Machine Learning

Online Learning

Dan Goldwasser

dgoldwas@purdue.edu

Online Learning and Linear models

We will introduce a new way to quantify performance,
by ***measuring and bounding the number of mistake an algorithm makes***.

We'll show that depending on the hypothesis class we choose, we can use algorithms that have better mistake bounds

Specifically, we'll compare learning disjunctions using Boolean functions and linear functions, and analyze their behavior.

Quantifying Performance

- We want to be able to say something rigorous about the performance of our **learning algorithm**
  - *Several ways of doing it*

Note the difference from the previous discussion about bounding the true error based on the observed error on the test set

- We will concentrate on discussing the **number of examples** one needs to **see** before we can say that our learned hypothesis is good.

Learning Conjunctions

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

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

- High dimensional space, but concept only uses a few
- *How many examples are needed to learn it ?*
  - **Protocol I**: The learner proposes instances as queries to the teacher ("active learning")
  - **Protocol II**: The teacher (*who knows f*) provides training examples ("hands-on teacher")
  - **Protocol III:** *Some random source (e.g., Nature) provides training examples; the Teacher (Nature) provides the labels (f(x))*
    - *Recall the definition of a data generating distribution P(X,Y)*

Learning Conjunctions

- Protocol III: Some random source (e.g., Nature) provides training examples
  - **Algorithm: Elimination**
  - Start with the set of all literals as candidates
  - ***Eliminate a literal that is not active (0) in a positive example***

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

Learning Conjunctions

- **Protocol III**: Some random source (e.g., Nature) provides training examples
  - **Algorithm: Elimination**
  - Start with the set of all literals as candidates
  - ***Eliminate a literal that is not active (0) in a positive example***

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

<(1,1,1,1,1,1,…,1,1), 1>
<(1,1,1,0,0,0,…,0,0), 0> ← learned nothing
<(1,1,1,1,1,0,…0,1,1), 1> ← **Eliminate literals**
<(1,0,1,1,1,0,…0,1,1), 0> ← learned nothing
<(1,1,1,1,1,0,…0,0,1), 1> ← **Eliminate literals**
<(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>

Final Hypothesis:
(*not the target!*)

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

Is that good ?
We only learned an approximation to the true concept!

Is it GOOD? *Two Directions*

- Follow a probabilistic intuition (PAC)
  - Considers the *Data generating distribution*
  - Never saw x_1=0 in positive examples, maybe we never will?
  - **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
- Mistake Driven Learning Algorithms
  - Learn from a stream of examples, update only when you make a mistake
  - Good: in terms of **how many mistakes** you make before you stop, happy with your hypothesis.

Online Learning

- **Online learning**
  - Learn from one example at a time (unlike batch)
  - Update current hypothesis based on that example
- **Mistake (*error*) driven learning**
  - **Update only on mistakes**
  - *Not all online learning algorithms are mistake driven*
- **Discuss two learning algorithms for linear functions**
  - Perceptron and Winnow

Online Learning: *Evaluation*

- Model:
  - Instance space: X (*dimensionality – n*)
  - Target: f: X →{0,1}, f ∈ C, concept class (***parameterized by n***)
- Protocol:
  - learner is given x ∈ X
  - learner predicts h(x), and is then given f(x) (*feedback*)
- Performance*: learner makes a mistake when h(x) ≠ f(x)*
  - # mistakes algorithm A makes on sequence S of examples, for 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** $M_A(c)$ is polynomial in n, the complexity parameter of the target concept.
  - Worse case model – No notion of distribution

Question:

Is it a realistic analysis?

Can we bound the number of mistakes?

Generic Mistake Bound Algorithms

- Let C be a concept class. Learn $f \in C$
- **CON**:
  - In the i-th stage of the algorithm:
    - C_i all concepts in C consistent with all $(i\text{-}1)$ 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 ∈ 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** (why?)
- In some cases Halving is optimal (C - class of all Boolean functions)

- We discuss these algorithms since they give us an idea of the ***theoretical bound for mistake driven learning***
  - *Can we find efficient algorithms that are close to the bounds?*

Learning Disjunctions

- There is a hidden disjunction the learner is to learn

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

- The number of disjunctions: 3^n
  - log(|C|) = n
- *Can you find a mistake bound algorithm for disjunctions?*
  - The elimination algorithm makes n mistakes
    - *How would you adapt it to the disjunctive case?*
  - The Halving Algorithm makes n mistakes as well

- Great news!
  - We have a mistake bound algorithm for disjunctions!

Learning K-Disjunctions

- **k-disjunctions:**
  - Assume that only k<<n attributes occur in the disjunction
- A reasonable scenario:
  - A spam email depends on a subset of words:

    "drugs" OR "credit" OR "pill"
- ***What is n and what is k in this example?***
- The number of k-disjunctions: $2^k C(n,k) \approx 2^k n^k$
  - How many mistakes: (1) elimination (2) Halving
  - Can we learn **efficiently** with this number of mistakes ?

