CS446 Introduction to Machine Learning (Fall 2013)
University of Illinois at Urbana-Champaign
http://courses.engr.illinois.edu/cs446

LECTURE 7: ONLINE LEARNING II (THEORETICAL ANALYSIS)

Prof. Julia Hockenmaier
juliahmr@illinois.edu

Thursday's key concepts

More on linear separability

Two new, mistake-driven update rules for learning linear classifiers:

– Perceptron (additive updates)

– Winnow (multiplicative updates)

Today's lecture

Theoretical analysis of online learning algorithms:

– Metric: Mistake bounds
– Setting: Concept learning

Concept learning

Concept learning

The notion of “concept” comes from psychology.

In machine learning:

- Concept = a Boolean-valued function

  Class labels are Boolean: {0, 1} or {false, true}, not {-1, +1}

  Instances are also represented in terms of Boolean variables

- Concept Learning = Inferring a concept from labeled training examples.

Boolean variables and functions

Boolean variables (literals): x can take the value 0 or 1.
- Positive literals: x
- Negative literals: $\neg x$

k-Conjunction: a conjunction of k literals:

$$k = 4: \; x_1 \wedge x_5 \wedge x_7 \wedge \neg x_{10}$$

k-Disjunction: a disjunction of k literals:

$$k = 5: \; x_2 \vee x_3 \vee x_7 \vee \neg x_{10} \vee \neg x_{100}$$

Monotone conjunction/disjunction: only positive literals

$$x_1 \wedge x_5 \wedge x_7 \wedge x_{10} \text{ but not } x_1 \wedge x_5 \wedge x_7 \wedge \neg x_{10}$$

Examples

$$f(\mathbf{x}) = x_1 \wedge x_5 \wedge x_7 \wedge \neg x_{10}$$

$f(\mathbf{x}) = 1$ whenever $x_1=1$ and $x_5=1$ and $x_7=1$ and $x_{10}=0$.

The values of the other variables don't matter.

$$f(\mathbf{x}) = x_3 \vee x_4 \vee x_5$$

$f(\mathbf{x}) = 1$ whenever $x_3=1$ or $x_4=1$ or $x_5=1$.

The values of the other variables don't matter.

Concept learning: Assumptions

We typically assume that the learner knows

– the instance space

(How many Boolean variables are used to represent the input?)

– the class of the target function:

e.g.: monotone conjunctions, or k-disjunctions, etc.

We also typically assume that the training data is noise-free.

Online learning

Online learning:
The learner can update its hypothesis after each training example it sees.

Number of examples needed:

How many training examples will the learner need?

Mistake bounds:

How many mistakes will the learner make before it has learned the target function?

Learning Conjunctions

Task: Learn a monotone conjunction $f(\mathbf{x})$

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- **Protocol I:** The learner proposes instances $\mathbf{x}$ as queries to the teacher, who returns $f(\mathbf{x})$
- **Protocol II:** The teacher provides labeled training examples $(\mathbf{x}, f(\mathbf{x}))$
- **Protocol III:** Some random source (Nature) provides training examples $\mathbf{x}$; the Teacher provides the labels $(f(\mathbf{x}))$

How many examples are needed to learn $f(\mathbf{x})$? How?

Protocol I

At each iteration:

1. The learner proposes an unlabeled instance $\mathbf{x}$ to the teacher
2. The teacher labels the instance (returns $f(\mathbf{x})$)
3. The learner updates its hypothesis

What is the best strategy for proposing examples?

Algorithm for Protocol I

Task: Learn monotone conjunctions

Monotone: No literal is negated

Hence: All target variables have to be true;
the value of irrelevant variables doesn't matter.

Algorithm: Check each variable x_i for i = 0…n

– At each iteration, learner wants to know: Is x_i in f ?

– For i=1, propose $\mathbf{x} = (0,1,1,\ldots,1,1)$ to the teacher

– If the teacher returns $f(\mathbf{x})=0$, x_i is in
If the teacher returns $f(\mathbf{x})=1$, x_i is out

This requires n queries (one per variable),
and will return the hidden conjunction (exactly).

Protocol II (unrealistic)

The teacher provides labeled training examples.

