Proof:
\[
W_{t+1} = W_t(1-F_t) + W_tF_t\beta = W_t(1-F_t+F_tF_t) = W_t(1-(1-\beta)F_t)
\]
Therefore, \( W_{t+1} = n \sum_{i=1,...,t} [1-(1-\beta)F_i] \)
- At the same time, \( W_{t+1} \geq \beta^m \) (weight of best expert so far)
- So: \( n \sum_{i=1,...,t} [1-(1-\beta)F_i] \geq \beta^m \)
- \( \ln n + \sum \ln [1-(1-\beta)F_i] \geq m \ln \beta \) \( \) // taking \( \ln \) from both sides
- \( -\ln n - \sum \ln [1-(1-\beta)F_i] \leq -m \ln \beta \) \( \) // negating
- \( -\ln n + (1-\beta)\sum F_i \leq -m \ln \beta \) \( \) // since \( -\ln (1-x) > x \)
- \( -\ln n + (1-\beta)M \leq -m \ln \beta \) \( \) // recall that \( M=\sum F_i \)
- The result follows
Online Learning from Examples of a Concept Class

- Typical scenario is a binary classification task: given input instances, we should classify them as 0 or 1
  - We focus on input n-dimensional Boolean vectors, i.e. \( \{0,1\}^n \)
- Online setting: after making our classification prediction, we are given the true label of the input
- Goal is to learn a “good” function \( f: \{0,1\}^n \rightarrow \{0,1\} \)
  - This class: “good” means with bounded absolute error
- Concept-class learning: when we know the space (family) of functions that the true classification function \( f \) comes from
  - i.e. a subset of all Boolean functions over \( n \) variables

Concept Class of Sparse Monotone Disjunctions

- A Boolean disjunction over \( n \) variables \( (x_1,\ldots,x_n) \):
  \[ x_2 \lor \neg x_5 \lor x_9 \lor x_{11} \lor \neg x_{17} \]
- Monotone disjunction: no negation operators; all variables appear as-is or are not part of disjunction
  \[ x_2 \lor x_5 \lor x_9 \lor x_{11} \lor x_{17} \]
- Sparse disjunctions: number of literals in the disjunction is much smaller than \( n \)
  - Most variables do not participate, i.e. are irrelevant
- The concept class of interest in the next few slides: disjunctions that are both monotone and sparse
Winnow Algorithm for Learning Sparse Monotone Disjunctions

1. \( w_1 = w_2 = ... = w_n \leftarrow 1 \)  // feature weights
2. Foreach input Boolean vector \( x = (x_1, ..., x_n) \):
   1. \( z = (\sum w_i x_i \geq n) \); output \( z \)
   2. Receive the correct label \( y \)
   3. If (\( z = 0 \) and \( y = 1 \))  // false negative
      1. Foreach \( j \) s.t. \( x_j = 1, w_j \leftarrow w_j \times 2 \)
   4. If (\( z = 1 \) and \( y = 0 \))  // false positive
      1. Foreach \( j \) s.t. \( x_j = 1, w_j \leftarrow 0 \)

- Theorem: on a monotone disjunction of \( r \) variables, Winnow makes at most \( 1+2r(1+\log n) \) mistakes
- Note: there are \( 2^{O(r \log n)} \) such disjunctions

Winnow Algorithm: Analysis

- Theorem: on a monotone disjunction of \( r \) variables, Winnow makes at most \( 1+2r(1+\log n) \) mistakes
- Proof:
  - Let \( R \) denote the set of the \( r \) variables in the disjunction
  - The weight of any \( x_j \in R \) never decreases, as \( x_j = 0 \) in any false positive
  - Once the weight of any \( x_j \in R \) reaches \( n \), it will never participate in a false negative again
  - Any \( x_j \in R \) will be doubled at most \( 1+\log(n) \) times
  - Any false negative doubles the weight of at least one of the \( r \) members of \( R \)
  - Hence, the number of false negatives is bounded by \( r(1+\log n) \)
Winnow Algorithm: Analysis (cont.)

