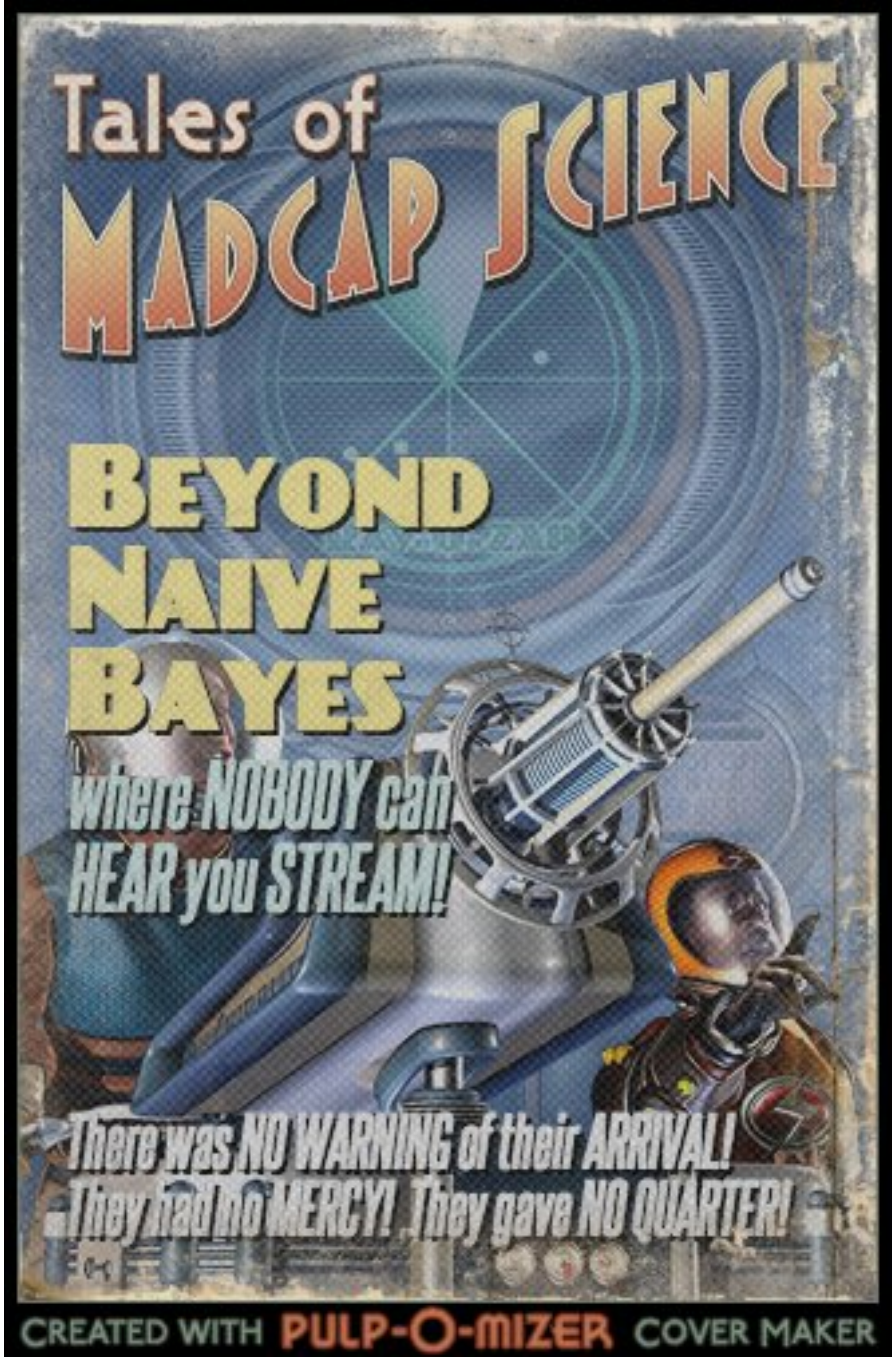

Tales of
MADCAP SCIENCE
BEYOND
NAIVE
BAYES
where NOBODY can
HEAR you STREAM!
There was NO WARNING of their ARRIVAL!
They had no MERCY! They gave NO QUARTER!
CREATED WITH PULP-O-MIZER COVER MAKER