- First iteration: Teacher provides a **positive** example that consists of a superset of the target variables set to 1:

  $\langle(0,1,1,1,1,0,\ldots,0,1),\ 1\rangle$
- In each following iteration, the teacher provides a **negative** example in which one of the target variables is set to 0.
  This tells the learner which variables are required

  $\langle(0,0,1,1,1,0,\ldots,0,1),\ 0\rangle$ x_2 is in

  $\langle(0,1,0,1,1,0,\ldots,0,1),\ 0\rangle$ x_3 is in

This requires k examples to learn a k-conjunction.

Protocol III (more realistic)

Some random source (e.g., Nature) provides labeled training examples:

$\langle(1,1,1,1,1,1,\ldots,1,1), 1\rangle$

$\langle(1,1,1,0,0,0,\ldots,0,0), 0\rangle$

$\langle(1,1,1,1,1,0,\ldots 0,1,1), 1\rangle$

$\langle(1,0,1,1,1,0,\ldots 0,1,1), 0\rangle$

We still assume training data is noise-free

Protocol III: Elimination

Elimination algorithm (for monotone conjunctions)

- Start with $h = x_1 \wedge \ldots \wedge x_n$ (a conjunction over all literals)
- Eliminate every literal from h that is 0 in a positive example

This algorithm

- doesn't learn from negative examples
- might not learn the target hypothesis from the training data

Learning Conjunctions with Elimination

Data

<(1,1,1,1,1,1,...,1,1), 1>

<(1,1,1,0,0,0,...,0,0), 0>

<(1,1,1,1,1,0,...,0,1,1), 1>

<(1,0,1,1,0,0,...0,0,1), 0>

<(1,1,1,1,1,0,...0,0,1), 1>

<(1,0,1,0,0,0,...0,1,1), 0>

<(1,1,1,1,1,1,...,0,1), 1>

<(0,1,0,1,0,0,...0,1,1), 0>

Target function

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

Learned 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 learn an "approximation" of the true concept

Two Directions

Probabilistic intuition (PAC framework)

- 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 = performance on future data

Mistake Driven Learning algorithms

- Update your hypothesis only when you make mistakes
- Good= how many mistakes will you make before you stop, happy with your hypothesis.
- Useful for some online algorithms

Mistake bounds for online learning (worst-case analysis)

Assumptions: *A* is an online learning algorithm:

At each iteration, *A* is given **x** and f(**x**), and predicts h(**x**).

A makes a mistake when h(**x**) ≠ f(**x**)

A knows the target concept class C.

Mistake bound of algorithm *A* on class C:

The mistake bound of algorithm *A* on class C, $M_A(C)$, is the maximum number of mistakes *A* makes on any sequence of examples *S* for any concept $f \in C$.

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

The CON(SISTENT) algorithm

Assumption: The learner knows the target class C (e.g. C = all monotone conjunctions)

At *i*th iteration:

- The learner has a set of hypotheses C_i ($C_0 = C$)
  C_i = all concepts in C consistent with i-1 previous examples
- Choose $h \in C_i$ randomly.
  Keep h if it labels ith example correctly.
  Otherwise, discard h.

$C_{i+1} \subseteq C_i$ and, if a mistake is made, $|C_{i+1}| < |C_i|$

The CON algorithm makes at most $|C|-1$ mistakes to learn f

The Halving Algorithm

In the *i*th stage of the algorithm:

- C_i: all concepts in C consistent with all previous examples
- Given example **x**, compute $h_k(\mathbf{x})$ for all $h_k \in C_i$
- Predict the value predicted by the majority of the h_k
- If the majority prediction is correct, keep the correct h_k and discard the rest. Otherwise, discard the majority.

If the majority vote is a mistake, $| C_{i+1} | < ½| C_i |$

This algorithm makes at most log(|C|) mistakes

Optimal for Boolean functions.

But: each halving iteration is expensive to compute

Choice of representation

If you want to learn conjunctions, should your hypothesis space be the class of conjunctions?

No: We cannot learn conjunctions efficiently *as 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.

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 disjunction implies a minimal cover.

But we can, if we are willing to learn the concept as a linear classifier.

In a more expressive class, the search for a good hypothesis may become combinatorially easier.

Linear classifiers for Boolean functions

Which Boolean functions can be captured by a linear classifier?

Disjunctions: $y = x_j \vee \neg x_k$

Linear classifier: $f(\mathbf{x}) = 1$ iff $\sum_i w_i x_i \geq \theta$

Disjunctions with a linear classifier:

$w_j = 1$ (x_j is a positive literal)

$w_k = -1$ (x_k is a negative literal),

all other $w_i = 0$

$$f(\mathbf{x}) = 1 \text{ iff } \sum_i w_i x_i \geq 1$$

Which Boolean functions can be captured by a linear classifier?