The Importance of Representation

- **Assume that you want to learn disjunctions. Should your hypothesis space be the class of disjunctions?**

> ***Theorem [Haussler 1988]:*** *Given a sample on n attributes consistent with a disjunctive concept, it is NP-hard to find a pure disjunctive hypothesis that is both consistent with the sample and has the minimum number of attributes*

- Intuition: Reduction to minimum set cover problem.
- ➔ Cannot learn the concept efficiently **as a disjunction.**
- 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

- **We want to discuss learning in a formal way**
  - *Did our algorithm learn?*
- Difficult question! *Two approaches*
  - **Probabilistic Intuition**: Data generating distribution
    - Provides a probabilistic measure of "what's important"
  - ***Mistake driven learning***: simpler (and **strict**) way to discuss learning
    - "when have we made enough mistakes"

Linear Models

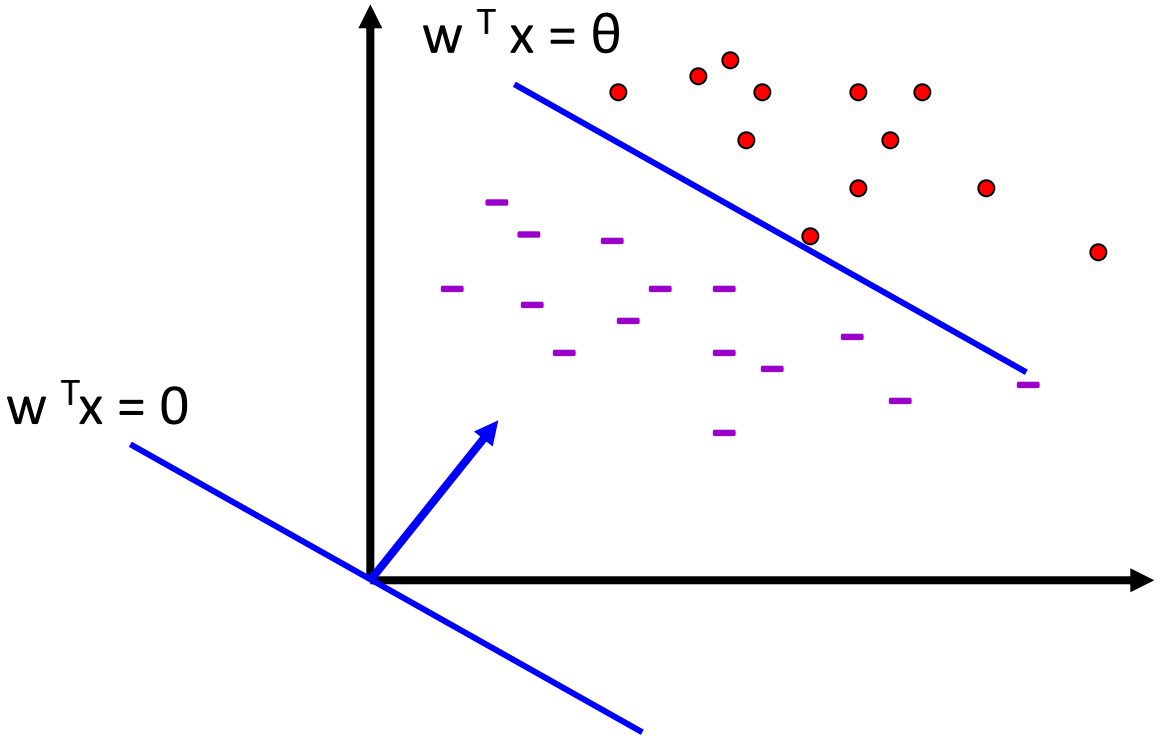
- *Input is a n-dimensional vector* ($\boldsymbol{x}$)
- *Output label* $y \in \{-1,1\}$

- *Linear threshold functions classify examples* ($\boldsymbol{x}$) *using a parameter vector* ($\boldsymbol{w}$) *and a real number* (b)

- $y = sign(\boldsymbol{w}^T\boldsymbol{x}+b) = sign(b + \sum_i w_i x_i)$
  - $\boldsymbol{w}^T\boldsymbol{x}+b \geq 0 \Rightarrow y=1$
  - $\boldsymbol{w}^T\boldsymbol{x}+b < 0 \Rightarrow y=-1$

Linear Model

A linear classifier represents a hyperplane (line in 2D), that separates the space into two half spaces.



Expressivity of Linear Functions

- **What can Linear Functions Represent?**
  - *Linear classifiers are an expressive hypothesis class*
- Many Boolean functions can be compactly represented using linear functions
  - We refer to it as *linear separablity*
- Some Boolean functions are not linearly separable

Expressivity of Linear Functions

- **Conjunctions** and **Disjunctions**
  - $y=x_1 \wedge x_2 \wedge x_3$ is equivalent to $y=1$ *if* $x_1 + x_2+x_3 \geq 3$

- Question:
  - How about disjunctions? $y=x_1 \vee x_2 \vee x_3$
  - $y=x_1 \vee x_2 \vee x_3$ is equivalent to $y=1$ if $x_1 + x_2+x_3 \geq 1$
  - How can you represent **negations**? $y=x_1 \wedge x_2 \wedge \neg x_3$
  - $y=x_1 \wedge x_2 \wedge \neg x_3$ is equivalent to $y= x_1 + x_2-x_3 \geq 2$