Announcements

- Guest lectures schedule:
  - D. Sculley, Google Pgh, 3/26
  - Alex Beutel, SGD for tensors, 4/7
  - Alex Smola, something cool, 4/9

Projects

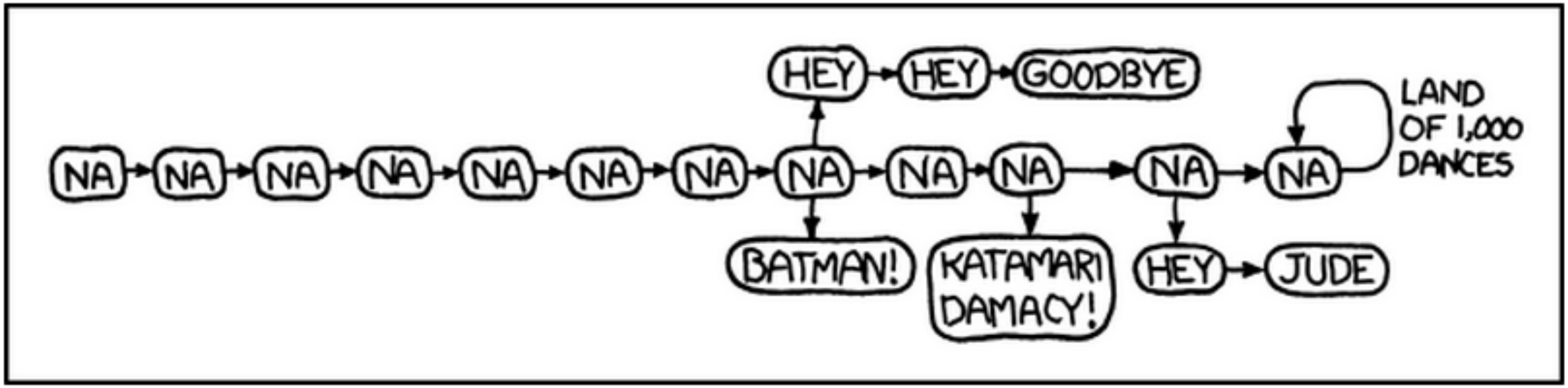
- Students in 805:
  - First draft of project proposal due 2/17.
  - Some more detail on projects is on the wiki.

Quiz

- https://qna-app.appspot.com/view.html?aglzfnFuYS1hcHByGQsSDFF1ZXN0aW9uTGlzdBiAgICAg-n-Cww

How do you debug a learning algorithm?

- Unit tests
- Simple artificial problems



How do you debug a learning algorithm?

- Unit tests
- Simple artificial problems

[rain | sleet | snow | showers |
[snow flurries | snow showers | light snow | …]]
[Monday | Tuesday | …]
and overcast

Beyond Naïve Bayes: Other Efficient Learning Methods

William W. Cohen

Two fast algorithms

- Naïve Bayes: one pass
- Rocchio: two passes
  - if vocabulary fits in memory
- Both method are algorithmically similar
  - count and combine
- Thought experiment: what if we duplicated some features in our dataset many times?
  - e.g., Repeat all words that start with "t" 10 times.

Limitations of Naïve Bayes/Rocchio

This isn't silly – often there are features that are "noisy" duplicates, or important phrases of different length

- Naïve Bayes: one pass
- Rocchio: two passes
  - if vocabulary fits in memory
- Both method are algorithmically similar
  - count and combine
- Thought thought thought thought thought thought thought thought thought thought experiment: what if we duplicated some features in our dataset many times times times times times times times times times times?
  - e.g., Repeat all words that start with "t" "t" "t" "t" "t" "t" "t" "t" "t" "t" ten ten ten ten ten ten ten ten ten ten times times times times times times times times times times.
  - Result: those features will be **over-weighted** in classifier by a factor of 10

Limitations of Naïve Bayes/Rocchio

This isn't silly – often there are features that are "noisy" duplicates, or important phrases of different length

