Machine Learning

Mistake Driven Learning

Dan Goldwasser

dgoldwas@purdue.edu

Mistake driven learning using Winnow and Perceptron

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

We will introduce and analyze two mistake-driven algorithms for learning
Linear functions

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

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.

Linear Models

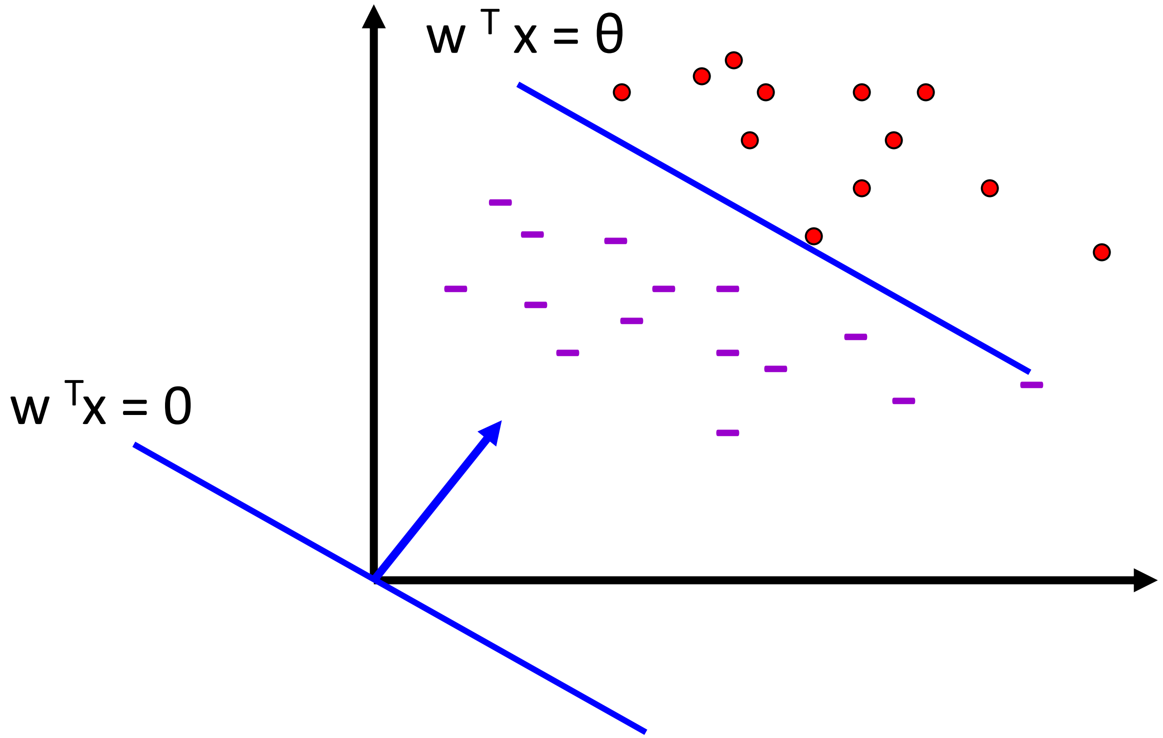
- *Input is a n-dimensional vector* (**x**)
- *Output label* y∈{-1,1}

- *Linear threshold functions classify examples (**x**) using a parameter vector (**w**) and a real number (b)*

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

Linear Model

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



*To simplify notation we will include the **bias term** as an "always on" feature*

