Perceptron

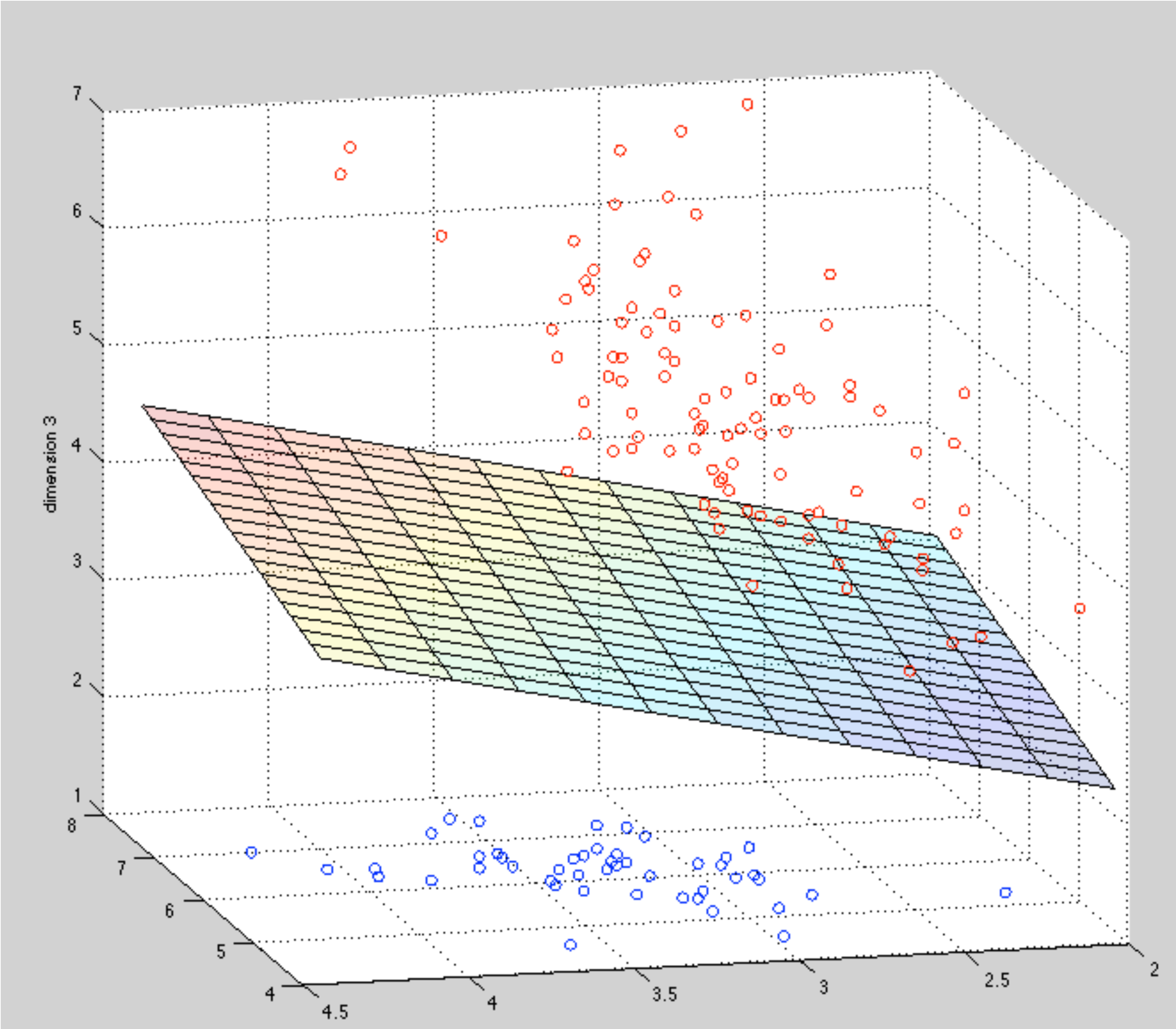
Machine Learning 10-601B

Seyoung Kim

Many of these slides are derived from William Cohen. Thanks!

Perceptron

- Logistic regression is a linear classifier
- Another famous linear classifier
  - The perceptron



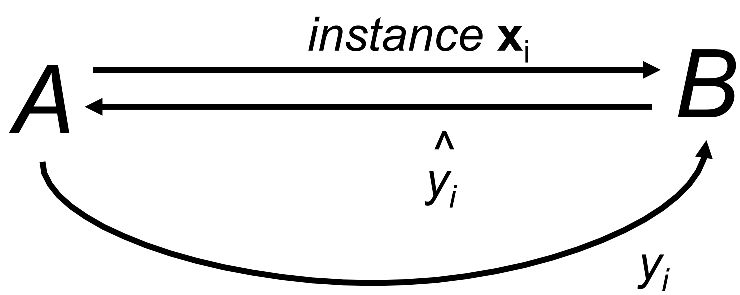
dimension 3
7
6
5
4
3
2
1
8
7
6
5
4
4.5
4
3.5
3
2.5
2

Probabilistic vs Margin-based Learning

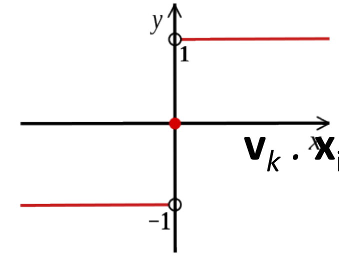
- It's not *all* probabilities: many other types of analysis are used
- We also want to
  - capture *geometric* intuitions about what makes learning hard or easy
  - analyze performance worst-case settings
- This particular analysis is simple enough to give some insight into "margin" learning
- See: Freund & Schapire, 1998

The perceptron

[Rosenblatt, 1957]



Compute: $\hat{y}_i = \mathrm{sign}(\mathbf{v}_k \cdot \mathbf{x}_i)$

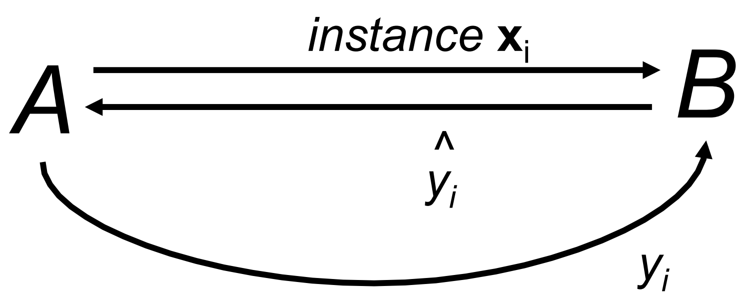


If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + y_i \mathbf{x}_i$

- On-line setting: data samples arrive one sample at a time
  - Adversary A provides student B with an *instance* **x**
  - Student B predicts a class (+1, -1) according to a simple linear classifier: sign($\mathbf{v}_k \cdot \mathbf{x}$)
  - Adversary gives student the answer (+1,-1) for that instance
- Will do a *worst-case* analysis of the mistakes made by the student over *any* sequence of instances from the adversary
  - … that follow a few rules