- Theorem: on a monotone disjunction of \( r \) variables, Winnow makes at most \( 1+2r(1+\log n) \) mistakes

Proof (cont.):
- From prev. slide: the number of false negatives \( \leq r(1+\log n) \)
- Let \( W \) denote the sum of all weights, \( W=\sum w_i \)
- Any false negative increases \( W \) by at most \( n \)
- Any false positive decreases \( W \) by at least \( n \)
- \( W \) starts at \( n \), and never becomes negative
- Hence, number of false positives \( \leq 1 + \) number of false negatives
- Conclusion: number of mistakes \( \leq 1+2r(1+\log n) \)

Concept Class of Linearly Separable Sets of Points in \( \mathbb{R}^d \)

- Let \( P \) (=positive) and \( N \) (=negative) be two sets of points in \( \mathbb{R}^d \).
- \( P \) and \( N \) are called linearly separable if there exists a weights vector \( w \in \mathbb{R}^d \) such that:
  - For all \( x \in P \), \( \langle w, x \rangle > 0 \)
  - For all \( x \in N \), \( \langle w, x \rangle < 0 \)
- The hyperplane \( \langle w, x \rangle = 0 \) separates the two sets
- The vector \( w \) is perpendicular to the separating hyperplane
Online Learning a Separating Plane

- We will receive, in streaming fashion, vectors in $\mathbb{R}^d$
- At each time $t$, we will hold a vector $w_t$ and decide whether $x_t$ belongs to $P$ or $N$ according to $\text{sign}(\langle w_t, x_t \rangle)$
- We will get the correct association of $x$, and if wrong, we will update $w$
- Goal: do not make too many mistakes

Preliminaries

- Let $w^{**}$ be a separating vector, and let $P$ and $N$ be finite sets of points
- Since $P$, $N$ are finite, there is some minimal $\epsilon$ such that
  - For all $x \in P$, $\langle w^{**}, x \rangle \geq \epsilon$
  - For all $x \in N$, $\langle w^{**}, x \rangle \leq \epsilon$
- Since any scaling of $w^{**}$ is also a separating vector, w.l.o.g. we can assume that $w^*$ is the smallest-norm vector satisfying:
  - For all $x \in P$, $\langle w^*, x \rangle \geq 1$
  - For all $x \in N$, $\langle w^*, x \rangle \leq 1$
The Perceptron Algorithm

1. \( w_0 \leftarrow 0, \ t \leftarrow 0 \) // initialization
2. Foreach input vector \( x \in \mathbb{R}^d \):
   1. \( z = \text{sign}(\langle w, x \rangle) \); output \( z \)
   2. Receive the correct label \( y \in \{-1, +1\} \)
   3. If \( (z = y) \) // mistake
      1. \( w_{t+1} = w_t + y \times x \) // add/subtract \( x \) to \( w \)
      2. \( t++ \)

Let \( m \) denote the total number of mistakes made
In the next slides, we will prove that \( m \) cannot grow too much – the number of mistakes is bounded

The Perceptron Algorithm - Analysis

Observations:
- The cosine of \( w^* \) and \( w_m \): \( \langle w^*/||w^*||, w_m/||w_m|| \rangle \geq 1 \)
- Let \( x_t \) be the example of the \( t+1 \) error. Then,
  - \( \langle w_{t+1}, w^* \rangle = \langle w_t + y_t x_t, w^* \rangle = \langle w_t, w^* \rangle + y_t \langle x_t, w^* \rangle \geq \langle w_t, w^* \rangle + 1 \)
  - Hence, \( \langle w_m, w^* \rangle \geq m \)
- \( ||w_{t+1}||^2 = ||w_t||^2 + 2 y_t \langle x_t, w_t \rangle + ||x_t||^2 \leq ||w_t||^2 + ||x_t||^2 \)
  - Denote by \( R \) the maximal value of \( ||x|| \) in the input
  - Hence, \( ||w_{t+1}||^2 \leq ||w_t||^2 + R^2 \), so \( ||w_m||^2 \leq mR^2 \)
The Perceptron Algorithm - Analysis

Our three observations:
1. \( \langle w^*/\|w^*\|,w_m/\|w_m\| \rangle \leq 1 \)
2. \( \langle w_m,w^* \rangle \geq m \)
3. \( \|w_m\|^2 \leq mR^2 \)

Together, we reach a bound on \( m \):
- \( m/\|w^*\|R\sqrt{m} \leq \langle w^*/\|w^*\|,w_m/\|w_m\| \rangle \leq 1 \)
- \( m \leq \|w^*\|R\sqrt{m} \)
- \( m \leq \|w^*\|^2R^2 \)

The bound on \( m \) depends on the norm of \( w^* \) and the maximal norm of any input vector \( x \)