Even if it is not explicitly mentioned in the slides,

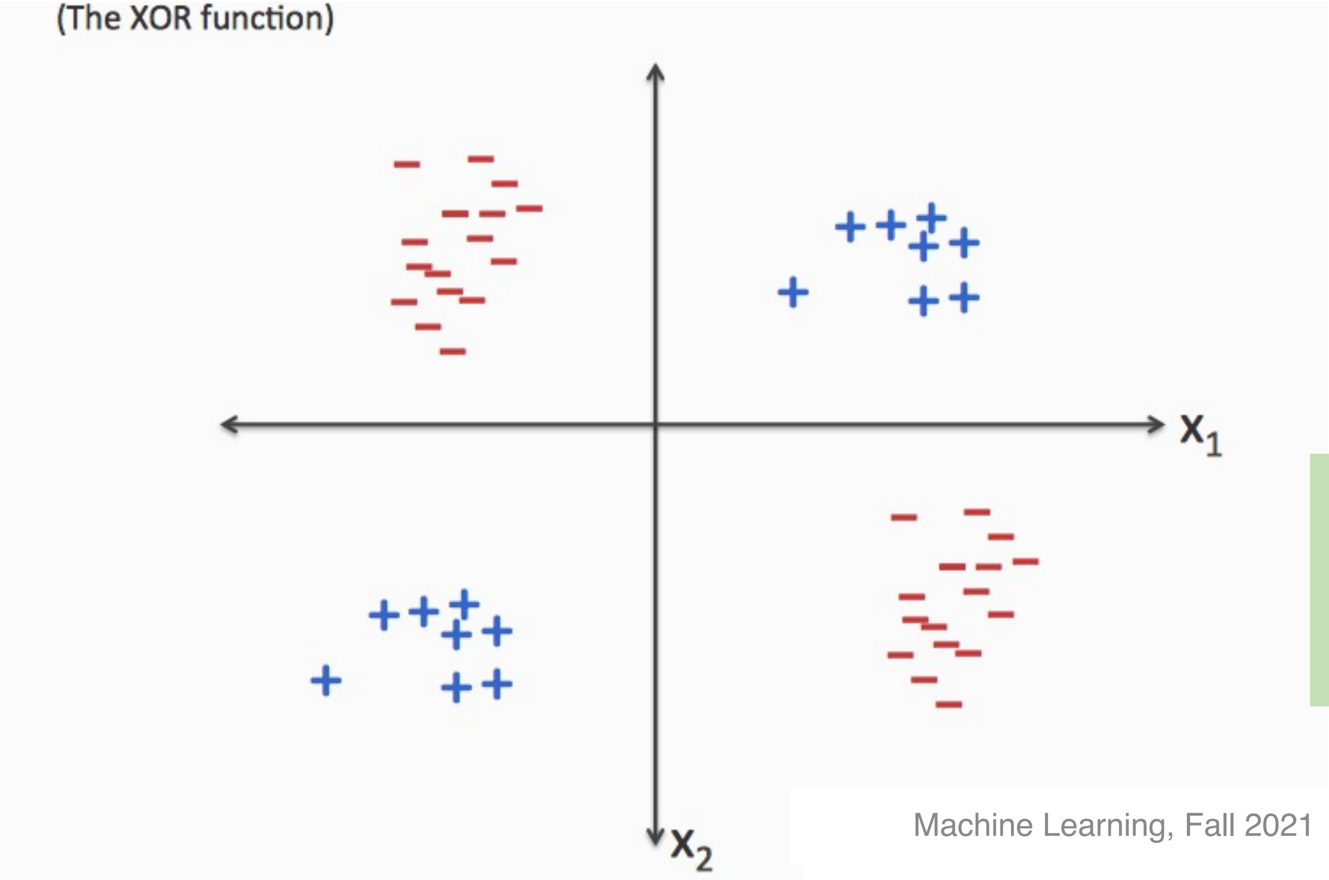
$w^T x \geq 0$ includes the bias term

Expressivity of Linear Functions

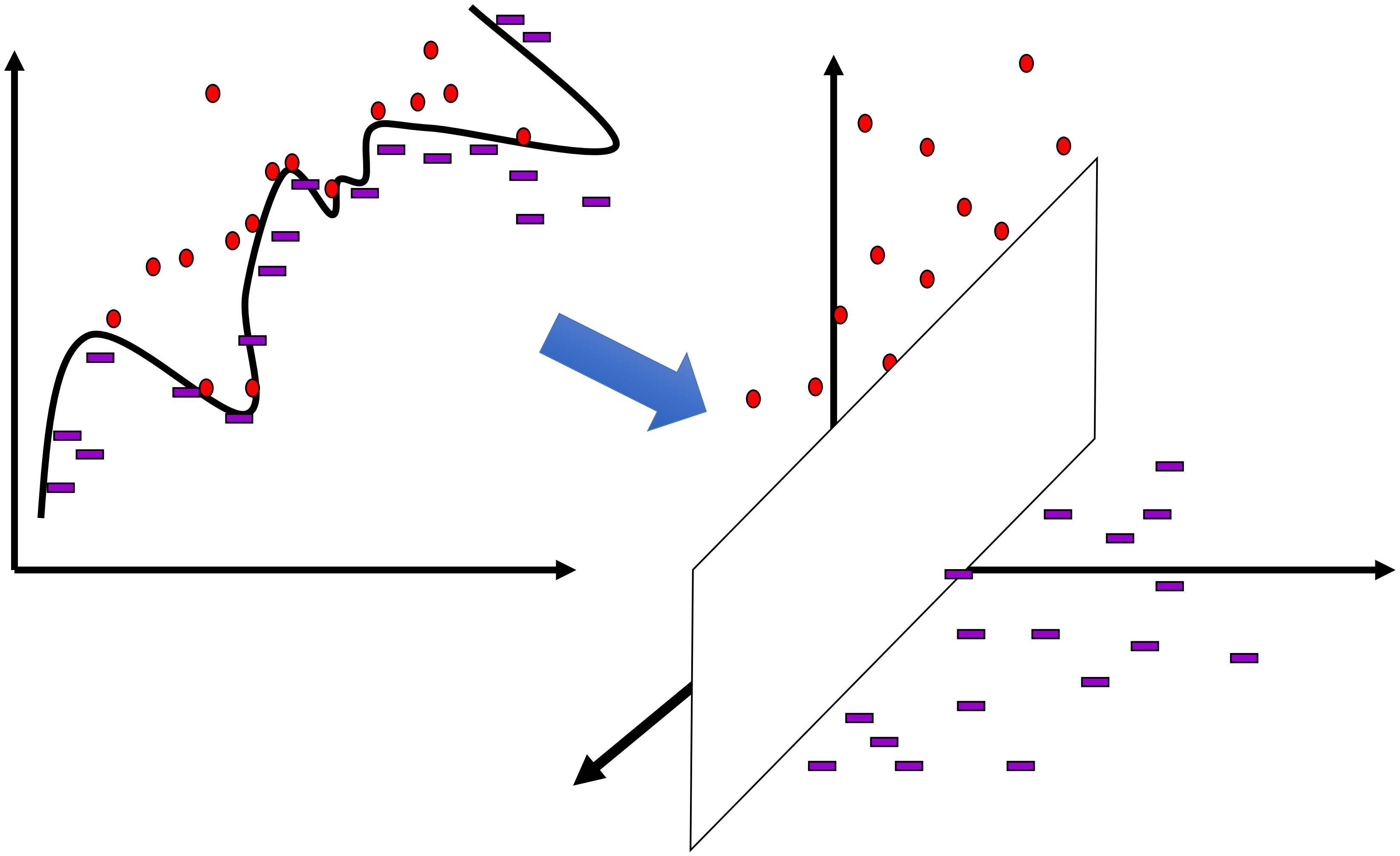
- **What can Linear Functions Represent?**
  - *Linear classifiers are an expressive hypothesis class*
- Many Boolean functions can be compactly represented using linear functions
  - We refer to it as *linear separablity*
- Some Boolean functions are not linearly separable

Expressivity of Linear Functions

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

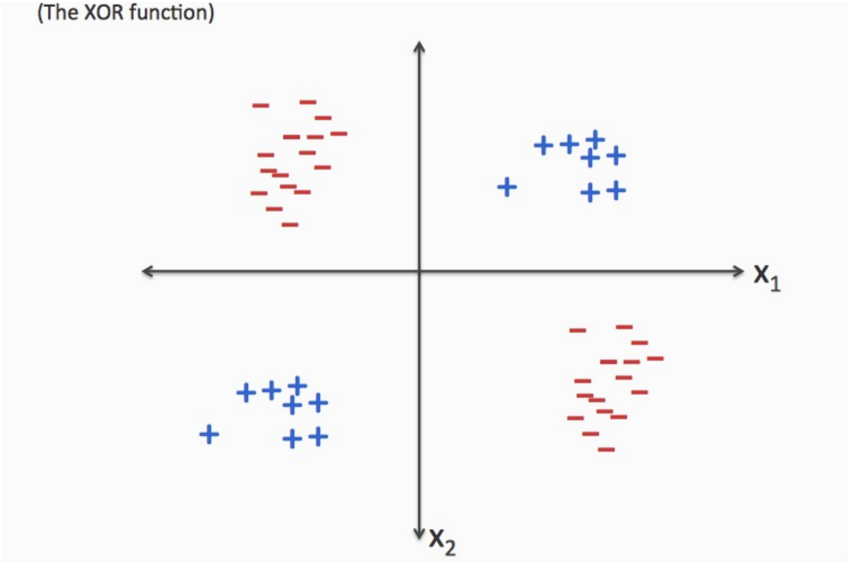


Parity function (number of ‘1’ is even) is another well-known example



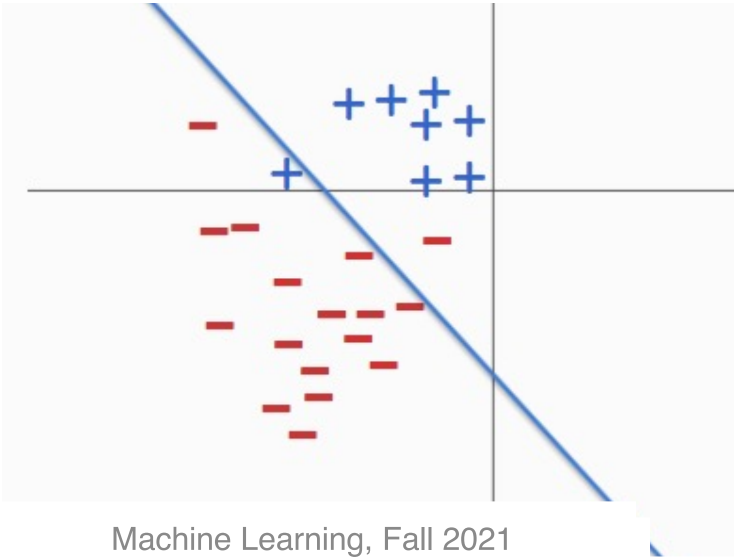
Question

*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?*



Winnow

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

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

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

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

$<(1,1,\ldots,1),+>$	$w \bullet x \geq \theta$	$w = (1,1,\ldots,1)$	ok
$<(0\ ,0,\ldots,0),->$	$w \bullet x < \theta$	$w = (1,1,\ldots.,1)$	ok
$<(0,0,111,,,,0),->$	$w \bullet x < \theta$	$w = (1,1,\ldots.,1)$	ok
$<(1,0,0,\ldots,0),+>$	$w \bullet x < \theta$	$w = (2,1,\ldots.,1)$	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 : $\theta = 1024; w = (1,1,\dots,1)$

$<(1,1,\dots,1),+> \quad w \bullet x \geq \theta \quad w = (1,1,\dots,1)$ ok

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

$<(0,0,111,,,,0),-> \quad w \bullet x < \theta \quad w = (1,1,\dots.,1)$ ok

$<(1,0,0,\dots,0),+> \quad w \bullet x < \theta \quad w = (2,1,\dots.,1)$ mistake

$<(1,0,1,1,0..,0),+> \quad w \bullet x < \theta \quad w = (4,1,2,2\dots,1)$ mistake

$<(1,0,1,0,0..,1),+> \quad w \bullet x < \theta \quad w = (8,1,4,2\dots,2)$ mistake

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

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

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

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

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

....

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

Winnow – Mistake Bound

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

$$\begin{aligned}
&\textbf{Initialize}: \theta = n;\ w_i = 1 \\
&\textbf{Prediction is 1 iff } w \bullet x \geq \theta \\
&\textbf{If no mistake: do nothing} \\
&\textbf{If } f(x) = 1 \textbf{ but } w \bullet x < \theta,\quad w_i \leftarrow 2w_i \ (\text{if } x_i = 1)\ (\text{promotion}) \\
&\textbf{If } f(x) = 0 \textbf{ but } w \bullet x \geq \theta,\quad w_i \leftarrow w_i/2\ (\text{if } x_i = 1)\ (\text{demotion})
\end{aligned}$$

- **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***

$$\begin{aligned}
&\textbf{Initialize}: \theta = n;\ \mathbf{w}_i = 1 \\
&\textbf{Prediction is 1 iff } \mathbf{w} \bullet \mathbf{x} \geq \theta \\
&\textbf{If no mistake: do nothing} \\
&\textbf{If } f(x) = 1 \textbf{ but } \mathbf{w} \bullet \mathbf{x} < \theta,\ \mathbf{w}_i \leftarrow 2\mathbf{w}_i \ (\text{if } x_i = 1)\ (\text{promotion}) \\
&\textbf{If } f(x) = 0 \textbf{ but } \mathbf{w} \bullet \mathbf{x} \geq \theta,\ \mathbf{w}_i \leftarrow \mathbf{w}_i/2 \ (\text{if } x_i = 1)\ (\text{demotion})
\end{aligned}$$

- **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*)