The perceptron

[Rosenblatt, 1957]



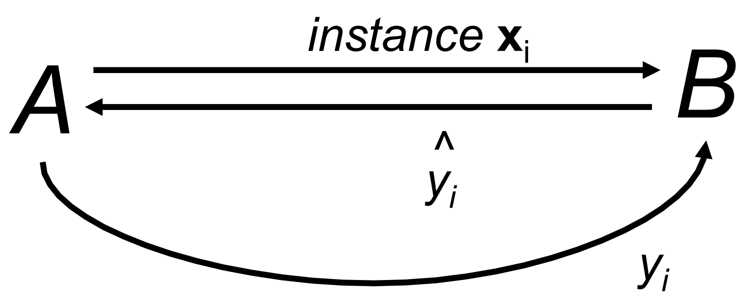
Compute: $\hat{y}_i = \mathrm{sign}(\mathbf{v}_k \cdot \mathbf{x}_i)$

If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + y_i \mathbf{x}_i$

- Amazingly simple algorithm
- Quite effective
- Very easy to *understand* if you do a little linear algebra

The perceptron

[Rosenblatt, 1957]



Compute: $\hat{y}_i = \mathrm{sign}(\mathbf{v}_k \,.\, \mathbf{x}_i)$

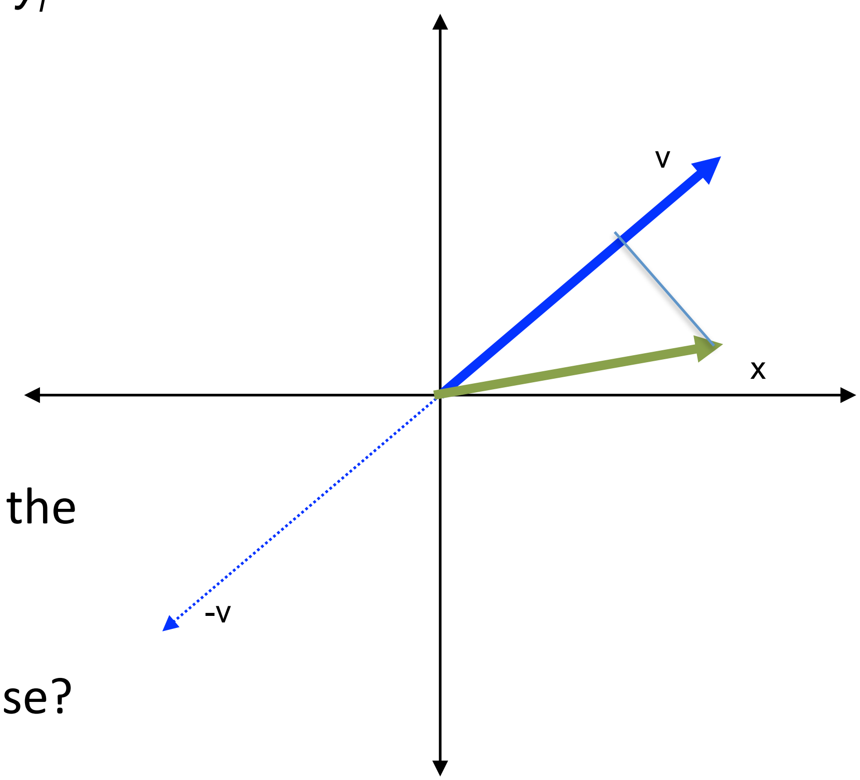
If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + y_i \, \mathbf{x}_i$

- Recall dot product definition:

$$\mathbf{x} \bullet \mathbf{v} = \sum_{\mathbf{i}} x_i v_i$$

•and intuition:

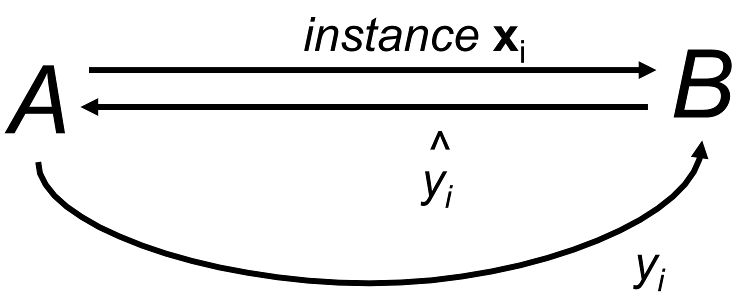
  - project vector **x** onto vector **v**
  - dot product is the distance from the origin to that projection



- So why does this algorithm make sense?

