Online Learning & The Perceptron Algorithm

Lecture 6

Machine Learning
Fall 2015

THE UNIVERSITY OF UTAH

Some slides based on lectures from Dan Roth, Avrim Blum and others

Outline

- How good is a learning algorithm?
- Online learning

Generic mistake bound learning

- The Perceptron Algorithm
- Perceptron Mistake Bound
- Variants of Perceptron

Back to linear threshold units

Where are we?

- How good is a learning algorithm?
- Online learning
- The Perceptron Algorithm
- Perceptron Mistake Bound
- Variants of Perceptron

Quantifying Performance

- How can we rigorously quantify the performance of our learning algorithm?

- **One approach**: Compute how many examples should the learning algorithm see before we can say that our learned hypothesis is *good* (or *good enough*)

Example: Learning Conjunctions

There is a hidden (monotone) conjunction for the learner (you) to learn

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

There are 100 Boolean variables. But you don't know that only these ***five*** are relevant

How many examples are needed to learn it? How does learning proceed?

- ***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))

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

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, x_{100} is in f)
  - Is x_{99} in? <(1,1,…1,0,1), ?> f(x)=1 (conclusion: No, x_{99} is not in f)
  - …
  - Is x_2 in ? <(1,0,…1,1,1), ?> f(x)=0 (conclusion: Yes, x_2 is in f)
  - Is x_1 in ? <(0,1,…1,1,1), ?> f(x)=1 (conclusion: No, x_1 is not in f)
- A straight forward algorithm requires n=100 queries, and will produce the hidden conjunction (exactly)

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

What happens here if the conjunction is not known to be monotone?
If we know of a positive example, the same algorithm works.

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

Learning Conjunctions

Protocol II: The teacher (who knows f) provides training examples

- First: Teacher gives a superset of the good variables

  <(0,1,1,1,1,0,…,0,1), 1>

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

Learning Conjunctions

Protocol II: The teacher (who knows f) provides training examples

- First: Teacher gives a superset of the good variables

  <(0,1,1,1,1,0,…,0,1), 1>

- Next: Teacher proves that each of these variables are required
  - <(0,0,1,1,1,0,…,0,1), 0>   need x_2
  - <(0,1,0,1,1,0,…,0,1), 0>   need x_3
  - …
  - <(0,1,1,1,1,0,…,0,0), 0>   need x_{100}

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

Learning Conjunctions

Protocol II: The teacher (who knows f) provides training examples

- First: Teacher gives a superset of the good variables

  <(0,1,1,1,1,0,…,0,1), 1>

These variables are sufficient

- Next: Teacher proves that each of these variables are required
  - <(0,0,1,1,1,0,…,0,1), 0> need x_2
  - <(0,1,0,1,1,0,…,0,1), 0> need x_3
  - …
  - <(0,1,1,1,1,0,…,0,0), 0> need x_{100}

All the variables are necessary

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

Learning Conjunctions

Protocol II: The teacher (who knows f) provides training examples

- First: Teacher gives a superset of the good variables

  <(0,1,1,1,1,0,…,0,1), 1>

- Next: Teacher proves that each of these variables are required
  - <(0,0,1,1,1,0,…,0,1), 0> need x_2
  - <(0,1,0,1,1,0,…,0,1), 0> need x_3
  - …
  - <(0,1,1,1,1,0,…,0,0), 0> need x_{100}

A straight forward algorithm requires k = 6 examples to produce the hidden conjunction (exactly)

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

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

Learning Conjunctions

Protocol II: The teacher (who knows f) provides training examples

- First: Teacher gives a superset of the good variables

  <(0,1,1,1,1,0,…,0,1), 1>

- Next: Teacher proves that each of these variables are required
  - <(0,0,1,1,1,0,…,0,1), 0> need x_2
  - <(0,1,0,1,1,0,…,0,1), 0> need x_3
  - …
  - <(0,1,1,1,1,0,…,0,0), 0> need x_{100}

Modeling teaching can be very difficult, unfortunately

A straight forward algorithm requires k = 6 examples to produce the hidden conjunction (exactly)

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

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

Learning Conjunctions

Protocol III: Some random source (nature) provides training examples

Teacher (Nature) provides the labels (f(x))

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

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

Learning Conjunctions

Protocol III: Some random source (nature) provides training examples

Teacher (Nature) provides the labels (f(x))

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

Look for the variables that are present in *all* positive examples

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

Learning Conjunctions

Protocol III: Some random source (nature) provides training examples

Teacher (Nature) provides the labels (f(x))

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

For a reasonable learning algorithm (by *elimination*), the final hypothesis will be

$$h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

How good is our learning algorithm?

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

Learning Conjunctions

Protocol III: Some random source (nature) provides training examples

Teacher (Nature) provides the labels (f(x))

- <($\boxed{1}$,1,1,1,1,1,…,1,1), $\boxed{1}$>
- <(1,1,1,0,0,0,…,0,0), 0>
- <($\boxed{1}$,1,1,1,1,0,...0,1,1), $\boxed{1}$>
- <(1,0,1,1,1,0,...0,1,1), 0>
- <($\boxed{1}$,1,1,1,1,0,...0,0,1), $\boxed{1}$>
- <(1,0,1,0,0,0,...0,1,1), 0>
- <($\boxed{1}$,1,1,1,1,1,…,0,1), $\boxed{1}$>
- <(0,1,0,1,0,0,...0,1,1), 0>

For a reasonable learning algorithm (by *elimination*), the final hypothesis will be

$$h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Whenever the output is 1, x_1 is present

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

Learning Conjunctions

Protocol III: Some random source (nature) provides training examples

Teacher (Nature) provides the labels (f(x))

- <($\boxed{1}$,1,1,1,1,1,…,1,1), $\boxed{1}$>
- <(1,1,1,0,0,0,…,0,0), 0>
- <($\boxed{1}$,1,1,1,1,0,...0,1,1), $\boxed{1}$>
- <(1,0,1,1,1,0,...0,1,1), 0>
- <($\boxed{1}$,1,1,1,1,0,...0,0,1), $\boxed{1}$>
- <(1,0,1,0,0,0,...0,1,1), 0>
- <($\boxed{1}$,1,1,1,1,1,…,0,1), $\boxed{1}$>
- <(0,1,0,1,0,0,...0,1,1), 0>

For a reasonable learning algorithm (by *elimination*), the final hypothesis will be

$$h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Whenever the output is 1, x_1 is present

With the given data, we only learned an *approximation* to the true concept.
Is it good enough?

Two Directions for How good is our learning algorithm?