At least *m* of *n*

y = at least 2 of (x_j, x_k, x_l)

At least *m* of *n* with a linear classifier:

$w_j = 1$, $w_k = 1$, $w_l = 1$, and all other $w_k = 0$

$f(\mathbf{x}) = 1$ iff $\sum_i w_i x_i \geq m$

Mistake bounds: Perceptron vs Winnow

Online perceptron

Assumptions: class labels $y \in \{+1, -1\}$;
learning rate $\alpha > 0$

Initial weight vector $\mathbf{w}^0 := (0,\ldots,0)$

$i = 0$

for $m = 0 \ldots M$:

Perceptron rule

if $y_m \cdot f(\mathbf{x_m}) = y_m \cdot \mathbf{w}^i \cdot \mathbf{x_m} < 0$:

($\mathbf{x_m}$ is misclassified – add $\alpha \cdot y_m \cdot \mathbf{x_m}$ to $\mathbf{w}$!)

$\mathbf{w}^{i+1} := \mathbf{w}^i + \alpha \cdot y_m \cdot \mathbf{x_m}$

$i := i+1$

return $\mathbf{w}^{i+1}$ when all examples correctly classified

Perceptron Convergence Theorem
(Block & Novikoff)

Assumptions:

- the data are linearly separable with margin γ (by a unit norm hyperplane $\mathbf{u}$)
- the l_2-norm of the data is bounded by a constant R

We can show that the perceptron algorithm makes at most $k \leq R^2/\gamma^2$ mistakes during training.

Winnow update

if $\mathbf{w}^k$ misclassifies $\mathbf{x}^i$:

if $y^i = +1$:

($\mathbf{x}^i$ should be above the decision boundary, but is currently below it)

double the weights of the features that are active in $\mathbf{x}^i$

if $y^i = -1$:

($\mathbf{x}^i$ should be below the decision boundary, but is currently above it)

halve the weights of the features that are active in $\mathbf{x}^i$

(don't touch weights of inactive features)

Winnow convergence: Learning monotone k-disjunctions

Monotone k-disjunction:
Disjunction of k features (out of n total) in which no feature is negated

Claim:
Winnow makes no more than $O(k \log n)$ mistakes on k-disjunctions before it converges

Winnow for *k*-disjunctions

Initialization:

$\theta = n \quad \mathbf{w} = (1, 1, \ldots, 1)$

Prediction:

return 1 iff $\mathbf{w}\cdot\mathbf{x} \geq \theta$, 0 otherwise

Learning:

if $\mathbf{w}\cdot\mathbf{x} < \theta$ but $y = 1$:

Promote active feature weights: $w_i := 2w_i$

if $\mathbf{w}\cdot\mathbf{x} \geq \theta$ but $y = 0$:

Demote active feature weights: $w_i := w_i/2$

Winnow for k-disjunctions: Mistake bound $O(k \log n)$

u: #mistakes on positive items (promotions): $u < k \log(2n)$

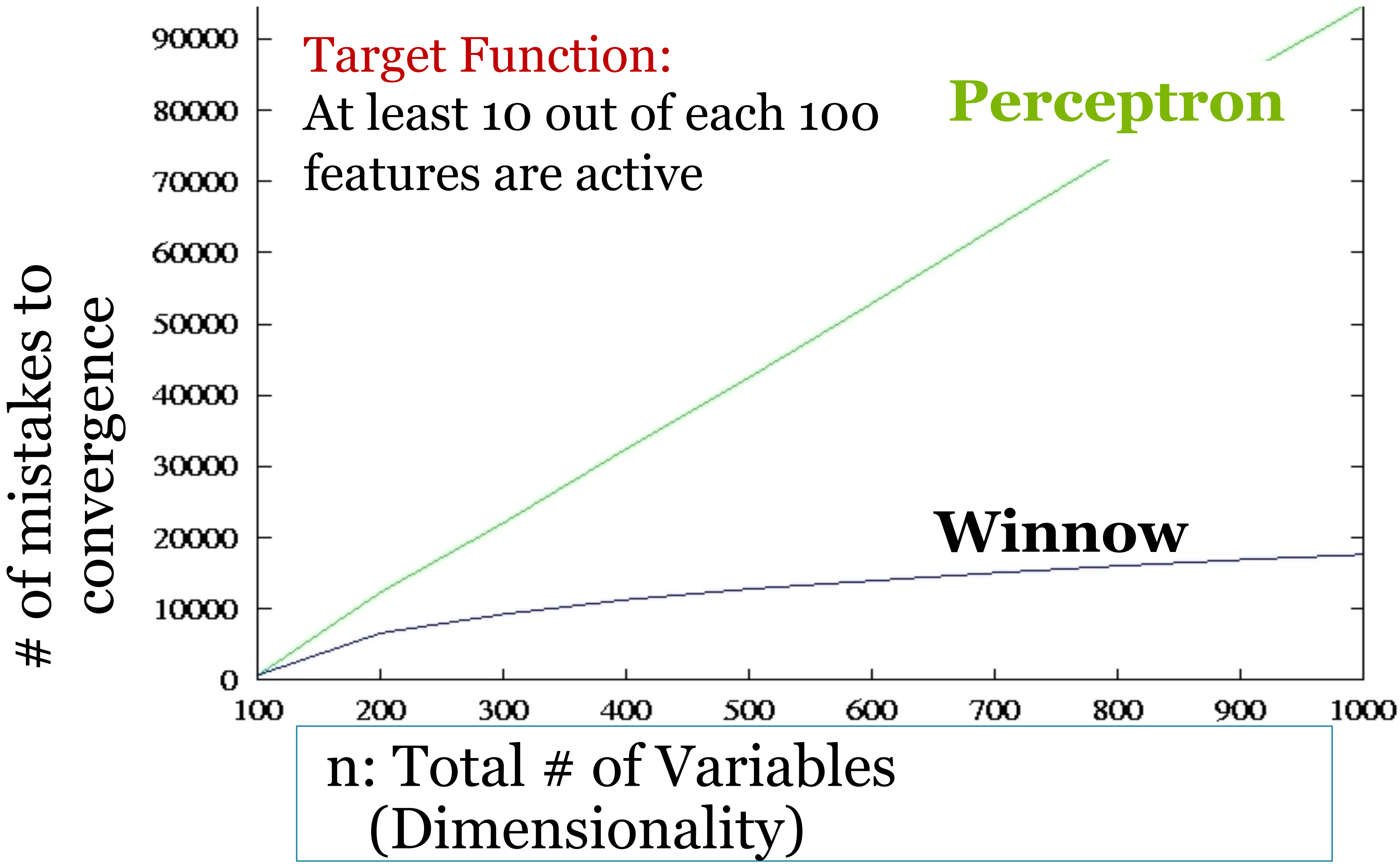
- There are k relevant weights;
- Each weight can only be promoted log(2n) times
- Disjunction: relevant weights cannot be demoted

v: #mistakes on negative items (demotions): $v < 2(u+1)$:

- Only active weights contribute to $\mathbf{w}_k\mathbf{x}$ and are changed
- Define $W = \sum_i w_i$ Initial weight $W = n$
- Mistake on positive item ($\mathbf{w}_k\mathbf{x} < n$): $W_{k+1} < W_k + n$
- Mistake on negative item ($\mathbf{w}_k\mathbf{x} \geq n$): $W_{k+1} < W_k - n/2$
- $0 < W < n + u \cdot n - v \cdot n/2$
  $\Rightarrow 0 < 1 + u - v/2 \Rightarrow v < 2(u+1)$

$u + v < 3u + 2 = O(k \log n)$

Mistakes bounds comparison



Comparing Winnow and Perceptron again

Perceptron: Convergence depends on the L2-norms of the data and the margin

Winnow (general case): Convergence depends on the L1 norm of the decision boundary **w** and on the L∞ norm of the data

Irrelevant features increase the dimensionality of **x**, hence the L2 norm of the data, but not its L∞ norm

How many updates are required?

Mistake bounds: How many mistakes will the learner make before it has converged?

- Multiplicative algorithms (e.g. Winnow)

  Bounds depend on $||\mathbf{u}||_1$, the l_1-norm of the separating hyperplane

  Advantage with few relevant features in concept

- Additive algorithms (e.g. Perceptron)

  Bounds depend on $||\mathbf{x}||$ (Kivinen / Warmuth, '95)

  Advantage with few active features per example

Today's key concepts

Theoretical analysis of online algorithms:
Mistake bounds

Mistake bounds of Winnow and Perceptron