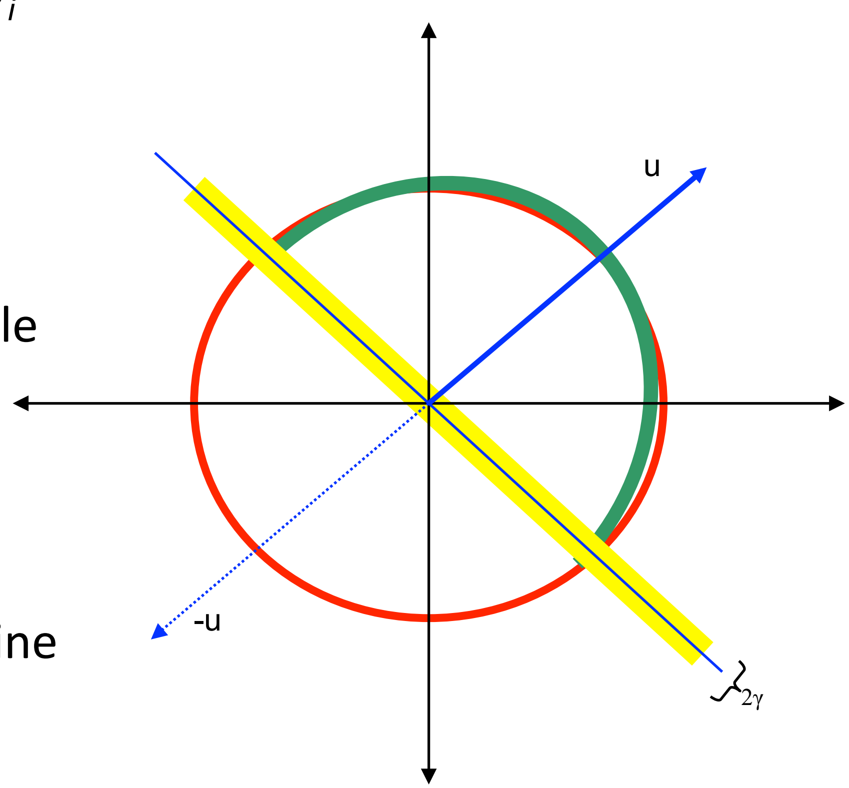
The perceptron

[Rosenblatt, 1957]



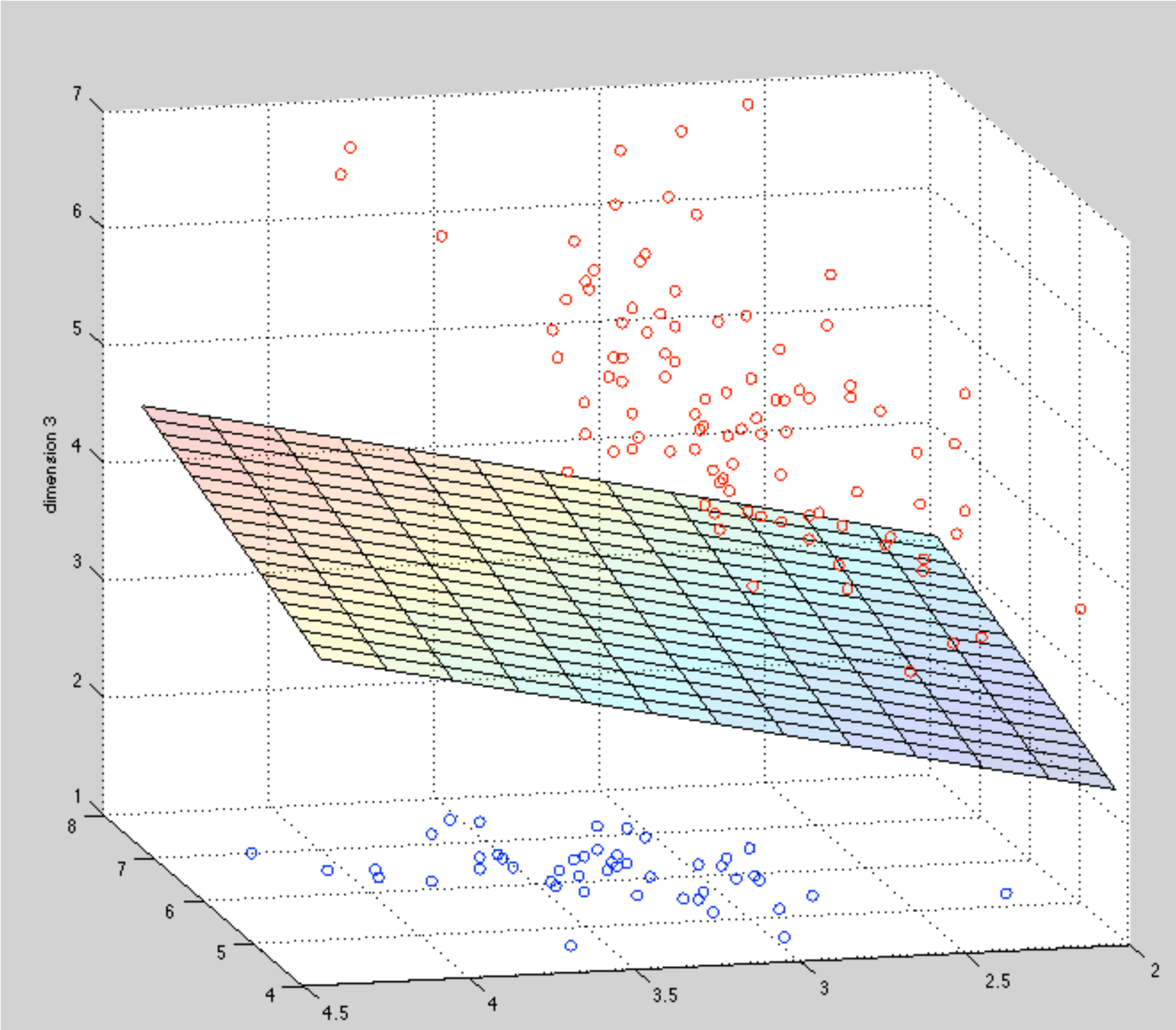
Compute: $\hat{y}_i = \mathrm{sign}(\mathbf{v}_k \cdot \mathbf{x}_i)$

If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + y_i \mathbf{x}_i$



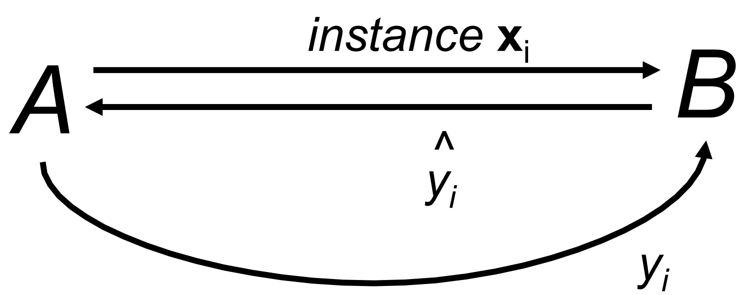
- Amazingly simple algorithm
- Quite effective
- Very easy to *understand* if you do a little linear algebra

- *Two rules:*
  - Examples are not too “big”
  - There is a “good” answer -- i.e. a line that clearly separates the pos/neg examples


dimension 3
7
6
5
4
3
2
1
8
7
6
5
4
4.5
4
3.5
3
2.5
2

The perceptron

[Rosenblatt, 1957]



