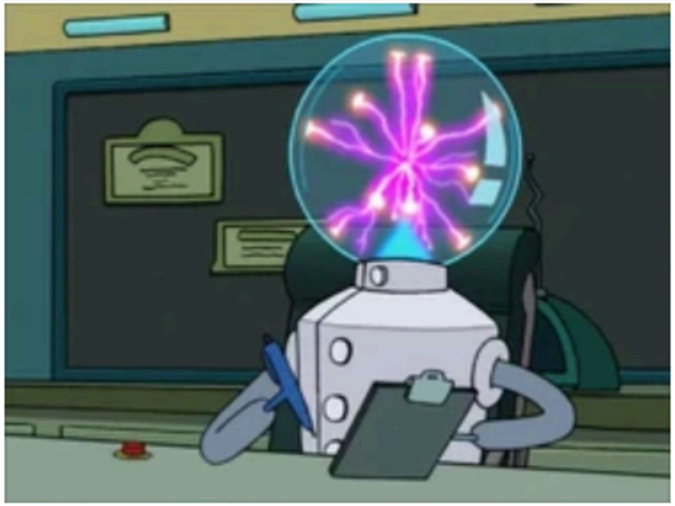
Dr. Perceptron

> *"Now, consider the following: You were admitted to this robot asylum. Therefore, you must be a robot. Diagnosis complete."*
> **—Dr. Perceptron to Fry**[source]

Dr. Perceptron is the head doctor at the Hal Institute for Criminally Insane Robots. He was destroyed briefly by Roberto during his escape from the Institute, but was apparently fixed/rebuilt and returned to work for Bender's second stay.

In 3008, Dr. Perceptron was damaged during a group therepy session, but like his encounter with Roberto, was quickly repaired to continue his duties at the Institute.

Dr. Perceptron	
	
Gender	Male ♂
Species	Robot
Planet	Earth
Profession	Doctor of Freudian Circuit Analysis
First appearance	Insane in the Mainframe
Voiced by	Maurice LaMarche

Appearances ✏Edit

- *Insane in the Mainframe*
- *Bender's Game*

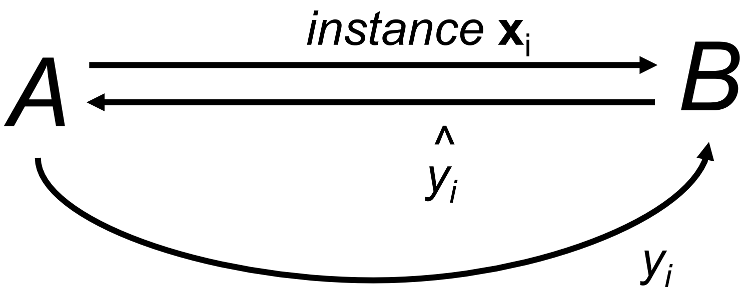
http://futurama.wikia.com/wiki/Dr._Perceptron

Where we are…

- Experiments with a hash-trick implementation of logistic regression
- Next question:
  - how do you parallelize SGD, or more generally, this kind of streaming algorithm?
  - each example affects the next prediction ➔ order matters ➔ parallelization changes the behavior
  - we will step back to perceptrons and then step forward to **parallel perceptrons**
  - then another nice parallel learning algorithm
  - then a midterm

Recap: perceptrons

The perceptron



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

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

Margin γ. A must provide examples that can be separated with some vector $\mathbf{u}$ with margin $\gamma > 0$, ie

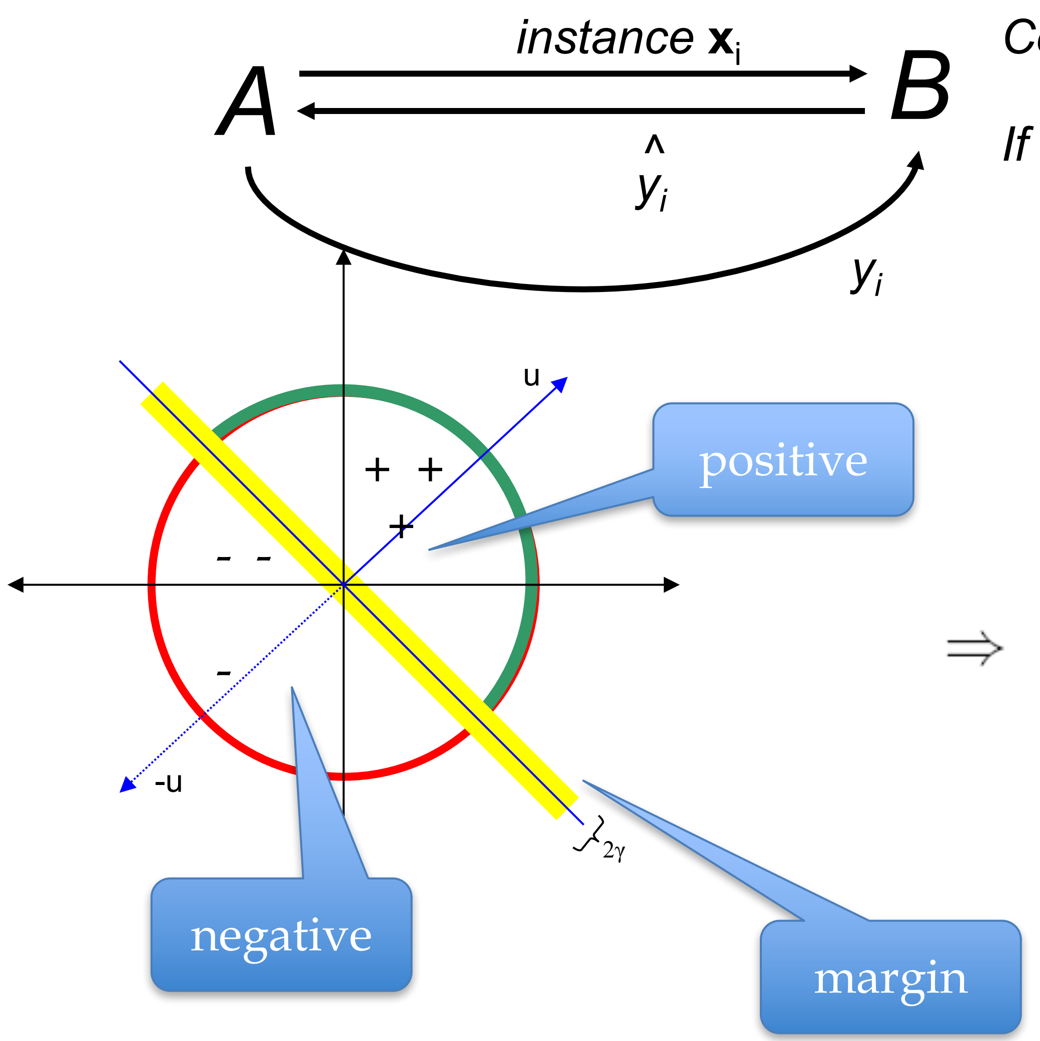
$$\exists \mathbf{u} : \forall (\mathbf{x}_i, y_i) \text{ given by } A,\ (\mathbf{u} \cdot \mathbf{x}) y_i > \gamma$$

and furthermore, $\|\mathbf{u}\| = 1$.

Radius R. A must provide examples "near the origin", ie

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

The perceptron



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

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

A lot like SGD update for logistic regression!

Mistake bound:

$$\Rightarrow \quad k \leq \frac{R^2}{\gamma^2} = \left(\frac{R}{\gamma}\right)^2$$

2

$$P(\text{error in } \mathbf{x}) = \sum_k P(\text{error on } \mathbf{x}|\text{picked } \mathbf{v}_k)P(\text{picked } \mathbf{v}_k)$$

$$= \sum_k \frac{1}{m_k}\frac{m_k}{m} = \sum_k \frac{1}{m} = \frac{k}{m}$$

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

$m_1=3$

$m_2=4$

$m=10$

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. (Actually, use some sort of deterministic approximation to this).

predict using sign(v*. **x**)

