Perceptrons – the story continues

On-line learning/regret analysis

- Optimization
  - is a great model of what you **want** to do
  - a less good model of what you have **time** to do

- Example:
  - How much to we lose when we replace gradient descent with SGD?
  - what if we can only approximate the local gradient?
  - what if the distribution changes over time?
  - …
- One powerful analytic approach: online-learning aka regret analysis (~aka on-line optimization)

Theory: the prediction game

- Player A:
  - picks a "target concept" c
    - for now - from a finite set of possibilities C (e.g., all decision trees of size *m*)
  - for t=1,….,
    - Player A picks $\mathbf{x}=(x_1,\ldots,x_n)$ and sends it to B
      - For now, from a finite set of possibilities (e.g., all binary vectors of length *n*)
    - B predicts a label, $\hat{\mathbf{y}}$, and sends it to A
    - A sends B the true label $y=c(\mathbf{x})$
    - we record if B made a *mistake* or not
  - We care about the *worst case* number of mistakes B will make over *all possible* concept & training sequences of any length
    - The "Mistake bound" for B, $M_B(C)$, is this bound

The voted perceptron: a simple algorithm with an easy mistake bound

$$\begin{tikzcd}
A \arrow[r, shift left, "\text{instance } \mathbf{x}_i"] \arrow[r, bend right=60, "y_i"'] & B \arrow[l, shift left, "\hat{y}_i"]
\end{tikzcd}$$

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 game

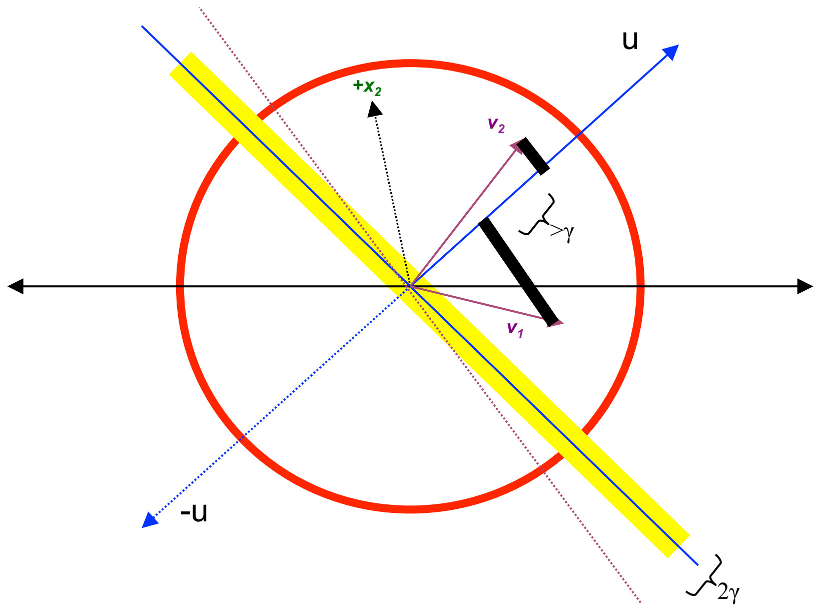
x is a vector
y is -1 or +1

$$\begin{tikzcd}
A \arrow[r, shift left, "\textit{instance } \mathbf{x}_i"] \arrow[r, bend right=60, "y_i"'] & B \arrow[l, shift left, "\hat{y}_i"]
\end{tikzcd}$$

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

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

mistake bound: $k \leq \left(\dfrac{R}{\gamma}\right)^2$

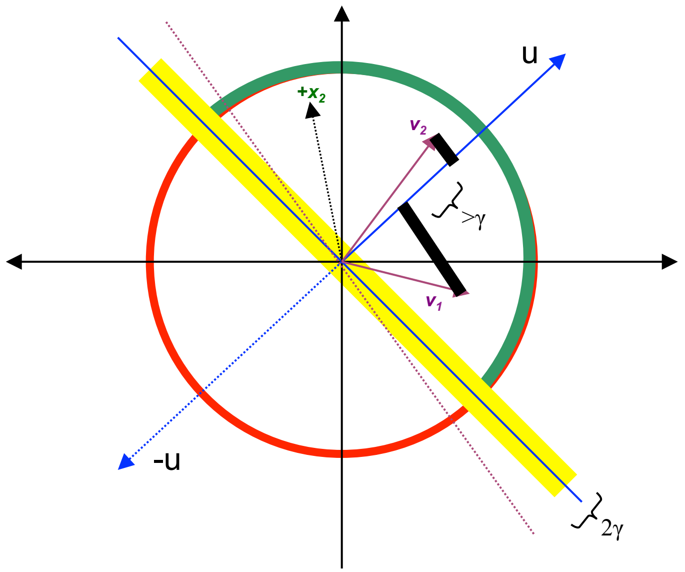


Depends on how easy the learning problem is, not dimension of vectors x

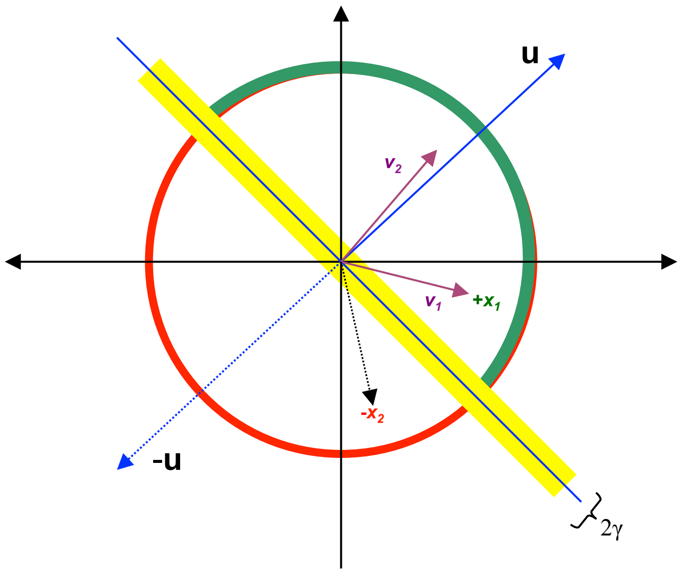
Fairly intuitive:

- "Similarity" of **v** to **u** looks like (**v.u**)/|**v.v**|
- (**v.u**) grows by >= γ after mistake
- (**v.v**) grows by <= R^2

(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: $\mathbf{v}_2=\mathbf{v}_1-\mathbf{x}_2$

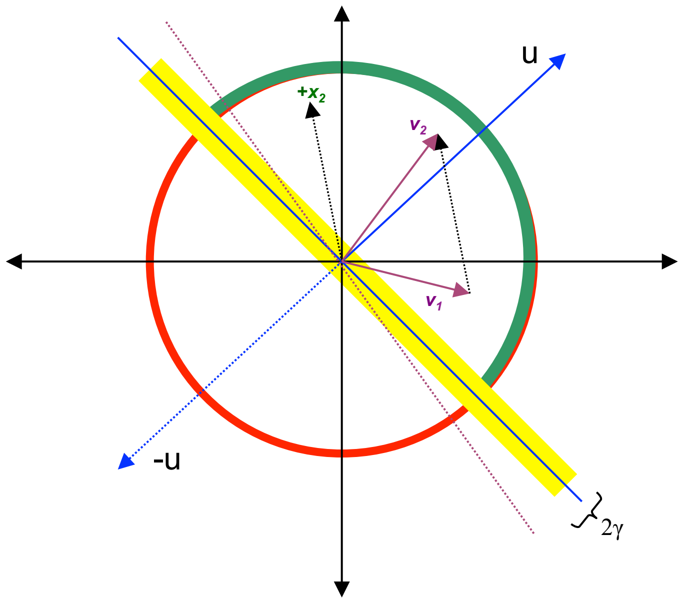


Lemma 1 $\forall k$, $\mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$. *In other words, the dot product between* $\mathbf{v}_k$ *and* $\mathbf{u}$ *increases with each mistake, at a rate depending on the margin* γ.

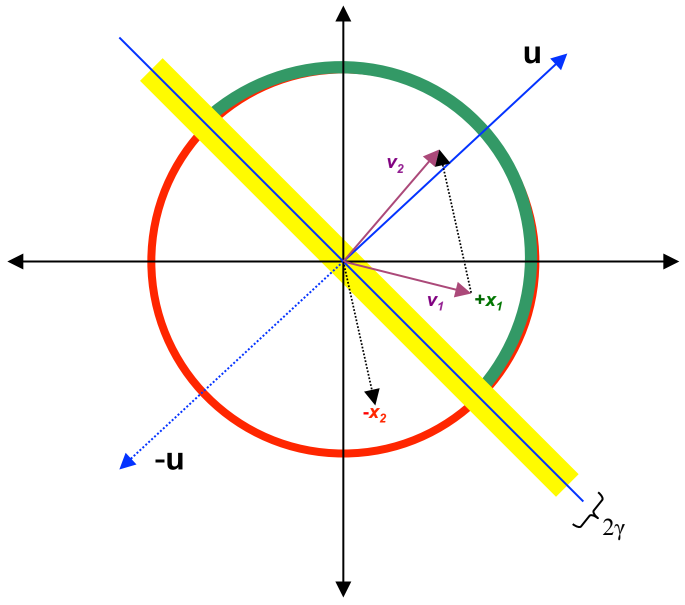
Proof:

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

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



(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 $\forall k$, $\|\mathbf{v}_k\|^2 \leq kR^2$. *In other words, the norm of* $\mathbf{v}_k$ *grows "slowly", at a rate depending on* R^2.

Proof:

$$\begin{aligned}
& \mathbf{v}_{k+1}\cdot\mathbf{v}_{k+1} = (\mathbf{v}_k + y_i\mathbf{x}_i)\cdot(\mathbf{v}_k + y_i\mathbf{x}_i) \\
\Rightarrow\ & \|\mathbf{v}_{k+1}\|^2 = \|\mathbf{v}_k\|^2 + 2y_i\mathbf{x}_i\cdot\mathbf{v}_k + y_i^2\|\mathbf{x}_i\|^2 \\
\Rightarrow\ & \|\mathbf{v}_{k+1}\|^2 = \|\mathbf{v}_k\|^2 + [\text{something negative}] + 1\|\mathbf{x}_i\|^2 \\
\Rightarrow\ & \|\mathbf{v}_{k+1}\|^2 \leq \|\mathbf{v}_k\|^2 + \|\mathbf{x}\|^2 \\
\Rightarrow\ & \|\mathbf{v}_{k+1}\|^2 \leq \|\mathbf{v}_k\|^2 + R^2 \\
\Rightarrow\ & \|\mathbf{v}_k\|^2 \leq kR^2
\end{aligned}$$

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.