Compute: $\hat{y}_i = \mathrm{sign}(\mathbf{v}_k \,.\, \mathbf{x}_i)$

If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + y_i \mathbf{x}_i$

Rule 1: Radius R: A must provide examples "near the origin"

$$\forall \mathbf{x}_i \text{ given by A, } \|\mathbf{x}_i\|_2^2 \leq R^2$$

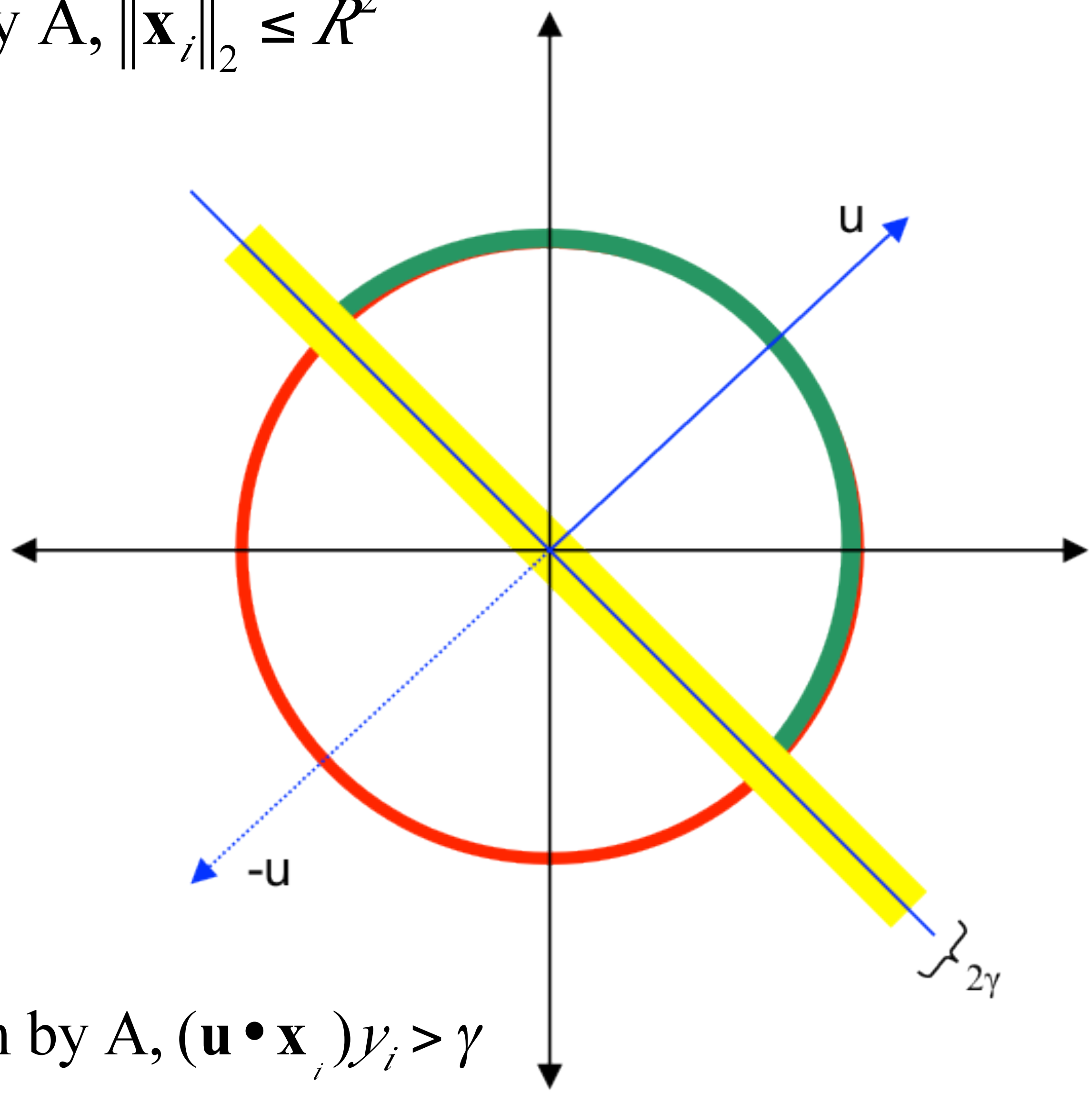
$$\|\mathbf{x}\|_2 = \sqrt{(x_1^2 + \ldots + x_n^2)}$$

Rule 2: Margin γ: A must provide examples that can be separated with some vector **u** with margin γ>0 and unit norm

$$\exists \mathbf{u} : \forall \mathbf{x}_i \text{ given by A, } (\mathbf{u} \bullet \mathbf{x}_i) y_i > \gamma$$

$$\text{and } \|\mathbf{u}\|_2 = 1$$

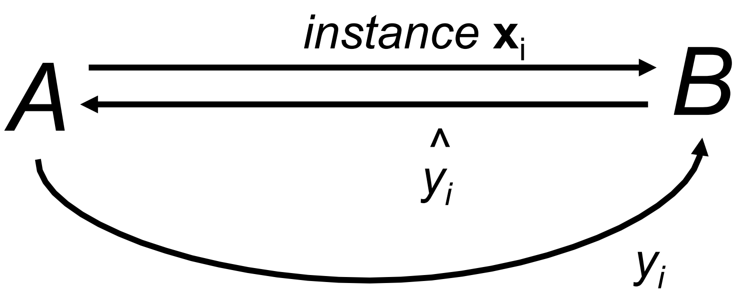
$\forall \mathbf{x}_i$ given by A, $\|\mathbf{x}_i\|_2 \leq R^2$



$\exists \mathbf{u} : \forall \mathbf{x}_i$ given by A, $(\mathbf{u} \bullet \mathbf{x}_i) y_i > \gamma$