$$\boxed{\mathbf{v}_* = \sum_k \left(\frac{m_k}{m}\mathbf{v}_k\right)}$$

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

$m_1=3$

$m_2=4$

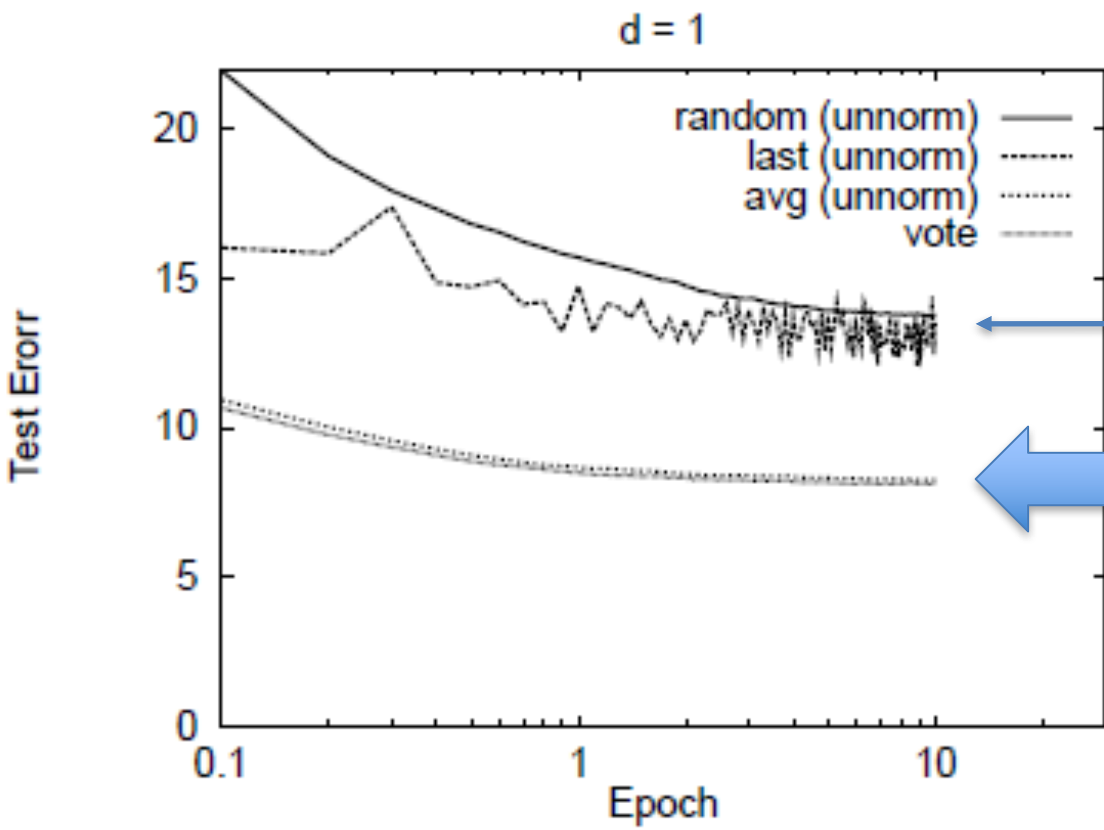
$m=10$

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. (Actually, use some sort of deterministic approximation to this).

predict using sign(v*. **x**)

$$\mathbf{v}_* = \sum_k \left(\frac{m_k}{m}\mathbf{v}_k\right)$$

Also: there's a sparsification trick that makes learning the averaged perceptron fast

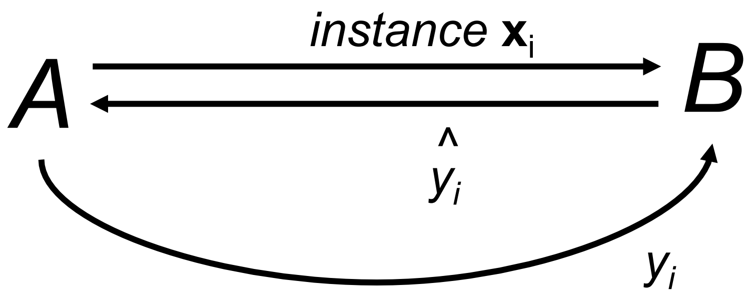


Last perceptron

Averaging/voting

KERNELS AND PERCEPTRONS

The kernel perceptron



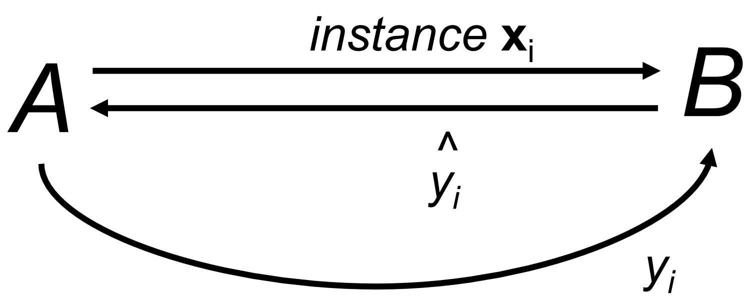
Compute: $\hat{y}_i = \mathbf{v}_k \,.\, \mathbf{x}_i$ ⟶ $\text{Compute} : \hat{y} = \sum_{\mathbf{x}_{k^+} \in FN} \mathbf{x}_i . \mathbf{x}_{k^+} - \sum_{\mathbf{x}_{k^-} \in FP} \mathbf{x}_i . \mathbf{x}_{k^-}$

If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + y_i \, \mathbf{x}_i$ ⟶ If false positive (too high) mistake : add $\mathbf{x}_i$ to FP

If false positive (too low) mistake : add $\mathbf{x}_i$ to FN

Mathematically the same as before … but allows use of the kernel trick

The kernel perceptron



$$\boxed{K(\mathbf{x}, \mathbf{x}_k) \equiv \mathbf{x} \cdot \mathbf{x}_k}$$

$$\boxed{\hat{y} = \sum_{\mathbf{x}_{k^+} \in FN} K(\mathbf{x}_i, \mathbf{x}_{k^+}) - \sum_{\mathbf{x}_{k^-} \in FP} K(\mathbf{x}_i, \mathbf{x}_{k^-})}$$

Compute: $\hat{y}_i = \mathbf{v}_k \cdot \mathbf{x}_i$ $\longrightarrow$ $\cancel{\text{Compute}: \hat{y} = \sum_{\mathbf{x}_{k^+} \in FN} \mathbf{x}_i . \mathbf{x}_{k^+} - \sum_{\mathbf{x}_{k^-} \in FP} \mathbf{x}_i . \mathbf{x}_{k^-}}$

If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + y_i \mathbf{x}_i$ $\longrightarrow$ $\text{If false positive (too high) mistake : add } \mathbf{x}_i \text{ to FP}$

$\text{If false positive (too low) mistake : add } \mathbf{x}_i \text{ to FN}$

Mathematically the same as before … but allows use of the “kernel trick”

Other kernel methods (SVM, Gaussian processes) aren’t constrained to limited set (+1/-1/0) of weights on the K(**x**,**v**) values.

Some common kernels

- Linear kernel:

$$\boxed{K(\mathbf{x},\mathbf{x}') \equiv \mathbf{x}\cdot\mathbf{x}'}$$

- Polynomial kernel:

$$\boxed{K(\mathbf{x},\mathbf{x}') \equiv (\mathbf{x}\cdot\mathbf{x}'+1)^d}$$

- Gaussian kernel:

$$\boxed{K(\mathbf{x},\mathbf{x}') \equiv e^{-\|\mathbf{x}-\mathbf{x}'\|^2/\sigma}}$$

Some common kernels

- Polynomial kernel:

$$\boxed{K(\mathbf{x},\mathbf{x}') \equiv (\mathbf{x}\cdot\mathbf{x}'+1)^d}$$

- for d=2

$$\left(\langle x_{1,}x_2\rangle \cdot \langle x'_{1,}x'_2\rangle + 1\right)^2$$

$$= \ (x_1x'_1 + x_2x'_2 + 1)^2$$

$$=(x_1x'_1 + x_2x'_2 + 1)(x_1x'_1 + x_2x'_2 + 1)$$

$$= (x_1x'_1)^2 + 2(x_1x'_1x_2x'_2) + 2(x_1x'_1) + (x_2x'_2)^2 + 2(x_2x'_2) + 1$$

$$\cancel{\cong \langle 1, x_{1,}x_2, x_1x_2, x_1^2, x_2^2\rangle \cdot \langle 1, x'_{1,}x'_2, x'_1x'_2, x'^2_1, x'^2_2\rangle}$$

$$= \langle 1, \sqrt{2}x_{1,}\sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2\rangle \cdot \langle 1, \sqrt{2}x'_{1,}\sqrt{2}x'_2, \sqrt{2}x'_1x'_2, x'^2_1, x'^2_2\rangle$$

Some common kernels

- Polynomial kernel: $$\boxed{K(\mathbf{x},\mathbf{x}') \equiv (\mathbf{x}\cdot\mathbf{x}'+1)^d}$$

- for d=2 $$\left(\langle x_{1,}x_2\rangle \cdot \langle x'_{1,}x'_2\rangle + 1\right)^2$$

$$= \langle 1, \sqrt{2}x_{1,}\sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2\rangle \cdot \langle 1, \sqrt{2}x'_{1,}\sqrt{2}x'_2, \sqrt{2}x'_1x'_2, x'^2_1, x'^2_2\rangle$$

Similarity with the kernel on x is **equivalent** to dot-product similarity on a **transformed** feature vector $\phi(\boldsymbol{x})$

Explicitly map from **x** to ϕ(**x**) – i.e. to the point corresponding to **x** in the *Hilbert space (RKHS)*

Kernels 101

Implicitly map from **x** to ϕ(**x**) by changing the kernel function K

- Duality: two ways to look at this

$$\hat{y} = \mathbf{x} \cdot \mathbf{w} = K(\mathbf{x}, \mathbf{w})$$

$$\mathbf{w} = \sum_{\mathbf{x}_{k^+} \in FN} \mathbf{x}_{k^+} - \sum_{\mathbf{x}_{k^-} \in FP} \mathbf{x}_{k^-}$$

Observation about perceptron

$$\hat{y} = \sum_{\mathbf{x}_{k^+} \in FN} K(\mathbf{x}_i, \mathbf{x}_{k^+}) - \sum_{\mathbf{x}_{k^-} \in FP} K(\mathbf{x}_i, \mathbf{x}_{k^-})$$

$$K(\mathbf{x}, \mathbf{x}_k) \equiv \phi(\mathbf{x}) \cdot \phi(\mathbf{x}_k)$$

Generalization of perceptron

$$\hat{y} = \phi(\mathbf{x}) \cdot \mathbf{w}$$

$$\mathbf{w} = \sum_{\mathbf{x}_{k^+} \in FN} \phi(\mathbf{x}_{k^+}) - \sum_{\mathbf{x}_{k^-} \in FP} \phi(\mathbf{x}_{k^-})$$

same behavior but compute time/space are different

*Generalization: add weights to the sums for **w***

Kernels 101

$K(x,x') = K(x',x)$ ➔ Gram matrix is *symmetric*

$K(x,x) > 0$ ➔ diagonal of K is positive ➔ K is "positive semi-definite" ➔ $\mathbf{z}^T \mathbf{K} \mathbf{z} >= 0$ for all $\mathbf{z}$