Winnow – Mistake Bound

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

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

- **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***

$$\begin{array}{l}
\textbf{Initialize}: \theta = n;\ \mathbf{w}_i = 1 \\
\textbf{Prediction is 1 iff } \mathbf{w} \bullet \mathbf{x} \geq \theta \\
\textbf{If no mistake: do nothing} \\
\textbf{If } f(x) = 1 \textbf{ but } \mathbf{w} \bullet \mathbf{x} < \theta,\ \mathbf{w}_i \leftarrow 2\mathbf{w}_i \ (\text{if } x_i = 1)\ (\textbf{promotion}) \\
\textbf{If } f(x) = 0 \textbf{ but } \mathbf{w} \bullet \mathbf{x} \geq \theta,\ \mathbf{w}_i \leftarrow \mathbf{w}_i/2 \ (\text{if } x_i = 1)\ (\textbf{demotion})
\end{array}$$

- **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***

$$
\begin{array}{l}
\textbf{Initialize}: \theta = n;\ \mathbf{w}_i = 1 \\
\textbf{Prediction is 1 iff } \mathbf{w} \bullet \mathbf{x} \geq \theta \\
\textbf{If no mistake : do nothing} \\
\textbf{If } f(x) = 1 \textbf{ but } \mathbf{w} \bullet \mathbf{x} < \theta,\quad \mathbf{w}_i \leftarrow 2\mathbf{w}_i \quad (\text{if } x_i = 1)\ (\textbf{promotion}) \\
\textbf{If } f(x) = 0 \textbf{ but } \mathbf{w} \bullet \mathbf{x} \geq \theta,\quad \mathbf{w}_i \leftarrow \mathbf{w}_i/2 \quad (\text{if } x_i = 1)\ (\textbf{demotion})
\end{array}
$$

- **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

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

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

Winnow – Mistake Bound

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

$$\begin{array}{l}
\textbf{Initialize}: \theta = n;\ \mathbf{w}_i = 1 \\
\textbf{Prediction is 1 iff } \mathbf{w} \bullet \mathbf{x} \geq \theta \\
\textbf{If no mistake: do nothing} \\
\textbf{If } f(\mathbf{x}) = 1 \textbf{ but } \mathbf{w} \bullet \mathbf{x} < \theta,\quad \mathbf{w}_i \leftarrow 2\mathbf{w}_i \ (\textbf{if } x_i = 1)\ (\textbf{promotion}) \\
\textbf{If } f(\mathbf{x}) = 0 \textbf{ but } \mathbf{w} \bullet \mathbf{x} \geq \theta,\quad \mathbf{w}_i \leftarrow \mathbf{w}_i/2 \ (\textbf{if } x_i = 1)\ (\textbf{demotion})
\end{array}$$

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

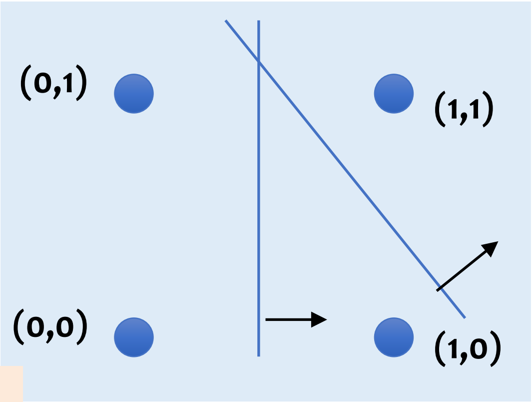
# of mistakes: $u + v < 3u + 2 = O(k \log n)$

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

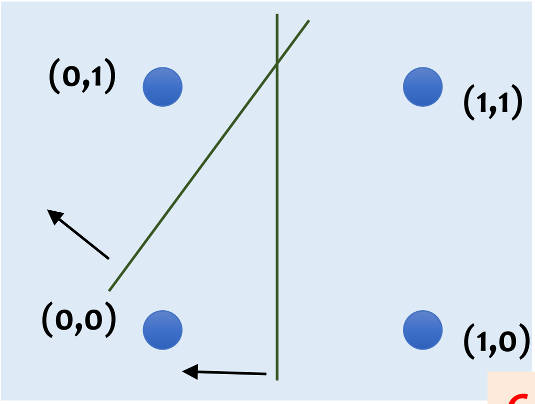
➔ **What did we just show**? *Recall definition of Mistake bounds*

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
$x_1 \vee x_2$
x_2



Cannot learn
$x_1 \vee !x_2$
$!x_2$

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

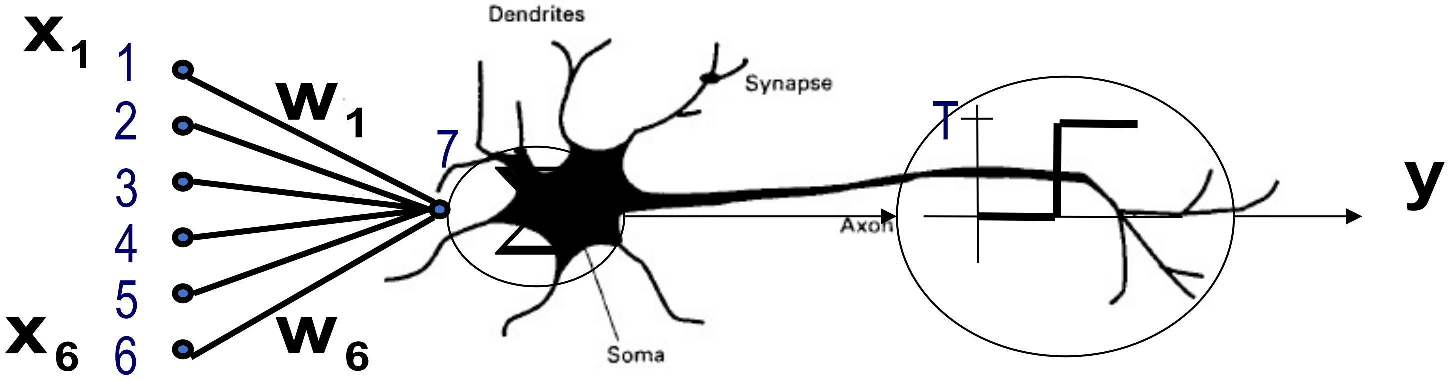
If $f(x)=0$ but $(w^+ - w^-)\bullet x \ge \theta$, $\quad w_i^+ \leftarrow \frac{1}{2}w_i^+ \quad w_i^- \leftarrow 2w_i^-$ where $x_i = 1$ (demotion)

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 <u>Perceptron learning rule</u>
- **Perceptron** == *Linear Threshold Unit*



Perceptron

- We learn $f: X \rightarrow \{-1,+1\}$ represented as $f = \text{sgn}\{w \bullet x)$
- 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 $w=0 \in \mathbf{R}^{\mathbf{n}}$
2. Cycle through all examples

   a. Predict the label of instance x to be $y' = \text{sgn}\{w \bullet x)$

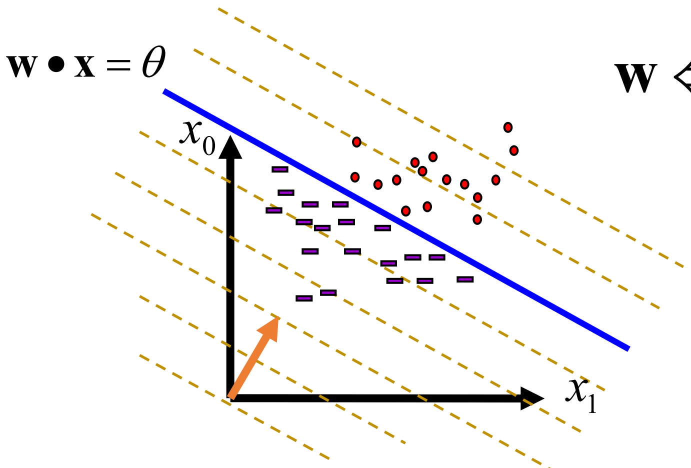
   b. If $y' \neq y$, update the weight vector:

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

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

Footnote About the Threshold

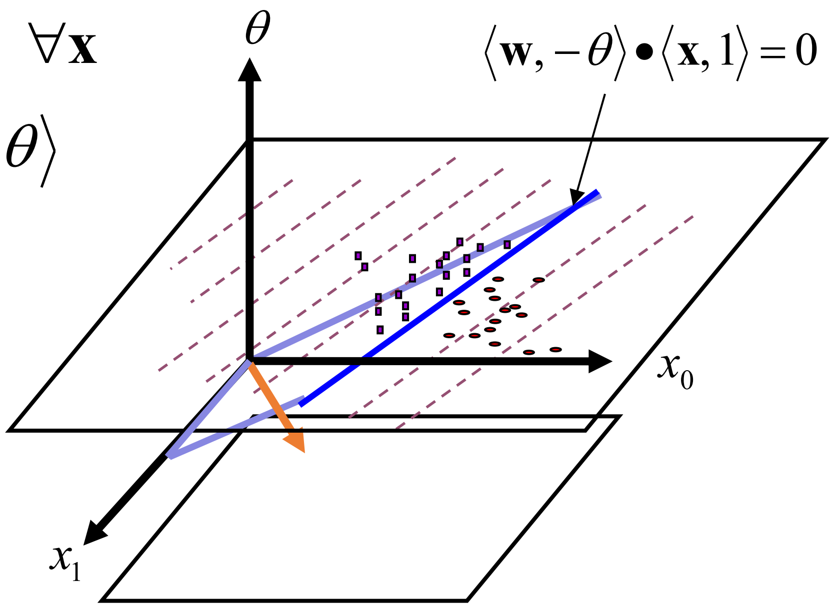
- On previous slide, Perceptron has no threshold
- But we don't lose generality:

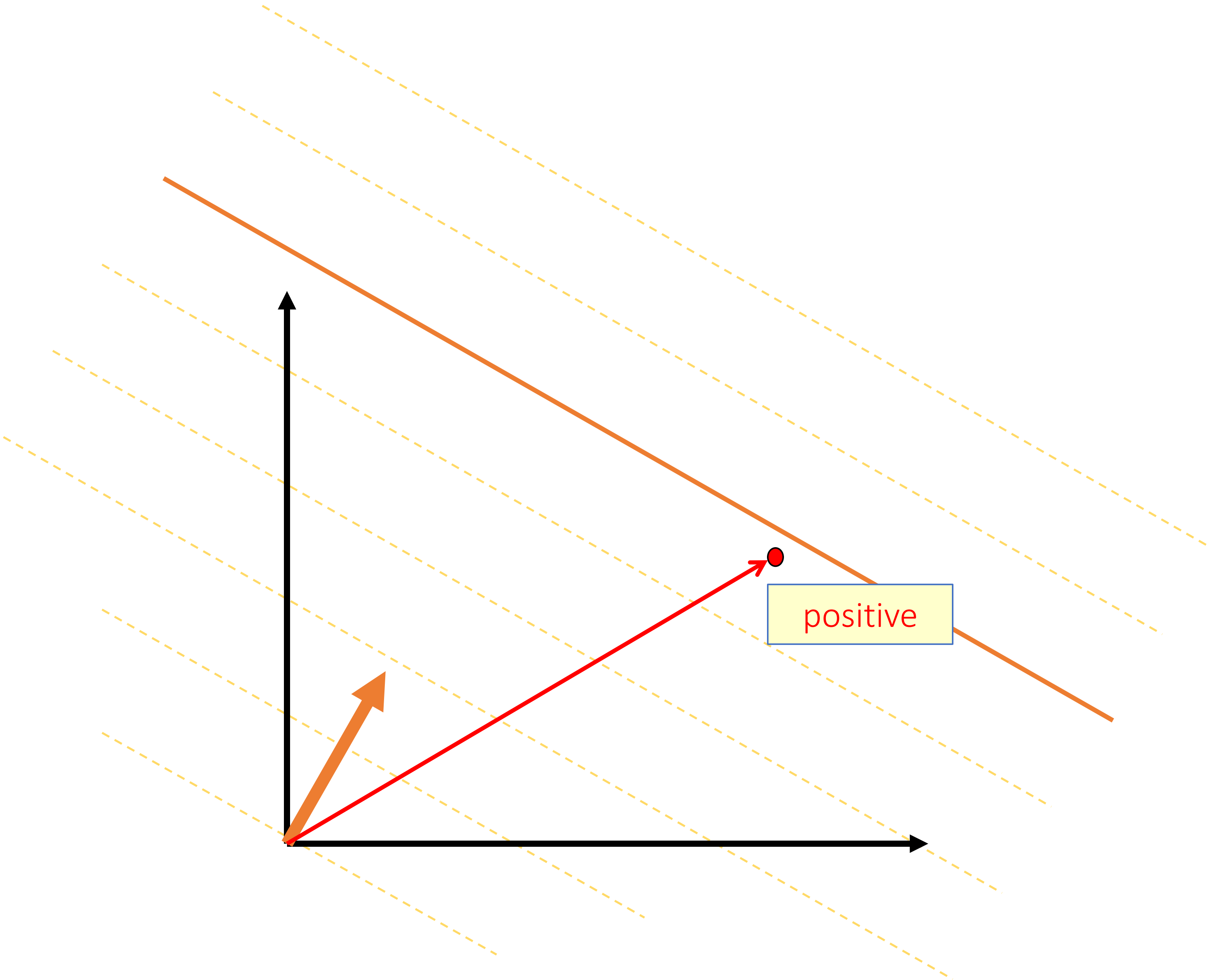


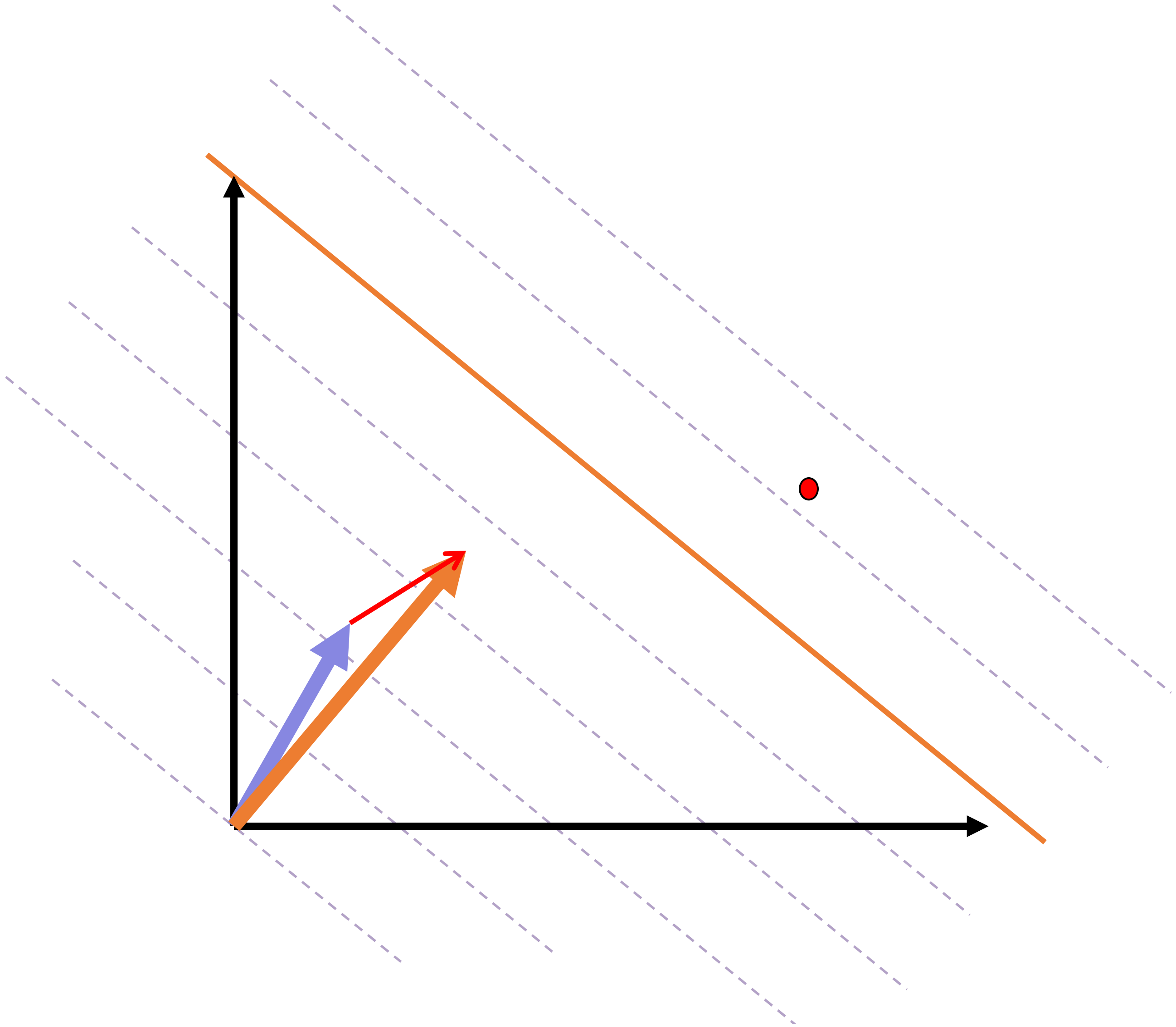
$$\mathbf{x} \Leftrightarrow \langle \mathbf{x}, 1 \rangle \quad \forall \mathbf{x}$$