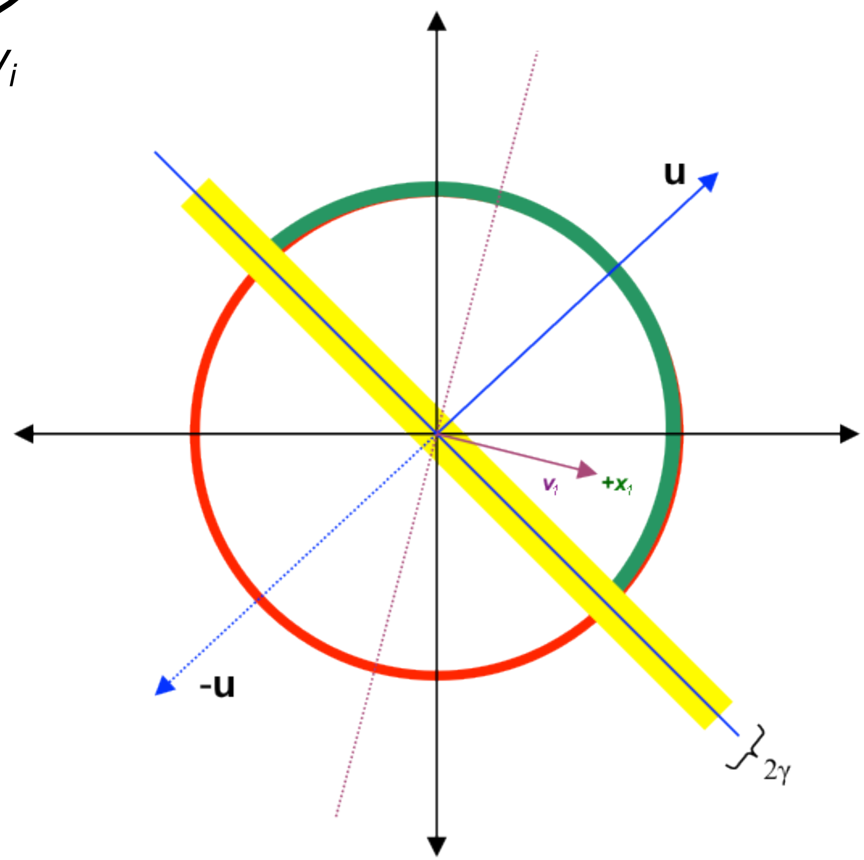
and $\|\mathbf{u}\|_2 = 1$

The perceptron: after one positive $\mathbf{x}_i$

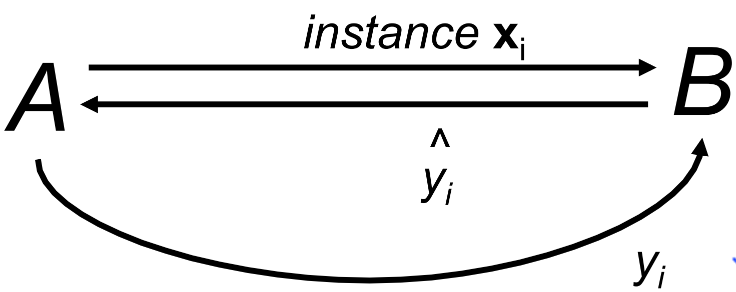


Compute: $\hat{y}_i = \mathrm{sign}(\mathbf{v}_k \,.\, \mathbf{x}_i)$

If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + y_i \, \mathbf{x}_i$

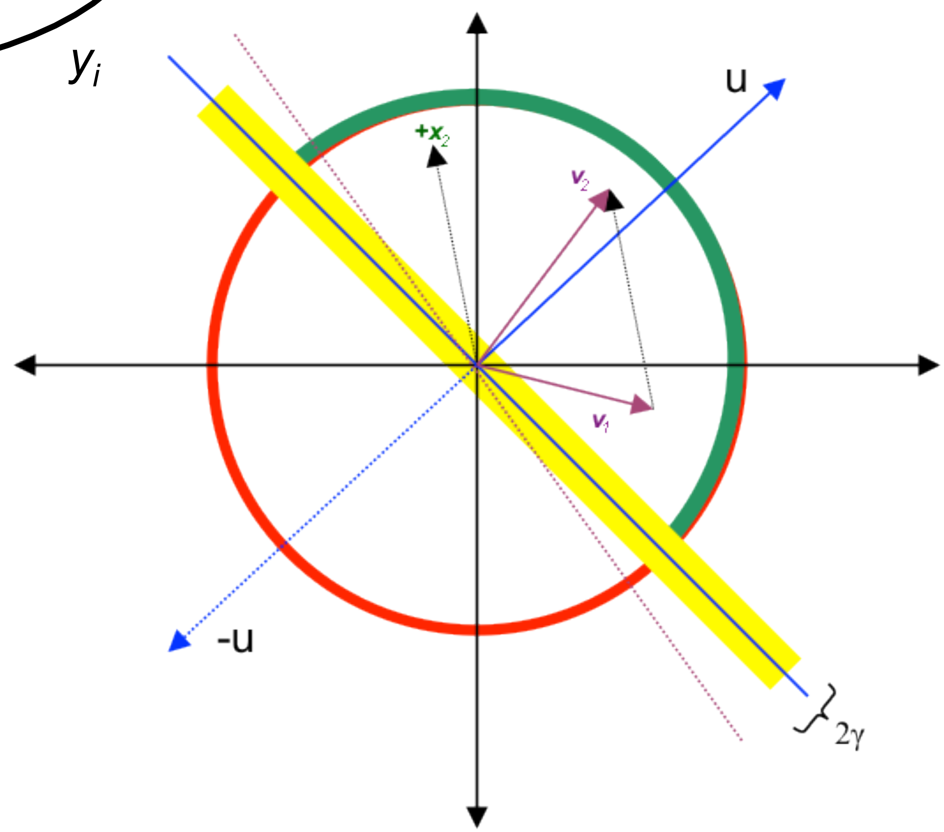


The perceptron: after two positive $\mathbf{x}_i$

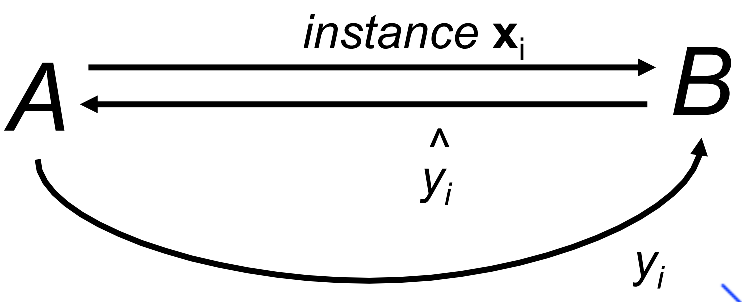


Compute: $\hat{y}_i = \mathbf{v}_k \, . \, \mathbf{x}_i$