Lemma 2 $\forall k$, $\|\mathbf{v}_k\|^2 \leq kR^2$. *In other words, the norm of* $\mathbf{v}_k$ *grows "slowly", at a rate depending on* R.

$$\begin{aligned}
(k\gamma)^2 &\leq (\mathbf{v}_k \cdot \mathbf{u})^2 \\
\Rightarrow \quad k^2\gamma^2 &\leq \|\mathbf{v}_k\|^2 \|\mathbf{u}\|^2 \\
\Rightarrow \quad k^2\gamma^2 &\leq \|\mathbf{v}_k\|^2
\end{aligned}$$

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

$$\begin{aligned}
\Rightarrow \quad k^2\gamma^2 &\leq kR^2 \\
\Rightarrow \quad k\gamma^2 &\leq R^2 \\
\Rightarrow \quad k &\leq \frac{R^2}{\gamma^2} = \left(\frac{R}{\gamma}\right)^2
\end{aligned}$$

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

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

Summary

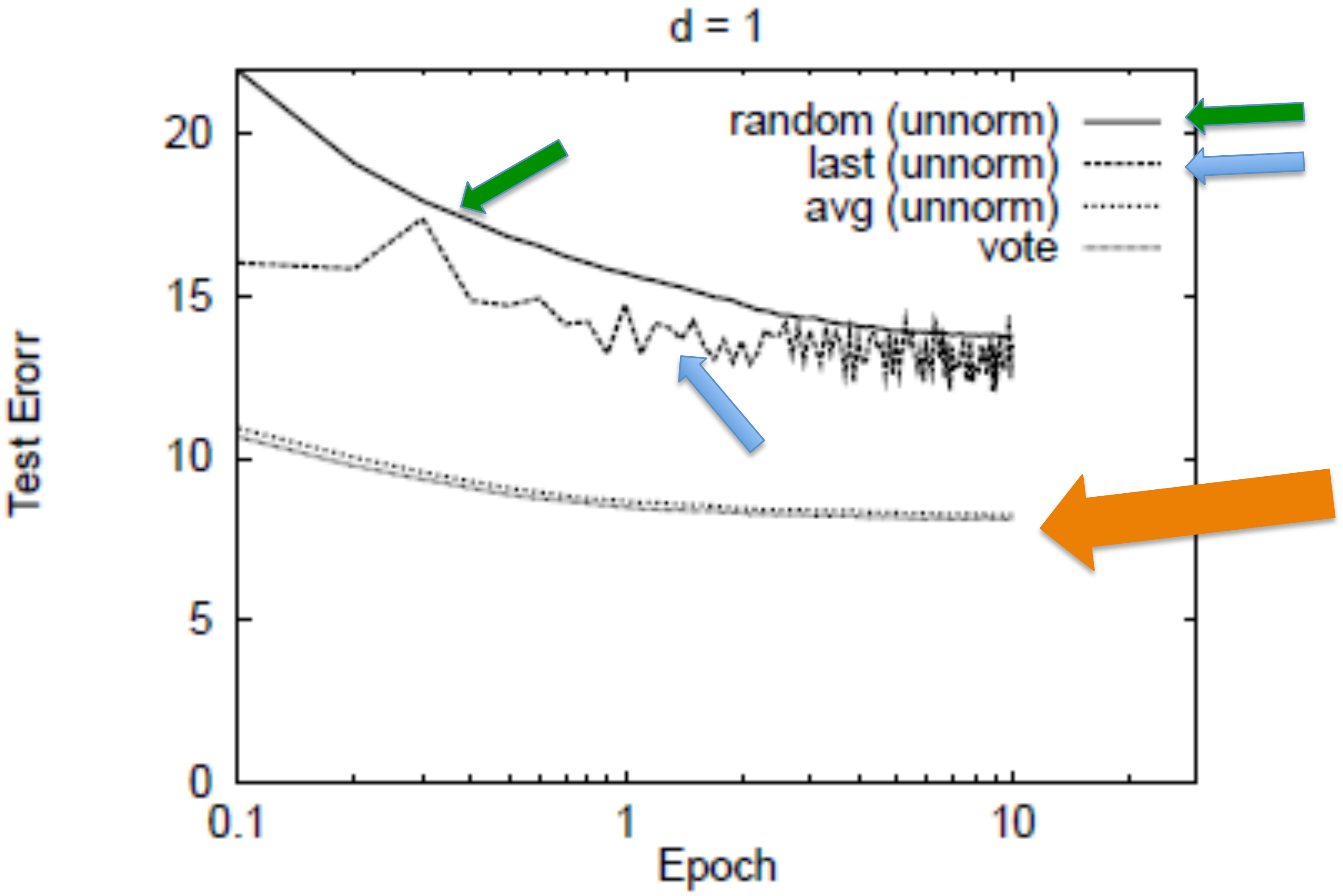
- We have shown that
  - *If* : exists a **u** with unit norm that has margin γ on examples in the seq $(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),\ldots$. (the data is easily separable)
  - *Then* : the perceptron algorithm makes $< R^2/\gamma^2$ mistakes on the sequence (where $R \geq \max_i ||\mathbf{x}_i||$)
  - *Independent* of dimension of the data or classifier (!)
- We can improve this (it's not *optimal,* just bounded)
  - There are many variants that rely on similar analysis (ROMMA, Passive-Aggressive, MIRA, …)
- We can extend the analysis to deal with noise
- We can extend the method to deal with other losses
- We can make the method more robust with averaging or voting

Theory: regret analysis

- In general: We care about the *worst case* "regret" of B
  - its loss on the *on*-line sequence
  - compared to the *best-case predictions it could make* if it had seem the entire sequence before making any predictions
- Mistakes for realizable case:
  - loss is 0/1 (mistake or not)
  - best case: 0 mistakes

Averaging and voting

- Theory says we get low error on the training set with a randomized algorithm:
  - pick a timepoint t
  - pick the perceptron $\mathbf{v}_\mathrm{k}$ active at t
  - predict on x with $\mathbf{v}_\mathrm{k}$
- We can approximate this by
  - voting the *predictions* of all the classifiers, weighted by their probability of being picked
  - averaging the classifiers themselves, with the same weights



perceptrons on the NIST digits data

Summary

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

- Very similar to SVMs:
  - in an SVM you search for a **u** that will *maximize margin* γ subject to a bound on **||u||**
  - or else, minimize **||u||** subject to constraint on the margin

  - like SVMs you can kernelize perceptrons….

SPARSIFYING THE AVERAGED PERCEPTRON UPDATE

Complexity of perceptron learning

- Algorithm: O(n)
- **v**=0
- for each example **x**,*y*:
  - if sign(**v**.**x**) != *y*
    - **v** = **v** + *y***x** O(|**x**|)=O(|d|)

- init hashtable
- for x_i!=0, v_i += yx_i

Complexity of perceptron learning

- Algorithm: O(n)
- **v**=0
- for each example **x**,*y*:
  - if sign(**v.x**) != *y*
    - **v** = **v** + *y***x** O(|**x**|)=O(|d|)

- init hashtable

- for x_i!=0, v_i += yx_i

Complexity of *averaged* perceptron

- Algorithm: ~~O(n)~~ O(n|V|)
- **vk=0**
- **va = 0**
- for each example **x**,*y*:
  - if sign(**vk.x**) != *y*
    - **va = va + vk** * mk
    - **vk = vk** + *y***x**
    - mk = 1
  - else
    - mk++

O(|V|)

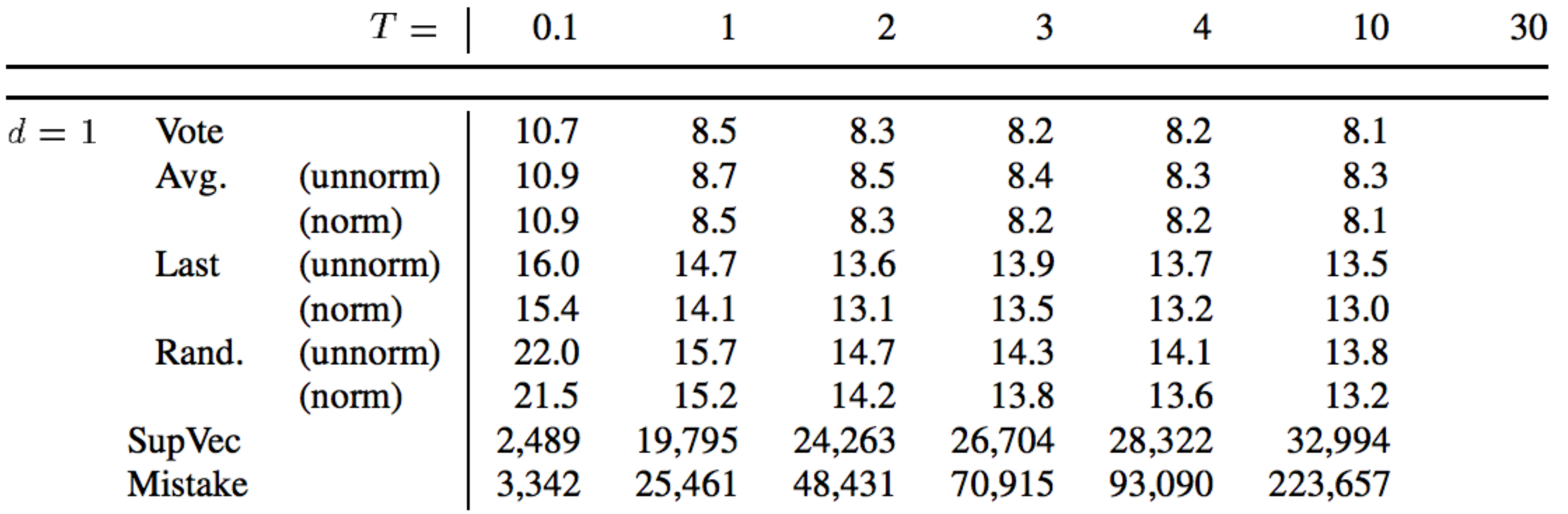
O(|**x**|)=O(|d|)

- init hashtables
- for vk_i!=0, va_i += vk_i
- for x_i!=0, v_i += yx_i

Complexity of *averaged* perceptron

```
• Algorithm:   ~~O(n)~~  O(n|V|)
• vk=0                                    • init hashtables
• va = 0
• for each example x,y:                   O(|V|)
    – if sign(vk.x) != y
        • va = va + vk * mk               • for $vk_i$!=0, $va_i$ += $vk_i$
        • vk = vk + yx                    • for $x_i$!=0, $v_i$ += $yx_i$
        • mk = 1
    – else                     O(|x|)=O(|d|)
        • mk++
• return va/k where
  k=#mistakes
```

There are lots of mistakes….