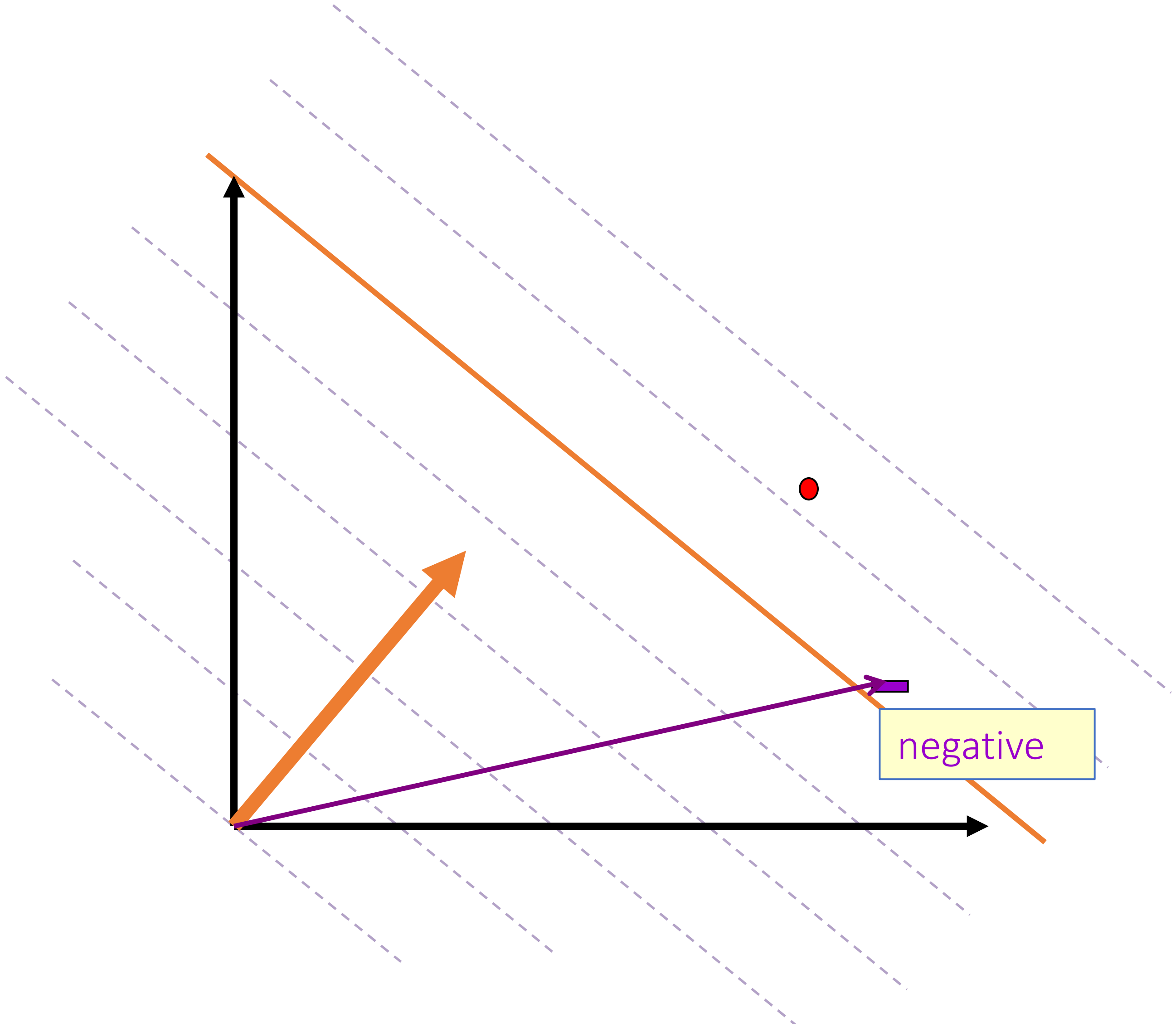
$$\mathbf{w} \Leftrightarrow \langle \mathbf{w}, -\theta \rangle$$

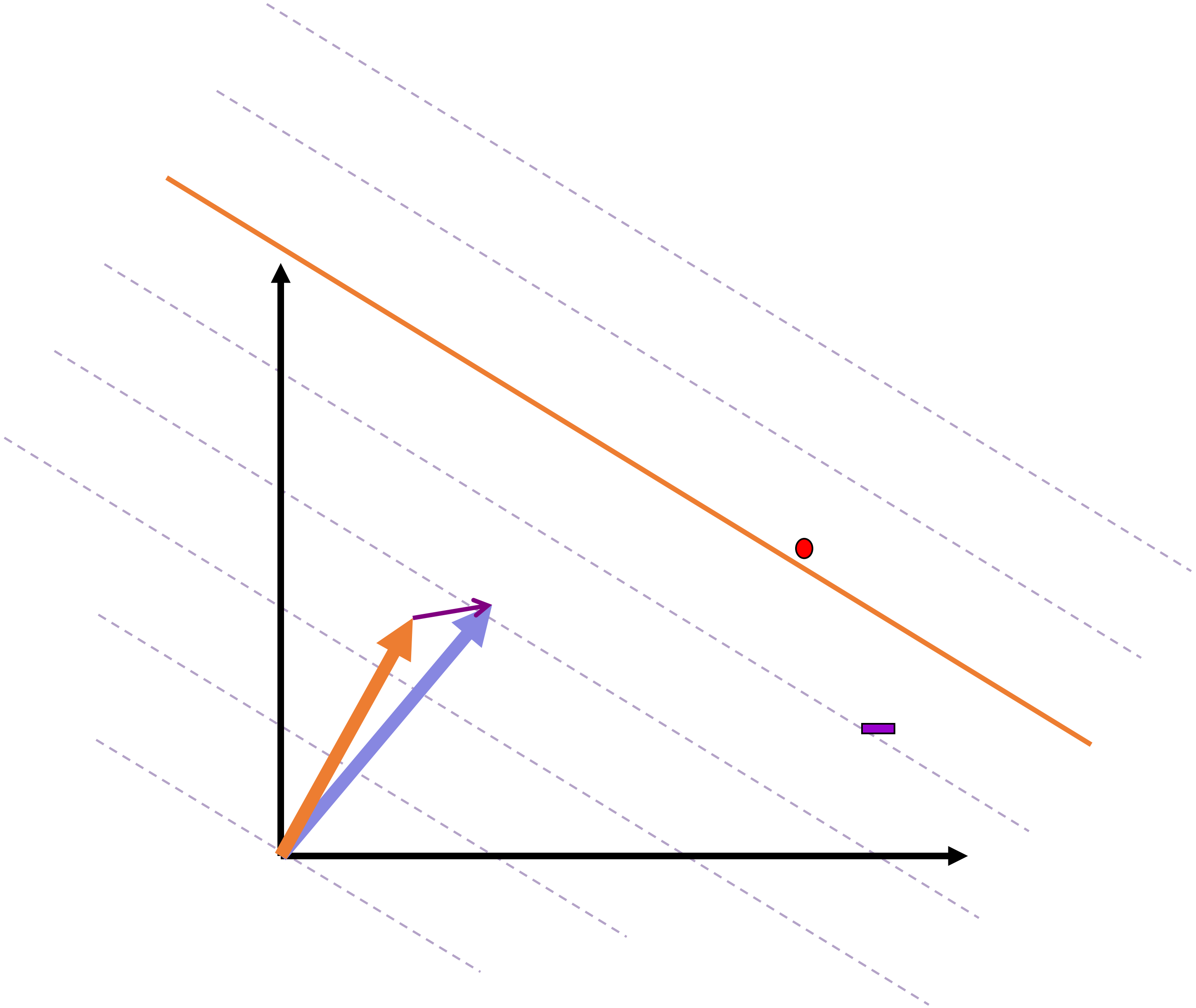
$$\Leftrightarrow$$




positive




negative



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 ?** *(will know soon)*

- **Perceptron Cycling Theorem:**

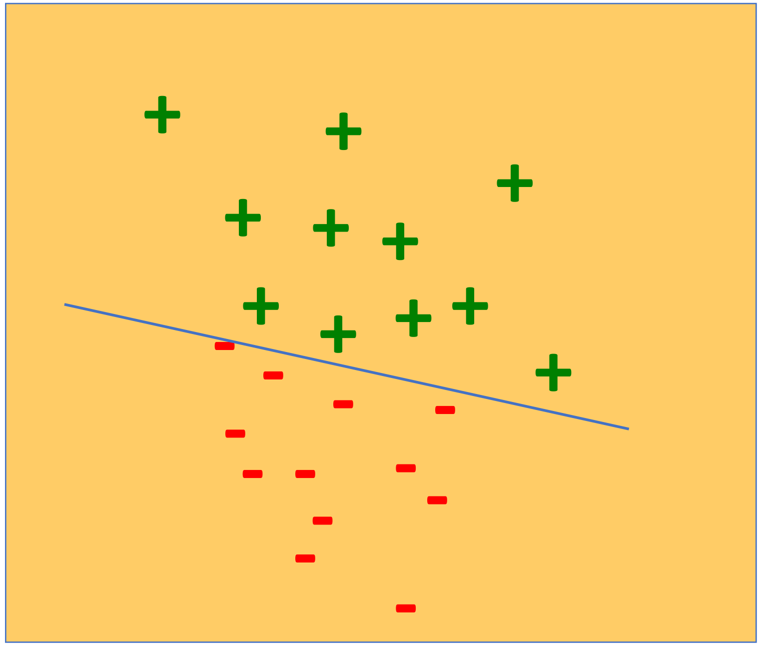
If the training data is **not linearly separable** the perceptron learning algorithm will eventually repeat the same set of weights and therefore **enter an infinite loop**.

