Registration

Hw2 is out
- Please start working on it as soon as possible
- Come to sections with questions

On Thursday we will have two lectures:
- Usual one, 12:30-11:45
- An additional one, 5pm-6:15pm; 1320 DCL
Projects

Projects proposals are due on Friday 10/16/15
Within a week we will give you an approval to continue with your project along with comments and/or a request to modify/augment/do a different project. There will also be a mechanism for peer comments.
We allow team projects – a team can be up to 3 people.

Please start thinking and working on the project now.
Your proposal is limited to 1-2 pages, but needs to include references and, ideally, some of the ideas you have developed in the direction of the project (maybe even some preliminary results).
Any project that has a significant Machine Learning component is good.
You can do experimental work, theoretical work, a combination of both or a critical survey of results in some specialized topic.
The work has to include some reading. Even if you do not do a survey, you must read (at least) two related papers or book chapters and relate your work to it.
Originality is not mandatory but is encouraged.
Try to make it interesting!
Examples

- **KDD Cup 2013:**
  - "Author-Paper Identification": given an author and a small set of papers, we are asked to identify which papers are really written by the author.
  - “Author disambiguation”: given a list of authors, we are asked to de-duplicate it or cluster them, so that the strings refer to the same author are in the same cluster.

- Adapt and NLP program to a new domain
- Work on making learned hypothesis (e.g., linear threshold functions) more comprehensible (medical domain example)
- Develop a (multi-modal) People Identifier
- Compare Regularization methods: e.g., Winnow vs. L1 Regularization
- Large scale clustering of documents + name the cluster
- Deep Networks: convert a state of the art NLP program to a deep network, efficient, architecture.
- Try to prove something
A Guide

- Learning Algorithms
  - Search: (Stochastic) Gradient Descent with LMS
  - Decision Trees & Rules
- Importance of hypothesis space (representation)
- How are we doing?
  - Simplest: Quantification in terms of cumulative # of mistakes
  - More later
- Perceptron
  - How to deal better with large features spaces & sparsity?
    - Winnow
  - Variations of Perceptron
    - Dealing with overfitting
  - Closing the loop: Back to Gradient Descent
  - Dual Representations & Kernels
- Beyond Binary Classification?
  - Multi-class classification and Structured Prediction
- More general way to quantify learning performance (PAC)
  - New Algorithms (SVM, Boosting)

Today:
Take a more general perspective and think more about learning, learning protocols, quantifying performance, etc. This will motivate some of the ideas we will see next.
Quantifying Performance

- We want to be able to say something rigorous about the performance of our learning algorithm.

- We will concentrate on discussing the number of examples one needs to see before we can say that our learned hypothesis is good.
There is a hidden (monotone) conjunction the learner (you) is to learn

\[ f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \]

How many examples are needed to learn it? How?

- Protocol I: The learner proposes instances as queries to the teacher
- Protocol II: The teacher (who knows f) provides training examples
- Protocol III: Some random source (e.g., Nature) provides training examples; the Teacher (Nature) provides the labels (f(x))
Learning Conjunctions

- Protocol I: The learner proposes instances as queries to the teacher.

- Since we know we are after a monotone conjunction:
  - Is \( x_{100} \) in? \(<1,1,1...,1,0>, ?> \, f(x)=0 \) (conclusion: Yes)
  - Is \( x_{99} \) in? \(<1,1,...1,0,1>, ?> \, f(x)=1 \) (conclusion: No)
  - Is \( x_{1} \) in? \(<0,1,...1,1,1>, ?> \, f(x)=1 \) (conclusion: No)

- A straightforward algorithm requires \( n=100 \) queries, and will produce as a result the hidden conjunction (exactly).

\[
h = x_2 \land x_3 \land x_4 \land x_5 \land x_{100}
\]

What happens here if the conjunction is not known to be monotone? If we know of a positive example, the same algorithm works.
Protocol II: The teacher (who knows $f$) provides training examples
Learning Conjunctions