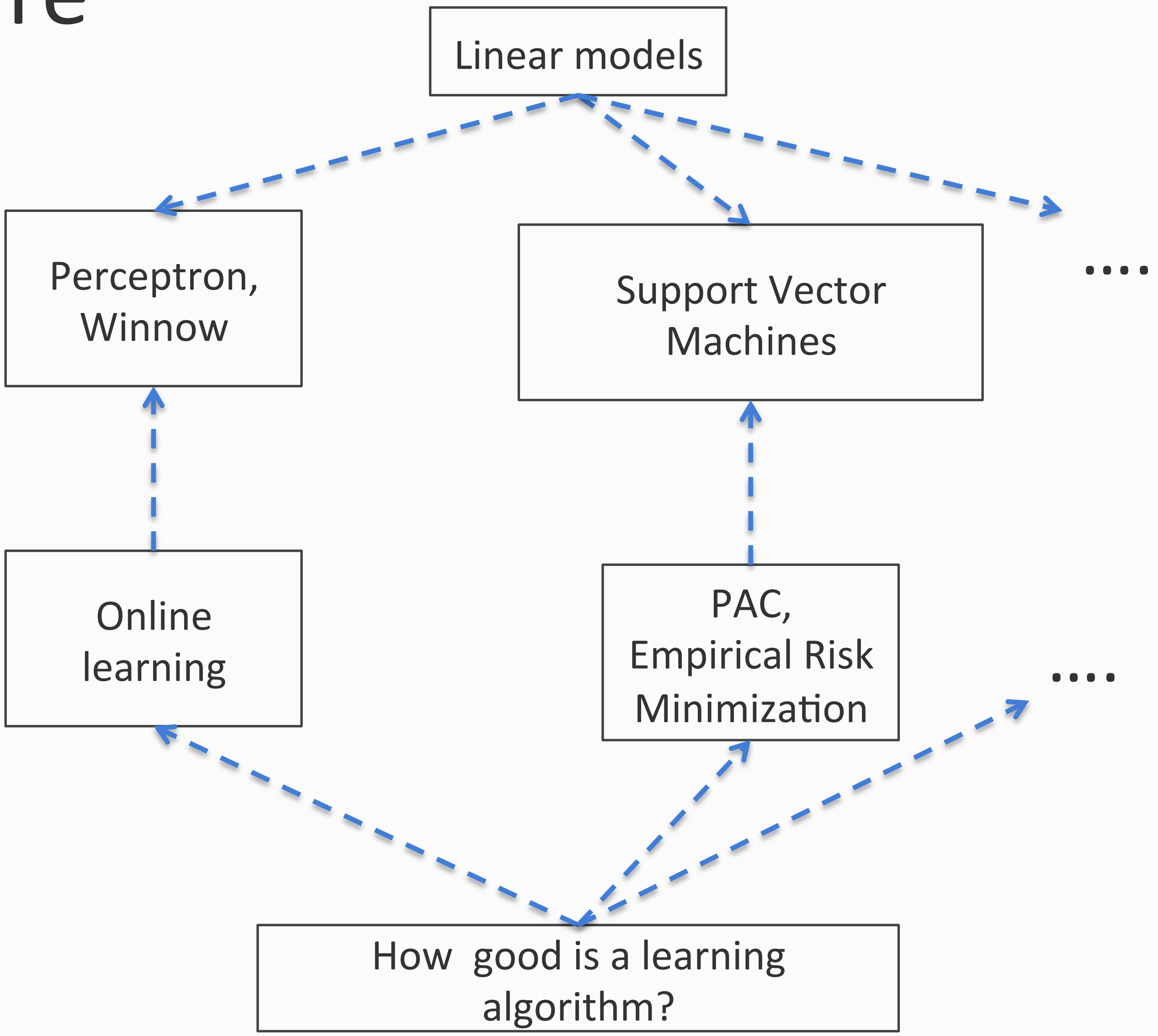
- Can continue to analyze the probabilistic intuition
  - Never saw x_1=0 in positive examples, maybe we'll never see it
  - And if we do, it will be with small probability, so the concepts we learn may be *pretty good*
    - *Pretty good:* In terms of performance on future data
  - ***PAC framework***

- ***Mistake Driven*** Learning algorithms
  - Update your hypothesis only when you make mistakes
  - Define *good* in terms of how many mistakes you make before you stop

Where are we?

- How good is a learning algorithm?
- Online learning
- The Perceptron Algorithm
- Perceptron Mistake Bound
- Variants of Perceptron

Big picture


Linear models
Perceptron, Winnow
Support Vector Machines
….
Online learning
PAC, Empirical Risk Minimization
….
How good is a learning algorithm?

Online Learning

Coming up

- Mistake-driven learning

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

- Issues to watch out for
  - 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

$$\{x_1, x_2, x_3, \ldots\ldots, x_{1000000}\}$$

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

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

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?

Mistake bound algorithms

- Not the most general setting for online learning
- Not the most general metric
- Other metrics: Regret, cumulative loss

- Setting:
  - Instance space: X (dimensionality n)
  - Target f: $X \rightarrow \{0,1\}$, $f \in C$, the concept class (parameterized by n)
- Learning Protocol:
  - Learner is given $\mathbf{x} \in X$, randomly chosen
  - Learner predicts h(**x**), and is then given f(**x**) (feedback)
- Performance: learner makes a mistake when $h(\mathbf{x}) \neq f(\mathbf{x})$
  - $M_A(f, S)$: Number of mistakes algorithm A makes on sequence S of examples for the target function f
  - $M_A(C) = \max_{f \in \mathcal{C}, \mathcal{S}} M_A(f, S)$: The maximum possible number of mistakes made by A for any target function in C and any sequence S of examples
- Algorithm A is a ***mistake bound algorithm*** for the concept class C if $M_A(C)$ is a polynomial in n

Online Learning

- No assumptions about the distribution of examples
- Examples are presented to the learning algorithm in a sequence. *Could be adversarial!*

  For each example:
  1. Learner gets an unlabeled example
  2. Learner makes a prediction
  3. Then, the true label is revealed
- Count the number of mistakes
- A concept class is learnable in the *mistake bound model* if there exists an algorithm that makes a polynomial number of mistakes for any sequence of examples
  - Polynomial in the size of the examples

Online Learning

- Simple and intuitive model, widely applicable
  - 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

Online/Mistake Bound Learning

- No notion of distribution; a worst case model
- No (or not much) 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

Is counting mistakes enough?

- Under the mistake bound model, we are not concerned about the number of examples needed to learn a function

- We only care about not making mistakes

- Eg: Suppose the learner is presented the *same example* over and over
  - *Under the mistake bound model, it is okay*
  - *We won't be able to learn the function, but we won't make any mistakes either!*

Can mistake bound algorithms exist?

Before getting to the ‘standard’ mistake bound algorithms, a proof-of-concept mistake bound algorithms

The Halving algorithm

Generic Mistake Bound Algorithms

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

- Algorithm CON:
  - In the ith 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, $C_{i+1} \subseteq C_i$

- If a mistake is made on the ith 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 finite concept class.

Goal: To learn a hidden $f \in C$

- Initialize $C_0 = C$, the set of all possible functions

$$\downarrow$$

We will construct a series of sets of functions C_i

$$\downarrow$$

- Learning ends when there is only one element in C_i

The Halving Algorithm

Let C be a finite concept class.

Goal: To learn a hidden f $\in$ C

- Initialize C_0 = C, the set of all possible functions
- When an example **x** arrives:
  - Predict the label for **x** as 1 if a majority of the functions in C_i predict 1. Otherwise 0. That is, output = 1 if

$$|\{h(\mathbf{x}) = 1 : h \in C_i\}| > |\{h(\mathbf{x}) = 0 : h \in C_i\}|$$

- Learning ends when there is only one element in C_i

The Halving Algorithm

Let C be a finite concept class.

Goal: To learn a hidden f $\in$ C

- Initialize C_0 = C, the set of all possible functions
- When an example **x** arrives:
    - Predict the label for **x** as 1 if a majority of the functions in C_i predict 1. Otherwise 0. That is, output = 1 if

$$|\{h(\mathbf{x}) = 1 : h \in C_i\}| > |\{h(\mathbf{x}) = 0 : h \in C_i\}|$$

    - If prediction ≠ *f(**x**)*: (i.e error)
        - Update C_{i+1} = all elements of C_i that agree with f(**x**)
- Learning ends when there is only one element in C_i

The Halving Algorithm

Let C be a finite concept class.