- Naïve Bayes: one pass
- Rocchio: two passes
  - if vocabulary fits in memory
- Both method are algorithmically similar
  - count and combine
- Thought oughthay experiment experiment-day: what we add a Pig latin version of each word starting with "t"?
  - Result: those features will be **over-weighted**
  - You need to look at interactions between features somehow

Naïve Bayes is a linear algorithm

Naïve Bayes

$$\log P(y, x_1, \ldots, x_n) = \left(\sum_j \log \frac{C(X = x_j \wedge Y = y) + mq_x}{C(X = ANY \wedge Y = y') + m} \right) + \log \frac{C(Y = y) + mq_y}{C(Y = ANY) + m}$$

$$= \left(\sum_j g(x_j, y) \right) + f(y)$$

where $g(x_j, y) = \log \frac{C(X = x_j \wedge Y = y) + mq_x}{C(X = ANY \wedge Y = y') + m}$

$$= \left(\sum_{x \in V} f(x, d) g(x, y) \right) + f(y)$$

$$= \mathbf{v}(y, d) \cdot \mathbf{w}(y)$$

where $f(x, d) = TF(x, d)$

sparse vector of TF values for each word in the document… plus a "bias" term for *f(y)*

dense vector of g(x,y) scores for each word in the vocabulary .. plus *f(y)* to match bias term

One way to look for interactions: on-line, incremental learning

Naïve Bayes

Scan thu data:

- whenever we see *x* with *y* we increase *g(x,y)*
- whenever we see *x* with *~y* we increase *g(x,~y)*

$$\log P(y, x_1, \ldots, x_n) = \left(\sum_j \log \frac{C(X = x_j \wedge Y = y) + mq_x}{C(X = ANY \wedge Y = y') + m} \right) + \log \frac{C(Y = y) + mq_y}{C(Y = ANY) + m}$$

$$= \left(\sum_j g(x_j, y) \right) + f(y)$$

$$\text{where } g(x_j, y) = \log \frac{C(X = x_j \wedge Y = y) + mq_x}{C(X = ANY \wedge Y = y') + m}$$

$$= \left(\sum_{x \in V} f(x,d) g(x,y) \right) + f(y)$$

$$= \mathbf{v}(y,d) \cdot \mathbf{w}(y)$$

dense vector of g(x,y) scores for each word in the vocabulary

One simple way to look for interactions

Naïve Bayes – two class version

Scan thru data:

- whenever we see x with y we increase $g(x,y)-g(x,\sim y)$
- whenever we see x with $\sim y$ we decrease $g(x,y)-g(x,\sim y)$

We do this regardless of whether it seems to help or not on the data….if there are duplications, the weights will become arbitrarily large

$$prediction = \left(\sum \right.$$

$$= \left(\sum_j g(x_j, y) \right) + f(y)$$

where $g(x_j, y) = \log \frac{C(X = x_j \wedge Y = y) + mq_x}{C(X = ANY \wedge Y = y') + m}$

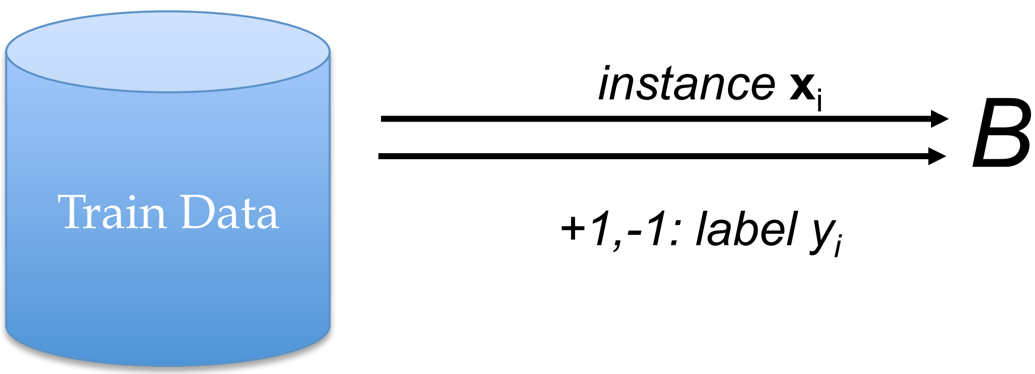
$$= \left(\sum f(x, d) g(x, y) \right) + f(y)$$