- Protocol II: The teacher (who knows f) provides training examples
- \((0,1,1,1,1,0,\ldots,0,1), 1\)
Protocol II: The teacher (who knows $f$) provides training examples

$(0,1,1,1,1,0,...,0,1), 1$ (We learned a superset of the good variables)
Learning Conjunctions

- Protocol II: The teacher (who knows f) provides training examples
- \(<(0,1,1,1,1,0,\ldots,0,1), 1>\) (We learned a superset of the good variables)
- To show you that all these variables are required...
Learning Conjunctions

- Protocol II: The teacher (who knows $f$) provides training examples
- $<(0,1,1,1,1,0,\ldots,0,1), 1>$ (We learned a superset of the good variables)
- To show you that all these variables are required...
  - $<(0,0,1,1,1,0,\ldots,0,1), 0>$ need $x_2$
  - $<(0,1,0,1,1,0,\ldots,0,1), 0>$ need $x_3$
  - ..... 
  - $<(0,1,1,1,1,0,\ldots,0,0), 0>$ need $x_{100}$

- A straightforward algorithm requires $k = 6$ examples to produce the hidden conjunction (exactly).

$$f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100}$$
Learning Conjunctions

- Protocol III: Some random source (e.g., Nature) provides training examples
- Teacher (Nature) provides the labels \( f(x) \)
  - \(<(1,1,1,1,1,...,1,1), 1>\)
  - \(<(1,1,1,0,0,0,...,0,0), 0>\)
  - \(<(1,1,1,1,1,0,...,0,1,1), 1>\)
  - \(<(1,0,1,1,1,0,...0,1,1), 0>\)
  - \(<(1,1,1,1,1,0,...0,0,1), 1>\)
  - \(<(1,0,1,0,0,0,...0,1,1), 0>\)
  - \(<(1,1,1,1,1,...,0,1), 1>\)
  - \(<(0,1,0,1,0,0,...0,1,1), 0>\)
Learning Conjunctions

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels (f(x))

- Algorithm: Elimination
  - \<(1,1,1,1,1,...,1,1), 1>  
  - \<(1,1,1,0,0,...,0,0), 0>  
  - \<(1,1,1,1,0,...0,1,1), 1>  
  - \<(1,0,1,1,0,...0,0,1), 0>  
  - \<(1,1,1,1,1,0,...0,0,1), 1>  
  - \<(1,0,1,0,0,...0,1,1), 0>  
  - \<(1,1,1,1,1,...,0,1), 1>  
  - \<(0,1,0,1,0,...0,1,1), 0>  

  Final hypothesis:
  \[ h = x_1 \land x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \]

- With the given data, we only learned an “approximation” to the true concept
Two Directions

Can continue to analyze the probabilistic intuition:

- Never saw $x_1=0$ in positive examples, maybe we’ll never see it?
- And if we will, it will be with small probability, so the concepts we learn may be pretty good
- Good: in terms of performance on future data
- PAC framework

Mistake Driven Learning algorithms

- Update your hypothesis only when you make mistakes
- Good: in terms of how many mistakes you make before you stop, happy with your hypothesis.
- Note: not all on-line algorithms are mistake driven, so performance measure could be different.
On-Line Learning

- Two new learning algorithms
  (learn a linear function over the feature space)
  - Perceptron (+ many variations)
  - Winnow
  - General Gradient Descent view

- Issues:
  - Importance of Representation
  - Complexity of Learning
  - Idea of Kernel Based Methods
  - More about features
Motivation

Consider a learning problem in a very high dimensional space \( \{x_1, x_2, x_3, \ldots, x_{1000000}\} \)

And assume that the function space is very sparse (every function of interest depends on a small number of attributes.)

\[ f = x_2 \land x_3 \land x_4 \land x_5 \land \ldots x_{100} \]

Middle Eastern deserts are known for their sweetness

Can we develop an algorithm that depends only weakly on the space dimensionality and mostly on the number of relevant attributes?