Goal: To learn a hidden f $\in$ C

- Initialize C_0 = C, the set of all possible functions
- When an example **x** arrives:
  - Predict the label for **x** as 1 if a majority of the functions in C_i predict 1. Otherwise 0. That is, output = 1 if

$$|\{h(\mathbf{x}) = 1 : h \in C_i\}| > |\{h(\mathbf{x}) = 0 : h \in C_i\}|$$

  - If prediction ≠ *f(**x**)*: (i.e error)
    - Update C_{i+1} = all elements of C_i that agree with f(**x**)
- Learning ends when there is only one element in C_i

How many mistakes will the Halving algorithm make?

How many mistakes will the Halving algorithm make?

Suppose it makes n mistakes. Finally, we will have the final set of concepts C_n with one element

C_n was created when a majority of the functions in C_{n-1} were incorrect

$$1 = |C_n| \quad < \quad \frac{1}{2}|C_{n-1}|$$

How many mistakes will the Halving algorithm make?

Suppose it makes n mistakes. Finally, we will have the final set of concepts C_n with one element

C_n was created when a majority of the functions in C_{n-1} were incorrect

$$\begin{aligned} 1 = |C_n| &< \frac{1}{2}|C_{n-1}| \\ &< \frac{1}{2} \cdot \frac{1}{2}|C_{n-2}| \\ &< \vdots \\ &< \frac{1}{2^n}|C_0| = \frac{1}{2^n}|C| \end{aligned}$$

How many mistakes will the Halving algorithm make?

Suppose it makes n mistakes. Finally, we will have the final set of concepts C_n with one element

C_n was created when a majority of the functions in C_{n-1} were incorrect

$$\begin{aligned}
1 = |C_n| &< \frac{1}{2}|C_{n-1}| \\
&< \frac{1}{2}\cdot\frac{1}{2}|C_{n-2}| \\
&< \vdots \\
&< \frac{1}{2^n}|C_0| = \frac{1}{2^n}|C|
\end{aligned}
\quad \dashrightarrow \quad |C| > 2^n$$

How many mistakes will the Halving algorithm make?

Suppose it makes n mistakes. Finally, we will have the final set of concepts C_n with one element

C_n was created when a majority of the functions in C_{n-1} were incorrect

$$\begin{aligned} 1 = |C_n| &< \frac{1}{2}|C_{n-1}| \\ &< \frac{1}{2}\cdot\frac{1}{2}|C_{n-2}| \\ &< \vdots \\ &< \frac{1}{2^n}|C_0| = \frac{1}{2^n}|C| \end{aligned} \qquad \rightarrow \qquad |C| > 2^n$$

The Halving algorithm will make at most *log(|C|)* mistakes

The Halving Algorithm

- Hard to compute

- In some concept classes, Halving is *optimal*
  - Eg: for class of all Boolean functions

For the most difficult concept in the class,

For the most difficult sequence of examples,

The optimal mistake bound algorithm makes the fewest number of mistakes

- 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

- Hidden function: conjunctions
  - The learner is to learn functions like $f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100}$
- Number of conjunctions with n variables = ? ?

Learning Conjunctions

- Hidden function: conjunctions
  - The learner is to learn functions like $f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$
- Number of conjunctions with n variables = 3^n
  - log(|C|) = O(n)
- The elimination algorithm makes at most n mistakes
  - Learn from positive examples; eliminate inactive literals.

- Hidden function: *k-conjunctions*
  - Assume that only k<<n attributes occur in the conjunction
- Number of k-conjunctions: $2^k C(n,k) \approx 2^k n^k$
  - log(|C|) = k log(n)
  - Can we learn efficiently with this number of mistakes ?

Representation and efficient learning

- Assume that you want to learn conjunctions. Should your hypothesis space be the class of conjunctions?
  - **Theorem** [Haussler 1988]: 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 (dis/conj)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

Summary: The Halving algorithm

- A simple algorithm for *finite* concept spaces
  - Stores a set of hypotheses that it iteratively refines
    - Receive an input
    - Prediction: the label of the majority of hypotheses still under consideration
    - Update: If incorrect, remove all inconsistent hypotheses
- Makes $\log|\mathrm{C}|$ mistakes for a concept class C
- Not always optimal because we care about minimizing the number of mistakes in the future!
  - What if, instead of eliminating functions that disagree with this example, we down-weight them
  - Perhaps via multiplicative or additive updates…

What you should know

- What is the mistake bound model?
- Simple proof-of-concept mistake bound algorithms
  - CON: Makes O(|C|) mistakes
  - The Halving algorithm
    - Can learn a concept with at most log(|C|) mistakes
    - Sadly, for non-trivial functions, only useful if we don't care about storage or computation time
- Even for simple Boolean functions (conjunctions and disjunctions), learning them as linear threshold units is computationally easier

Where are we?

- How good is a learning algorithm?
- Online learning

Generic mistake bound learning

- The Perceptron Algorithm
- Perceptron Mistake Bound
- Variants of Perceptron

Back to linear threshold units

Recall: Linear Classifiers

- Input is a n dimensional vector **x**
- Output is a label $y \in \{-1, 1\}$

- *Linear Threshold Units* (LTUs) classify an example **x** using the following classification rule
  - Output = $\mathrm{sgn}(\mathbf{w}^T\mathbf{x} + b) = \mathrm{sgn}(b + \sum w_i x_i)$

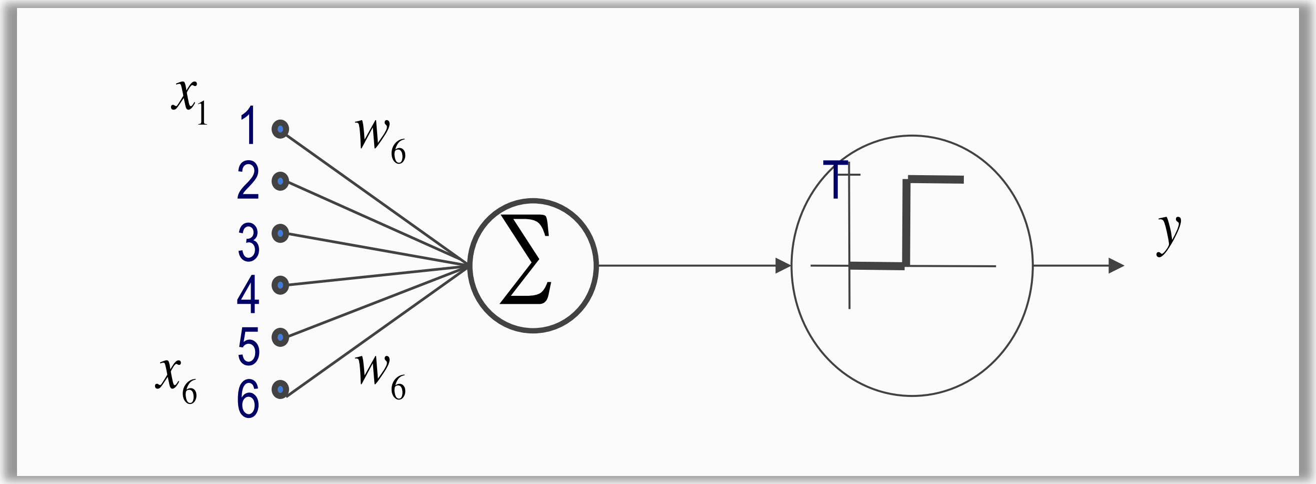
  - $\mathbf{w}^T\mathbf{x} + b \geq 0 \Rightarrow$ Predict $y = 1$
  - $\mathbf{w}^T\mathbf{x} + b < 0 \Rightarrow$ Predict $y = -1$