- Duality
- Gram matrix: $\mathbf{K}$: $k_{ij} = K(\mathbf{x}_i,\mathbf{x}_j)$

K=

K(1,1)	K(1,2)	K(1,3)	…	K(1,m)
K(2,1)	K(2,2)	K(2,3)	…	K(2,m)
…	…	…	…	…
K(m,1)	K(m,2)	K(m,3)	…	K(m,m)

A FAMILIAR KERNEL

Learning as optimization for regularized logistic regression + hashes

- Algorithm: $w^j = w^j + \lambda(y - p)x^j - \lambda 2\mu w^j$
- Initialize arrays *W, A* of size *R* and set *k=0*
- For each iteration t=1,…T
  - For each example ($\mathbf{x}_i$,y_i)

$$V[h] = \sum_{j:hash(j)\%R==h} x_i^j$$

    - V is a hash table
    - For $j : x^j>0$ increment $V[h[j]]$ by x^j
    - $p_i = \ldots ; k{+}{+}$
    - For each hash value h: $V[h]>0$:
      - » $W[h] \;\; {*}{=} (1 - \lambda 2\mu)^{k-A[h]}$
      - » $W[h] = W[h] + \lambda(y_i - p^i) V[h]$
      - » $A[h] = k$

Hash Kernels

Qinfeng Shi, James Petterson
Australian National University and NICTA,
Canberra, Australia

Gideon Dror
Department of Computer Science
Academic College of Tel-Aviv-Yaffo, Israel

John Langford, Alex Smola, Alex Strehl
Yahoo! Research
New York, NY and Santa Clara, CA, USA

Vishy Vishwanathan
Department of Statistics
Purdue University, IN, USA

Some details

Slightly different hash to avoid systematic bias

$$V[h] = \sum_{j: hash(j)\%R==h} x_i^j$$

$$\varphi[h] = \sum_{j: hash(j)\%m==h} \xi(j) x_i^j, \quad \text{where } \xi(j) \in \{-1, +1\}$$

m is the number of buckets you hash into (R in my discussion)

Some details

Slightly different hash to avoid systematic bias

$$\varphi[h] = \sum_{j:hash(j)\%m==h} \xi(j)x_i^j, \quad \text{where } \xi(j) \in \{-1,+1\}$$

Lemma 2 *The hash kernel is unbiased, that is* $\mathbf{E}_\phi[\langle x, x'\rangle_\phi] = \langle x, x'\rangle$. *Moreover, the variance is* $\sigma^2_{x,x'} = \frac{1}{m}\left(\sum_{i\neq j} x_i^2 {x'_j}^2 + x_i x'_i x_j x'_j\right)$, *and thus, for* $\|x\|_2 = \|x'\|_2 = 1$, $\sigma^2_{x,x'} = O\left(\frac{1}{m}\right)$.

I.e., for large feature sets the variance should be low

Some details

Theorem 3 *Let $\epsilon < 1$ be a fixed constant and x be a given instance. Let $\eta = \frac{\|x\|_\infty}{\|x\|_2}$. Under the assumptions above, the hash kernel satisfies the following inequality*

$$\Pr\left\{\frac{\left|\,\|x\|_\phi^2 - \|x\|_2^2\,\right|}{\|x\|_2^2} \geq \sqrt{2}\sigma_{x,x} + \epsilon\right\} \leq \exp\left(-\frac{\sqrt{\epsilon}}{4\eta}\right).$$

I.e. – a hashed vector is probably close to the original vector

Some details

Corollary 4 *For two vectors x and x', let us define*

$$\sigma := \max(\sigma_{x,x}, \sigma_{x',x'}, \sigma_{x-x',x-x'})$$
$$\eta := \min\left(\frac{\|x\|_\infty}{\|x\|_2}, \frac{\|x'\|_\infty}{\|x'\|_2}, \frac{\|x-x'\|_\infty}{\|x-x'\|_2}\right).$$

Also let $\Delta = \|x\|^2 + \|x'\|^2 + \|x-x'\|^2$. Under the assumptions above, we have that

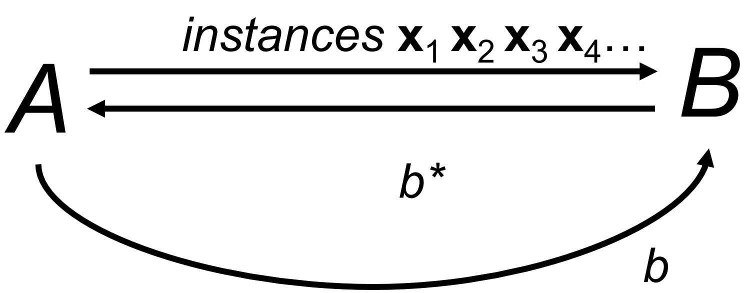
$$\Pr\left[\left|\langle x, x'\rangle_\phi - \langle x, x'\rangle\right| > (\sqrt{2}\sigma + \epsilon)\Delta/2\right] < 3e^{-\frac{\sqrt{\epsilon}}{4\eta}}.$$

I.e. the inner products between x and x′ are probably not changed too much by the hash function: a classifier will probably still work

The Voted Perceptron for Ranking and Structured Classification

William Cohen

The voted perceptron *for ranking*



Compute: $y_i = \hat{\mathbf{v}}_k \,.\, \mathbf{x}_i$
Return: the index b^* of the "best" $\mathbf{x}_i$

If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + \mathbf{x}_b - \mathbf{x}_{b^*}$

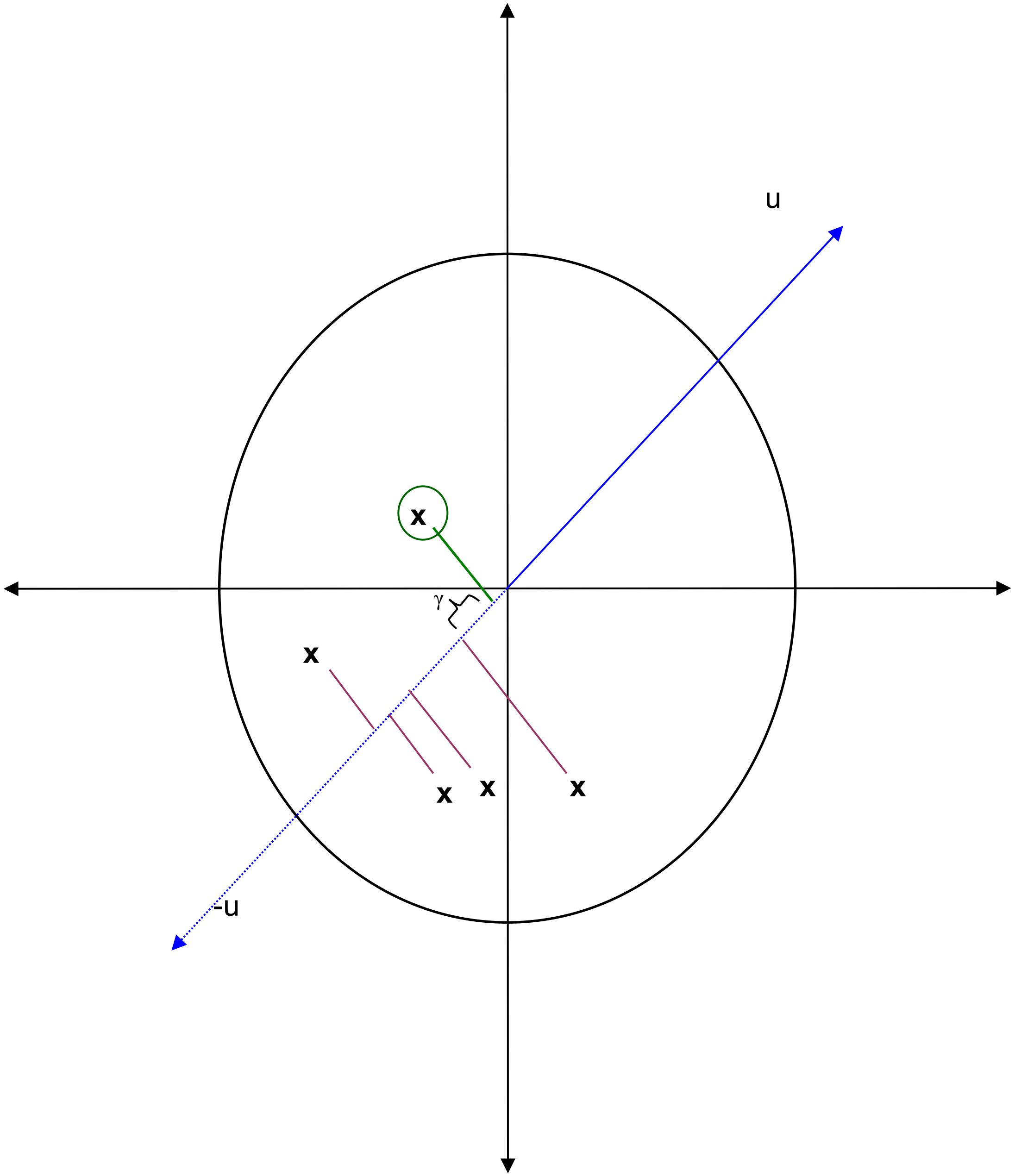
Margin γ. A must provide examples that can be correctly ranked with some vector $\mathbf{u}$ with margin $\gamma > 0$, ie

$$\exists \mathbf{u} : \forall \mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,n_i}, \ell \text{ given by } A,\ \forall j \neq \ell,\ \mathbf{u} \cdot \mathbf{x}_\ell - \mathbf{u} \cdot \mathbf{x}_j > \gamma$$