How should we represent the hypothesis?
On-Line Learning

- Of general interest; simple and intuitive model;
- Robot in an assembly line, language learning,…

- Important in the case of very large data sets, when the data cannot fit memory – Streaming data

- Evaluation: We will try to make the smallest number of mistakes in the long run.
  - What is the relation to the “real” goal?
  - Generate a hypothesis that does well on previously unseen data
On-Line Learning

- Model:
  - Instance space: $X$ (dimensionality – $n$)
  - Target: $f: X \rightarrow \{0,1\}$, $f \in C$, concept class (parameterized by $n$)

- Protocol:
  - learner is given $x \in X$
  - learner predicts $h(x)$, and is then given $f(x)$ (feedback)

- Performance: learner makes a mistake when $h(x) \neq f(x)$
  - number of mistakes algorithm $A$ makes on sequence $S$ of examples, for the target function $f$.

$$M_A(C) = \max_{f \in C,S} M_A(f,S)$$

- $A$ is a mistake bound algorithm for the concept class $C$, if $MA(c)$ is a polynomial in $n$, the complexity parameter of the target concept.
On-Line/Mistake Bound Learning

- We could ask: how many mistakes to get to $\epsilon$-$\delta$ (PAC) behavior?
  - Instead, looking for exact learning. (easier to analyze)
- No notion of distribution; a worst case model
- Memory: get example, update hypothesis, get rid of it (??)
On-Line/Mistake Bound Learning

- We could ask: how many mistakes to get to $\epsilon$-$\delta$ (PAC) behavior
  - Instead, looking for exact learning. (easier to analyze)
- No notion of distribution; a worst case model
- Memory: get example, update hypothesis, get rid of it (??)

Drawbacks:
- Too simple
- Global behavior: not clear when will the mistakes be made
On-Line/Mistake Bound Learning

- We could ask: how many mistakes to get to $\epsilon-\delta$ (PAC) behavior
  - Instead, looking for exact learning. (easier to analyze)
- No notion of distribution; a worst case model
- Memory: get example, update hypothesis, get rid of it (??)

Drawbacks:
- Too simple
- Global behavior: not clear when will the mistakes be made

Advantages:
- Simple
- Many issues arise already in this setting
- Generic conversion to other learning models
- “Equivalent” to PAC for “natural” problems (?)
Is it clear that we can bound the number of mistakes?

Let $C$ be a finite concept class. Learn $f \in C$

**CON:**
- In the $i$th stage of the algorithm:
- $C_i$ all concepts in $C$ consistent with all $i-1$ previously seen examples
- Choose randomly $f \in C_i$ and use to predict the next example
- Clearly, $C_{i+1} \subseteq C_i$ and, if a mistake is made on the $i$th example, then $|C_{i+1}| < |C_i|$ so progress is made.

The CON algorithm makes at most $|C|-1$ mistakes.

Can we do better?
The Halving Algorithm

- Let C be a concept class. Learn $f \in C$
- Halving:
  - In the $i$th stage of the algorithm:
    - $C_i$ all concepts in C consistent with all i-1 previously seen examples
  - Given an example $e_i$ consider the value $f_j(e_i)$ for all $f_j \in C_i$ and predict by majority.
Let $C$ be a concept class. Learn $f \in C$

**Halving:**

In the $i$th stage of the algorithm:

- $C_i$ all concepts in $C$ consistent with all $i-1$ previously seen examples

Given an example $e_i$ consider the value $f_j(e_i)$ for all $f_j \in C_i$ and predict by majority.

Predict 1 if $|\{f_j \in C_i; f_j(e_i) = 0\}| < |\{f_j \in C_i; f_j(e_i) = 1\}|$
Let C be a concept class. Learn \( f \in C \)

**Halving:**

In the \( i \)th stage of the algorithm:

- \( C_i \) all concepts in C consistent with all \( i-1 \) previously seen examples

Given an example \( e_i \) consider the value \( f_j(e_i) \) for all \( f_j \in C_i \) and predict by majority.