    b is called the bias term

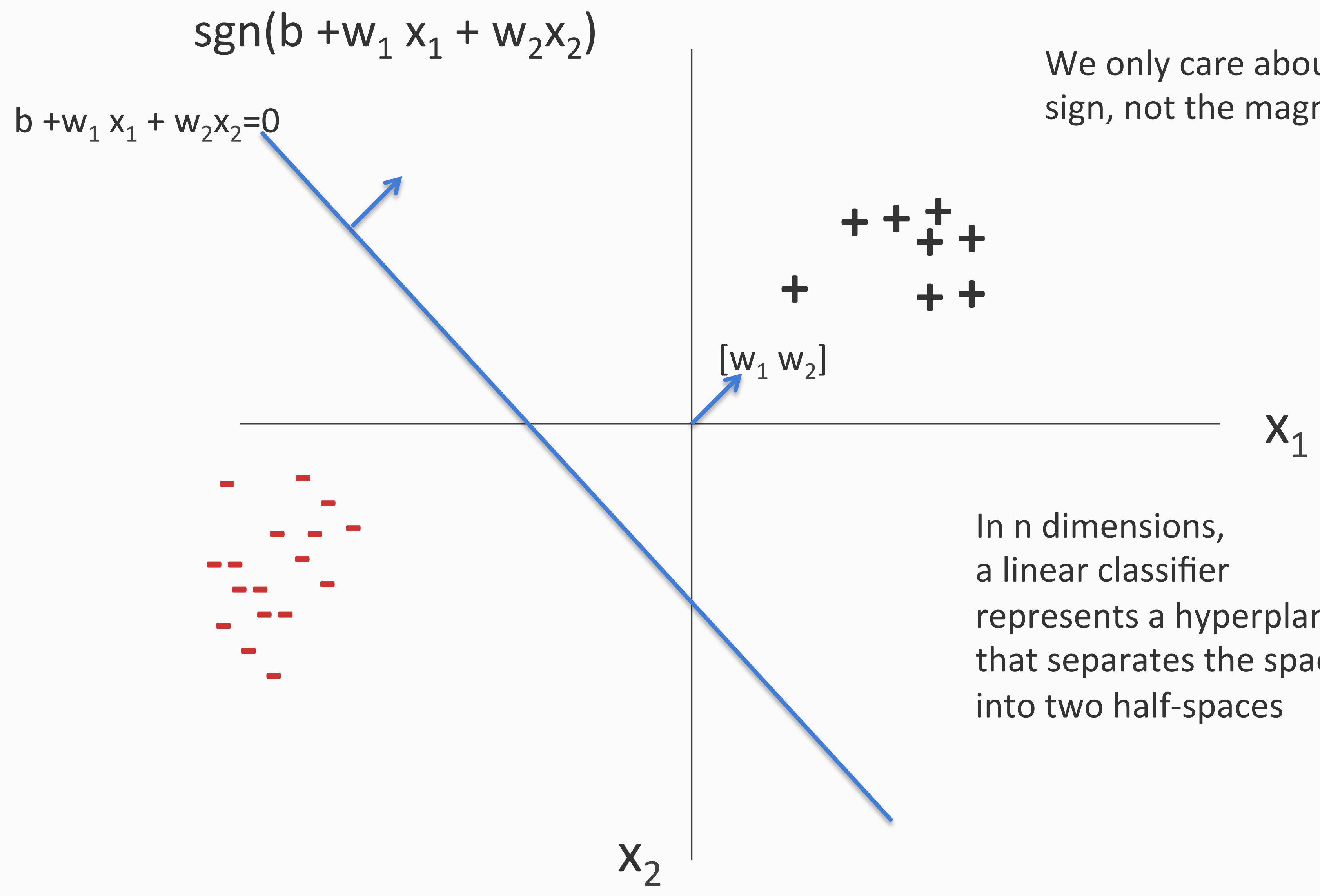
Recall: Linear Classifiers

- Input is a n dimensional vector **x**
- Output is a label y ∈ {-1, 1}

- *Linear Threshold Units* (LTUs) classify an example **x** using the following classification rule



The geometry of a linear classifier

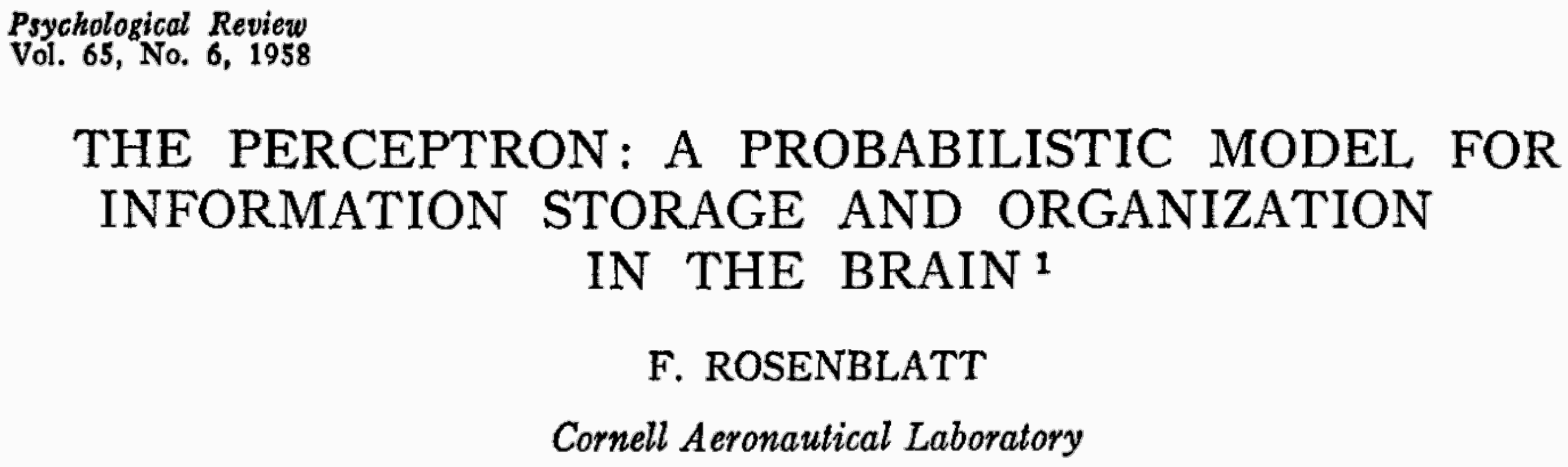


$\text{sgn}(b + w_1 x_1 + w_2 x_2)$
$b + w_1 x_1 + w_2 x_2 = 0$
$[w_1\ w_2]$
x_1
x_2

We only care about the sign, not the magnitude

In n dimensions,
a linear classifier represents a hyperplane that separates the space into two half-spaces

The Perceptron



Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

The Perceptron algorithm

- Rosenblatt 1958
- The goal is to find a separating hyperplane
  - For separable data, guaranteed to find one
- An online algorithm
  - Processes one example at a time
- Several variants exist (will discuss briefly at towards the end)

The Perceptron algorithm

Input: A sequence of training examples $(x_1, y_1), (x_2, y_2), \cdots$ where all $x_i \in \Re^n$, $y_i \in \{-1,1\}$

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example (x_i, y_i):
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T x_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, x_i)$
- Return final weight vector

Remember:
Prediction = $\text{sgn}(\mathbf{w}^T\mathbf{x})$

There is typically a bias term also $(\mathbf{w}^T\mathbf{x} + b)$, but the bias may be treated as a constant feature and folded into $\mathbf{w}$

Footnote: For some algorithms it is mathematically easier to represent False as -1, and at other times, as 0. For the Perceptron algorithm, treat -1 as false and +1 as true.