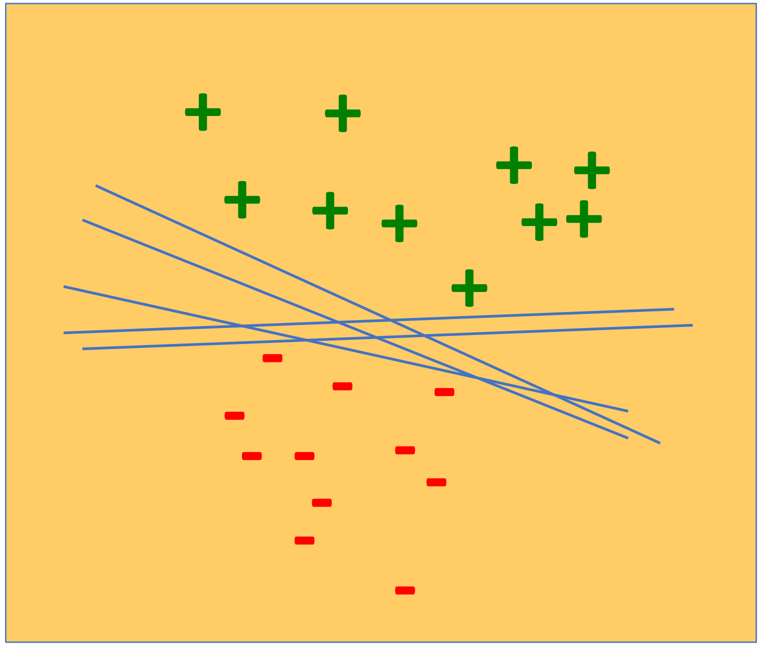
  - **How to provide robustness, more expressivity ?**

Perceptron Convergence

- **We have just learned**: *If the data is linearly separable: perceptron will converge*
- We would like to have a bound on the number of mistakes
  - **Intuition:** *The perceptron algorithm should converge faster on easier problems*
- Can we come up with a way to describe easy and hard problems?

Which one is *easier*?





You can quantify difficulty using the notion of **margin**, the distance between the hyperplane (*w*) and the nearest point.

Problems for which there are w with large margins should be ???

Margin

- Given a dataset D, a weight vector w that separates D
- **Margin(w,D)**: distance between w and the nearest point

$$\min_{(x,y)\in D} y(wx)/|||w||$$

  - *Essentially the point with minimum activation (absolute value)*
    - *Similar to distance when* ||w|| = 1 (*functional margin = geometric margin*)

- **The margin of a dataset (Margin(D))**

$$margin(D) = \sup_w margin(D,w)$$

  - *The margin of a data set is the largest attainable margin on this data*
- *Perceptron mistake bound depends on Margin(D)*

Perceptron Mistake Bound

- Let D be a linearly separable data set with a margin $\gamma > 0$. Assume the $||x|| \leq 1$ for all $x \in D$. Then the algorithm will converge after at most $1/\gamma^2$

- **Linearly Separable**
  - Exists a "good" weight vector v
  - *Pick v s.t $||v||=1$*
- **Scaling of examples to a norm of 1**
  - Scaling to $||x||=1$ doesn't change sign (wx)
- **Set learning rate to 1**

Proof (1)

- $$\cos(w,v) = \frac{w \cdot v}{\| w \| \| v \|} = \frac{w \cdot v}{\| w \|}$$

We assume the norm of ||v||=1

- Cosine is bounded by one
- Let's bound wv when a mistake is made

- **Each mistake:** $w^{new}=w^{old}+yx$
- $v \cdot w^{new} = v \cdot (w^{old}+yx) \geq vw^{old}+\gamma$
- $v \cdot w^{new} \geq \gamma$ (#mistakes)

Since ɣ minimizes the dot product vx

By induction; since $w^0 = 0$

Proof (2)

- *Now let's bound* $\| w \|$ *after a mistake is made*

$$\| w^{new} \|^2 = \| w^{old} + yx \|^2$$

$$= \| w^{old} \|^2 + 2y(w^{old} x) + \| x \|^2$$

$$\leq \| w^{old} \|^2 + 1$$

$$Therefore, \| w^{new} \|^2 \leq (\# mistakes)$$

$$\| w^{new} \| \leq \sqrt{(\# mistakes)}$$

Assumption :
x·x=1

Since a mistake was made:
$w^{old}x \le 0$

By induction since w init to 0

Proof (2)

- Thus we have the inequalities:

$$1 \geq \frac{v \cdot w}{\| w \|} \geq \frac{\gamma(\# mistakes)}{\sqrt{(\# mistakes)}}$$

$$(\# mistakes) \leq \frac{1}{\gamma^2}$$

Proof (2)

- Thus we have the inequalities:

$$1 \geq \frac{v \cdot w}{\| w \|} \geq \frac{\gamma(\# mistakes)}{\sqrt{(\# mistakes)}}$$

$$(\# mistakes) \leq \frac{1}{\gamma^2}$$

$$(\# mistakes) \leq \frac{R^2}{\gamma^2}$$

We can also relax the assumption that ||x||=1 and assume instead - ||x|| = R

Perceptron in Practice: *order of examples*

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

   a. Predict the label of instance x to be $y' = \text{sgn}\{w \bullet x)$

   b. If $y' \neq y$, update the weight vector:

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

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

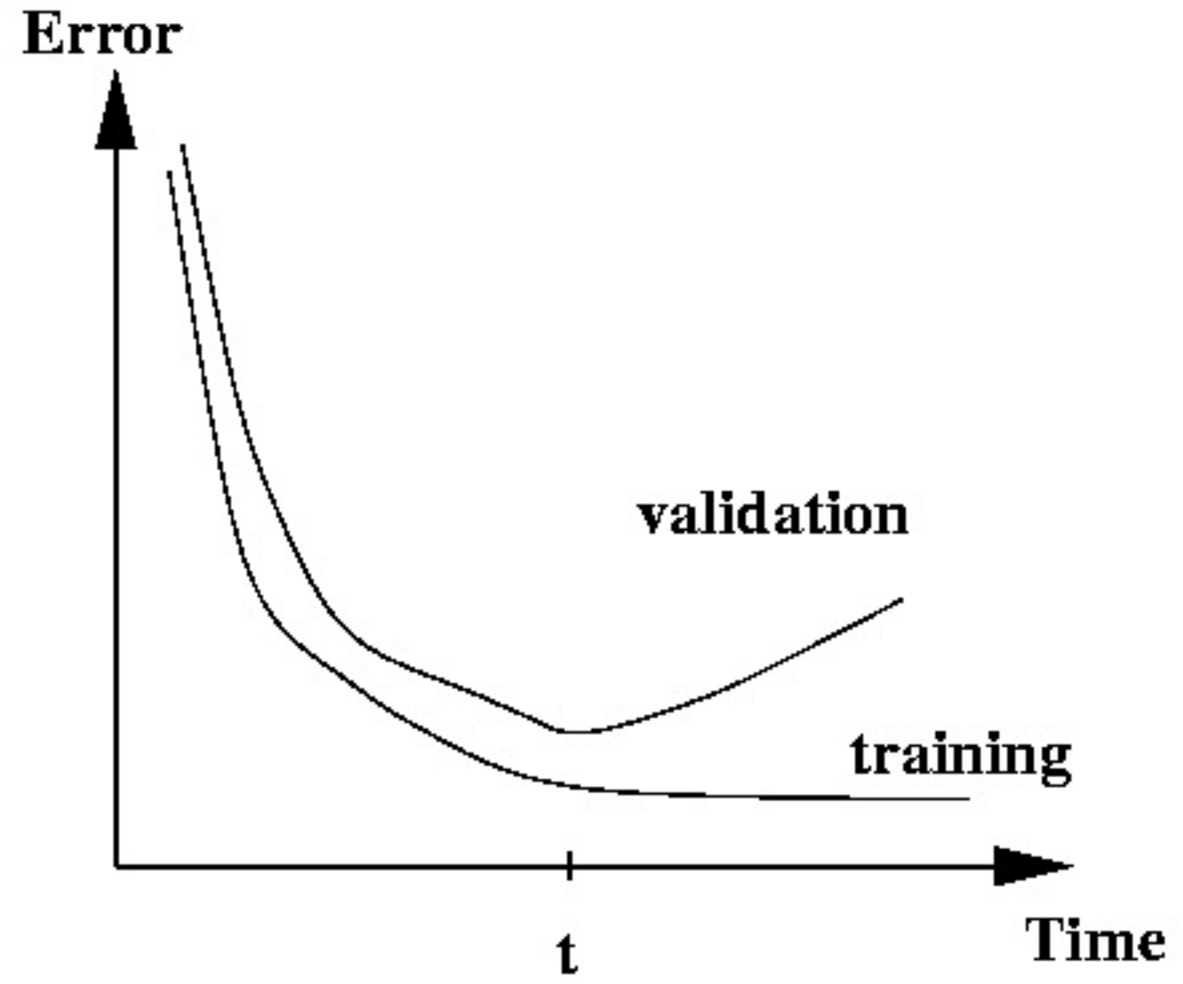
- The perceptron algorithm iterates over the data. **Does the order of examples matter?**
- Example: we see ***500 positive*** examples and **500 negative** examples
  - ***What will perceptron learn?***