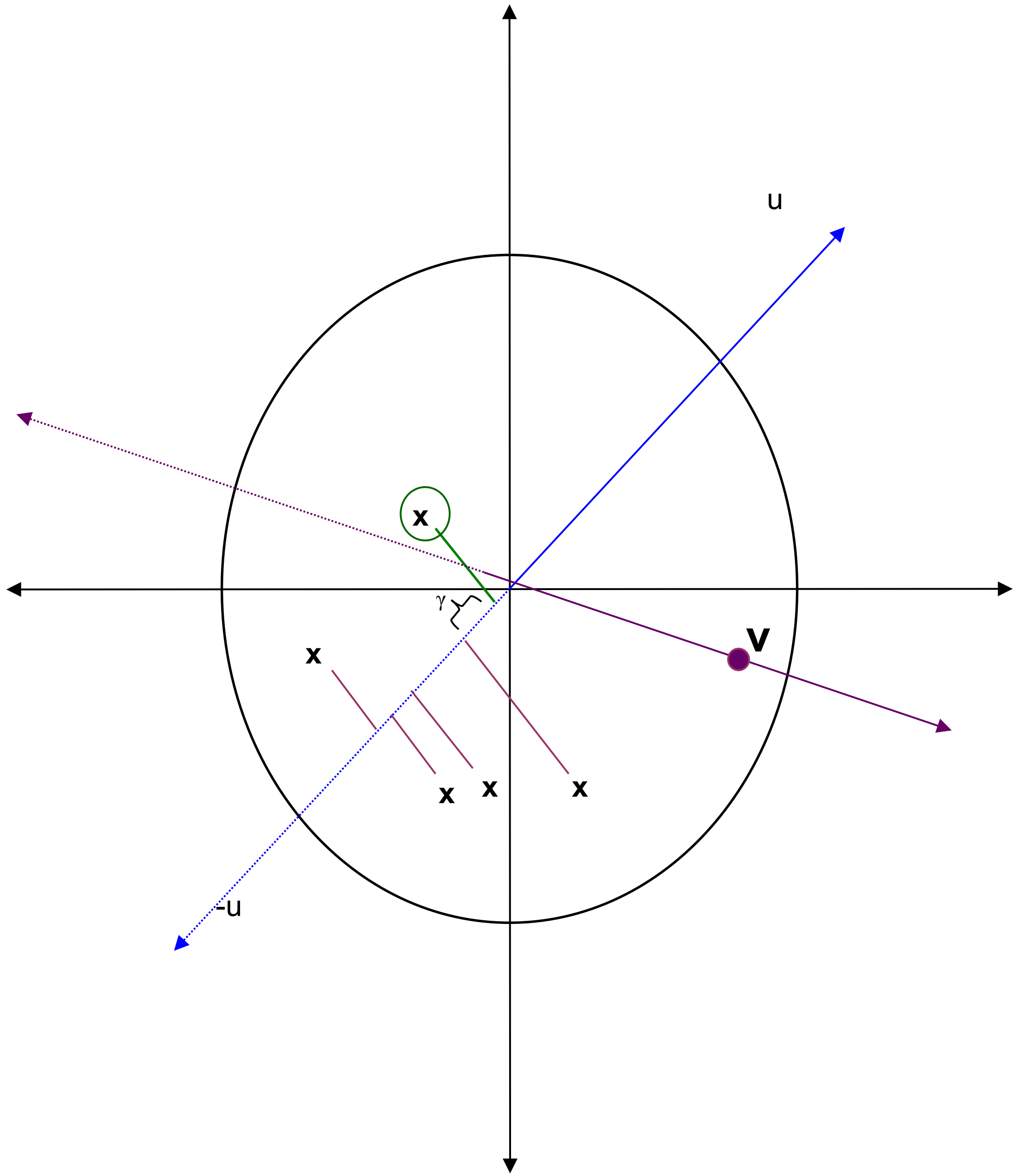
and furthermore, $\|\mathbf{u}\|^2 = 1$.

Radius R. A must provide examples "near the origin", ie

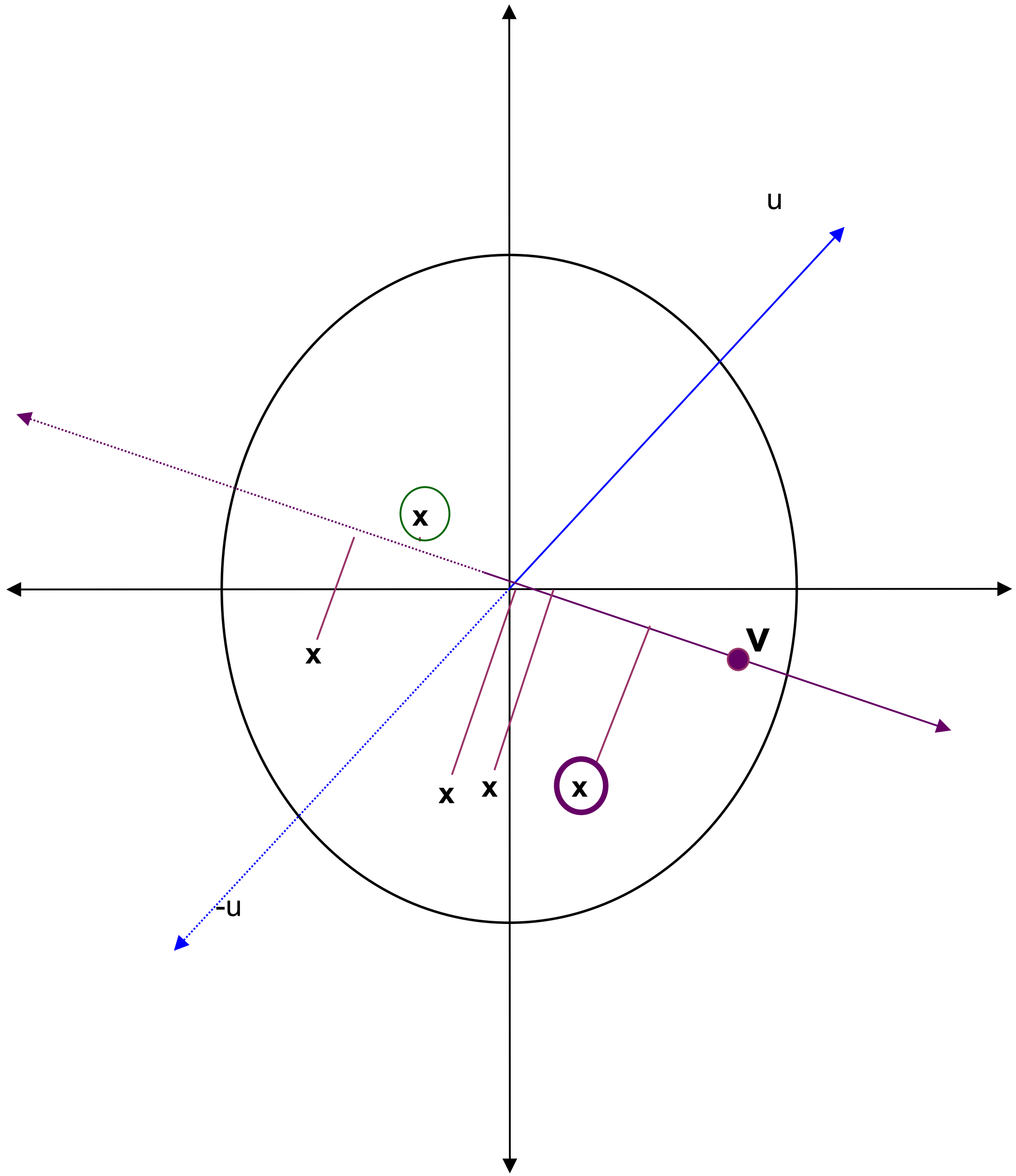
$$\forall \mathbf{x}_i \text{ given by } A,\ \|\mathbf{x}\|^2 < R^2$$