The Perceptron algorithm

Input: A sequence of training examples $(x_1, y_1), (x_2, y_2), \cdots$ where all $x_i \in \Re^n$, $y_i \in \{-1,1\}$

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example (x_i, y_i):
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T x_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, x_i)$
- Return final weight vector

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\, x_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\, x_i$

r is the learning rate, a small positive number less than 1

Update only on error. A mistake-driven algorithm

This is the simplest version. We will see more robust versions at the end

Mistake can be written as $y_i \mathbf{w}_t^T x_i \leq 0$

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\, x_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\, x_i$

Intuition behind the update

Suppose we have made a mistake on a positive example

That is, y = +1 and $\mathbf{w}_t^\mathsf{T} x < 0$

Call the new weight vector $\mathbf{w}_{t+1} = \mathbf{w}_t + x$ (say r = 1)

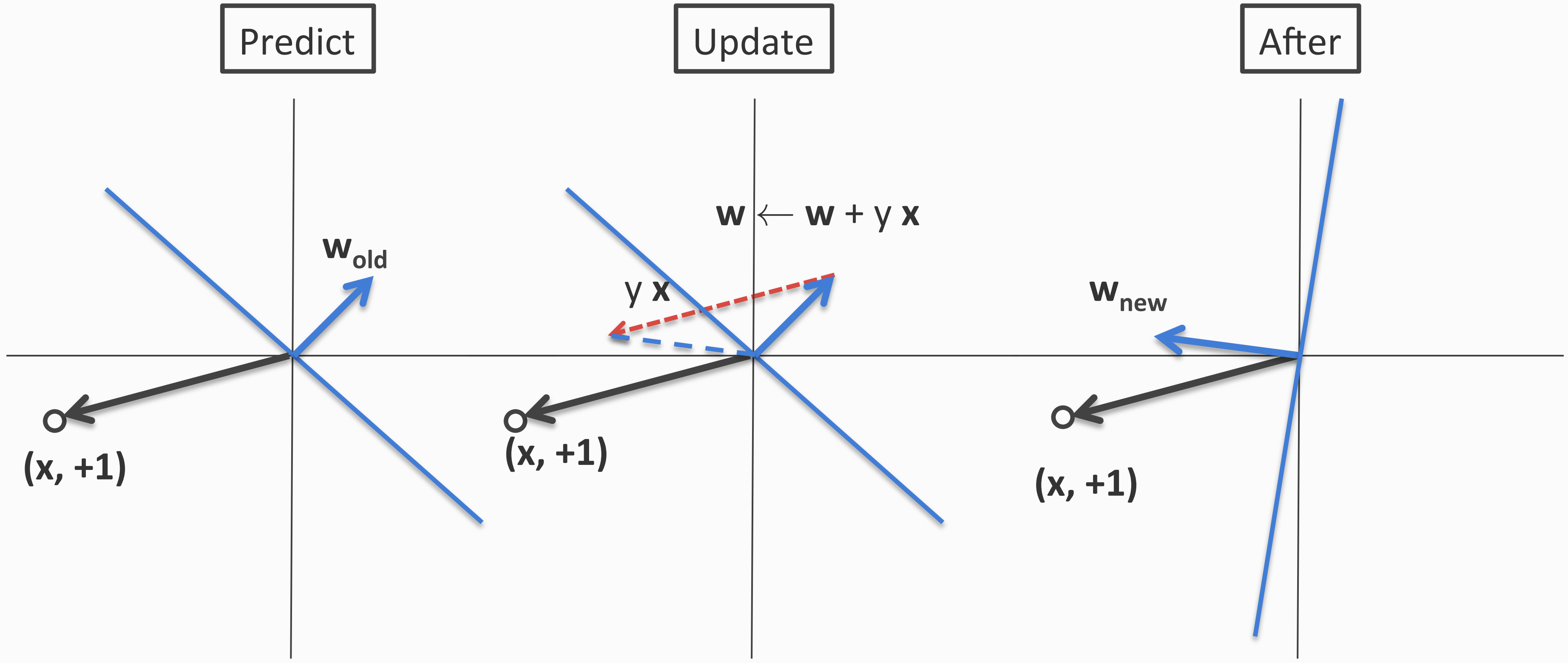
The new dot product will be $\mathbf{w}_{t+1}^\mathsf{T} x = (\mathbf{w}_t + x)^\mathsf{T} x = \mathbf{w}_t^\mathsf{T} x + x^\mathsf{T} x \geq \mathbf{w}_t^\mathsf{T} x$

For a positive example, the Perceptron update will increase the score assigned to the same input

Similar reasoning for negative examples

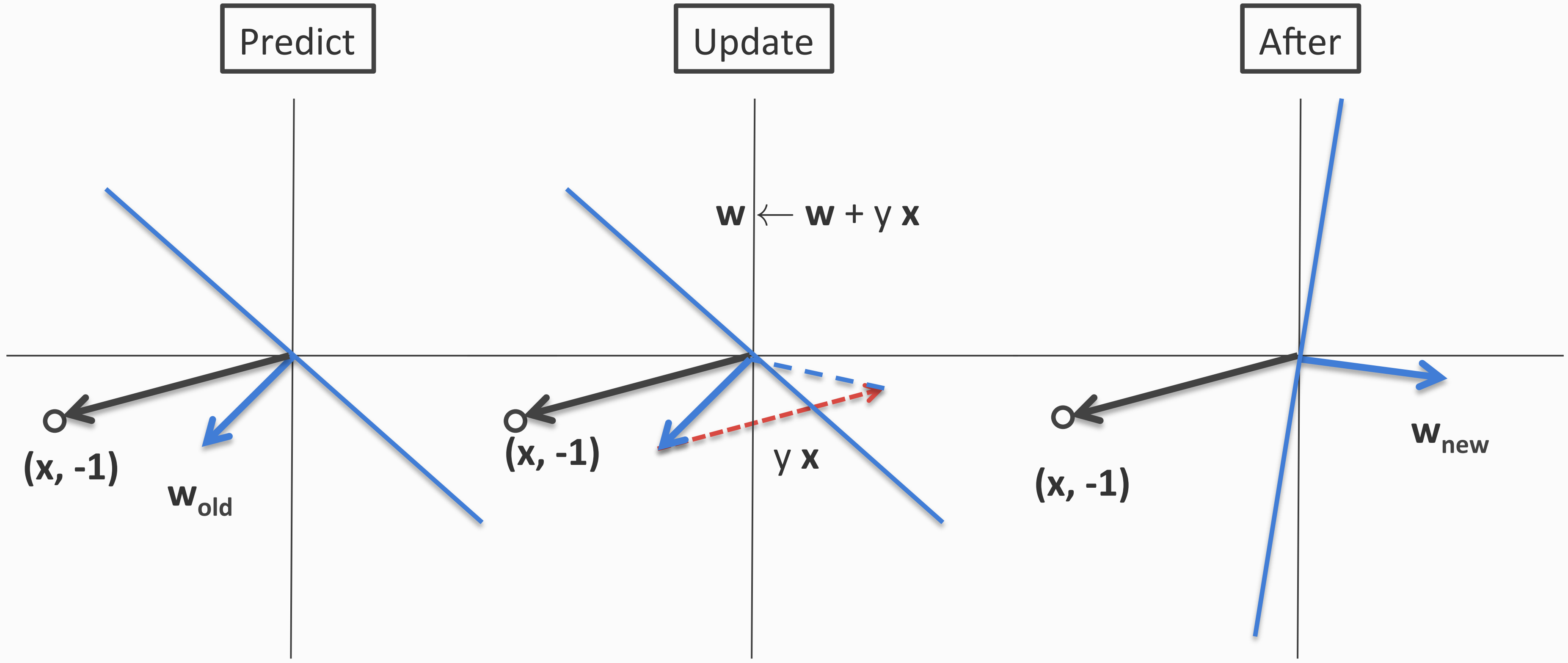
Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\, x_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\, x_i$