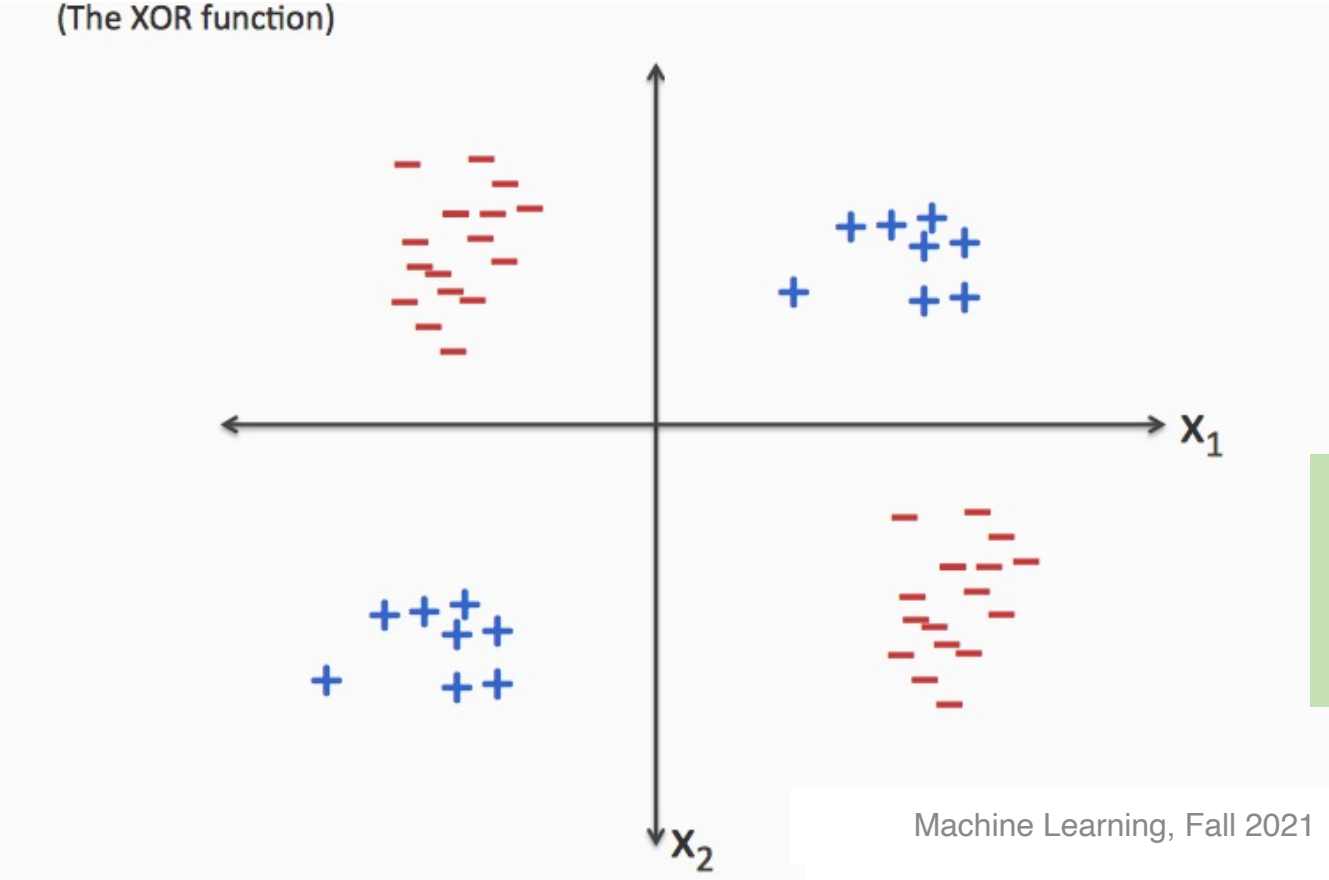
Expressivity of Linear Functions

- M-of-N rule
  - Pick N variables (N can be all of the variables or a subset)
  - Y=1 if at least M are active

- How can you represent 3 of $\{x_1,x_2,x_3,x_4,x_5\}$
  - As a Boolean function?
  - As a Linear function?
    - $x1+x_2+x_3+x_4+x_5 \geq 3$

Expressivity of Linear Functions

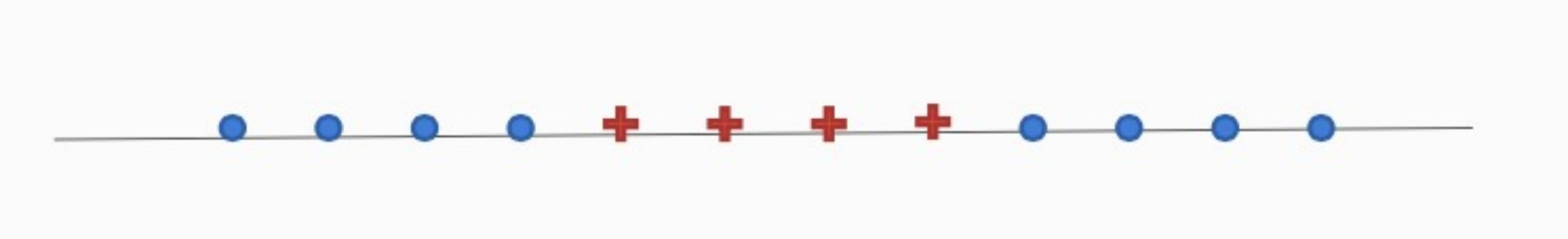
- ***Not*** *all functions are linearly separable*