		$T =$	0.1	1	2	3	4	10	30
$d = 1$	Vote		10.7	8.5	8.3	8.2	8.2	8.1	
	Avg.	(unnorm)	10.9	8.7	8.5	8.4	8.3	8.3	
		(norm)	10.9	8.5	8.3	8.2	8.2	8.1	
	Last	(unnorm)	16.0	14.7	13.6	13.9	13.7	13.5	
		(norm)	15.4	14.1	13.1	13.5	13.2	13.0	
	Rand.	(unnorm)	22.0	15.7	14.7	14.3	14.1	13.8	
		(norm)	21.5	15.2	14.2	13.8	13.6	13.2	
	SupVec		2,489	19,795	24,263	26,704	28,322	32,994	
	Mistake		3,342	25,461	48,431	70,915	93,090	223,657	

Alternative averaged perceptron

- Algorithm:
- **vk** = 0
- **va** = 0
- for each example **x**,*y*:
  - **va** = **va** + **vk**
  - t = t+1
  - if sign(**vk.x**) != *y*
    - **vk** = **vk** + *y**x
- Return **va**/t

Observe:

$$\mathbf{vk} = \sum_{j \in S_k} y_j \mathbf{x}_j$$

S_k is the set of examples including the first k mistakes

Alternative averaged perceptron

- Algorithm:
- **vk** = 0
- **va** = 0
- for each example **x**,*y*:
  - **va** = **va** + $\sum_{j \in S_k} y_j \mathbf{x}_j$
  - t = t+1
  - if sign(**vk.x**) != *y*
    - **vk** = **vk** + *y**x
- Return **va**/t

So when there's a mistake at time t on **x**,*y*:

*y****x** is added to **va** on *every subsequent iteration*

Suppose you know T, the total number of examples in the stream…

Alternative averaged perceptron

- Algorithm:
- **vk** = 0
- **va** = 0
- for each example **x**,*y*:
  - ~~**va** = **va** + $\sum_{j \in S_k} y_j \mathbf{x}_j$~~
  - t = t+1
  - if sign(**vk.x**) != *y*
    - **vk** = **vk** + *y**x
    - **va** = **va** + (T-t)**y****x**
- Return **va**/T

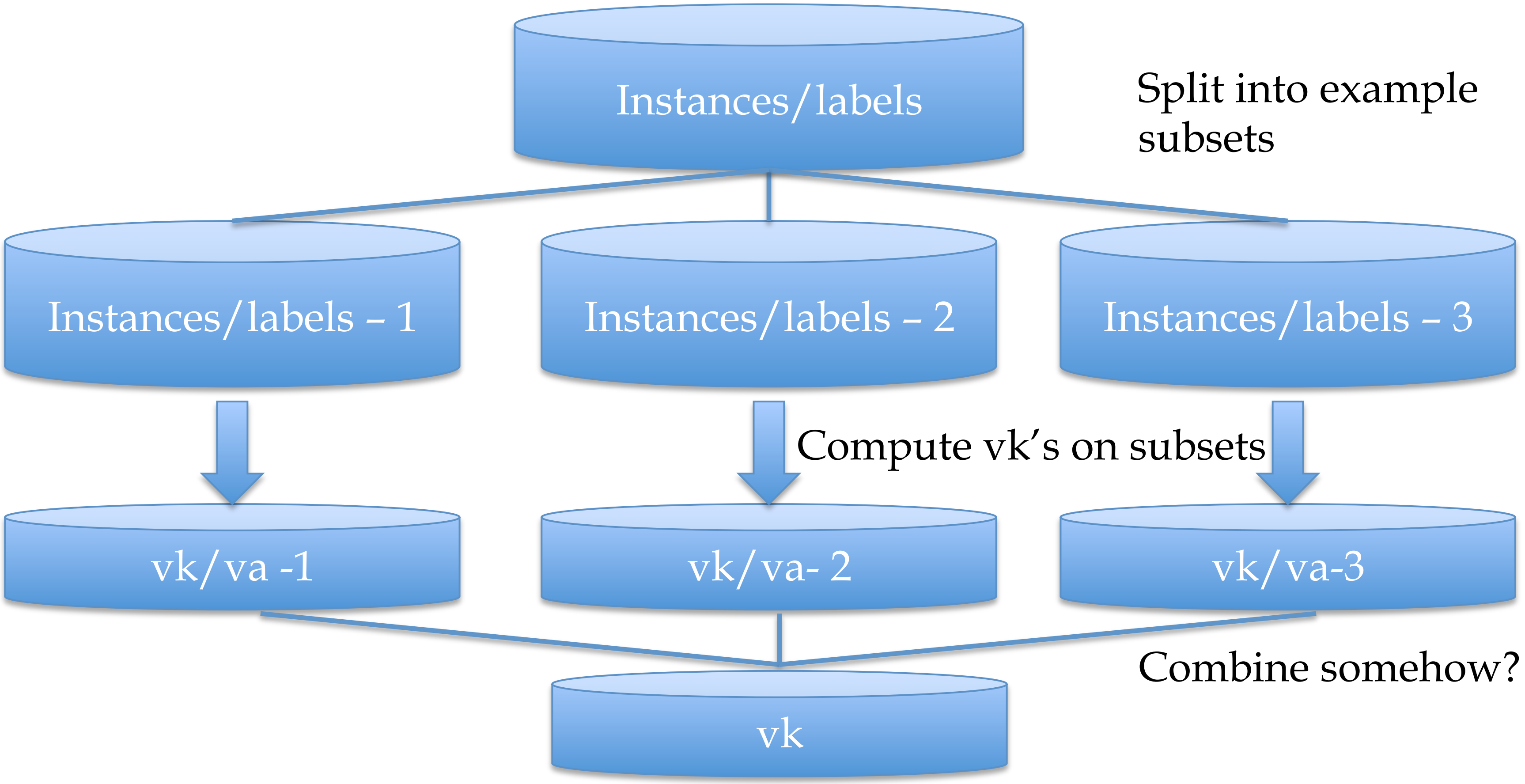
T = the total number of examples in the stream…

All subsequent additions of **x** to **va**

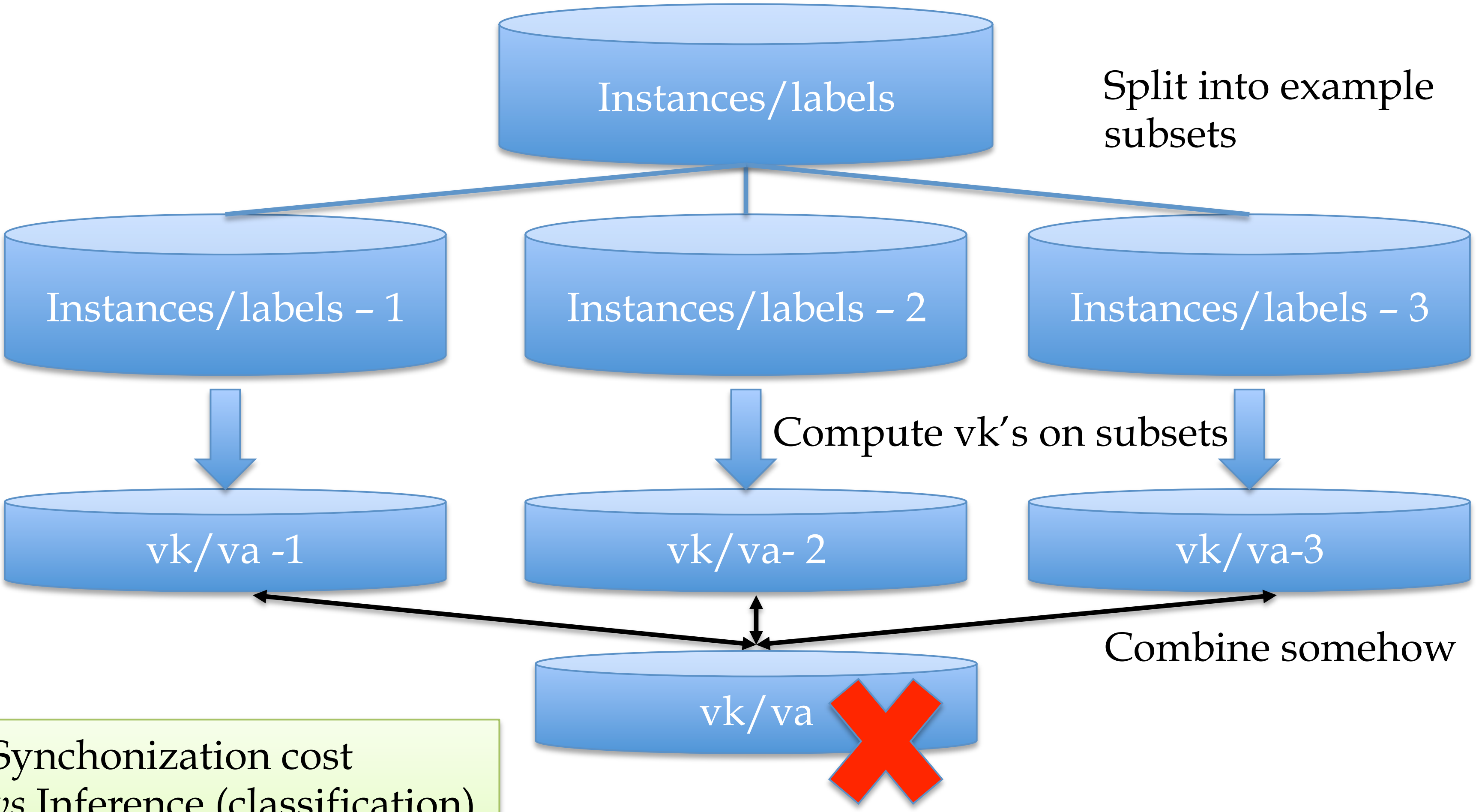
Unpublished? I figured this out recently, Leon Bottou knows it too