Geometry of the perceptron update



For a mistake on a positive example

Geometry of the perceptron update



For a mistake on a negative example

Perceptron Learnability

- Obviously Perceptron cannot learn what it cannot 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)
    - But we already know that XOR is not linearly separable

  - In vision, if patterns are represented with local features, can't represent symmetry, connectivity

What you need to know

- The Perceptron algorithm
- The geometry of the update
- What can it represent

Where are we?

- How good is a learning algorithm?
- Online learning
- The Perceptron Algorithm
- Perceptron Mistake Bound
- Variants of Perceptron

Convergence

Convergence theorem

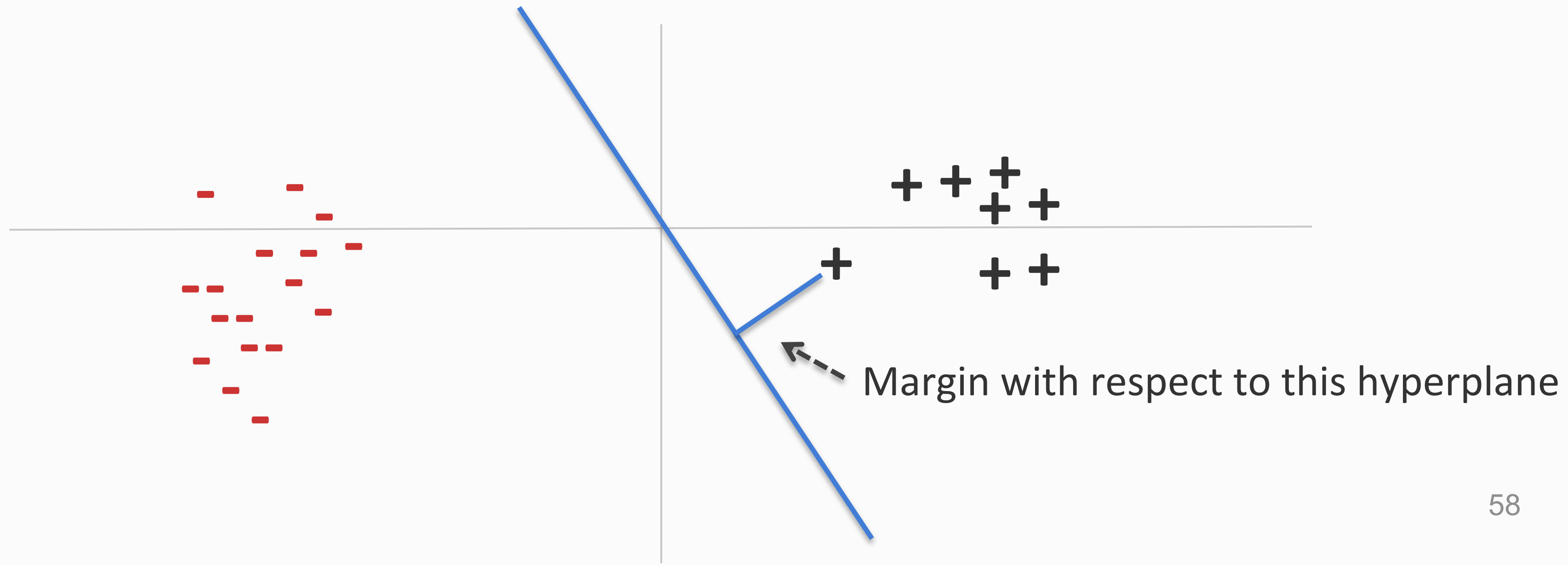
- If there exist a set of weights that are consistent with the data (i.e. the data is linearly separable), the perceptron algorithm will converge.

Cycling theorem

- If the training data is *not* linearly separable, then the learning algorithm will eventually repeat the same set of weights and enter an infinite loop

Margin

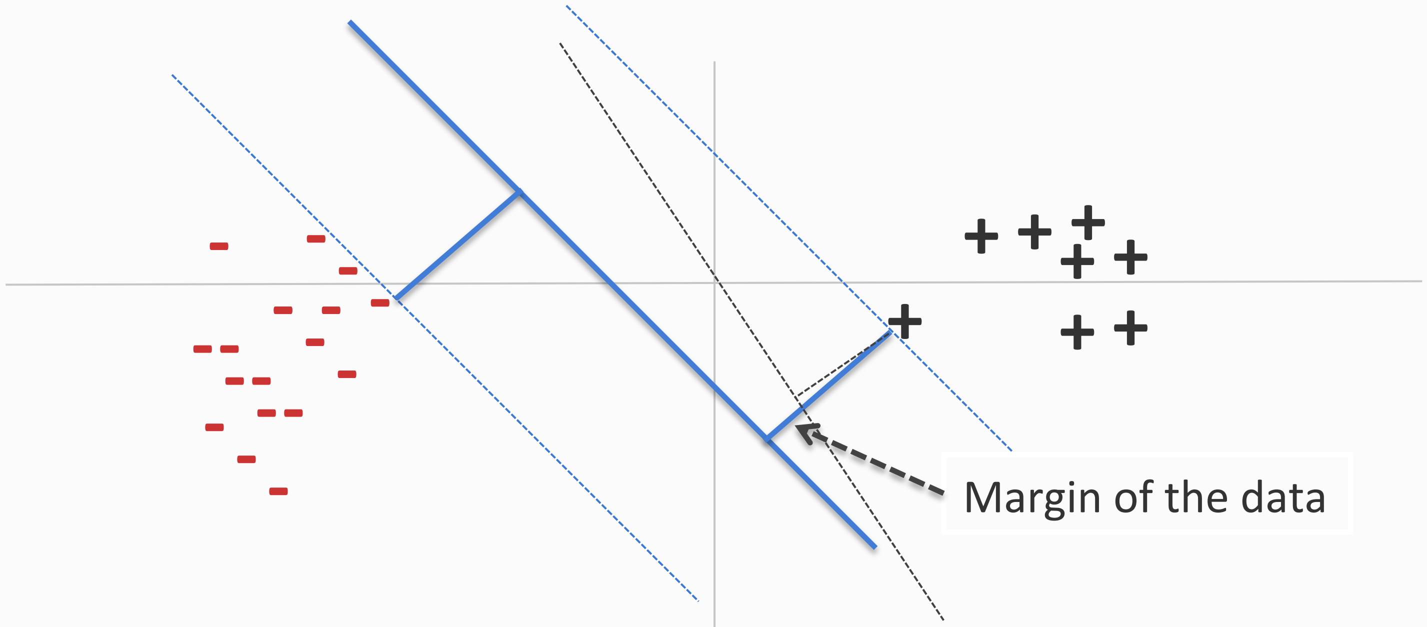
- The margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.