Parity function (number of '1' is even) is another well-known example

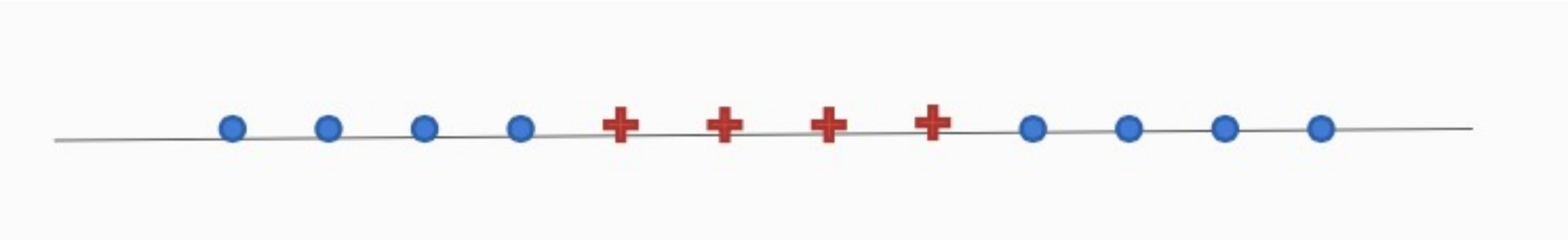
Expressivity of Linear Functions

- Non-Linear functions can be made linear
  - *We can change the representation of the input instances to represent functions that are not linear in the original space.*