Ranking some x' s with the target vector **u**

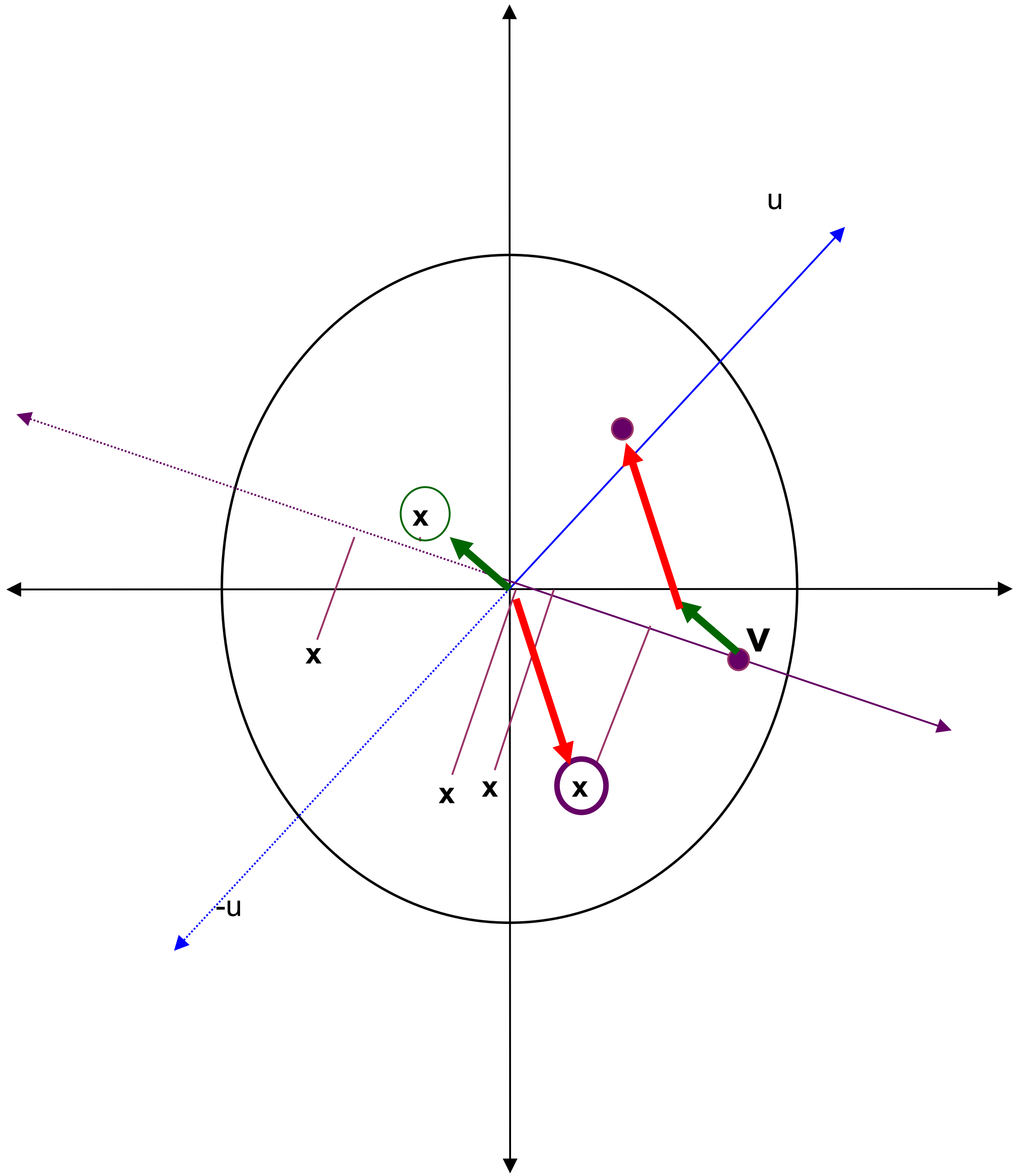


Ranking some x' s with some guess vector **v** – part 1

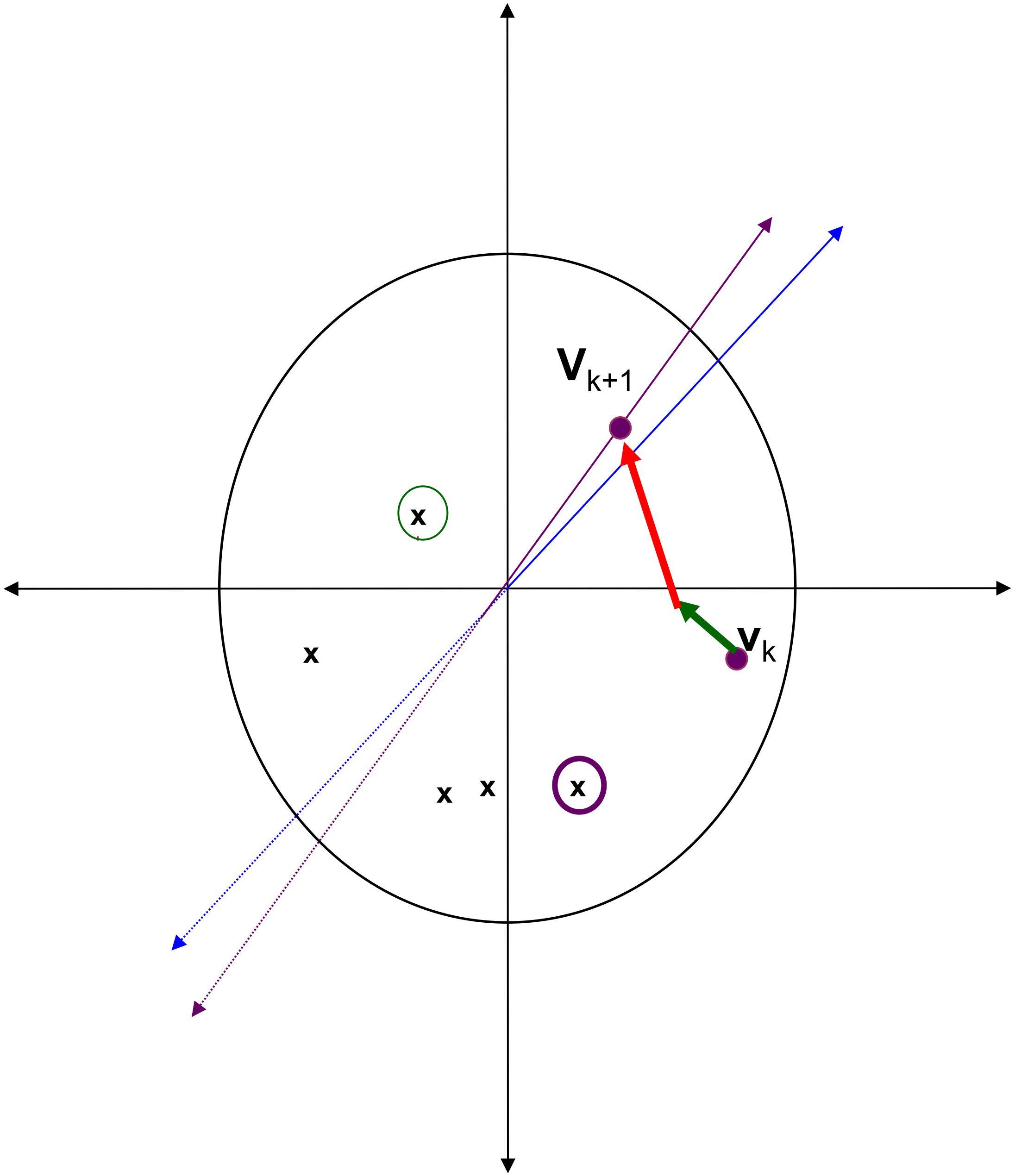


Ranking some x' s with some guess vector **v –** part 2.

The purple-circled x is x_{b^*} - the one the learner has chosen to rank highest. The green circled x is x_b, the right answer.



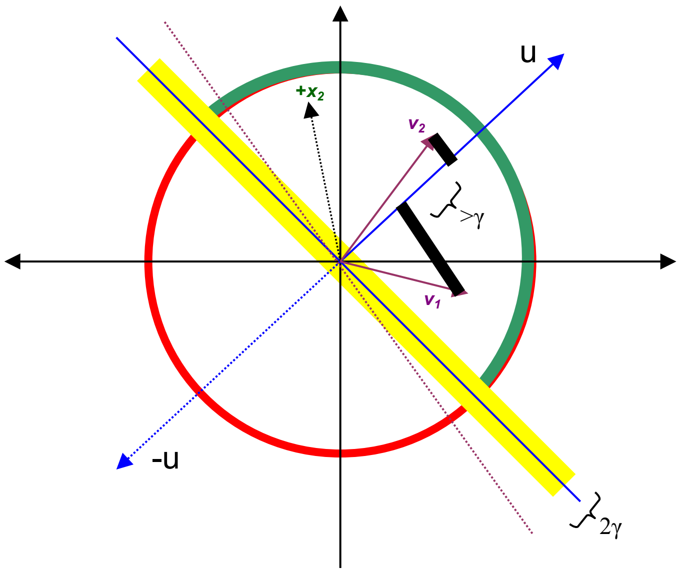
Correcting **v** by adding $x_b - x_{b^*}$

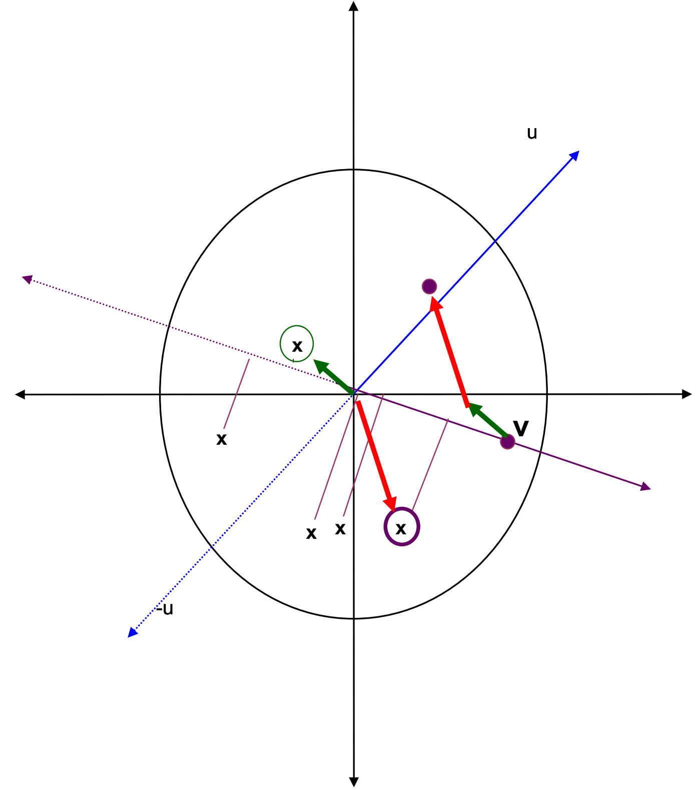


Correcting **v** by adding $x_b - x_{b^*}$

(part 2)