To detect interactions:

- increase/decrease g(x,y)-g(x,~y) only if we need to (for that example)
- otherwise, leave it unchanged

ch
word in the vocabulary

One simple way to look for interactions



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

If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + \textit{correction}$

To detect interactions:

- increase/decrease $\mathbf{v}_k$ only if we need to (for that example)
- otherwise, leave it unchanged

- We can be sensitive to duplication by stopping updates when we get better performance

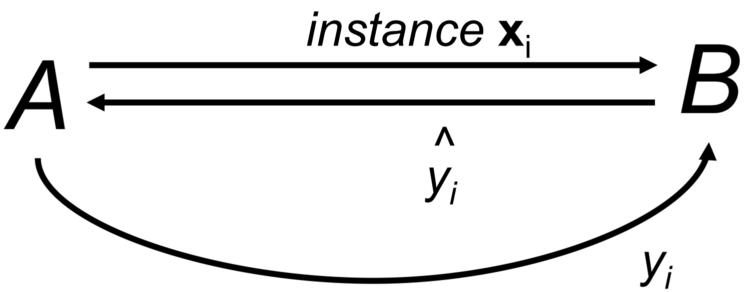
Theory: the prediction game

- 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 prediction game

- Are there practical algorithms where we can compute the mistake bound?

The voted perceptron



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

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

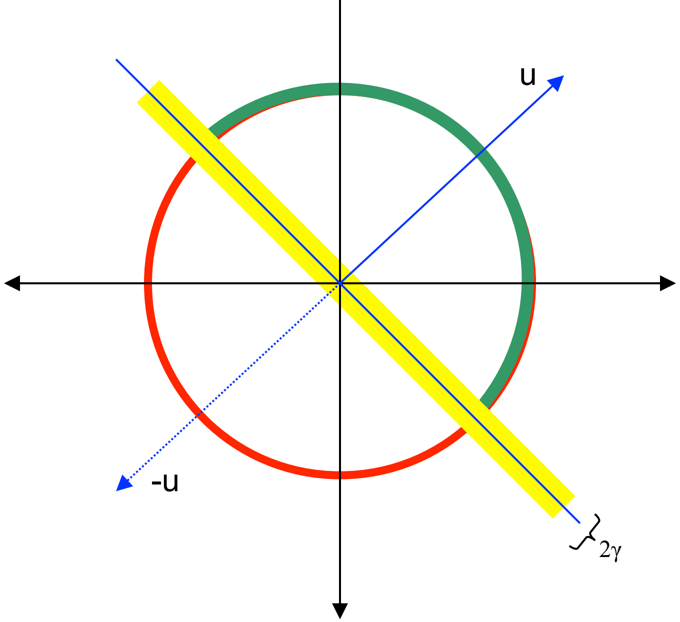
$$\exists \mathbf{u} : \forall (\mathbf{x}_i, y_i) \text{ given by } A, (\mathbf{u} \cdot \mathbf{x}) y_i > \gamma$$

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

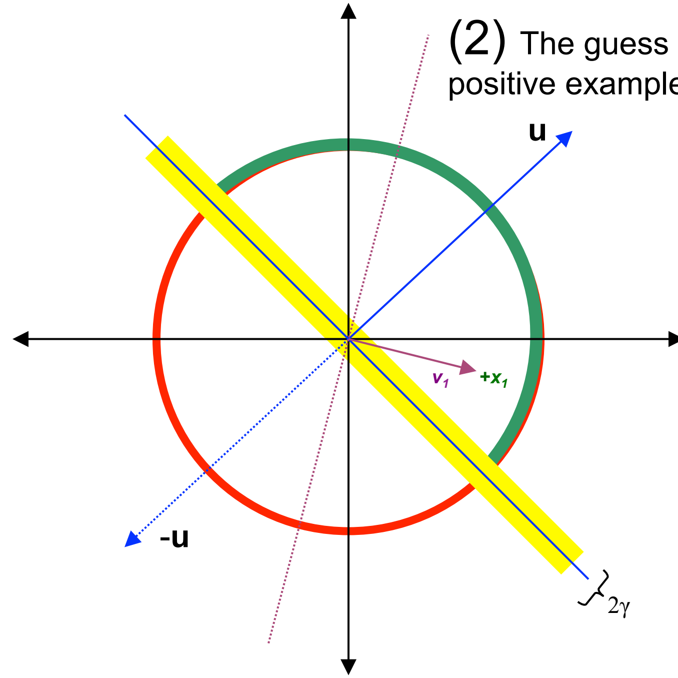
Radius R**.** A must provide examples "near the origin", ie