PERCEPTRONS IN PARALLEL

Parallelizing perceptrons



Parallelizing perceptrons



Synchonization cost
vs Inference (classification) cost

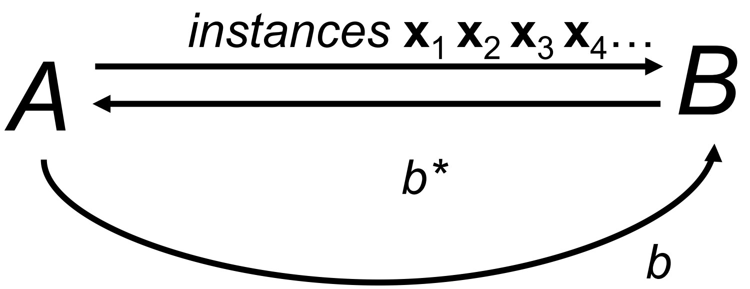
A hidden agenda

- Part of machine learning is good grasp of theory
- Part of ML is a good grasp of what hacks tend to work
- These are not always the same
  - Especially in big-data situations

- Catalog of useful tricks so far
  - Brute-force estimation of a joint distribution
  - Naive Bayes
  - Stream-and-sort, request-and-answer patterns
  - BLRT and KL-divergence (and when to use them)
  - TF-IDF weighting – especially IDF
    - it's often useful even when we don't understand why
  - Perceptron/mistake bound model
    - often leads to fast, competitive, easy-to-implement methods
    - parallel versions are non-trivial to implement/understand

The Voted Perceptron for Ranking and Structured Classification

The voted perceptron *for ranking*



instances $\mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3\ \mathbf{x}_4 \ldots$
A
B
*b**
b

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

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

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

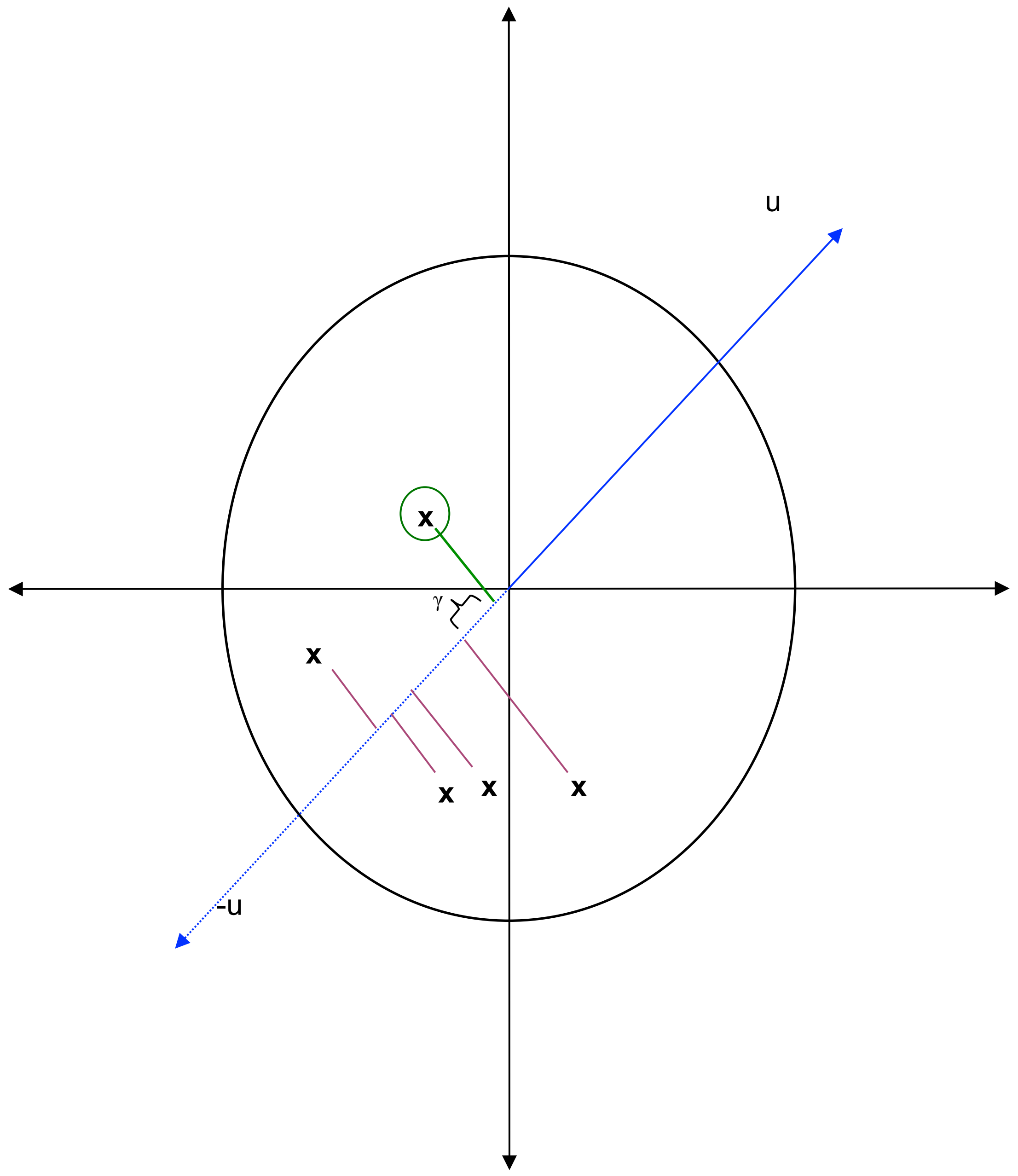
Margin γ. A must provide examples that can be correctly ranked with some vector $\mathbf{u}$ with margin $\gamma > 0$, ie

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

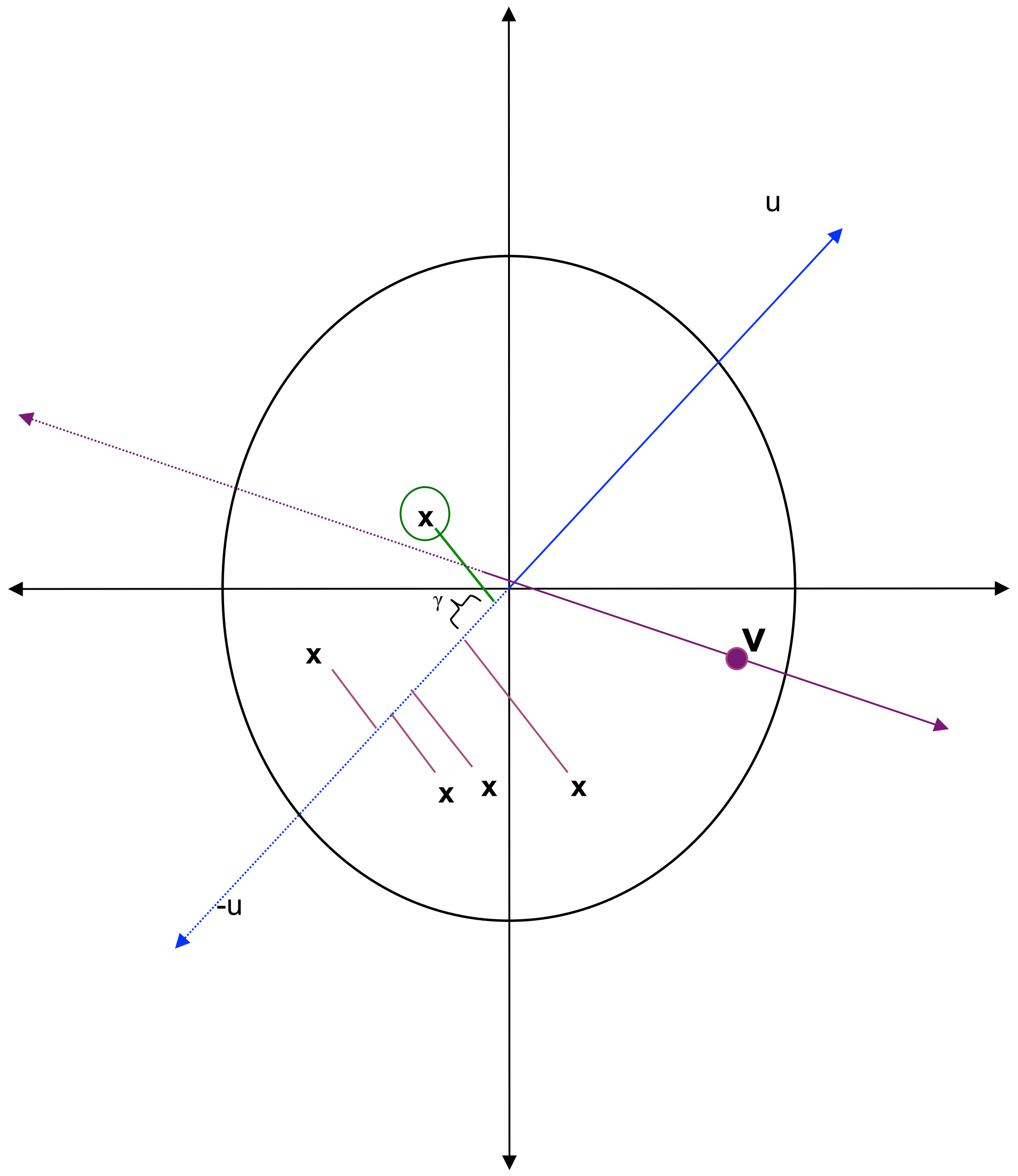
and furthermore, $\|\mathbf{u}\|^2 = 1$.