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



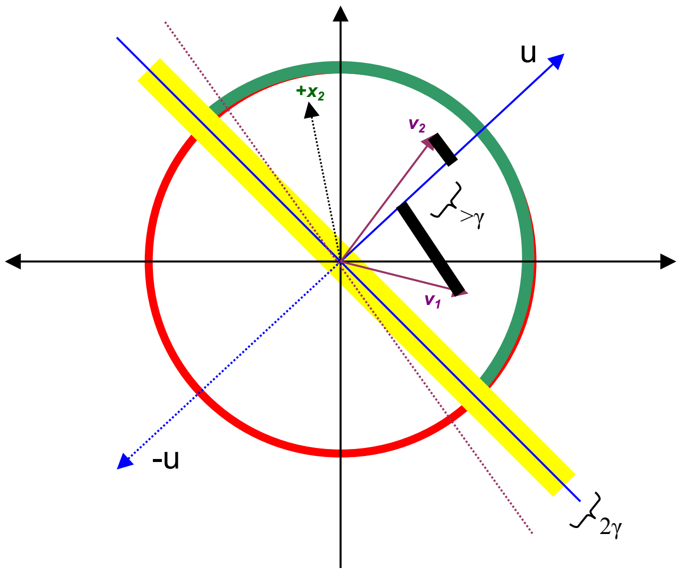


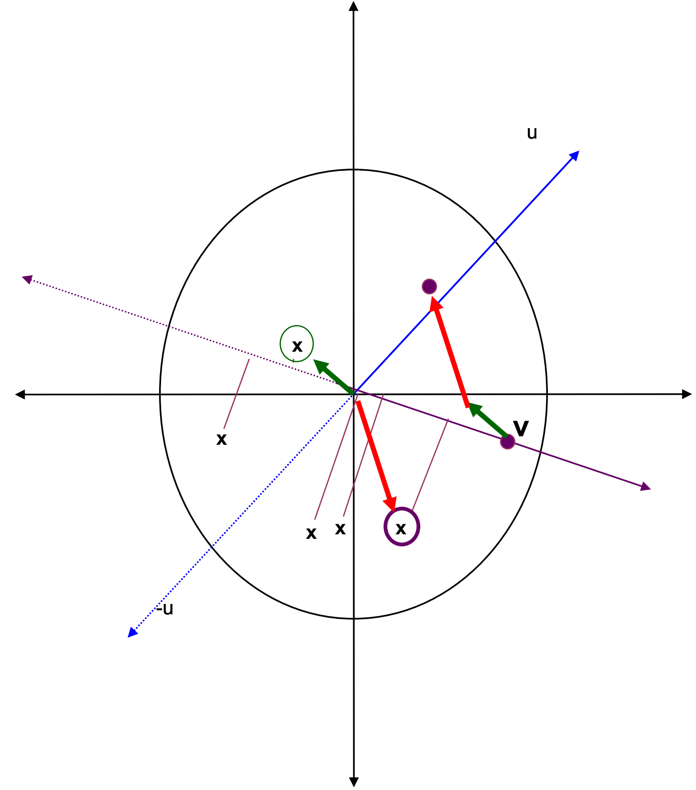
Lemma 1 $\forall k$, $\mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$. *In other words, the dot product between* $\mathbf{v}_k$ *and* $\mathbf{u}$ *increases with each mistake, at a rate depending on the margin* γ.

Proof:

$$
\begin{aligned}
& \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k + y_i \mathbf{x}_i) \cdot \mathbf{u} \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k \cdot \mathbf{u}) + y_i (\mathbf{x}_i \cdot \mathbf{u}) \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} \geq \mathbf{v}_k \cdot \mathbf{u} + \gamma \\
\Rightarrow \quad & \mathbf{v}_k \cdot \mathbf{u} \geq k\gamma
\end{aligned}
$$

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



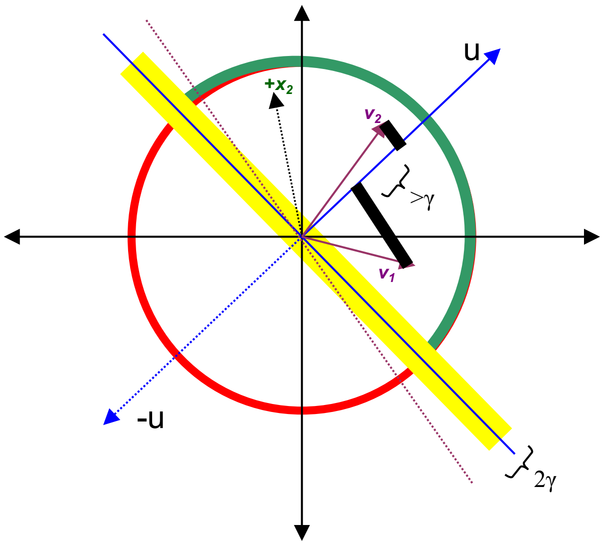


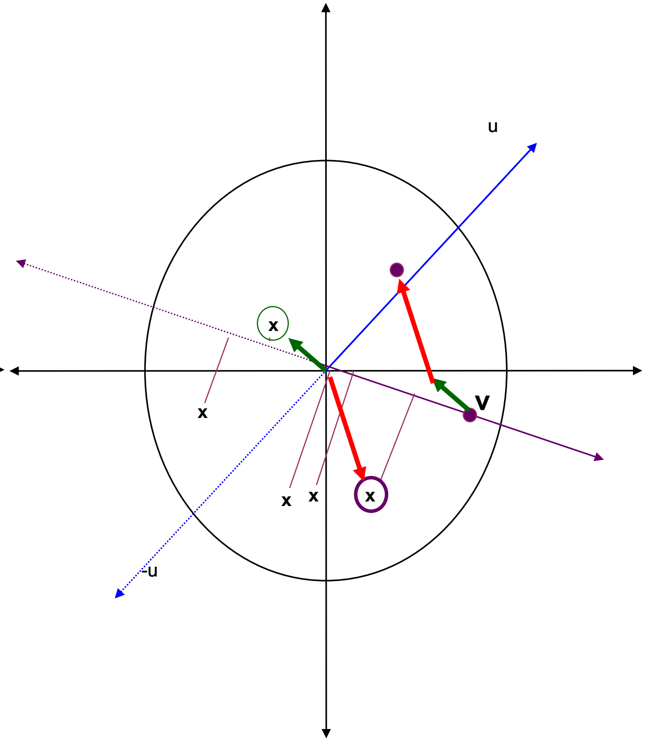
Lemma ~~1~~ 3 $\forall k$, $\mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$. *In other words, the dot product between* $\mathbf{v}_k$ *and* $\mathbf{u}$ *increases with each mistake, at a rate depending on the margin* γ.

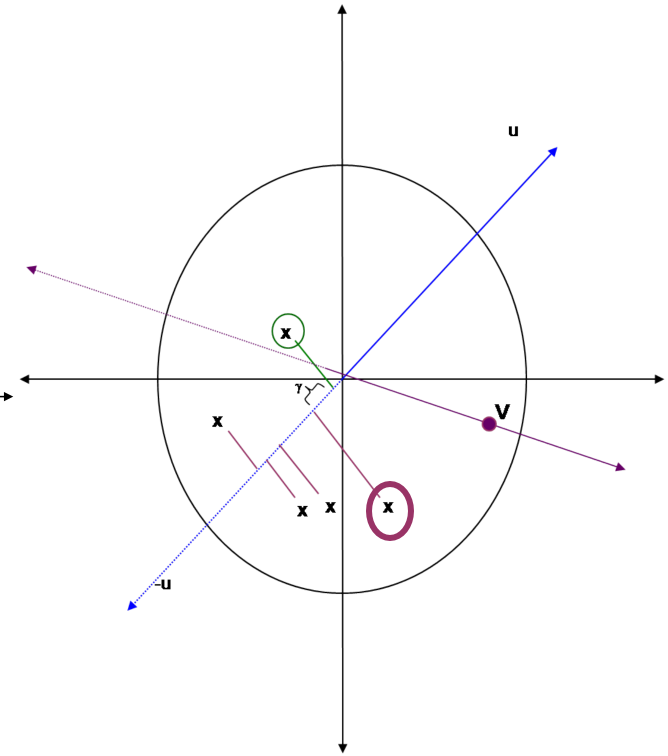
$$\begin{aligned}
& \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k + y_i \mathbf{x}_i) \cdot \mathbf{u} \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k \cdot \mathbf{u}) + y_i (\mathbf{x}_i \cdot \mathbf{u}) \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} \geq \mathbf{v}_k \cdot \mathbf{u} + \gamma \\
\Rightarrow \quad & \mathbf{v}_k \cdot \mathbf{u} \geq k\gamma
\end{aligned}$$

$$\begin{aligned}
& \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k + \mathbf{x}_{i,\ell} - \mathbf{x}_{i,\hat{\ell}}) \cdot \mathbf{u} \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} = \mathbf{v}_k \cdot \mathbf{u} + \mathbf{x}_{i,\ell} \cdot \mathbf{u} - \mathbf{x}_{i,\hat{\ell}} \cdot \mathbf{u} \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} \geq \mathbf{v}_k \cdot \mathbf{u} + \gamma \\
\Rightarrow \quad & \mathbf{v}_k \cdot \mathbf{u} \geq k\gamma
\end{aligned}$$