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

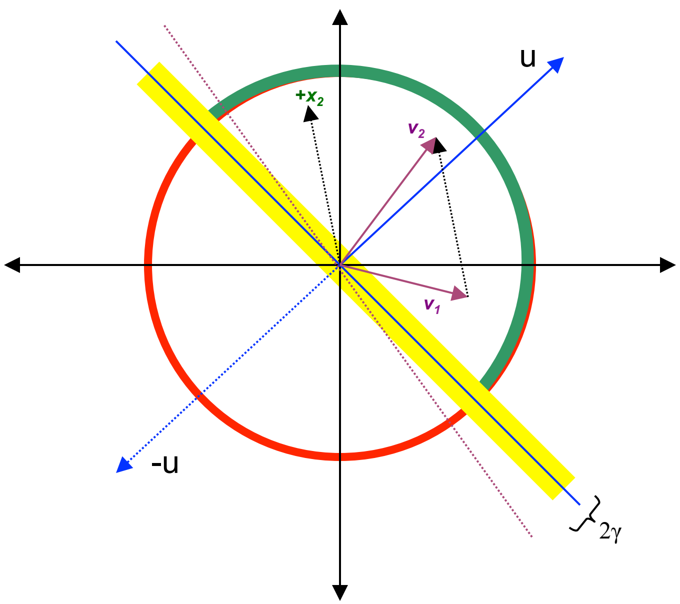
(1) A target **u**



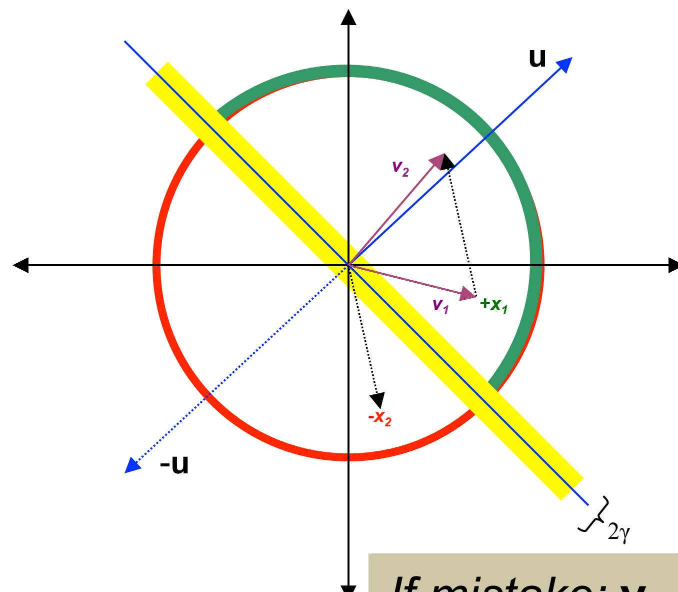
(2) The guess $\mathbf{v_1}$ after one positive example.