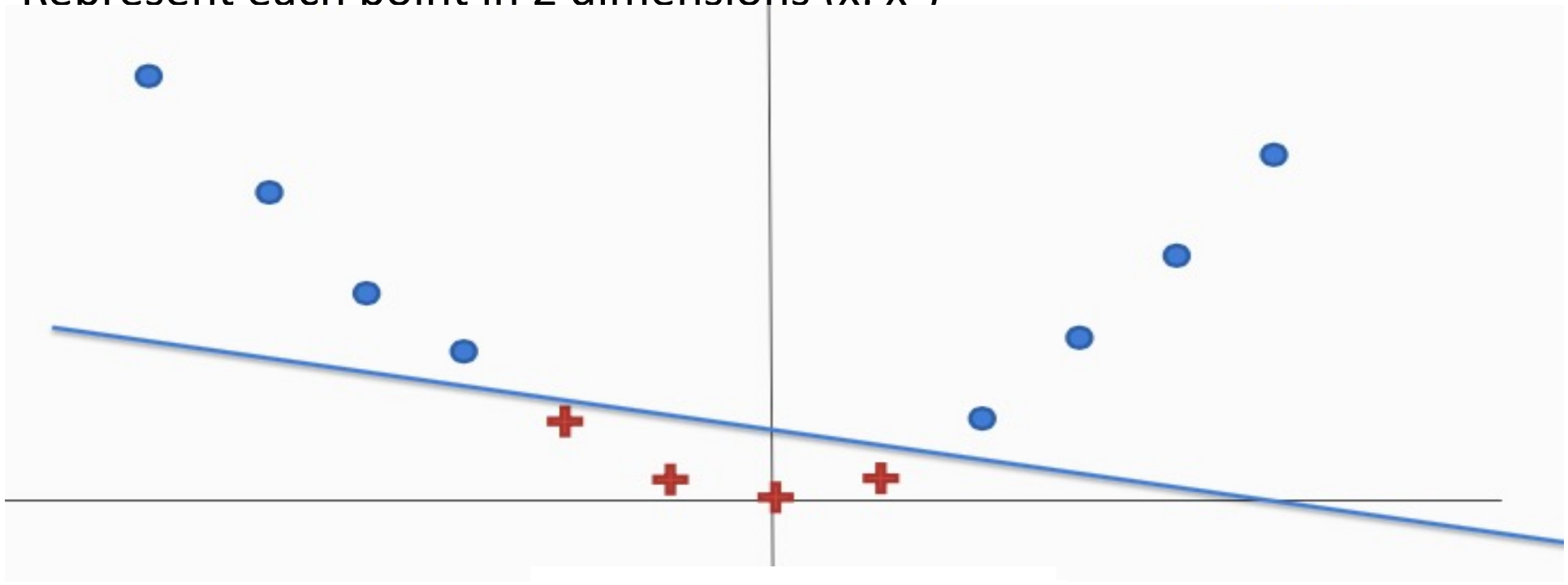
Expressivity of Linear Functions

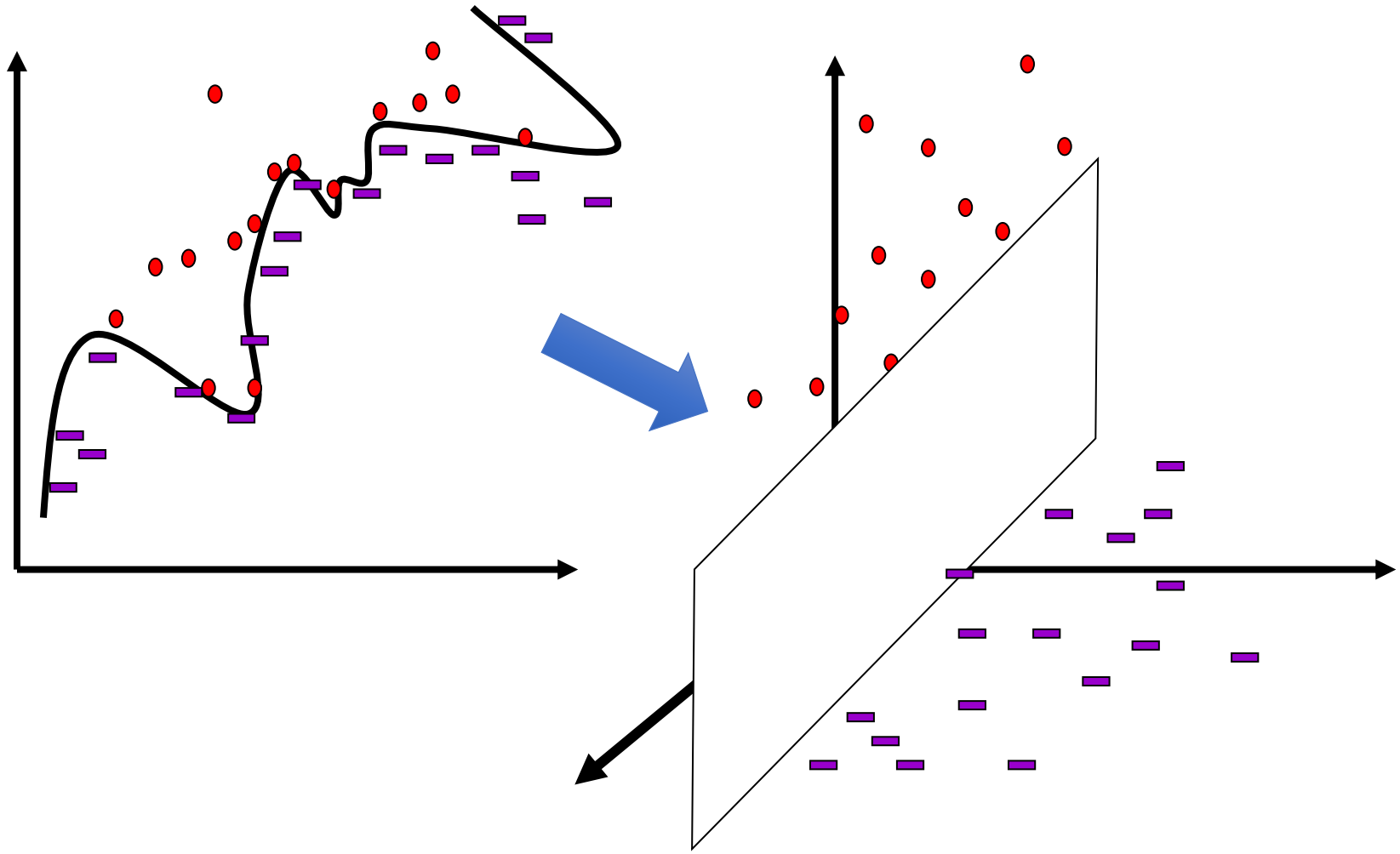
- **This function is not linearly separable**
- **Solution**: *use feature conjunctions*
  - Represent each point in 2 dimensions (x, x^2)



Expressivity of Linear Functions

- This function is not linearly separable
- Solution: use feature conjunctions
  - Represent each point in 2 dimensions (x, x^2)



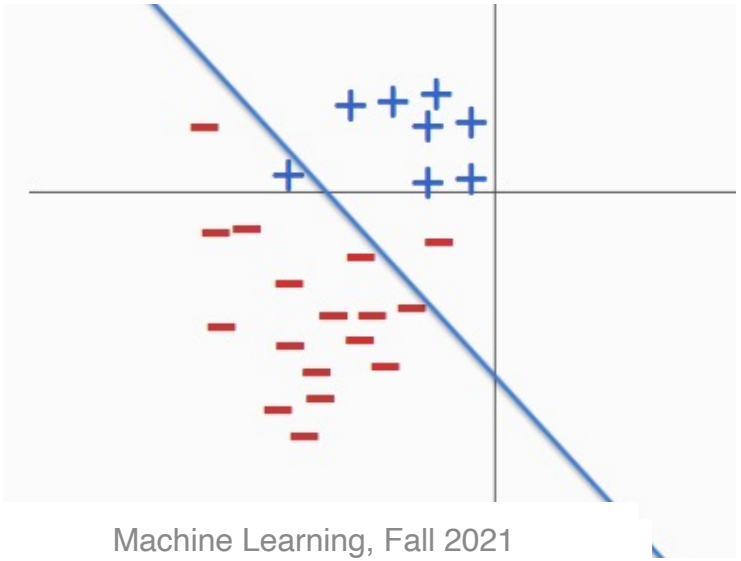


Question

*Which feature transformation would you use to make 2d **XOR** into a linearly separable function?*

Almost Linearly Separable Data

- In many cases the data is “almost” linearly separable
  - We can think about these examples as “noise”
  - *How much noise should we allow?*
  - *When should we change the expressivity of the learned function to account for it?*



Note on the Bias Term

- *To simplify notation we will include the bias term as an "always on" feature*
  - Instead of $b+w^Tx$ we concatenate b to x and set its value to 1
  - Increase the space dimension by 1
  - We can learn the right bias by tuning the value of that feature

Even if we don't explicitly mention it, remember that it is still there!

Learning Linear Functions

- We will look into several ways of learning linear functions
  - Both batch and online

- We will start with two *mistake driven* algorithms for linear functions
  - Winnow and Perceptron
  - Fun Fact*:*
    - *Perceptron is one of the first learning algorithms!*

Winnow

Initialize : $\theta = n;\ w_i = 1$

Prediction is 1 iff $w \bullet x \geq \theta$

If no mistake : do nothing

If $f(x) = 1$ **but** $w \bullet x < \theta$, $w_i \leftarrow 2w_i$ **(if** $x_i = 1$**) (promotion)**

If $f(x) = 0$ **but** $w \bullet x \geq \theta$, $w_i \leftarrow w_i/2$ **(if** $x_i = 1$**) (demotion)**

Only update the active features!