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







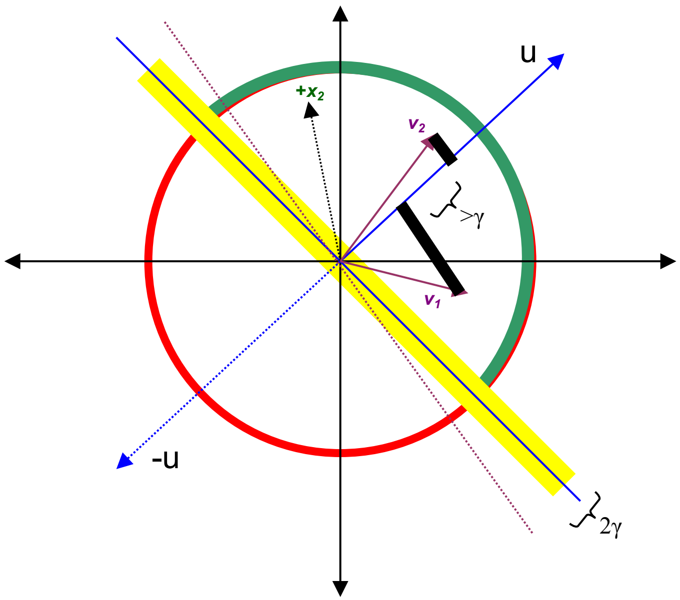
Lemma ~~1~~ 3 $\forall k$, $\mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$. *In other words, the dot product between* $\mathbf{v}_k$ *and* $\mathbf{u}$ *increases with each mistake, at a rate depending on the margin* γ.

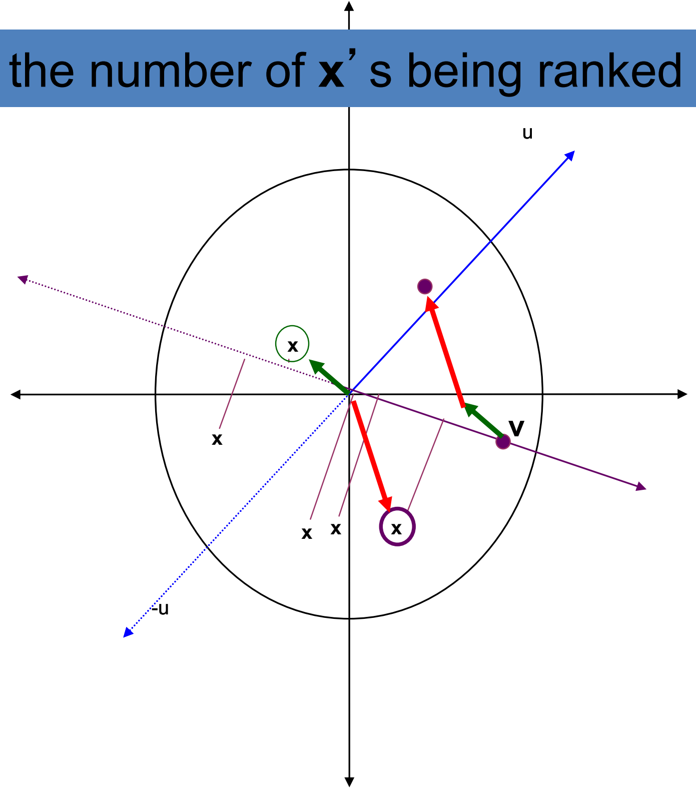
$$\begin{aligned}
& \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k + y_i \mathbf{x}_i) \cdot \mathbf{u} \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k \cdot \mathbf{u}) + y_i(\mathbf{x}_i \cdot \mathbf{u}) \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} \geq \mathbf{v}_k \cdot \mathbf{u} + \gamma \\
\Rightarrow \quad & \mathbf{v}_k \cdot \mathbf{u} \geq k\gamma
\end{aligned}$$

$$\begin{aligned}
& \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k + \mathbf{x}_{i,\ell} - \mathbf{x}_{i,\hat{\ell}}) \cdot \mathbf{u} \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} = \mathbf{v}_k \cdot \mathbf{u} + \mathbf{x}_{i,\ell} \cdot \mathbf{u} - \mathbf{x}_{i,\hat{\ell}} \cdot \mathbf{u} \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} \geq \mathbf{v}_k \cdot \mathbf{u} + \gamma \\
\Rightarrow \quad & \mathbf{v}_k \cdot \mathbf{u} \geq k\gamma
\end{aligned}$$

Notice this doesn't depend *at all* on the number of **x**'s being ranked

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





Lemma 4 $\forall k,\ \|\mathbf{v}_k\|^2 \leq 2kR.$

Theorem 2 *Under the rules of the ranking perceptron game, it is always the case that* $k < 2R/\gamma^2$.

Neither proof depends on the *dimension* of the **x**'s.

Ranking perceptrons ➔ structured perceptrons

- The API:
  - A sends B a (maybe **huge**) set of items to rank
  - B finds the single **best** one according to the current weight vector
  - A tells B which one was actually best
- Structured classification on a sequence
  - Input: list of words: $\mathbf{x}=(w_1,\ldots,w_n)$
  - Output: list of labels: $\mathbf{y}=(y_1,\ldots,y_n)$
  - If there are K classes, there are K^n labels possible for $\mathbf{x}$

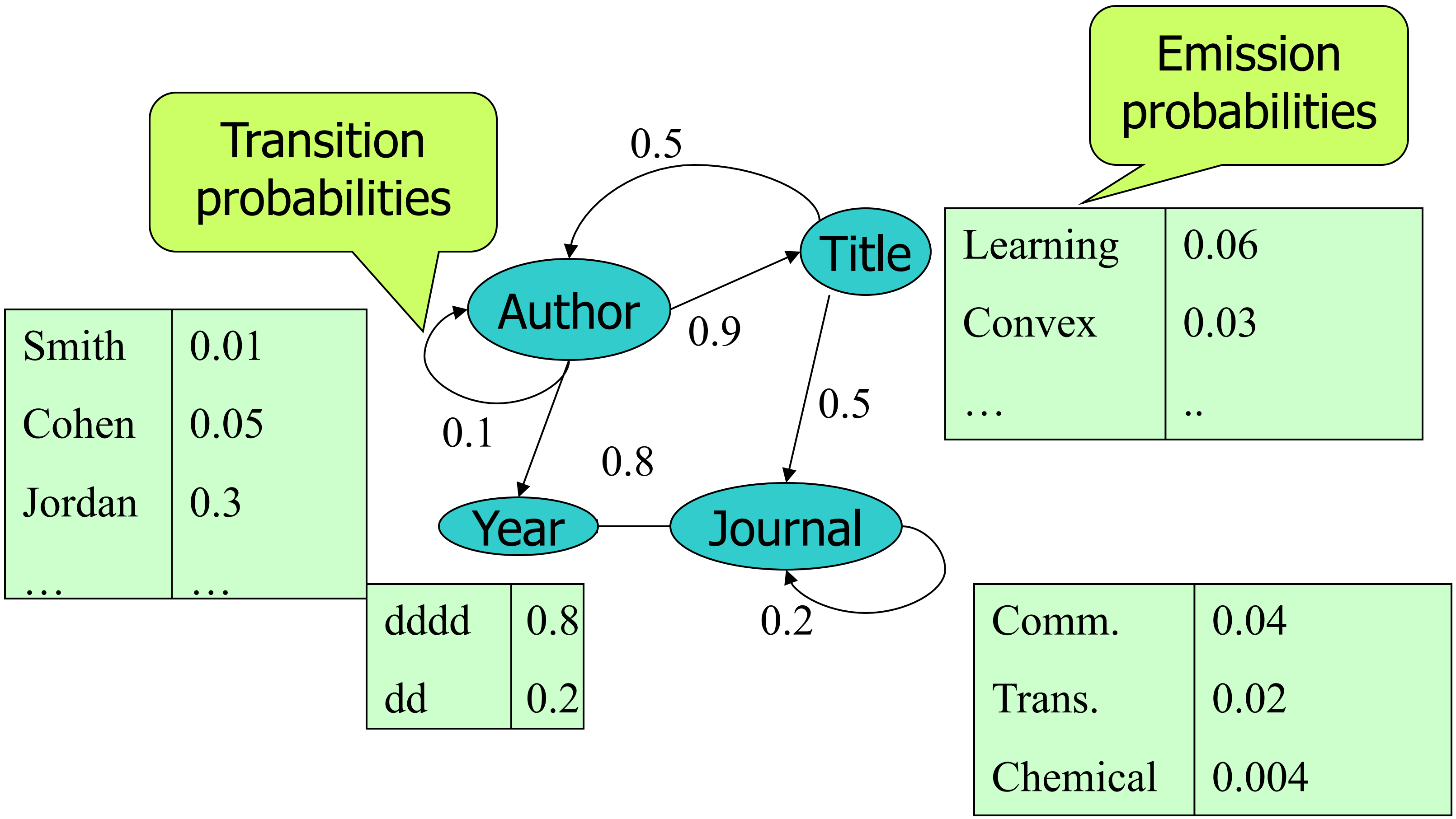
Borkar et al's: HMMs for segmentation

- Example: Addresses, bib records
- Problem: some DBs may split records up differently (eg no "mail stop" field, combine address and apt #, …) or not at all
- Solution: Learn to segment textual form of records

Author Year Title Journal Volume Page

P.P.Wangikar, T.P. Graycar, D.A. Estell, D.S. Clark, J.S. Dordick (1993)
Protein and Solvent Engineering of Subtilising BPN' in Nearly
Anhydrous Organic Media J.Amer. Chem. Soc. 115, 12231-12237.