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

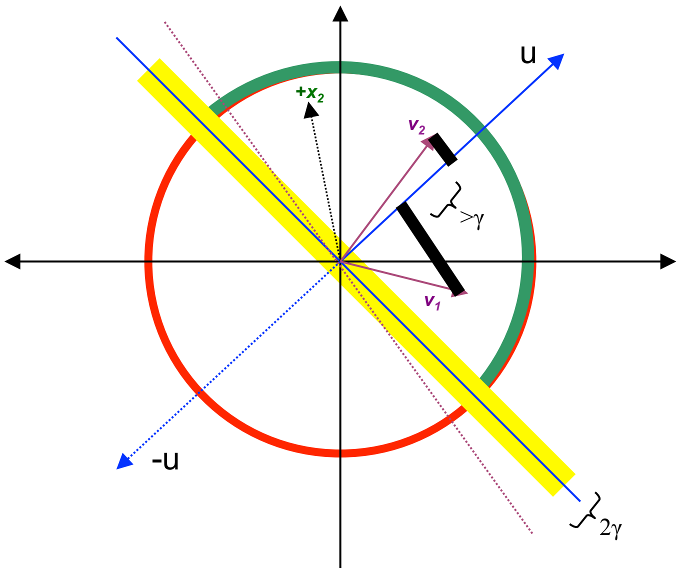


(3b) The guess $\mathbf{v_2}$ after the one positive and one negative example: $\mathbf{v}_2=\mathbf{v}_1-\mathbf{x}_2$

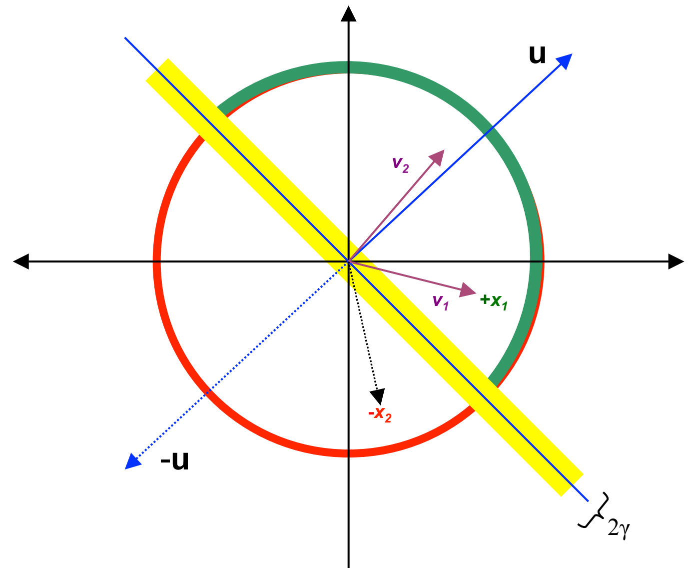


If mistake: $\mathbf{v}_{k+1} = \mathbf{v}_k + y_i\,\mathbf{x}_i$

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

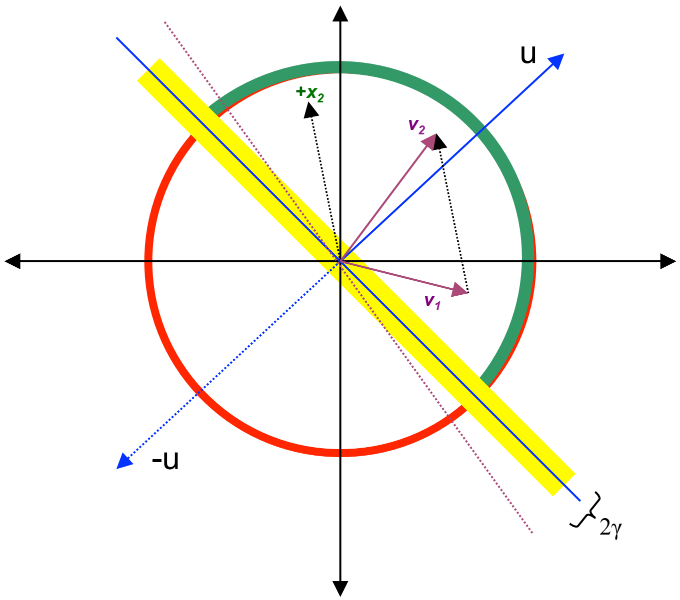


Lemma 1 $\forall k,\ \mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$. *In other words, the dot product between* $\mathbf{v}_k$ *and* $\mathbf{u}$ *increases with each mistake, at a rate depending on the margin* γ.