- The Winnow Algorithm learns Linear Threshold Functions.
  - We will prove a mistake bound for learning K-disjunctions with Winnow

Winnow Example

$f = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$

$\text{Initialize}: \quad \theta = 1024; w = (1,1,\ldots,1)$

Initialize weight to 1, threshold to *n*

Winnow Example

$f = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$

Initialize : $\theta = 1024; w = (1,1,...,1)$

$<(1,1,...,1),+>$	$w \bullet x \geq \theta$	$w = (1,1,...,1)$	ok
$<(0\ ,0,...,0),->$	$w \bullet x < \theta$	$w = (1,1,....,1)$	ok

So far no update..

Winnow Example

$f = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$

Initialize : $\theta = 1024; w = (1,1,...,1)$

$<(1,1,...,1),+>$	$w \bullet x \geq \theta$	$w = (1,1,...,1)$	ok
$<(0\ ,0,...,0),->$	$w \bullet x < \theta$	$w = (1,1,....,1)$	ok
$<(0,0,111,,,,0),->$	$w \bullet x < \theta$	$w = (1,1,....,1)$	ok
$<(1,0,0,...,0),+>$	$w \bullet x < \theta$	$w = (2,1,....,1)$	mistake

Mistake!
Dot product is **positive, but below the threshold**
(only x_1 is active..)

Winnow Example

$f = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$

Initialize : $\theta = 1024; w = (1,1,...,1)$

$<(1,1,...,1),+>$	$w \bullet x \geq \theta$	$w = (1,1,...,1)$	ok
$<(0\ ,0,...,0),->$	$w \bullet x < \theta$	$w = (1,1,....,1)$	ok
$<(0,0,111,,,,0),->$	$w \bullet x < \theta$	$w = (1,1,....,1)$	ok
$<(1,0,0,...,0),+>$	$w \bullet x < \theta$	$w = (2,1,....,1)$	mistake
$<(1,0,1,1,0..,0),+>$	$w \bullet x < \theta$	$w = (4,1,2,2...,1)$	mistake

Many variables are active now, but still not enough to go over the threshold

Winnow Example

$f = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$

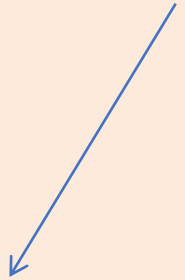
Initialize : $\theta = 1024; w = (1,1,...,1)$

$<(1,1,...,1),+>$	$w \bullet x \geq \theta$	$w = (1,1,...,1)$	ok
$<(0\ ,0,...,0),->$	$w \bullet x < \theta$	$w = (1,1,....,1)$	ok
$<(0,0,111,,,,0),->$	$w \bullet x < \theta$	$w = (1,1,....,1)$	ok
$<(1,0,0,...,0),+>$	$w \bullet x < \theta$	$w = (2,1,....,1)$	mistake
$<(1,0,1,1,0..,0),+>$	$w \bullet x < \theta$	$w = (4,1,2,2...,1)$	mistake
$<(1,0,1,0,0..,1),+>$	$w \bullet x < \theta$	$w = (8,1,4,2...,2)$	mistake

..... log(n/2) (for each good variable)

$w = (512,1,256,256,....,256)$

After log(n) mistakes for each "good" variable we stop making mistakes on positives



$<(1,0,1,0...,1),+>$	$w \bullet x \geq \theta$	$w = (512,1,256,256,....,256)$	ok	
$<(0,0,1,0.111..,0),->$	$w \bullet x \geq \theta$	$w = (512,1,0,..0,...,256)$	mistake	**(elimination version)**

.....

$w = (1024,1024,0,0,0,1,32,...,1024,1024)$ **(final hypothesis)**

Winnow Example

$f = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$

Initialize : $\theta = 1024; w = (1,1,...,1)$

$<(1,1,...,1),+>$	$w \bullet x \geq \theta$	$w = (1,1,...,1)$	ok
$<(0\ ,0,...,0),->$	$w \bullet x < \theta$	$w = (1,1,....,1)$	ok
$<(0,0,111,,,,0),->$	$w \bullet x < \theta$	$w = (1,1,....,1)$	ok
$<(1,0,0,...,0),+>$	$w \bullet x < \theta$	$w = (2,1,....,1)$	mistake
$<(1,0,1,1,0..,0),+>$	$w \bullet x < \theta$	$w = (4,1,2,2...,1)$	mistake
$<(1,0,1,0,0..,1),+>$	$w \bullet x < \theta$	$w = (8,1,4,2...,2)$	mistake

.....

$w = (512,1,256,256,....,256)$ log(n/2) (for each good variable)

$<(1,0,1,0...,1),+>$	$w \bullet x \geq \theta$	$w = (512,1,256,256,....,256)$	ok
$<(0,0,1,0.111..,0),->$	$w \bullet x \geq \theta$	$w = (512,1,0,..0,...,256)$	mistake

.....

$w = (1024,1024,0,0,0,1,32,...,1024,1024)$ **(final hypothesis)**

(elimination version)

And.. Mistakes on negatives (eliminations) don't effect the weights of "good" variables (why?)