IE with Hidden Markov Models



Inference for linear-chain CRFs

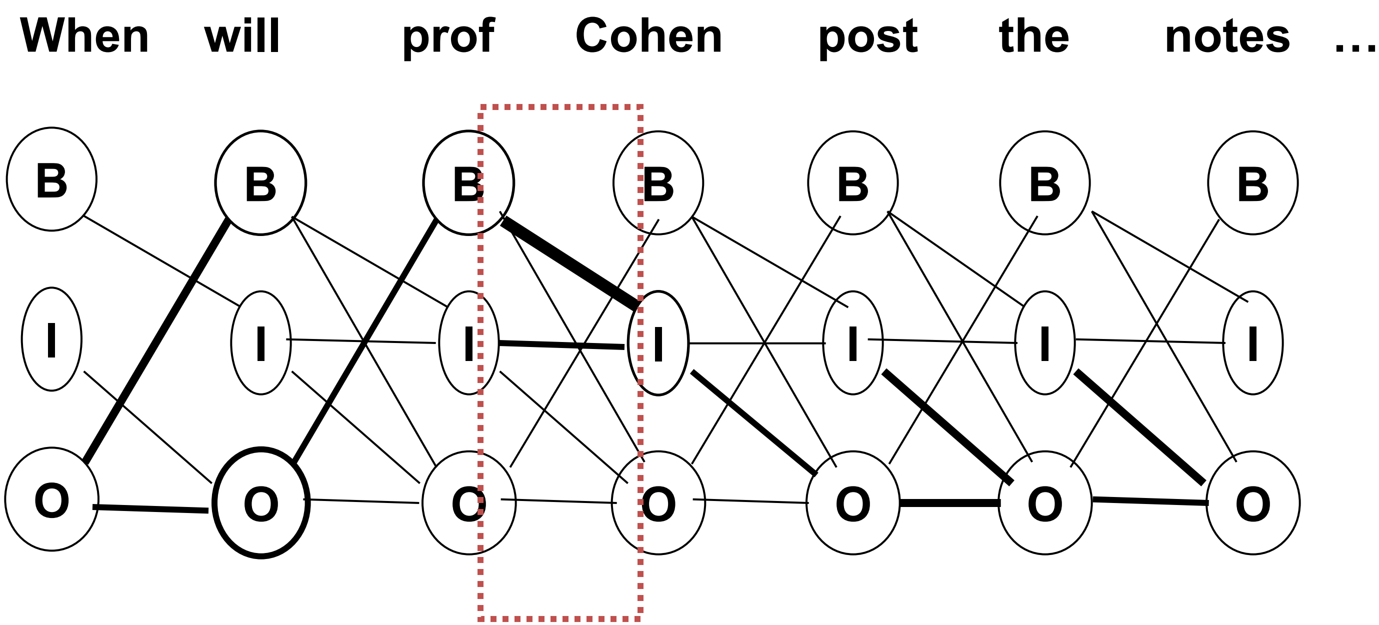
When will prof Cohen post the notes …

Idea 1: features are properties of *two adjacent tokens*, and the *pair* of labels assigned to them (Begin,Inside,Outside)

- (y(i)==B or y(i)==I) and (token(i) is capitalized)
- (y(i)==I and y(i-1)==B) and (token(i) is hyphenated)
- (y(i)==B and y(i-1)==B)
  - eg "tell Rose William is on the way"

Idea 2: construct a graph where each *path* is a possible sequence labeling.

Inference for a linear-chain CRF



- Inference: find the highest-weight path given a weighting of features
- This can be done efficiently using dynamic programming (Viterbi)

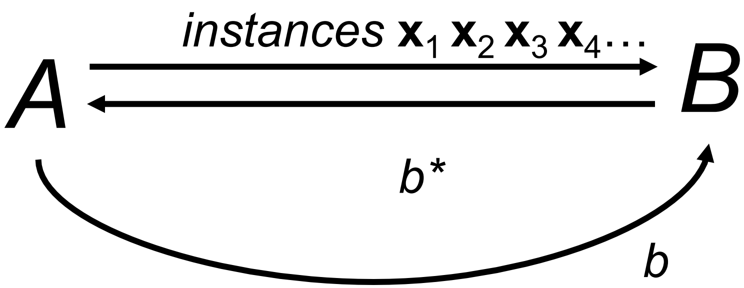
Ranking perceptrons → structured perceptrons

- The API:
  - A sends B a (maybe **huge**) set of items to rank
  - B finds the single **best** one according to the current weight vector
  - A tells B which one was actually best
- Structured classification on a sequence
  - Input: list of words: $\mathbf{x}=(w_1,\ldots,w_n)$
  - Output: list of labels: $\mathbf{y}=(y_1,\ldots,y_n)$
  - If there are K classes, there are K^n labels possible for $\mathbf{x}$

Ranking perceptrons → structured perceptrons

- New API:
  - A sends B the word sequence **x**
  - B finds the single **best y** according to the current weight vector using Viterbi
  - A tells B which **y** was actually best
  - This is equivalent to ranking pairs g=(**x**,**y'**)
- Structured classification on a sequence
  - Input: list of words: $\mathbf{x}=(w_1,\ldots,w_n)$
  - Output: list of labels: $\mathbf{y}=(y_1,\ldots,y_n)$
  - If there are K classes, there are K^n labels possible for **x**

The voted perceptron *for ranking*



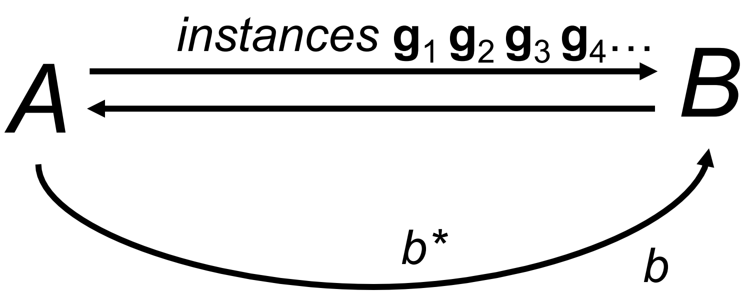
Compute: $\hat{y}_i = \mathbf{v}_k \cdot \mathbf{x}_i$

Return: the index b^* of the "best" $\mathbf{x}_i$

If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + \mathbf{x}_b - \mathbf{x}_{b^*}$

Change number one is notation: replace **x** with **g**

The voted perceptron *for structured classification tasks*



Compute: $y_i = \hat{\mathbf{v}}_k \cdot \mathbf{g}_i$
Return: the index b^* of the "best" $\mathbf{g}_i$

If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + \mathbf{g}_b - \mathbf{g}_{b^*}$

1. A sends B feature functions, and instructions for creating the instances $\mathbf{g}$:
   - A sends a word vector $\mathbf{x}_i$. Then B could create the instances $\mathbf{g}_1 = \mathbf{F}(\mathbf{x}_i,\mathbf{y}_1)$, $\mathbf{g}_2 = \mathbf{F}(\mathbf{x}_i,\mathbf{y}_2)$, …
   - but instead B just returns the $\mathbf{y}^*$ that gives the best score for the dot product $\mathbf{v}_k \cdot \mathbf{F}(\mathbf{x}_i,\mathbf{y}^*)$ by using Viterbi.
2. A sends B the correct label sequence $\mathbf{y}_i$.
3. On errors, B sets $\mathbf{v}_{k+1} = \mathbf{v}_k + \mathbf{g}_b - \mathbf{g}_{b^*} = \mathbf{v}_k + \mathbf{F}(\mathbf{x}_i,\mathbf{y}) - \mathbf{F}(\mathbf{x}_i,\mathbf{y}^*)$

Results from the original paper….

Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms

Michael Collins
AT&T Labs-Research, Florham Park, New Jersey.
mcollins@research.att.com

EMNLP 2002, Best paper

Collins' Experiments

- POS tagging
- NP Chunking (words and POS tags from Brill's tagger as features) and BIO output tags
- Compared logistic regression methods (MaxEnt) and "Voted Perceptron trained HMM's"
  - With and w/o averaging
  - With and w/o feature selection (count>5)

Collins' results

NP Chunking Results

Method	F-Measure	Numits
Perc, avg, cc=0	93.53	13
Perc, noavg, cc=0	93.04	35
Perc, avg, cc=5	93.33	9
Perc, noavg, cc=5	91.88	39
ME, cc=0	92.34	900
ME, cc=5	92.65	200

POS Tagging Results

Method	Error rate/%	Numits
Perc, avg, cc=0	2.93	10
Perc, noavg, cc=0	3.68	20
Perc, avg, cc=5	3.03	6
Perc, noavg, cc=5	4.04	17
ME, cc=0	3.4	100
ME, cc=5	3.28	200

Figure 4: Results for various methods on the part-of-speech tagging and chunking tasks on development data. All scores are error percentages. Numits is the number of training iterations at which the best score is achieved. Perc is the perceptron algorithm, ME is the maximum entropy method. Avg/noavg is the perceptron with or without averaged parameter vectors. cc=5 means only features occurring 5 times or more in training are included, cc=0 means all features in training are included.

Where we are…

- Experiments with a hash-trick implementation of logistic regression
- Next question:
  - how do you parallelize SGD, or more generally, this kind of streaming algorithm?
  - each example affects the next prediction ➔ order matters ➔ parallelization changes the behavior
  - we will step back to perceptrons and then step forward to **parallel perceptrons**
  - then another nice parallel learning algorithm
  - then a midterm