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

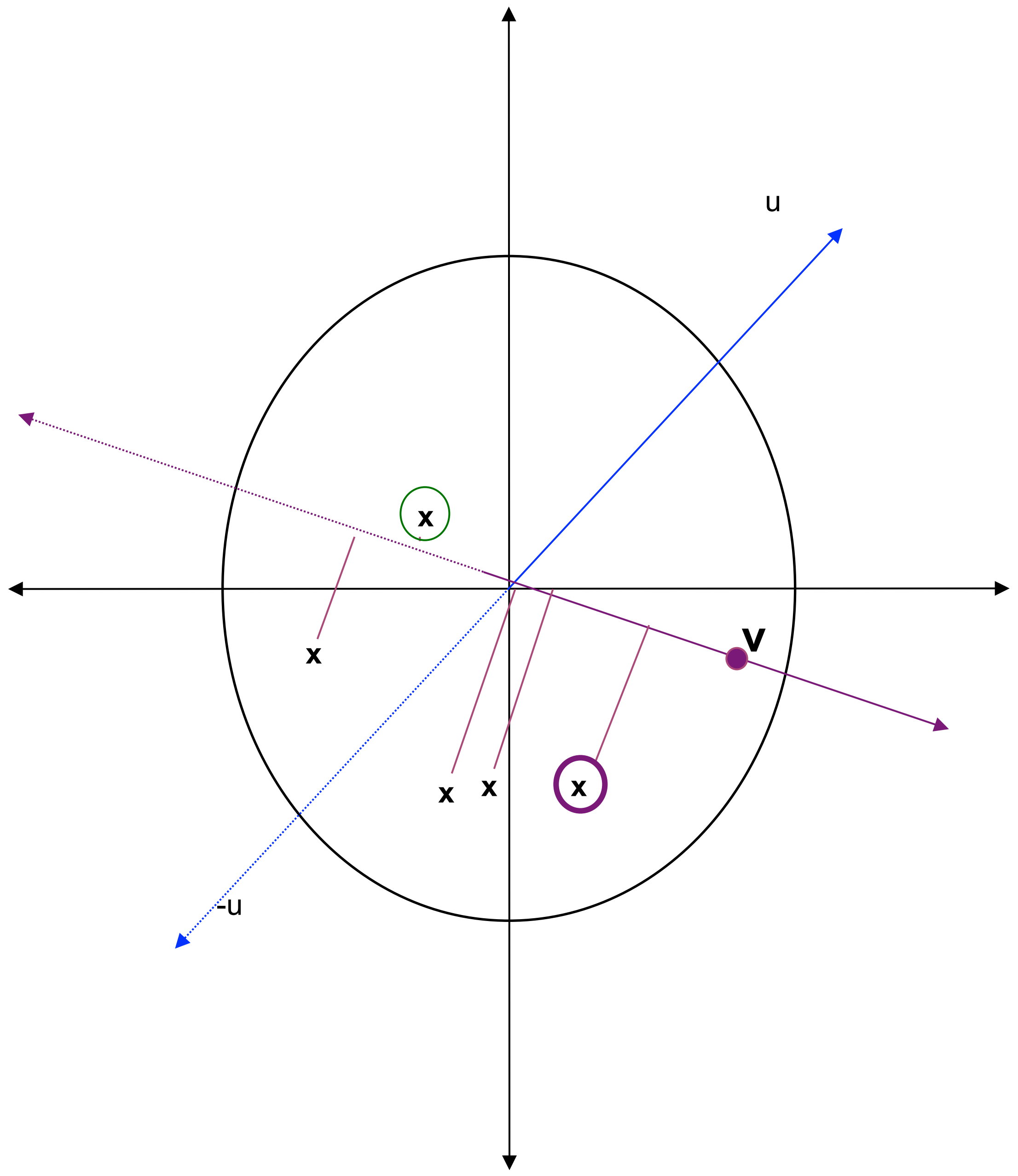
$$\forall \mathbf{x}_i \text{ given by } A, \|\mathbf{x}\|^2 < R^2$$



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

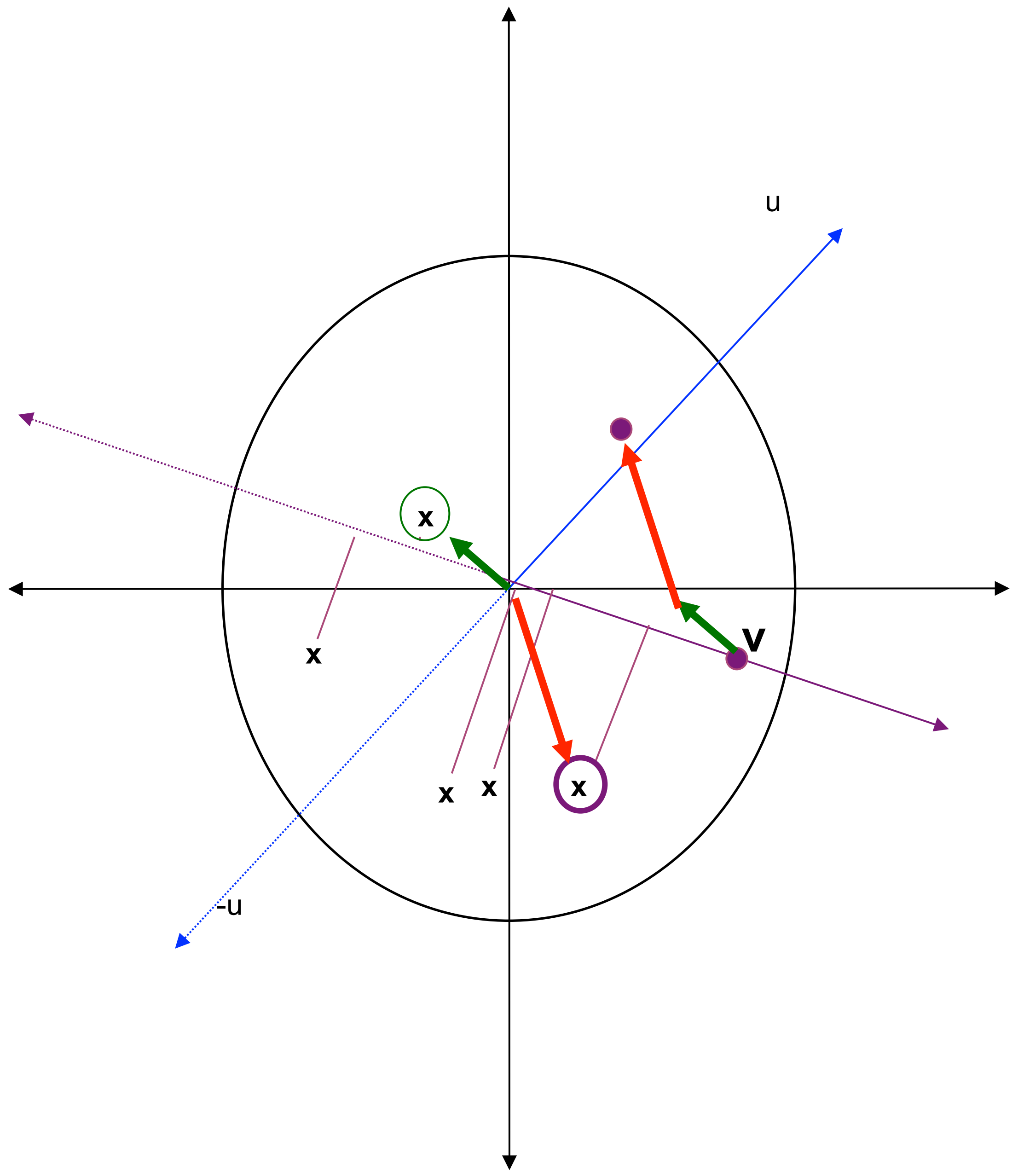


Ranking some x’ s with some guess vector **v** – part 1



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

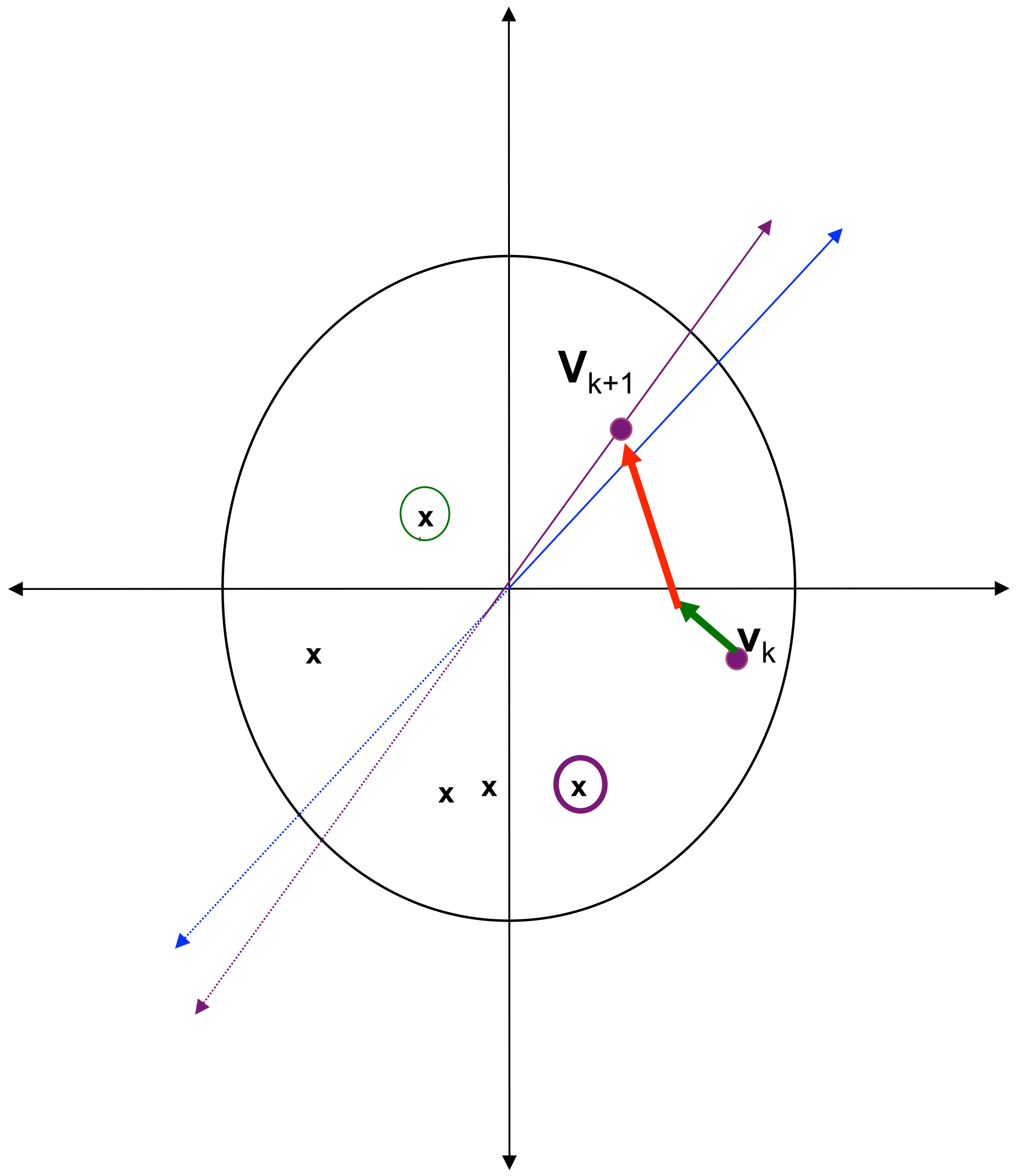
The purple-circled x is x_{b^*} - the one the learner has chosen to rank highest. The green circled x is x_b, the right answer.