Perceptron Convergence

- **We have just learned**: *If the data is linearly separable: perceptron will converge*
- What happens if the data is not linearly separable?
  - Option 1*: all is lost..*
  - *Option 2: **how much do we really care about the training error anyway!?***

- ***What is a good practical measure of when to stop?***

Overfitting an the Perceptron Algorithm



Perceptron in Practice: hyperparameters

$\in$

1. Initialize w=0 $\mathbf{R}^n$
2. Cycle **max_iter** times through the 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.

- Iterating over the data until convergence might not be a good idea (***why***?)
- Introduce a new hyperparamter: max_iter
- *How can we find the best assignment for it?*

Footnote: Tuning Hyperparameters

- Hyperprameter are parameters of the learning algorithm
  - Control the operation of the algorithm
  - “Wrong” assignment can lead to poor generalization
- *How can you assign “good” values to hyper-parameter?*
  - *Use the validation set*
- ***What are the hyper-parameters of the Perceptron algorithm?***

Regularization: *Perceptron with Margin*

- *Weights with better margin generalize better*
  - Perceptron finds *any* separating hyperplane
- Thick Separator (aka as Perceptron with Margin)

- **Predict positive**

  $w\,x - \theta > \gamma$

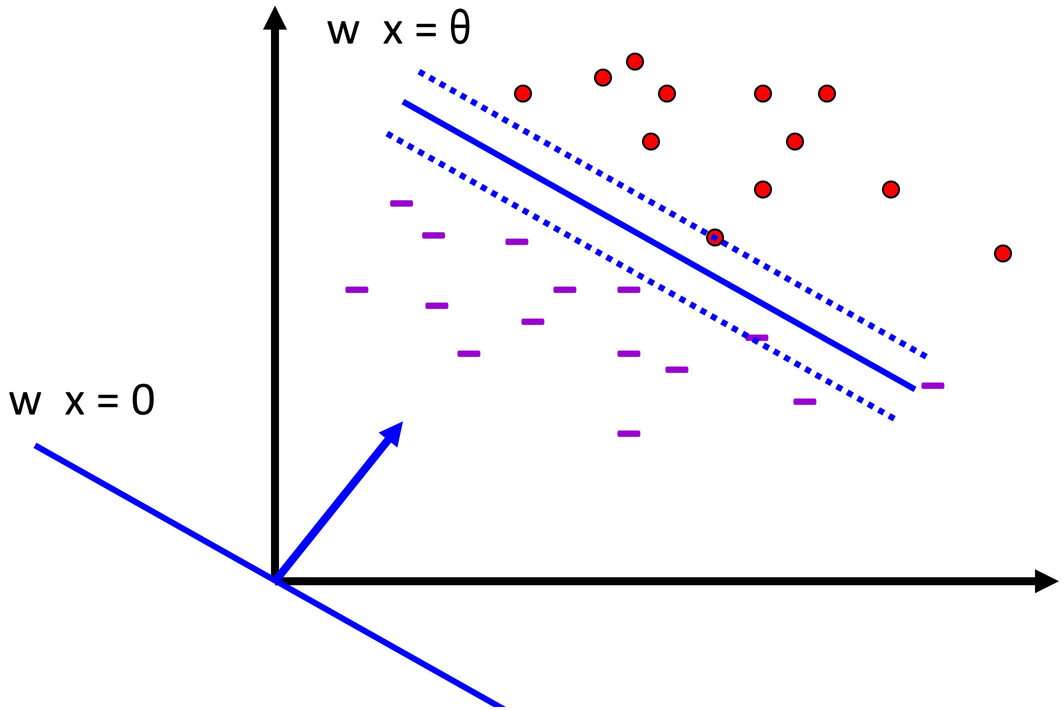
- **Predict negative**

  $w\,x - \theta < -\gamma$

- **Mistake:**

  $\gamma > (w\,x - \theta) > -\gamma$

Note: only at training time!



Regularization: *Perceptron with Margin*

- **Perceptron margin** : hyperparameter that has to be tuned using the validation set (try different values)
  - **In the future**: *the data will decide the margin*
- The effect of margin regularization in perceptron becomes smaller as w grows
  - Can we control the growth of w?
- In practice, **very effective**

Perceptron: Robust Variation

- *The perceptron algorithm counts later points more than earlier points*

1: (0,1,..,1,0,1)

2: (0,1,..,1,0,1)

…

100: (0,1,..,1,0,1)

…

10000: (0,1,..,1,0,1)

Makes some mistakes, update..
After 100 examples, learner stops making mistakes

We keep going…

BUT, at the 10,000 example the learner makes a mistake!

Why is this a problem?

Voted Perceptron

- *Training:*
  - *Learner remembers how long each hypothesis survived (no mistakes on w)*
- Test:
  - Weighted vote of all participating hypotheses

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \text{sign}\left(\boldsymbol{w}^{(k)} \cdot \hat{\boldsymbol{x}} + b^{(k)}\right)\right)$$

- Heavy: (1) Computational effort (2) **Storage**

Averaged Perceptron

- **Training**: *Maintain a running weighted average of survived hypotheses*
- **Test**: *Predict according to the averag ight vector*

Voted Perceptron $\longrightarrow$

$$\hat{y} = \operatorname{sign}\left(\sum_{k=1}^{K} c^{(k)} \operatorname{sign}\left(\boldsymbol{w}^{(k)} \cdot \hat{\boldsymbol{x}} + b^{(k)}\right)\right)$$

$$\hat{y} = \operatorname{sign}\left(\sum_{k=1}^{K} c^{(k)} \left(\boldsymbol{w}^{(k)} \cdot \hat{\boldsymbol{x}} + b^{(k)}\right)\right)$$

$\longleftarrow$ Averaged Perceptron

- *An efficient approximation of voted perceptron*
- *Almost always better than regular perceptron!*

Averaged Perceptron

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

   a. Predict the label of instance x to be $y' = \text{sgn}\{w \bullet x)$

   b. If $y' \neq y$, update the weight vector:

   $$w = w + r\, y\, x \qquad \text{(r - a constant, learning rate)}$$

   a = a+w

3. Return a

Summary: Perceptron/ Winnow

- Examples: $x \in \{0,1\}^n$; Hypothesis: $w \in R^n$
- Prediction: $y \in \{-1,+1\}$: y = 1 iff w x >µ
- Additive weight update algorithm
  - (Perceptron, Rosenblatt, 1958. Variations exist)