Winnow – Mistake Bound

Claim: ***Winnow makes O(k log n) mistakes on k-disjunctions***

Initialize : $\theta = n;\ w_i = 1$

Prediction is 1 iff $w \bullet x \geq \theta$

If no mistake : do nothing

If $f(x) = 1$ **but** $w \bullet x < \theta$, $w_i \leftarrow 2w_i$ **(if** $x_i = 1$**) (promotion)**

If $f(x) = 0$ **but** $w \bullet x \geq \theta$, $w_i \leftarrow w_i/2$ **(if** $x_i = 1$**) (demotion)**

- **u** - # of mistakes on positive examples (promotions)
- **v** - # of mistakes on negative examples (demotions)

1. **u < k log(n)**

A weight that corresponds to a good variable is only promoted

When these weights get to n there will be no more mistakes on positives

Winnow – Mistake Bound

Claim: ***Winnow makes O(k log n) mistakes on k-disjunctions***

Initialize : $\theta = n;\ w_i = 1$

Prediction is 1 iff $w \bullet x \geq \theta$

If no mistake : do nothing

If $f(x) = 1$ **but** $w \bullet x < \theta$, $w_i \leftarrow 2w_i$ **(if** $x_i = 1$**) (promotion)**

If $f(x) = 0$ **but** $w \bullet x \geq \theta$, $w_i \leftarrow w_i/2$ **(if** $x_i = 1$**) (demotion)**

- **u** - # of mistakes on positive examples (promotions)
- **v** - # of mistakes on negative examples (demotions)

2. v < 2(u + 1)

Total weight (TW)= **n *initially*** (*we initialize weights to 1*)

Mistake on positive: TW(t+1) < TW(t) + n

Winnow – Mistake Bound

Claim: ***Winnow makes O(k log n) mistakes on k-disjunctions***

Initialize : $\theta = n;\ w_i = 1$

Prediction is 1 iff $w \bullet x \geq \theta$

If no mistake : do nothing

If f(x) = 1 but $w \bullet x < \theta$ **,** $w_i \leftarrow 2w_i$ **(if** $x_i = 1$**) (promotion)**

If f(x) = 0 but $w \bullet x \geq \theta$ **,** $w_i \leftarrow w_i/2$ **(if** $x_i = 1$**) (demotion)**

- **u** - # of mistakes on positive examples (promotions)
- **v** - # of mistakes on negative examples (demotions)

2. v < 2(u + 1)

Total weight (TW)= n initially

Mistake on positive: TW(t+1) < TW(t) + n

Updates after a mistake on positive:
Can promote less than n, *otherwise no mistake in the previous step*

Winnow – Mistake Bound

Claim: ***Winnow makes O(k log n) mistakes on k-disjunctions***

Initialize : $\theta = n;\ w_i = 1$

Prediction is 1 iff $w \bullet x \geq \theta$

If no mistake : do nothing

If f(x) = 1 but $w \bullet x < \theta$ **,** $w_i \leftarrow 2w_i$ **(if** $x_i = 1$**) (promotion)**

If f(x) = 0 but $w \bullet x \geq \theta$ **,** $w_i \leftarrow w_i/2$ **(if** $x_i = 1$**) (demotion)**

- **u** - # of mistakes on positive examples (promotions)
- **v** - # of mistakes on negative examples (demotions)

2. v < 2(u + 1)

Total weight (TW)= n initially

Mistake on positive: TW(t+1) < TW(t) + n

Mistake on negative: TW(t+1) < TW(t) - n/2

Mistakes on negative examples have to demote more than n/2, *otherwise the dot product will be below the threshold and we wouldn't make a mistake*

Winnow – Mistake Bound

Claim: ***Winnow makes O(k log n) mistakes on k-disjunctions***

Initialize : $\theta = n;\ w_i = 1$
Prediction is 1 iff $w \bullet x \geq \theta$
If no mistake : do nothing
If f(x) = 1 but $w \bullet x < \theta$, $w_i \leftarrow 2w_i$ **(if** $x_i = 1$**) (promotion)**
If f(x) = 0 but $w \bullet x \geq \theta$, $w_i \leftarrow w_i/2$ **(if** $x_i = 1$**) (demotion)**

- **u** - # of mistakes on positive examples (promotions)
- **v** - # of mistakes on negative examples (demotions)

2. v < 2(u + 1)

Total weight (TW)= n initially

Mistake on positive: TW(t+1) < TW(t) + n

Mistake on negative: TW(t+1) < TW(t) - n/2

0 < TW < n + u n - v n/2 ➔ **v < 2(u+1)**

TW is always positive which allow us to bound the number of negative mistakes

Winnow – Mistake Bound

Claim: ***Winnow makes O(k log n) mistakes on k-disjunctions***

Initialize : $\theta = n;\ w_i = 1$

Prediction is 1 iff $w \bullet x \geq \theta$

If no mistake : do nothing

If $f(x) = 1$ **but** $w \bullet x < \theta$, $w_i \leftarrow 2w_i$ **(if** $x_i = 1$**) (promotion)**

If $f(x) = 0$ **but** $w \bullet x \geq \theta$, $w_i \leftarrow w_i/2$ **(if** $x_i = 1$**) (demotion)**

- **u** - # of mistakes on positive examples (promotions)
- **v** - # of mistakes on negative examples (demotions)

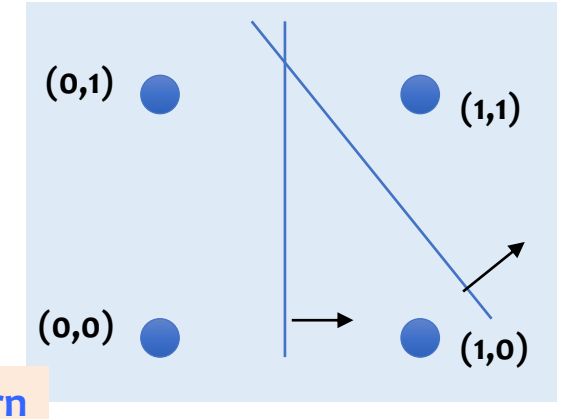
# of mistakes: $u + v < 3u + 2 = O(k \log n)$