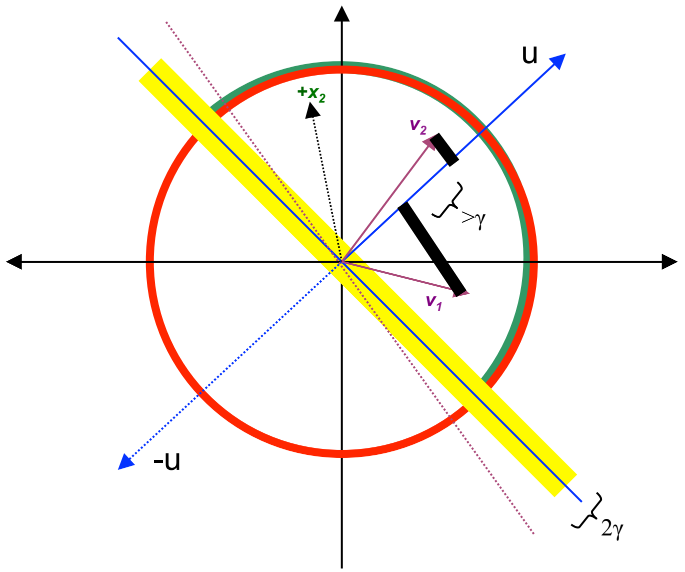
Correcting **v** by adding $x_b - x_{b^*}$

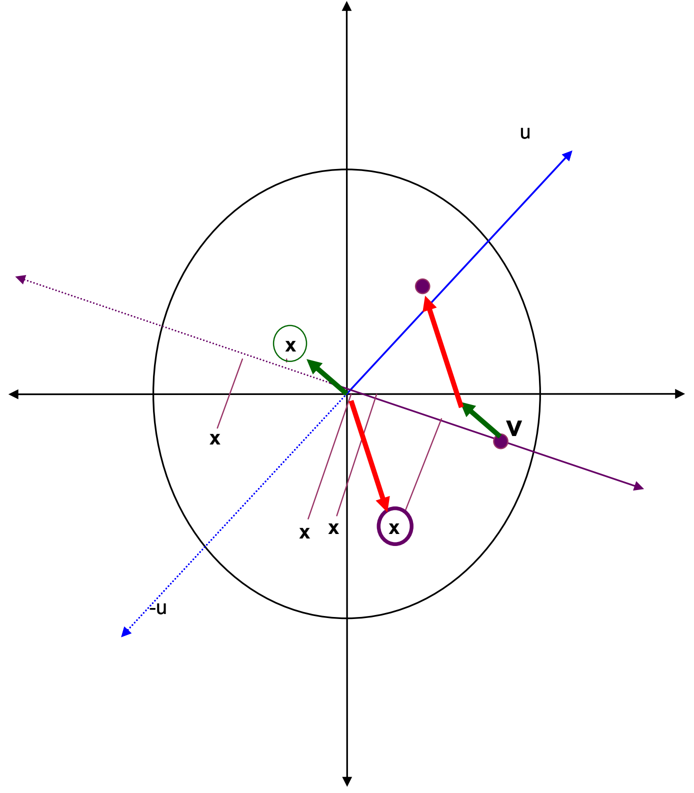
Correcting **v** by adding $x_b - x_{b^*}$

(part 2)



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



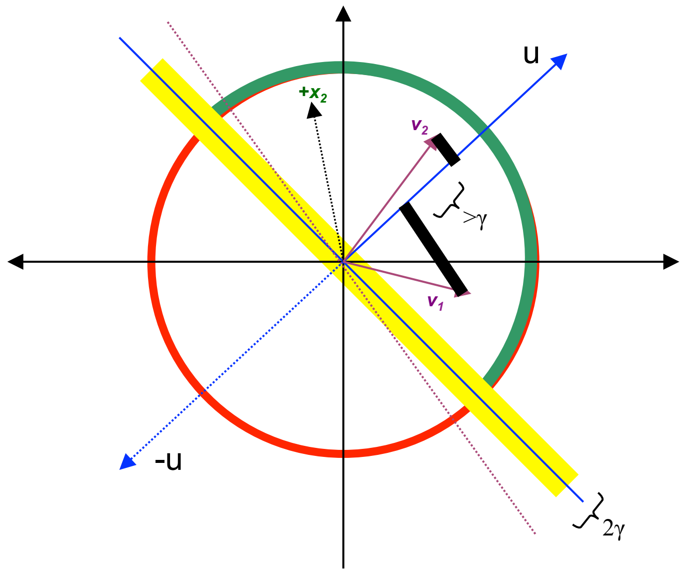


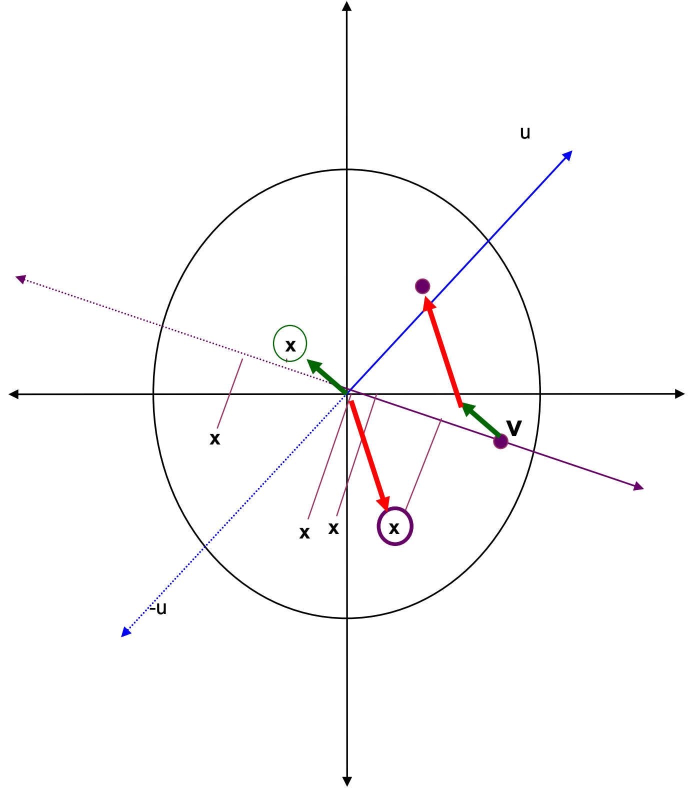
Lemma 1 *$\forall k$, $\mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$. In other words, the dot product between $\mathbf{v}_k$ and $\mathbf{u}$ increases with each mistake, at a rate depending on the margin γ.*

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: $\mathbf{v}_2$=$\mathbf{v}_1$+$\mathbf{x}_2$



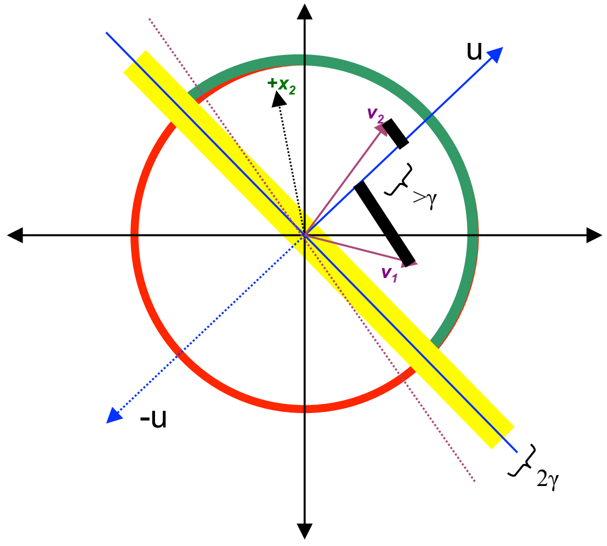


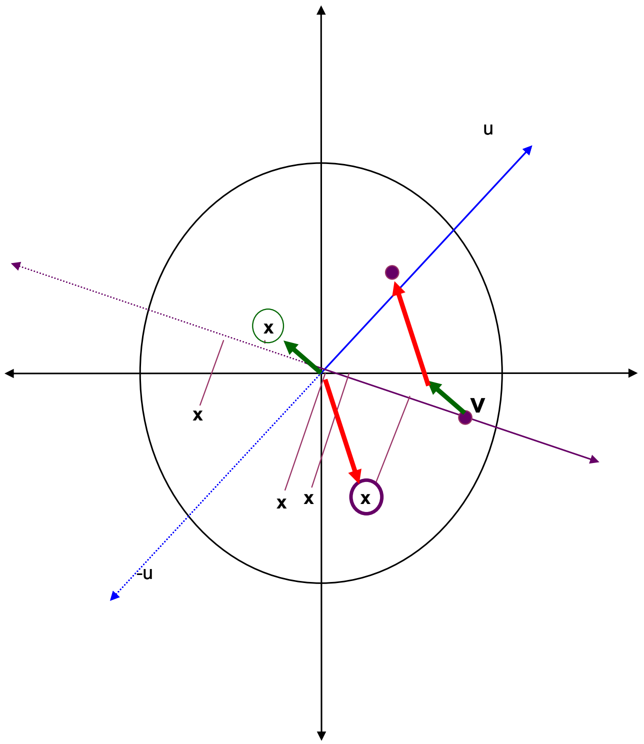
Lemma ~~1~~ 3 $\forall k$, $\mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$. *In other words, the dot product between* $\mathbf{v}_k$ *and* $\mathbf{u}$ *increases with each mistake, at a rate depending on the margin* γ.