If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + y_i \, \mathbf{x}_i$

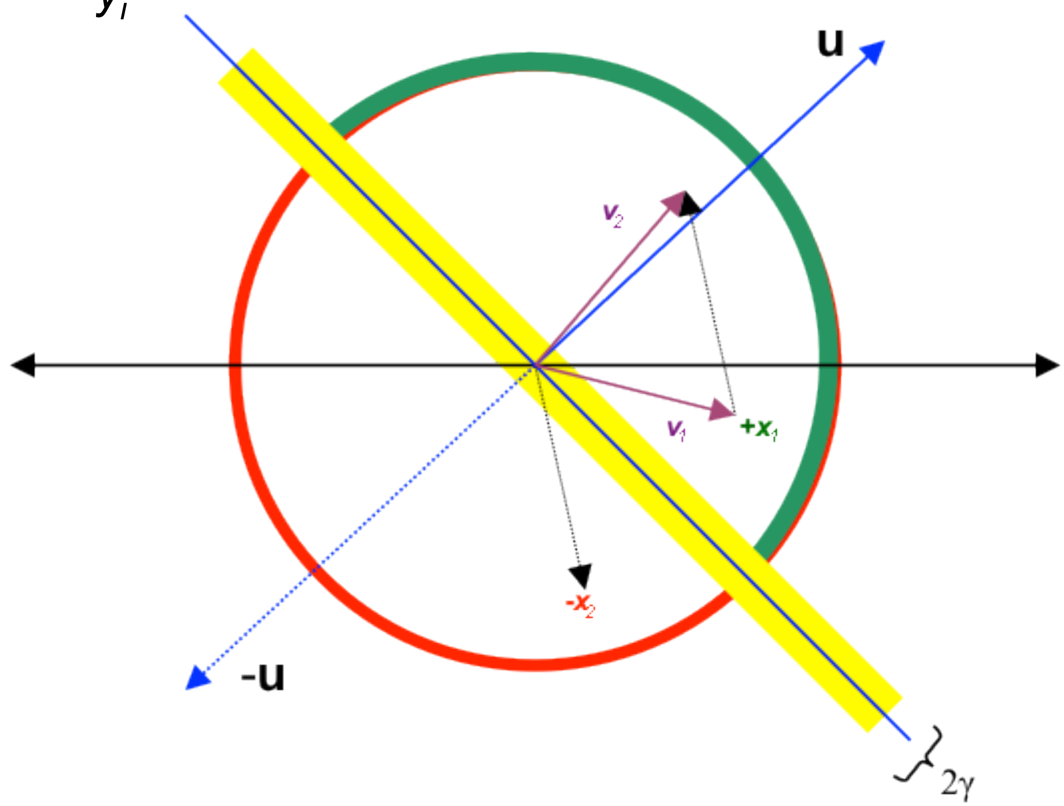


The perceptron: after one pos + one neg $\mathbf{x}_i$

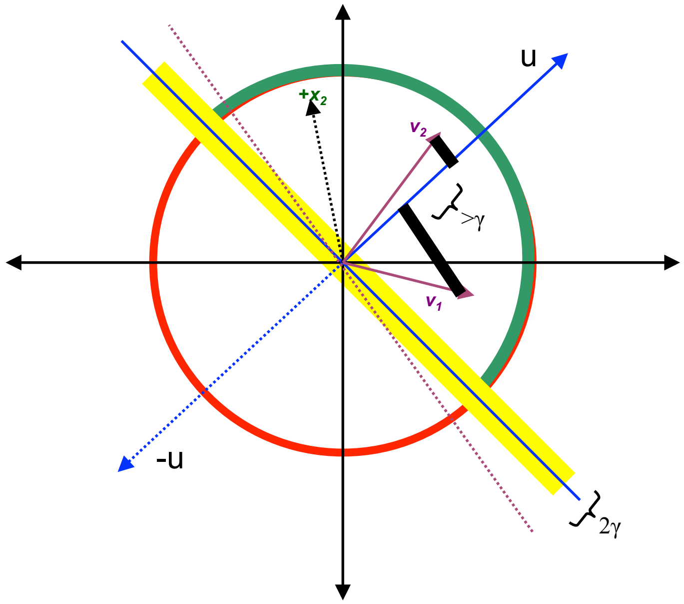


Compute: $\hat{y}_i = \text{sign}(\mathbf{v}_k \cdot \mathbf{x}_i)$

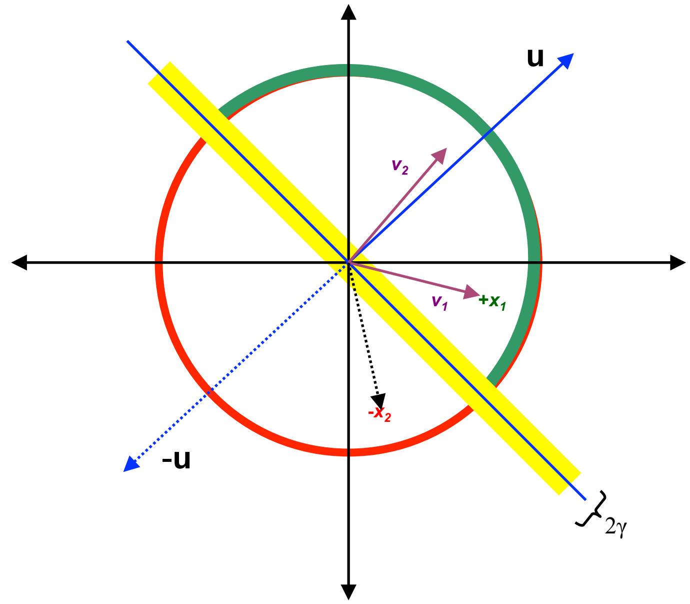
If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + y_i \mathbf{x}_i$



The guess $\mathbf{v_2}$ after the two positive examples: $\mathbf{v}_2=\mathbf{v}_1+\mathbf{x}_2$