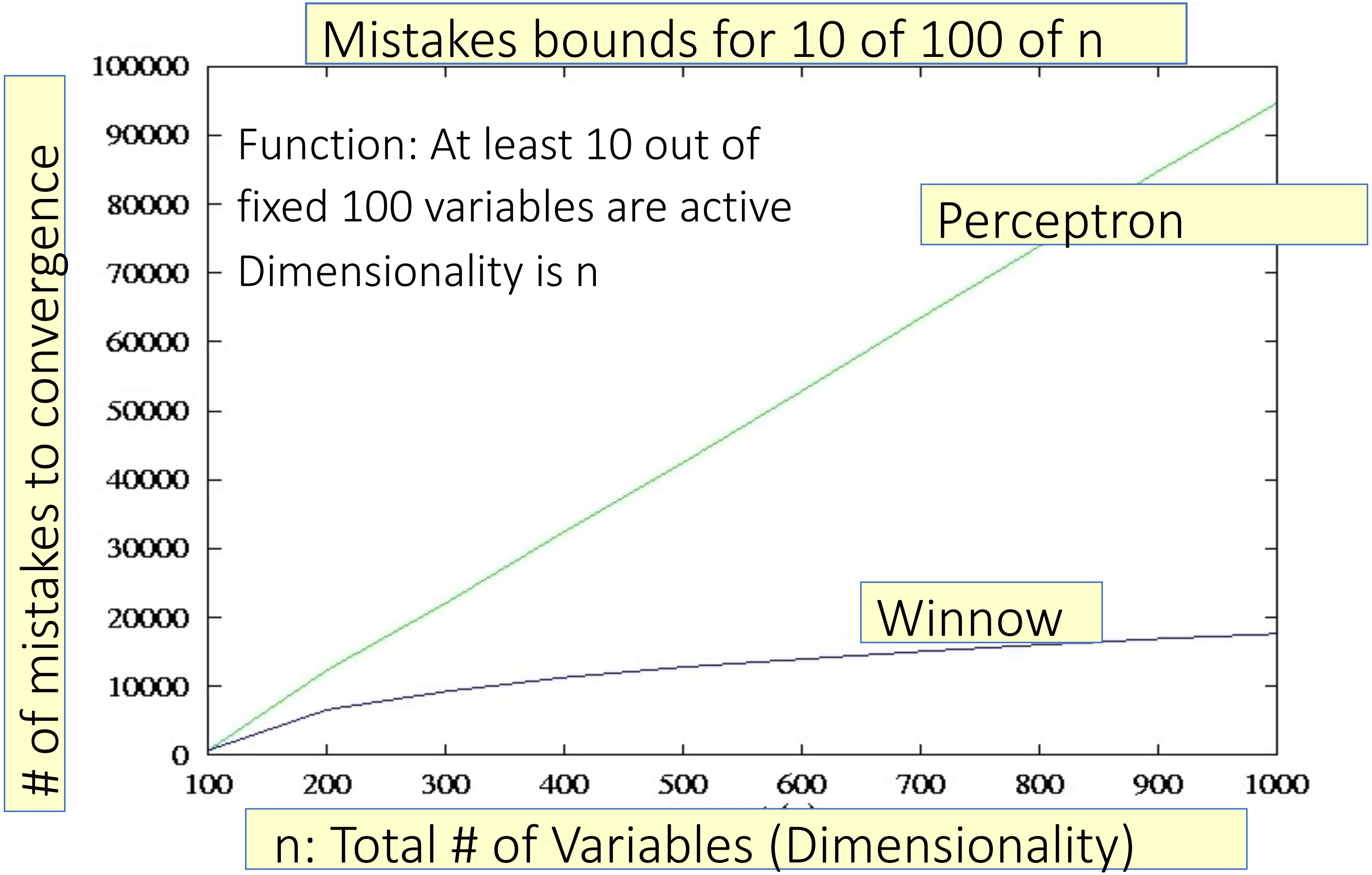
$$\boxed{\begin{array}{l} \text{If Class} = 1 \text{ but } w \bullet x \le \theta, \quad w_i \leftarrow w_i + 1 \text{ (if } x_i = 1) \text{ (promotion)} \\ \text{If Class} = 0 \text{ but } w \bullet x \ge \theta, \quad w_i \leftarrow w_i - 1 \text{ (if } x_i = 1) \text{ (demotion)} \end{array}}$$

- Multiplicative weight update algorithm
  - (Winnow, Littlestone, 1988. Variations exist)

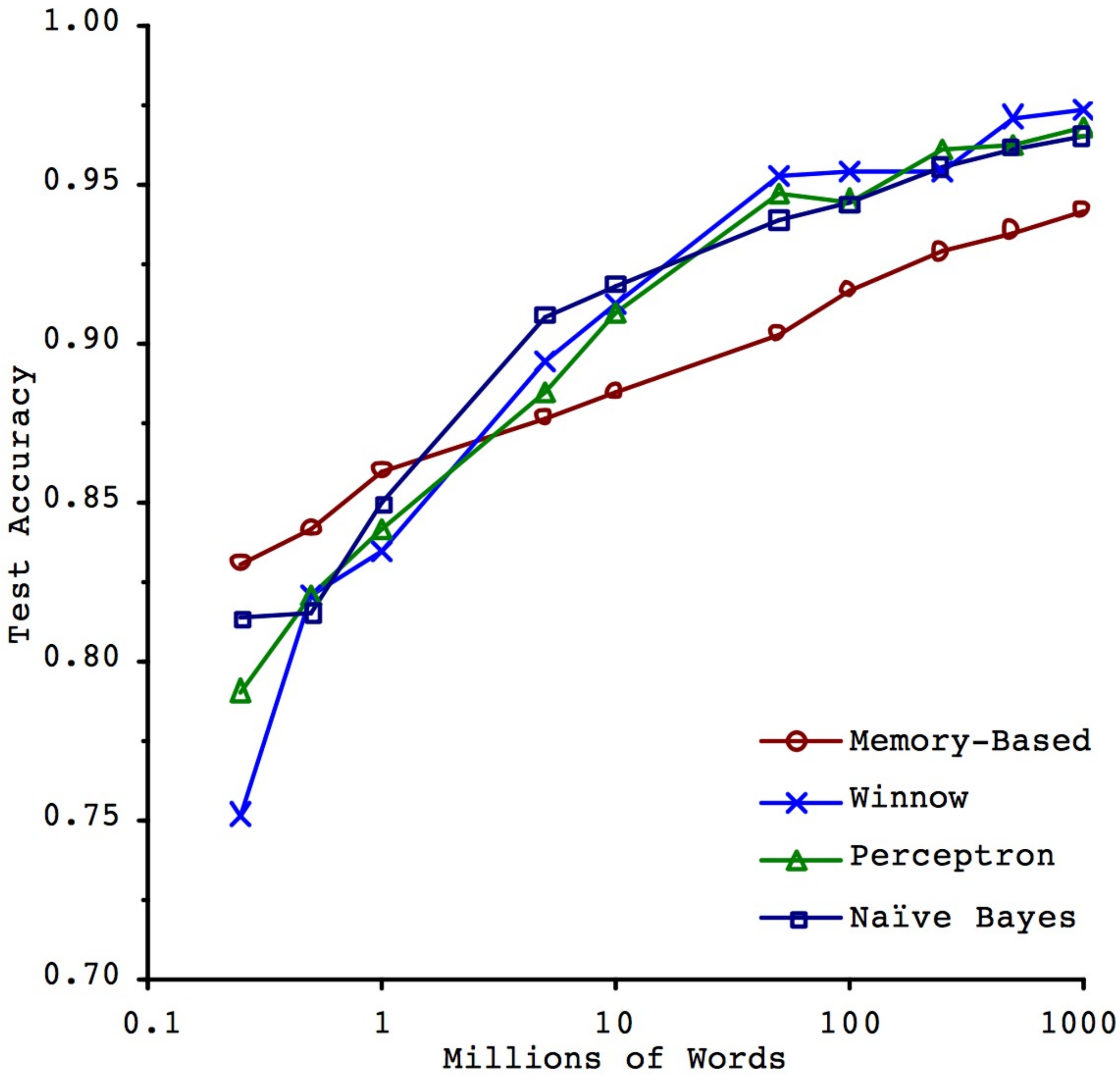
$$\boxed{\begin{array}{l} \text{If Class} = 1 \text{ but } w \bullet x \le \theta, \quad w_i \leftarrow 2w_i \text{ (if } x_i = 1) \text{ (promotion)} \\ \text{If Class} = 0 \text{ but } w \bullet x \ge \theta, \quad w_i \leftarrow w_i/2 \text{ (if } x_i = 1) \text{ (demotion)} \end{array}}$$

How to Compare the Algorithms?

- **Fair Question!**
  - **Same representation**, comparison pertains to the algorithms
  - Both mistake bound algorithm
- *Both algorithms are robust to noise*
  - *Small variations to for the noisy settings*
- ***Which algorithm should you choose?***
  - ***Multiplicative:*** *many irrelevant attributes (sparse target functions)*



Sparseness in the function space



Scaling to Very Very Large Corpora for Natural Language Disambiguation. Banko Brill 2001

Summary

- We saw –
  - Theoretical framework for analyzing learning
    - Probabilistic framework (PAC) *discuss in a few weeks*
    - Mistake bounds
  - Mistake bound for disjunctions (theoretical)
    - Halving algorithm: unrealistic, used as a theoretical bound
  - Linear threshold units
    - Perceptron and Winnow
    - Mistake bounds for Winnow and Perceptron
    - Realistic considerations (averaged parameters, thick separator, order of examples)
  - ***Can you implement these algorithms?***