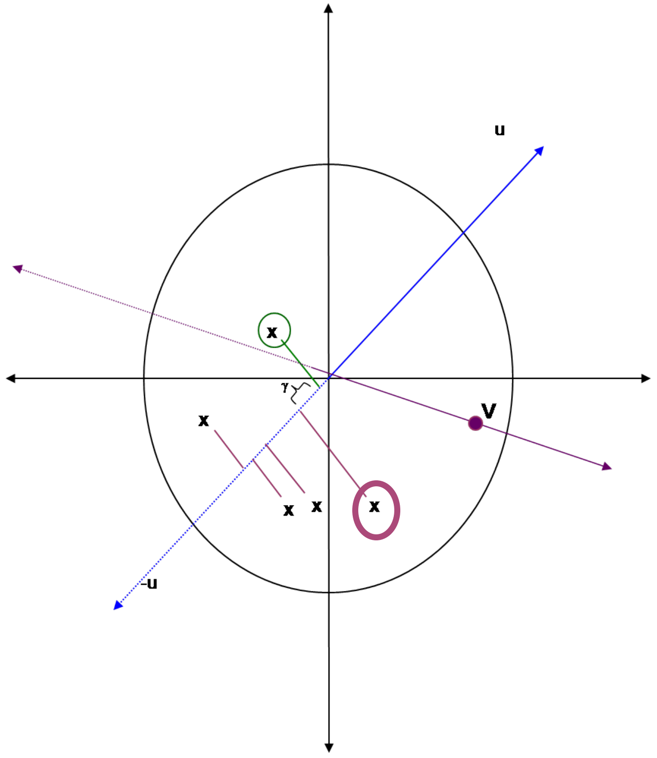
$$\begin{aligned}
& \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k + y_i \mathbf{x}_i) \cdot \mathbf{u} \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k \cdot \mathbf{u}) + y_i(\mathbf{x}_i \cdot \mathbf{u}) \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} \geq \mathbf{v}_k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

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

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







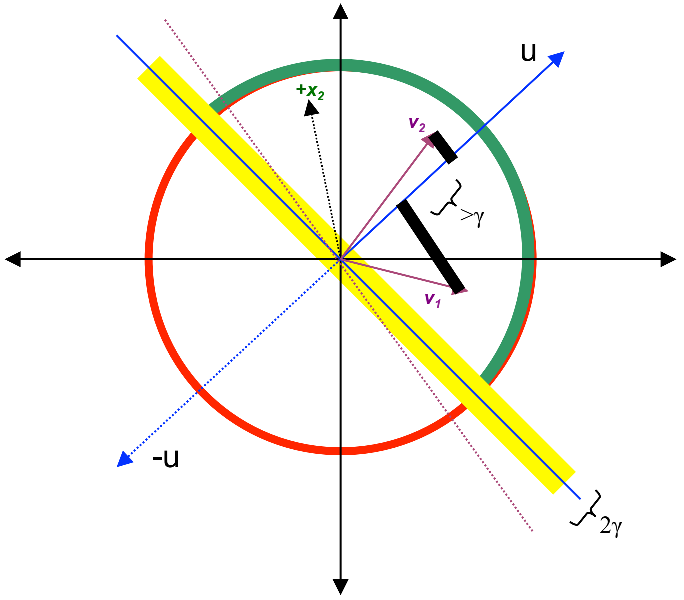
Lemma ~~1~~ 3 $\forall k$, $\mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$. *In other words, the dot product between* $\mathbf{v}_k$ *and* $\mathbf{u}$ *increases with each mistake, at a rate depending on the margin* γ.

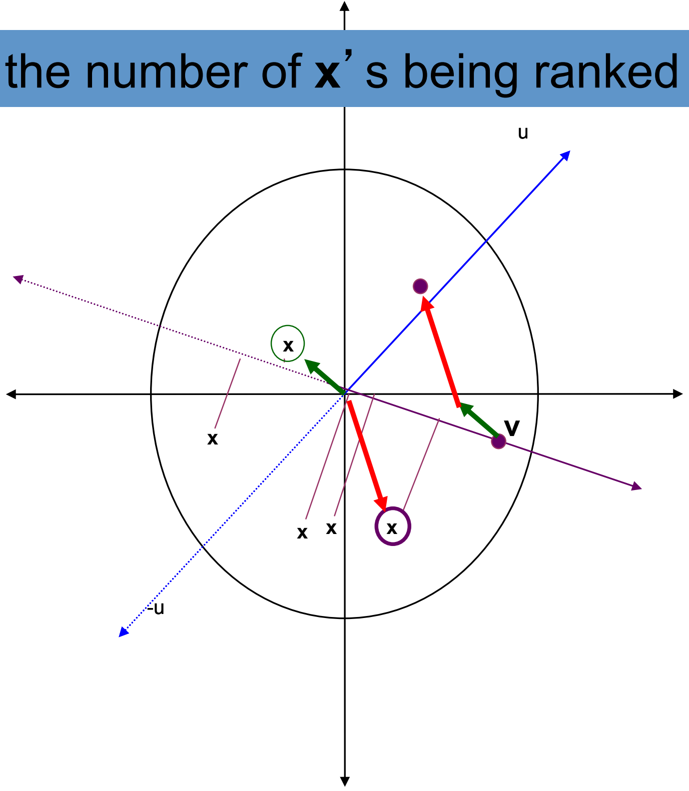
$$\begin{aligned}
& \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k + y_i \mathbf{x}_i) \cdot \mathbf{u} \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k \cdot \mathbf{u}) + y_i (\mathbf{x}_i \cdot \mathbf{u}) \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} \geq \mathbf{v}_k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

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

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

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

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 $\mathbf{x}$

Ranking perceptrons → structured perceptrons

- Suppose we can
  1. Given **x** and **y**, form a feature vector **F(x,y)**
     - Then we can score **x,y** using a weight vector: **w.F(x,y)**

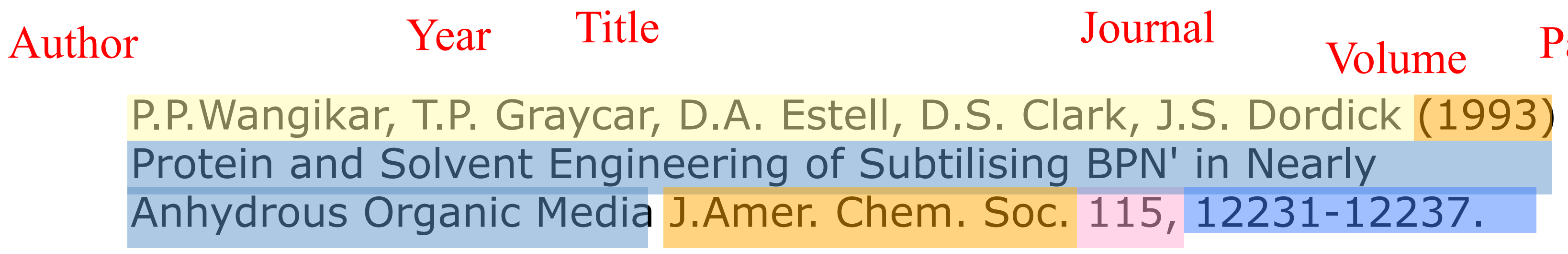
  2. Given **x,** find the *top-scoring* **y**, with respect to **w.F(x,y)**

  Then we can *learn*….

- Structured classification on a sequence
  - Input: list of words: **x**=$(w_1,\ldots,w_n)$
  - Output: list of labels: **y**=$(y_{1,}\ldots,y_n)$
  - If there are K classes, there are K^n labels possible for **x**

Example Structure classification problem: segmentation or NER

- 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



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.