Margin

- The margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.

- The margin of a data set (γ) is the maximum margin possible for that dataset using any weight vector.



Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $\|\mathbf{x}_i\| \leq R$ and the label $y_i \in \{-1, +1\}$.

Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $\|\mathbf{u}\| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i (\mathbf{u}^T \mathbf{x}_i) \geq \gamma$.

Then, the perceptron algorithm will make at most $(R/\gamma)^2$ mistakes on the training sequence.

Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $\|\mathbf{x}_i\| \leq R$ and the label $y_i \in \{-1, +1\}$.

We can always find such an R. Just look for the farthest data point from the origin.

Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $\|\mathbf{u}\| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i (\mathbf{u}^T \mathbf{x}_i) \geq \gamma$.

Then, the perceptron algorithm will make at most $(R/\gamma)^2$ mistakes on the training sequence.

Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $\|\mathbf{x}_i\| \leq R$ and the label $y_i \in \{-1, +1\}$.

Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $\|\mathbf{u}\| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i\,(\mathbf{u}^T \mathbf{x}_i) \geq \gamma$.

Then, the perceptron algorithm will make at most $(R/\gamma)^2$ mistakes on the training sequence.

The data has a margin γ.
Importantly, the data is *separable*.
γ is the complexity parameter that defines the seperability of data.

Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $\|\mathbf{x}_i\| \leq R$ and the label $y_i \in \{-1, +1\}$.

Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $\|\mathbf{u}\| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i (\mathbf{u}^T \mathbf{x}_i) \geq \gamma$.

Then, the perceptron algorithm will make at most $(R/\gamma)^2$ mistakes on the training sequence.

If **u** hadn't been a unit vector, then we could scale γ in the mistake bound. This will change the final mistake bound to $(\|\mathbf{u}\| R/\gamma)^2$

Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $\|\mathbf{x}_i\| \leq R$ and the label $y_i \in \{-1, +1\}$.

Suppose we have a binary classification dataset with n dimensional inputs.

Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $\|\mathbf{u}\| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i (\mathbf{u}^T \mathbf{x}_i) \geq \gamma$.

If the data is separable,…

Then, the perceptron algorithm will make at most $(R/\gamma)^2$ mistakes on the training sequence.

…then the Perceptron algorithm will find a separating hyperplane after making a finite number of mistakes

- Receive an input $(\mathbf{x}_i, y_i)$
- if sgn($\mathbf{w}_t^T \mathbf{x}_i$) ≠ y_i:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i \mathbf{x}_i$

Proof (preliminaries)

The setting

- Initial weight vector **w** is all zeros
- Learning rate = 1
  - Effectively scales inputs, but does not change the behavior
- All training examples are contained in a ball of size R
  - $\|\mathbf{x}_i\| \leq R$
- The training data is separable by margin γ using a unit vector **u**
  - $y_i (\mathbf{u}^T \mathbf{x}_i) \geq \gamma$

- Receive an input (x_i, y_i)
- if sgn($\mathbf{w}_t^T x_i$) ≠ y_i:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i\, x_i$

Proof (1/3)

1. After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\gamma$

$$\mathbf{u}^T\mathbf{w}_{t+1} = \mathbf{u}^T\mathbf{w}_t + y_i\mathbf{u}^T\mathbf{x}_i$$

Proof (1/3)

- Receive an input (x_i, y_i)
- if $\text{sgn}(\mathbf{w}_t^T x_i) \neq y_i$:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i\, x_i$

1. After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\gamma$

$$\begin{aligned} \mathbf{u}^T\mathbf{w}_{t+1} &= \mathbf{u}^T\mathbf{w}_t + y_i\mathbf{u}^T\mathbf{x}_i \\ &\geq \mathbf{u}^T\mathbf{w}_t + \gamma \end{aligned}$$

Because the data is separable by a margin γ

- Receive an input (x_i, y_i)
- if $\text{sgn}(\mathbf{w}_t^T x_i) \neq y_i$:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i x_i$

Proof (1/3)

1. After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\gamma$

$$\begin{aligned} \mathbf{u}^T\mathbf{w}_{t+1} &= \mathbf{u}^T\mathbf{w}_t + y_i\mathbf{u}^T\mathbf{x}_i \\ &\geq \mathbf{u}^T\mathbf{w}_t + \gamma \end{aligned}$$

Because the data is separable by a margin γ

Because $\mathbf{w}_0 = \mathbf{0}$ (i.e $\mathbf{u}^T\mathbf{w}_0 = 0$), straightforward induction gives us $\mathbf{u}^T\mathbf{w}_t \geq t\gamma$

- Receive an input (x_i, y_i)
- if sgn($\mathbf{w}_t^{\mathrm{T}} x_i$) ≠ y_i:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i\, x_i$

Proof (2/3)

2. After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

$$\begin{aligned} \left\|\mathbf{w}_{t+1}\right\|^2 &= \left\|\mathbf{w}_t + y_i \mathbf{x}_i\right\|^2 \\ &= \left\|\mathbf{w}_t\right\|^2 + 2y_i(\mathbf{w}_t^T \mathbf{x}_i) + \left\|\mathbf{x}_i\right\|^2 \end{aligned}$$

- Receive an input (x_i, y_i)
- if $\text{sgn}(\mathbf{w}_t^T x_i) \neq y_i$:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i\, x_i$

Proof (2/3)

2. After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

$$\begin{aligned} \|\mathbf{w}_{t+1}\|^2 &= \|\mathbf{w}_t + y_i \mathbf{x}_i\|^2 \\ &= \|\mathbf{w}_t\|^2 + 2y_i(\mathbf{w}_t^T \mathbf{x}_i) + \|\mathbf{x}_i\|^2 \end{aligned}$$

The weight is updated only when there is a mistake. That is when $y_i\, \mathbf{w}_t^T\, \mathbf{x}_i < 0$.

$\|\mathbf{x}_i\| \leq R$, by definition of R

- Receive an input (x_i, y_i)
- if sgn($\mathbf{w}_t^T x_i$) ≠ y_i:
  Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_i\, x_i$

Proof (2/3)

2. After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

$$\begin{aligned} \left\|\mathbf{w}_{t+1}\right\|^2 &= \left\|\mathbf{w}_t + y_i \mathbf{x}_i\right\|^2 \\ &= \left\|\mathbf{w}_t\right\|^2 + 2y_i(\mathbf{w}_t^T \mathbf{x}_i) + \left\|\mathbf{x}_i\right\|^2 \\ &\leq \left\|\mathbf{w}_t\right\|^2 + R^2 \end{aligned}$$

Because $\mathbf{w}_0 = \mathbf{0}$ (i.e $\mathbf{u}^T\mathbf{w}_0 = 0$), straightforward induction gives us $\|\mathbf{w}_t\|^2 \leq tR^2$

Proof (3/3)

What we know:

1. After t mistakes, $\mathbf{u}^\mathsf{T}\mathbf{w}_t \geq t\gamma$
2. After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

Proof (3/3)

What we know:

1. After t mistakes, $\mathbf{u}^\mathsf{T}\mathbf{w}_t \geq t\gamma$
2. After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

$$R\sqrt{t} \geq \|\mathbf{w}_t\|$$

From (2)

Proof (3/3)

What we know:

1. After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\gamma$
2. After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

$$\underbrace{R\sqrt{t} \geq \|\mathbf{w}_t\|}_{\boxed{\text{From (2)}}} \geq \mathbf{u}^T\mathbf{w}_t$$