Predict 1 if \(| \{ f_j \in C_i ; f_j(e_i) = 0 \} | < | \{ f_j \in C_i ; f_j(e_i) = 1 \} | \)

Clearly \( C_{i+1} \subseteq C_i \) and if a mistake is made in the \( i \)th example, then \(| C_{i+1} | < \frac{1}{2} | C_i | \)

The Halving algorithm makes at most \( \log(|C|) \) mistakes
The Halving Algorithm

- Hard to compute
- In some cases Halving is optimal (C - class of all Boolean functions)
- In general, to be optimal, instead of guessing in accordance with the majority of the valid concepts, we should guess according to the concept group that gives the least number of expected mistakes (even harder to compute)
Learning Conjunctions

- There is a hidden conjunctions the learner is to learn
  \[ f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \]
- The number of conjunctions: \( 3^n \)
- \( \log(|\mathcal{C}|) = n \)
- The elimination algorithm makes \( n \) mistakes
  - Learn from positive examples; eliminate active literals.
- \( k \)-conjunctions:
  - Assume that only \( k \ll n \) attributes occur in the disjunction
- The number of \( k \)-conjunctions: \( 2^k C(n, k) \approx 2^k n^k \)
  - \( \log(|\mathcal{C}|) = k \log n \)
  - Can we learn efficiently with this number of mistakes?
Assume that you want to learn conjunctions. Should your hypothesis space be the class of conjunctions?

- **Theorem:** Given a sample on \( n \) attributes that is consistent with a conjunctive concept, it is NP-hard to find a pure conjunctive hypothesis that is both consistent with the sample and has the minimum number of attributes.
  - [David Haussler, AIJ’88: “Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework”]

- Same holds for Disjunctions.
- Intuition: Reduction to minimum set cover problem.
  - Given a collection of sets that cover \( X \), define a set of examples so that learning the best (dis/con)junction implies a minimal cover.

- Consequently, we cannot learn the concept efficiently as a (dis/con)junction.
- But, we will see that we can do that, if we are willing to learn the concept as a Linear Threshold function.
- In a more expressive class, the search for a good hypothesis sometimes becomes combinatorially easier.
Linear Functions

\[ f(x) = \begin{cases} 
1 & \text{if } W_1 x_1 + W_2 x_2 + \ldots + W_n x_n \geq \theta \\
0 & \text{Otherwise} 
\end{cases} \]

- **Disjunctions**
  \[ y = x_1 \lor x_3 \lor x_5 \]
  \[ y = (1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 \geq 1) \]

- **At least m of n**:
  \[ y = \text{at least 2 of } \{x_1, x_3, x_5\} \]
  \[ y = (1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 \geq 2) \]

- **Exclusive-Or**:
  \[ y = (x_1 \land x_2 \lor ) (x_1 \land x_2) \]

- **Non-trivial DNF**
  \[ y = (x_1 \land x_2) \lor (x_3 \land x_4) \]
$w \cdot x = \theta$

$w \cdot x = 0$
Footnote About the Threshold

- On previous slide, Perceptron has no threshold
- But we don’t lose generality:

\[ w \cdot x = \theta \]

\[ x \Leftrightarrow \langle x, 1 \rangle \quad \forall x \]

\[ w \Leftrightarrow \langle w, -\theta \rangle \]

\[ \langle w, -\theta \rangle \cdot \langle x, 1 \rangle = 0 \]
Perceptron learning rule

- On-line, mistake driven algorithm.
- Rosenblatt (1959) suggested that when a target output value is provided for a single neuron with fixed input, it can incrementally change weights and learn to produce the output using the **Perceptron learning rule**
- (Perceptron == Linear Threshold Unit)
## Perceptron learning rule