When will prof Cohen post the notes …

Converting segmentation to a feature set (Begin,In,Out)

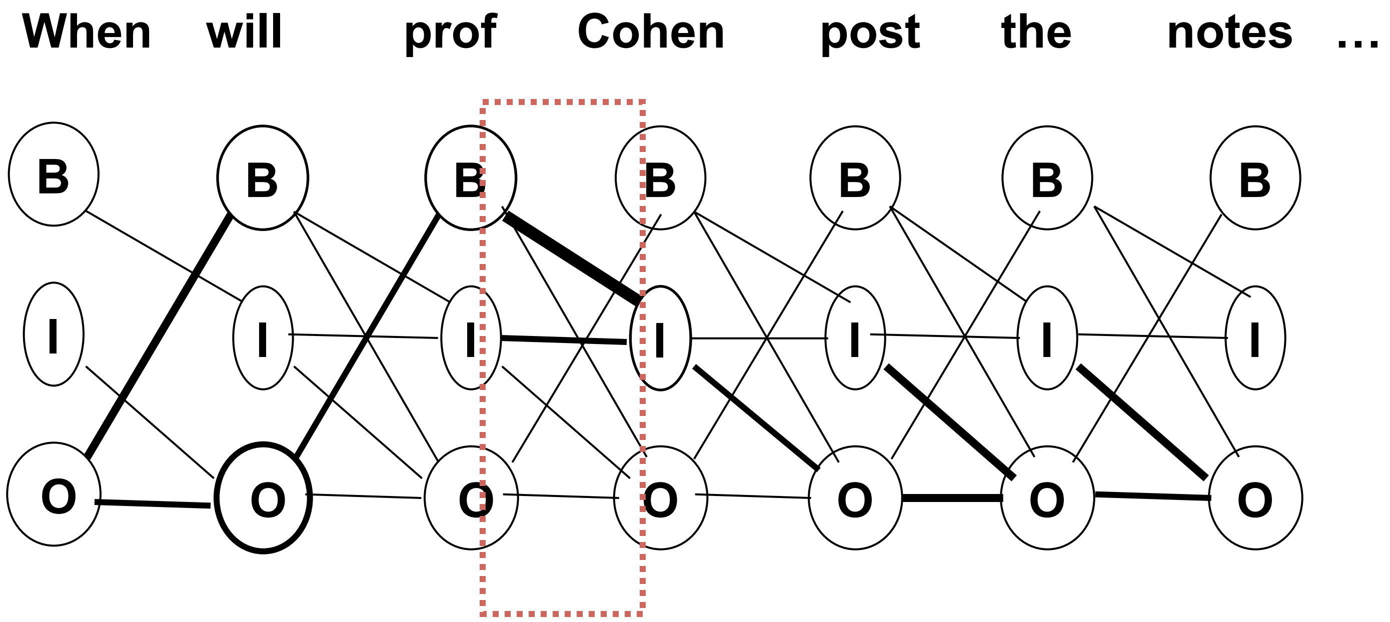
When will prof Cohen post the notes …

Idea 1: features are properties of *two adjacent tokens*, and the *pair* of labels assigned to them.

- (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 Rahul William is on the way"

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

Find the top-scoring y



•Inference: find the highest-weight path
•This can be done efficiently using dynamic programming (Viterbi)

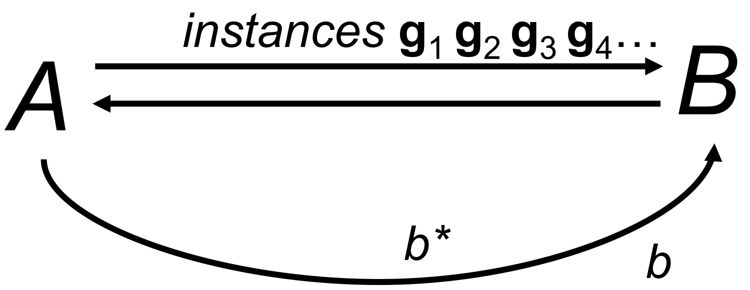
Ranking perceptrons → structured perceptrons

- 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'**) based on **w.F(x,y)**

- Structured classification on a sequence
  - Input: list of words: **x**=$(w_1,\ldots,w_n)$
  - Output: list of labels: **y**=$(y_1,\ldots,y_n)$
  - If there are K classes, there are K^n labels possible for **x**

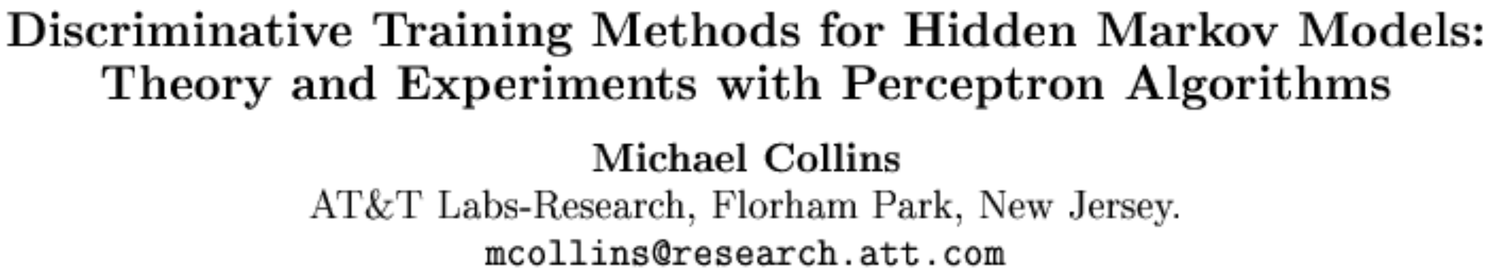
The voted perceptron *for NER*


instances $\mathbf{g}_1\ \mathbf{g}_2\ \mathbf{g}_3\ \mathbf{g}_4\ldots$
A
B
*b**
b

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 **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 $\mathbf{v}_{k+1} = \mathbf{v}_k + \mathbf{g}_b - \mathbf{g}_{b^*} = \mathbf{v}_k + \mathbf{F}(\mathbf{x}_i,\mathbf{y}) - \mathbf{F}(\mathbf{x}_i,\mathbf{y}^*)$



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

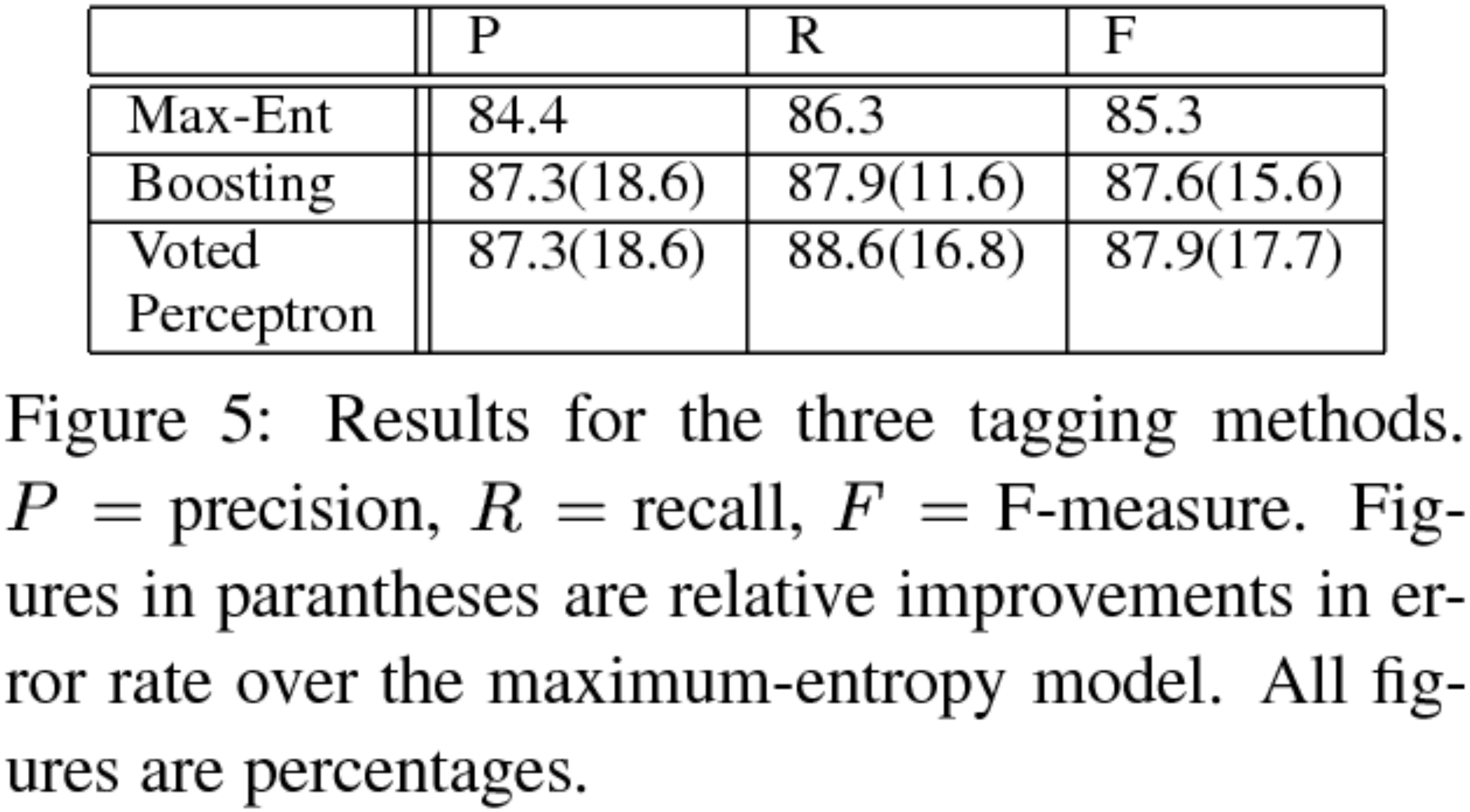
Michael Collins
AT&T Labs-Research, Florham Park, New Jersey.
mcollins@research.att.com

EMNLP 2002

Some background…

- Collins' parser: generative model…
- …New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron, Collins and Duffy, ACL 2002.
- …Ranking Algorithms for Named-Entity Extraction: Boosting and the Voted Perceptron, Collins, ACL 2002.
  - Propose entities using a MaxEnt tagger (as in MXPOST)
  - Use beam search to get *multiple* taggings for each document (20)
  - Learn to *rerank* the candidates to push correct ones to the top, using some new candidate-specific features:
    - Value of the "whole entity" (e.g., "Professor_Cohen")
    - Capitalization features for the whole entity (e.g., "Xx+_Xx+")
    - Last word in entity, and capitalization features of last word
    - Bigrams/Trigrams of words and capitalization features before and after the entity

Some background…



	P	R	F
Max-Ent	84.4	86.3	85.3
Boosting	87.3(18.6)	87.9(11.6)	87.6(15.6)
Voted Perceptron	87.3(18.6)	88.6(16.8)	87.9(17.7)

Figure 5: Results for the three tagging methods. P = precision, R = recall, F = F-measure. Figures in parantheses are relative improvements in error rate over the maximum-entropy model. All figures are percentages.

Collins' Experiments

- POS tagging
- NP Chunking (words and POS tags from Brill's tagger as features) and BIO output tags
- Compared Maxent Tagging/MEMM's (with iterative scaling) 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.