---

Stochastic k-Multi-Armed Bandit

(unknown) expected rewards

\( \mu_1 \quad \mu_2 \quad \mu_k \)

Arms pulls are independent

All rewards are in \([0,1]\)

An instance of both Reinforcement Learning (online learning from observed rewards) and Online Optimization

11 June 2017 236620 Big Data Technology
**Target: Regret Minimization**

- A stochastic k-MAB algorithm accrues reward as time goes by.
- How much reward does it “leave on the table” due to not knowing the reward distributions of the arms?
- Definition: the regret of an algorithm till time $t$ is:
  \[
  R(t) = t \mu - \sum_{i=1}^{t} \mu(A_i)
  \]
  Where $\mu$ is the expected reward of the best arm and $A_i$ is the arm pulled by the algorithm at time $i$.

---

**UCB1 Algorithm**

[Auer, Cesa-Bianchi, Fischer; 1998]

1. Foreach $j = 1, ..., k$  // initialization
   1. Play arm $j$
   2. $x_j \leftarrow$ observed reward  // tracks avg. reward
   3. $n_j \leftarrow 1$  // $j$-plays counter
2. $nt \leftarrow k$  // trial counter
3. Foreach trial:
   1. Play arm $j$ that maximizes $x_j + \sqrt{2\ln(nt)/n_j}$
   2. $x_j \leftarrow (n_j \cdot x_j + \text{current trial's reward}) / (n_j + 1)$
   3. $n_j++$
   4. $nt++$
**UCB1 Algorithm: Claim**

- **Notation:** for any non-optimal arm $i$ with expected reward $\mu_i$, $\Delta_i$ is defined as $\mu - \mu_i$
- **Theorem:** The expected regret of UCB on a stochastic $k$-armed bandit instance after $n$ trials is at most
  $$8 \ln(n) \left( \sum_{i: \mu_i(0) < \mu} \Delta_i^{-1} \right) + \left( 1 + \frac{\Pi^2}{3} \right) \left( \sum \Delta_i \right)$$
- **Observe** that the expected regret can be written as
  $$8\ln(n) \times c_1 + c_2,$$
  where $c_1 = O(k/\Delta_{\min})$ and $c_2 = O(k\Delta_{\max})$
- **Algorithm exemplifies** “optimism in the face of uncertainty” principle

---

**$\varepsilon$-Greedy Randomized Policy**

[Auer, Cesa-Bianchi, Fischer; 1998]

- Assume we know some aspect of the hardness of our $k$-MAB instance – specifically, that we know $\Delta_{\min}$, the difference in expectation between the two best arms
- The $\varepsilon$-Greedy randomized policy for playing the arms is:

1. Choose a constant $d \leq \Delta_{\min}$ and let $c$ be a constant
2. Define the sequence $\varepsilon_n = \min\{1, ck/d^2n\}
3. Foreach $n = 1, 2, ...$
   1. Let $j$ be an arm with maximal reward till now; play arm $j$ w.p. $1-\varepsilon_n$ (exploit),
   2. and any other arm $i \neq j$ w.p $\varepsilon_n/(k-1)$ (explore)
ε-Greedy Randomized Policy

1. Choose a constant \(d \leq \Delta_{\min}\) and let \(c\) be a constant
2. Define the sequence \(\varepsilon_n = \min\{1,ck/d^2n\}\)
3. Foreach \(n = 1,2,...\)
   1. Let \(j\) be an arm with maximal reward till now;
      play arm \(j\) w.p. \(1-\varepsilon_n\) (exploit),
      and any other arm \(i \neq j\) w.p. \(\varepsilon_n/(k-1)\) (explore)

   - Whenever \(c > 5\) and whenever \(n \geq ck/d\), the probability of playing a suboptimal arm at time \(n\) is bounded by \(c/nd^2 + o(1/n)\)
   - This bounds the instantaneous regret – the regret at each step
   - Overall regret is logarithmic in \(n\)
   - In practice, typically performs better than UCB1

Many Variants of Stochastic K-MAB

- Mortal Bandits – the set of bandits changes (some die, some are born) at each timestamp
- Sleeping Bandits – the set of bandits is fixed, but some are unavailable (“asleep”) at each timestamp
- Dueling Bandits – pull two arms at each time \(t\), and receive a binary indication of which of them was better
- Contextual Bandits – arm pulls are associated with context, which affects arms’ effectiveness
- ...

11 June 2017 236620 Big Data Technology 27