Mission Accomplished! Efficient algorithm with similar mistake bound as the Halving algorithm

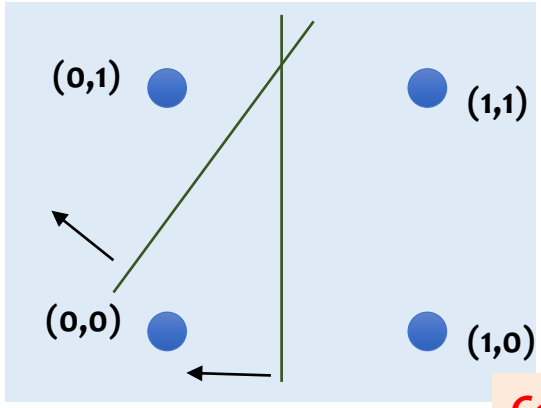
➔ **What did we just show**? *Recall definition of Mistake bounds*

What can Winnow represent?

- The version of Winnow we saw cannot represent all disjunctions. **Why?**
- In fact, it can only represent monotone functions
  - **Multiplicative updates cannot change the sign of the weights (no negative weights)**



Can learn
x_1 V x_2
x_2



Cannot learn
x_1 V !x_2
!x_2

Winnow Extension

- The algorithm we saw learns **monotone functions**
- For the general case: *Duplicate variables*
- For the negation of variable x, introduce a new variable x^{neg}.
- Learn monotone functions over 2n variables ***(down side?)***
- Balanced version:
  - Keep two weights for each variable; effective weight is the difference

Update Rule :

If $f(x)=1$ but $(w^+ - w^-)\bullet x \le \theta$, $w_i^+ \leftarrow 2w_i^+$ $w_i^- \leftarrow \frac{1}{2} w_i^-$ where $x_i = 1$ (promotion)

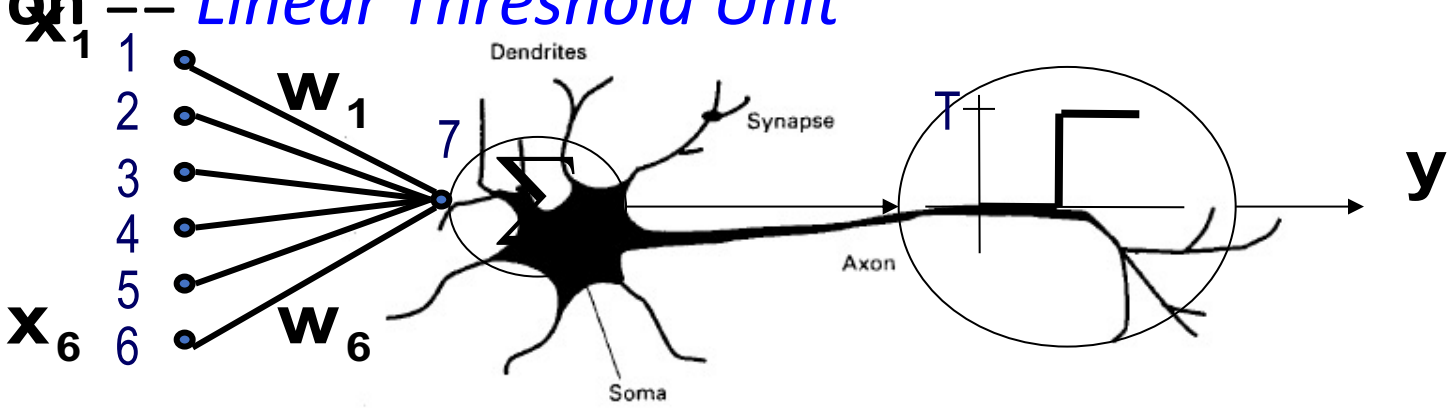
If $f(x)=0$ but $(w^+ - w^-)\bullet x \ge \theta$, $w_i^+ \leftarrow \frac{1}{2} w_i^+$ $w_i^- \leftarrow 2w_i^-$ where $x_i = 1$ (demotion)

Summary

- **Learning Linear Function**
  - Very popular choice when the number of attributes is high
  - Expressive, but not all functions can be represented
    - Real world examples?
    - Several ways to deal with limited expressivity
      - Simplest: change the input space, rethink feature choices
- **Winnow**
  - First Learning algorithm for linear functions
    - Multiplicative updates
  - Applicable when concept depends on few relevant attributes
  - Nice theoretical properties: mistake bound only weakly depends on number of attributes

Perceptron Learning Algorithm

- 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

- We learn f:X→ {-1,+1} represented as f =sgn{w•x)
- Where X= $\{0,1\}^n$ or X= R^n and w ² R^n
- Given Labeled examples: $\{(x_1, y_1), (x_2, y_2),\ldots(x_m, y_m)\}$

1. Initialize w=0 ϵ $\mathbf{R}^n$
2. Cycle through all examples

   a. Predict the label of instance x to be y' = sgn{w•x)

   b. If *y'≠y*, update the weight vector:

   w = w + r y x (r - a constant, learning rate)

   Otherwise, if y'=y, leave weights unchanged.

Next time – *More on perceptron*

- We will see that the small change has strong implications
  - Multiplicative vs. additive update
  - **Mistake bound for perceptron**
- Also-
  - **Practical considerations**
    - How to get perceptron to work? What can be tuned?