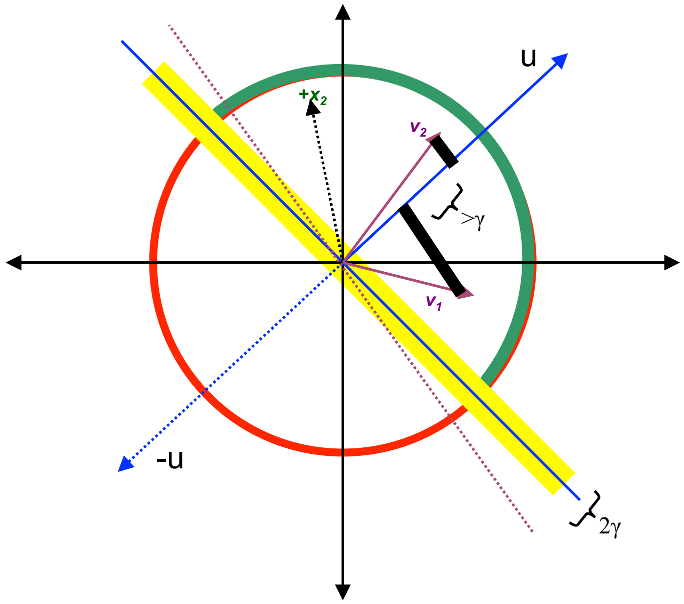
The guess $\mathbf{v_2}$ after the one positive and one negative example: $\mathbf{v}_2=\mathbf{v}_1-\mathbf{x}_2$



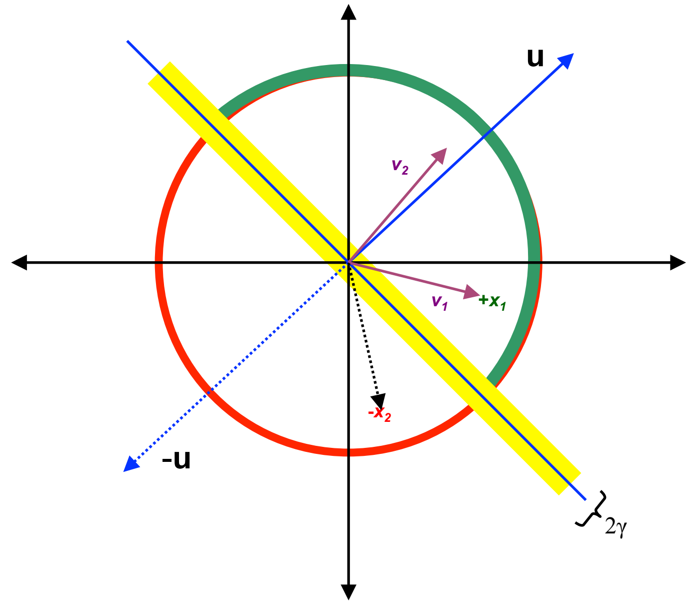
Lemma 1: the dot product between $\mathbf{v}_k$ and $\mathbf{u}$ increases with each mistake by at least γ: i.e.,

$$\forall k : \mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$$

The guess $\mathbf{v_2}$ after the two positive examples: $\mathbf{v}_2=\mathbf{v}_1+\mathbf{x}_2$



The guess $\mathbf{v_2}$ after the one positive and one negative example: $\mathbf{v}_2=\mathbf{v}_1-\mathbf{x}_2$



Lemma 1: the dot product between $\mathbf{v}_k$ and $\mathbf{u}$ increases with each mistake by at least γ: i.e.,

$$\forall k : \mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$$

$$\mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k + y_i \mathbf{x}_i) \cdot \mathbf{u}$$

$$\mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k \cdot \mathbf{u}) + y_i (\mathbf{x}_i \cdot \mathbf{u})$$

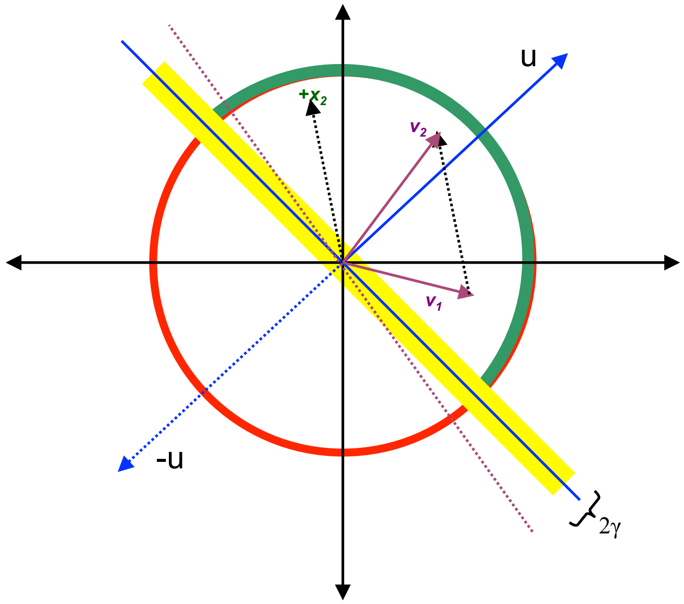
$$\mathbf{v}_{k+1} \cdot \mathbf{u} \geq (\mathbf{v}_k \cdot \mathbf{u}) + \gamma$$

$\exists \mathbf{u} : \forall \mathbf{x}_i$ given by A, $(\mathbf{u} \bullet \mathbf{x}_i) y_i > \gamma$

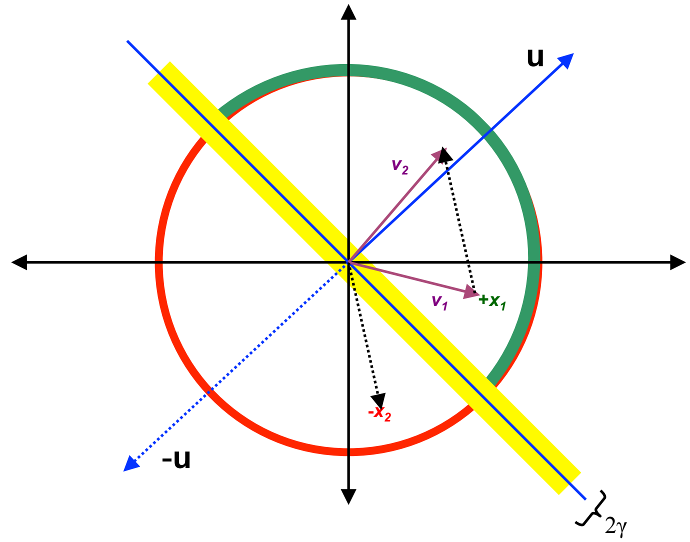
so …

$$\mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$$

(3a) The guess $\mathbf{v_2}$ after the two positive examples: $\mathbf{v}_2=\mathbf{v}_1+\mathbf{x}_2$