$\mathbf{u}^T \mathbf{w}_t = \|\mathbf{u}\| \|\mathbf{w}_t\| \; cos$(<angle between them>)

But $\|\mathbf{u}\|$ = 1 and cosine is less than 1

So $\mathbf{u}^T \mathbf{w}_t \leq \|\mathbf{w}_t\|$ *(Cauchy-Schwarz inequality)*

Proof (3/3)

What we know:

1. After t mistakes, $\mathbf{u}^T\mathbf{w}_t \geq t\gamma$
2. After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

$$\underbrace{R\sqrt{t} \geq \|\mathbf{w}_t\|}_{\boxed{\text{From (2)}}} \geq \mathbf{u}^T\mathbf{w}_t \underbrace{\geq t\gamma}_{\boxed{\text{From (1)}}}$$

$\mathbf{u}^T \mathbf{w}_t = \|\mathbf{u}\| \, \|\mathbf{w}_t\| \, cos$(<angle between them>)

But $\|\mathbf{u}\| = 1$ and cosine is less than 1

So $\mathbf{u}^T \mathbf{w}_t \leq \|\mathbf{w}_t\|$

Proof (3/3)

What we know:

1. After t mistakes, $\mathbf{u}^{\mathsf{T}}\mathbf{w}_t \geq t\gamma$
2. After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

$$R\sqrt{t} \geq \|\mathbf{w}_t\| \geq \mathbf{u}^T\mathbf{w}_t \geq t\gamma$$

Number of mistakes $t \leq \frac{R^2}{\gamma^2}$

Bounds the total number of mistakes!

Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $\|\mathbf{x}_i\| \leq R$ and the label $y_i \in \{-1, +1\}$.

Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $\|\mathbf{u}\| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i (\mathbf{u}^T \mathbf{x}_i) \geq \gamma$.

Then, the perceptron algorithm will make at most $(R/\gamma)^2$ mistakes on the training sequence.

The Perceptron Mistake bound

Number of mistakes $\leq \frac{R^2}{\gamma^2}$

- R is a property of the dimensionality. How?
  - For Boolean functions with n attributes, show that $R^2 = n$.
- γ is a property of the data
- Exercises:
  - How many mistakes will the Perceptron algorithm make for disjunctions with n attributes?
    - What are R and γ?
  - How many mistakes will the Perceptron algorithm make for k-disjunctions with n attributes?
  - Find a sequence of examples that will force the Perceptron algorithm to make O(n) mistakes for a concept that is a k-disjunction.

Beyond the separable case

- Good news
  - Perceptron makes no assumption about data distribution, could be even adversarial
  - After a fixed number of mistakes, you are done. Don't even need to see any more data

- Bad news: Real world is not linearly separable
  - Can't expect to *never* make mistakes again
  - What can we do: more features, try to be linearly separable if you can, use averaging

What you need to know

- What is the perceptron mistake bound?

- How to prove it

Where are we?

- How good is a learning algorithm?
- Online learning
- The Perceptron Algorithm
- Perceptron Mistake Bound
- Variants of Perceptron

The Perceptron algorithm

Input: A sequence of training examples $(x_1, y_1), (x_2, y_2), \cdots$ where all $x_i \in \Re^n$, $y_i \in \{-1,1\}$

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example (x_i, y_i):
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T x_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, x_i)$
- Return final weight vector

Mistake Bound Theorem [Novikoff 1962, Block 1962]

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$ be a sequence of training examples such that for all i, the feature vector $\mathbf{x}_i \in \Re^n$, $\|\mathbf{x}_i\| \leq R$ and the label $y_i \in \{-1, +1\}$.

Suppose there exists a unit vector $\mathbf{u} \in \Re^n$ (i.e $\|\mathbf{u}\| = 1$) such that for some $\gamma \in \Re$ and $\gamma > 0$ we have $y_i (\mathbf{u}^T \mathbf{x}_i) \geq \gamma$.

Then, the perceptron algorithm will make at most $(R/\gamma)^2$ mistakes on the training sequence.

Beyond the separable case

- Good news
  - Perceptron makes no assumption about data distribution, could be even adversarial
  - After a fixed number of mistakes, you are done. Don't even need to see any more data

- Bad news: Real world is not linearly separable
  - Can't expect to *never* make mistakes again
  - What can we do: more features, try to be linearly separable if you can, use averaging

Where are we?

- How good is a learning algorithm?
- Online learning
- The Perceptron Algorithm
- Perceptron Mistake Bound
- Variants of Perceptron

Practical use of the Perceptron algorithm

1. Using the Perceptron algorithm with a finite dataset

2. Voting and Averaging

3. Margin Perceptron

1. The “standard” algorithm

Given a training set D = $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x}_i \in \Re^n$, $y_i \in \{-1,1\}$

```
1. Initialize w = 0 ∈ ℜⁿ
2. For epoch = 1 … T:
    1. For each training example (x_i, y_i) ∈ D:
        • If y_i wᵀx_i ≤ 0, update w ← w + r y_i x_i
3. Return w
```

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. For each training example $(\mathbf{x}_i, y_i) \in$ D:
      - If $y_i \mathbf{w}^T\mathbf{x}_i \leq 0$, update $\mathbf{w} \leftarrow \mathbf{w} + r\, y_i\, \mathbf{x}_i$
3. Return **w**

T is a hyper-parameter to the algorithm

In practice, good to shuffle D before the inner loop

Another way of writing that there is an error

Prediction: sgn($\mathbf{w}^T\mathbf{x}$)

2. Voting and Averaging

- So far: We return the final weight vector
- Voted perceptron
  - Remember every weight vector in your sequence of updates.
  - At final prediction time, each weight vector gets to vote on the label. The number of votes it gets is the number of iterations it survived before being updated
  - Comes with strong theoretical guarantees about generalization, impractical because of storage issues
- Averaged perceptron
  - Instead of using all weight vectors, use the average weight vector (i.e longer surviving weight vectors get more say)
  - More practical alternative and widely used

Averaged Perceptron

Given a training set D = $\{(x_i, y_i)\}$, $x_i \in \Re^n$, $y_i \in \{-1,1\}$

```
1. Initialize $\mathbf{w} = 0 \in \Re^n$ and $\mathbf{a} = 0 \in \Re^n$

2. For epoch = 1 … T:
   – For each training example $(x_i, y_i) \in D$:
       • If $y_i \mathbf{w}^T x_i \leq 0$
           – update $\mathbf{w} \leftarrow \mathbf{w} + r\, y_i\, x_i$
       • $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{w}$
3. Return $\mathbf{a}$
```

Prediction: sgn($\mathbf{a}^T\mathbf{x}$)

This is the simplest version of the averaged perceptron

There are some easy programming tricks to make sure that **a** is also updated only when there is an error

Extremely popular

If you want to use the Perceptron algorithm, use averaging

3. Margin Perceptron

- Perceptron makes updates only when the prediction is incorrect

  $$y_i \mathbf{w}^\mathsf{T} x_i < 0$$

- What if the prediction is close to being incorrect? That is, Pick a small positive η and update when

  $$y_i \mathbf{w}^\mathsf{T} x_i < \eta$$

- Can generalize better, but need to choose η
  - Exercise: Why is this a good idea?

Summary: Perceptron

- Online learning algorithm, very widely used, easy to implement
- Additive updates to weights
- Geometric interpretation
- Mistake bound
- Practical variants abound
- You should be able to implement the Perceptron algorithm