- We learn \( f : X \rightarrow \{-1, +1\} \) represented as \( f = \text{sgn}(w \cdot x) \)
- \( X = \{0,1\}^n \) or \( X = \mathbb{R}^n \) and \( w \in \mathbb{R}^n \)
- Given Labeled examples: \( \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\} \)

1. Initialize \( w = 0 \in \mathbb{R}^n \)

2. Cycle through all examples
   a. Predict the label of instance \( x \) to be \( y' = \text{sgn}(w \cdot x) \)
   b. If \( y' \neq y \), **update** the weight vector:
      \[
      w = w + r \ y \ x
      \]
      \( (r - \text{a constant, learning rate}) \)
   Otherwise, if \( y' = y \), leave weights unchanged.
The Perceptron rule

- If $y = +1$: $x$ should be above the decision boundary
- Raise the decision boundary’s slope: $w^{i+1} := w^i + x$

- If $y = -1$: $x$ should be below the decision boundary
- Lower the decision boundary’s slope: $w^{i+1} := w^i - x$
Perceptron in action

\[ wx = 0 \]

Current decision boundary

\[ w \]

Current weight vector

\[ x \]

(x with \( y = +1 \)) next item to be classified

\[ w \]

New weight vector

\[ wx = 0 \]

New decision boundary

\[ x \text{ as a vector} \]

\[ w \text{ as a vector added to} \]

\[ x \text{ as a vector} \]

(Figures from Bishop 2006)
Perceptron in action

\[ \mathbf{x} \text{ (with } y = +1) \]

next item to be classified

\[ \mathbf{w} \]

Current weight vector

\[ \mathbf{w} \mathbf{x} = 0 \]

Current decision boundary

\[ \mathbf{x} \text{ as a vector added to } \mathbf{w} \]

\[ \mathbf{w} \]

New weight vector

\[ \mathbf{w} \mathbf{x} = 0 \]

New decision boundary

(Figures from Bishop 2006)
Perceptron learning rule

- If \( x \) is Boolean, only weights of active features are updated.
- Why is this important?

1. Initialize \( \mathbf{w} = 0 \in \mathbb{R}^n \)

2. Cycle through all examples
   - a. Predict the label of instance \( x \) to be \( y' = \text{sgn}\{\mathbf{w} \cdot \mathbf{x}\} \)
   - b. If \( y' \neq y \), update the weight vector to
     \[
     \mathbf{w} = \mathbf{w} + r \ y \ \mathbf{x}
     \] (\( r \) - a constant, learning rate)
   Otherwise, if \( y' = y \), leave weights unchanged.

\[
\mathbf{w} \cdot \mathbf{x} > 0 \text{ is equivalent to } \frac{1}{1 + \exp\{- (\mathbf{w} \cdot \mathbf{x})\}} > 1/2
\]
Perceptron Learnability

- Obviously can’t learn what it can’t represent (???)
  - Only linearly separable functions
- Minsky and Papert (1969) wrote an influential book demonstrating Perceptron’s representational limitations
  - Parity functions can’t be learned (XOR)
  - In vision, if patterns are represented with local features, can’t represent symmetry, connectivity
- Research on Neural Networks stopped for years

Rosenblatt himself (1959) asked,

> “What pattern recognition problems can be transformed so as to become linearly separable?”
\[(X_1 \Lambda X_2) \lor (X_3 \Lambda X_4)\] \hspace{1cm} y_1 \Lambda y_2
Perceptron Convergence

- **Perceptron Convergence Theorem:**
  - If there exist a set of weights that are consistent with the data (i.e., the data is linearly separable), the perceptron learning algorithm will converge.
  - How long would it take to converge?

- **Perceptron Cycling Theorem:**
  - If the training data is not linearly separable the perceptron learning algorithm will eventually repeat the same set of weights and therefore enter an infinite loop.
  - How to provide robustness, more expressivity?