(3b) The guess $\mathbf{v_2}$ after the one positive and one negative example: $\mathbf{v}_2=\mathbf{v}_1-\mathbf{x}_2$



Lemma 2: The norm of $\mathbf{v}_k$ grows slowly with each mistake, i.e.,

$$\forall k, \|\mathbf{v}_k\|_2^2 \leq kR^2$$

$$\mathbf{v}_{k+1} \cdot \mathbf{v}_{k+1} = (\mathbf{v}_k + y_i \mathbf{x}_i) \cdot (\mathbf{v}_k + y_i \mathbf{x}_i)$$

$$\|\mathbf{v}_{k+1}\|_2^2 = \|\mathbf{v}_k\|_2^2 + \underline{2 y_i \mathbf{v}_k \cdot \mathbf{x}_i} + y_i^2 \|\mathbf{x}_i\|_2^2$$

$$\|\mathbf{v}_{k+1}\|_2^2 \leq \|\mathbf{v}_k\|_2^2 + 1\|\mathbf{x}_i\|_2^2$$

$$\|\mathbf{v}_{k+1}\|_2^2 \leq \|\mathbf{v}_k\|_2^2 + R^2$$

Always negative, since it was a mistake

$$\forall \mathbf{x}_i \text{ given by A}, \|\mathbf{x}_i\|_2^2 \leq R^2$$

so …

$$\|\mathbf{v}_k\|_2^2 \leq kR^2$$

Lemma 1: the dot product between $\mathbf{v}_k$ and $\mathbf{u}$ increases with each mistake by at last γ: i.e.,

$$\forall k : \mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$$

Lemma 2: The norm of $\mathbf{v}_k$ grows slowly with each mistake, i.e.,

$$\forall k, \|\mathbf{v}_k\|_2^2 \leq kR^2$$

$$k\gamma \leq \mathbf{v}_k \cdot \mathbf{u} \quad \text{and} \quad \|\mathbf{v}_k\|_2^2 \leq kR^2$$

$$k^2\gamma^2 \leq \|\mathbf{v}_k \cdot \mathbf{u}\|_2^2 \quad \text{and} \quad \|\mathbf{v}_k\|_2^2 \leq kR^2$$

$$k^2\gamma^2 \leq \|\mathbf{v}_k\|_2^2 \cdot \|\mathbf{u}\|_2^2 \quad \text{and} \quad \|\mathbf{v}_k\|_2^2 \leq kR^2$$

...and $\|\mathbf{u}\|_2 = 1$

$$k^2\gamma^2 \leq \|\mathbf{v}_k\|_2^2 \quad \text{and} \quad \|\mathbf{v}_k\|_2^2 \leq kR^2$$

$$k^2\gamma^2 \leq \|\mathbf{v}_k\|_2^2 \leq kR^2$$

$$k^2\gamma^2 \leq kR^2$$

$$k < \left(\frac{R}{\gamma}\right)^2$$

证明算法收敛

Summary

- We have shown that
  - *If* : exists a $\mathbf{u}$ with unit norm that has margin γ on examples in the seq $(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),\ldots.$
  - *Then* : the perceptron algorithm makes $< R^2/\gamma^2$ mistakes on the sequence (where $R >= ||\mathbf{x}_i||$)
  - *Independent* of dimension of the data (!)

- We *don't* know what happens if the data's not separable

The voted perceptron

On-line to batch learning

Imagine we run the on-line perceptron and see this result.

i	guess	input	result
1	$\mathbf{v}_0$	$\mathbf{x}_1$	X (a mistake)
2	$\mathbf{v}_1$	$\mathbf{x}_2$	$\surd$ (correct!)
3	$\mathbf{v}_1$	$\mathbf{x}_3$	$\surd$
4	$\mathbf{v}_1$	$\mathbf{x}_4$	X (a mistake)
5	$\mathbf{v}_2$	$\mathbf{x}_5$	$\surd$
6	$\mathbf{v}_2$	$\mathbf{x}_6$	$\surd$
7	$\mathbf{v}_2$	$\mathbf{x}_7$	$\surd$
8	$\mathbf{v}_2$	$\mathbf{x}_8$	X
9	$\mathbf{v}_3$	$\mathbf{x}_9$	$\surd$
10	$\mathbf{v}_3$	$\mathbf{x}_{10}$	X

Which $\mathbf{v}_i$ should we use?

Maybe the *last* one?

Here it's never gotten any test cases right!
(Experimentally, the classifiers move around a lot.)

Maybe the "best one"?

But we "improved" it with later mistakes…

$$
\begin{aligned}
P(\text{error in } \mathbf{x}) &= \sum_k P(\text{error on } \mathbf{x}|\text{picked } \mathbf{v}_k) P(\text{picked } \mathbf{v}_k) \\
&= \sum_k \frac{1}{m_k} \frac{m_k}{m} = \sum_k \frac{1}{m} = \frac{k}{m}
\end{aligned}
$$

Imagine we run the on-line perceptron and see this result.

i	guess	input	result
1	$\mathbf{v}_0$	$\mathbf{x}_1$	X (a mistake)
2	$\mathbf{v}_1$	$\mathbf{x}_2$	$\surd$ (correct!)
3	$\mathbf{v}_1$	$\mathbf{x}_3$	$\surd$
4	$\mathbf{v}_1$	$\mathbf{x}_4$	X (a mistake)
5	$\mathbf{v}_2$	$\mathbf{x}_5$	$\surd$
6	$\mathbf{v}_2$	$\mathbf{x}_6$	$\surd$
7	$\mathbf{v}_2$	$\mathbf{x}_7$	$\surd$
8	$\mathbf{v}_2$	$\mathbf{x}_8$	X
9	$\mathbf{v}_3$	$\mathbf{x}_9$	$\surd$
10	$\mathbf{v}_3$	$\mathbf{x}_{10}$	X

1. Pick a $\mathbf{v}_k$ at random according to m_k/m, the fraction of examples it was used for.
2. Predict using the $\mathbf{v}_k$ you just picked.
3. Better: use a deterministic approximation to this*: a sum of the $\mathbf{v}_k$'s, weighted by m_k/m*

From Freund & Schapire, 1998: Classifying digits with VP