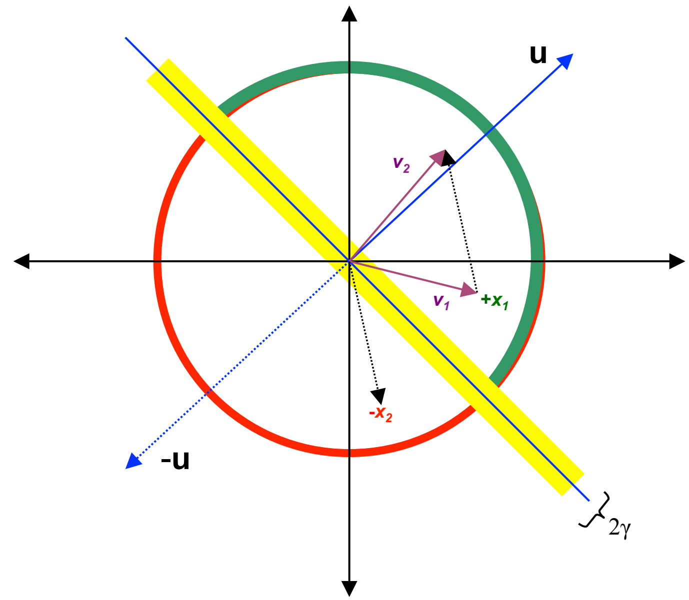
Proof:

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

(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:

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

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 $\forall k$, $\|\mathbf{v}_k\|^2 \leq kR$. *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 $(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),\ldots.$
  - *Then* : the perceptron algorithm makes $< R^2/\gamma^2$ mistakes on the sequence (where $R \geq ||\mathbf{x}_i||$)
  - *Independent* of dimension of the data or classifier (!)
  - This doesn't follow from M(C)<=VCDim(C)
- We *don't* know if this algorithm could be better
  - There are many variants that rely on similar analysis (ROMMA, Passive-Aggressive, MIRA, …)
- We *don't* know what happens if the data's not separable
  - Unless I explain the "Δ trick" to you
- We *don't* know what classifier to use "after" training

The Δ Trick

- The proof assumes the data is separable by a wide margin
- We can *make* that true by adding an “id” feature to each example
  - sort of like we added a constant feature

$$\mathbf{x}^1 = (x_1^1, x_2^1, \ldots, x_m^1) \rightarrow (x_1^1, x_2^1, \ldots, x_m^1, \overbrace{\Delta, 0, \ldots, 0}^{n \text{ new features}})$$

$$\mathbf{x}^2 = (x_1^2, x_2^2, \ldots, x_m^2) \rightarrow (x_1^2, x_2^2, \ldots, x_m^2, 0, \Delta, \ldots, 0)$$

$$\ldots$$

$$\mathbf{x}^n = (x_1^n, x_2^n, \ldots, x_m^n) \rightarrow (x_1^n, x_2^n, \ldots, x_m^n, 0, 0, \ldots, \Delta)$$

The Δ Trick

- Replace $\mathbf{x}_i$ with $\mathbf{x'}_i$ so $\mathbf{X}$ becomes $[\mathbf{X} \mid \mathbf{I}\,\Delta]$
- Replace R^2 in our bounds with $R^2 + \Delta^2$
- Let $d_i = \max(0, \gamma - y_i\, \mathbf{x}_i\, \mathbf{u})$
- Let $\mathbf{u'} = \mathbf{(}u_1,\ldots,u_n, y_1 d_1/\Delta, \ldots\, y_m d_m/\Delta) * 1/Z$
  - So $Z=\mathrm{sqrt}(1 + D^2/\Delta^2)$, for $D=\mathrm{sqrt}(d_1^2+\ldots+d_m^2)$
  - Now $[\mathbf{X}|\mathbf{I}\Delta]$ is separable by $\mathbf{u'}$ with margin γ
- Mistake bound is $(R^2 + \Delta^2)Z^2 / \gamma^2$
- Let $\Delta = \mathrm{sqrt}(RD)$ ➔ $k \leq ((R + D)/\gamma)^2$
- Conclusion: a little noise is ok

Summary

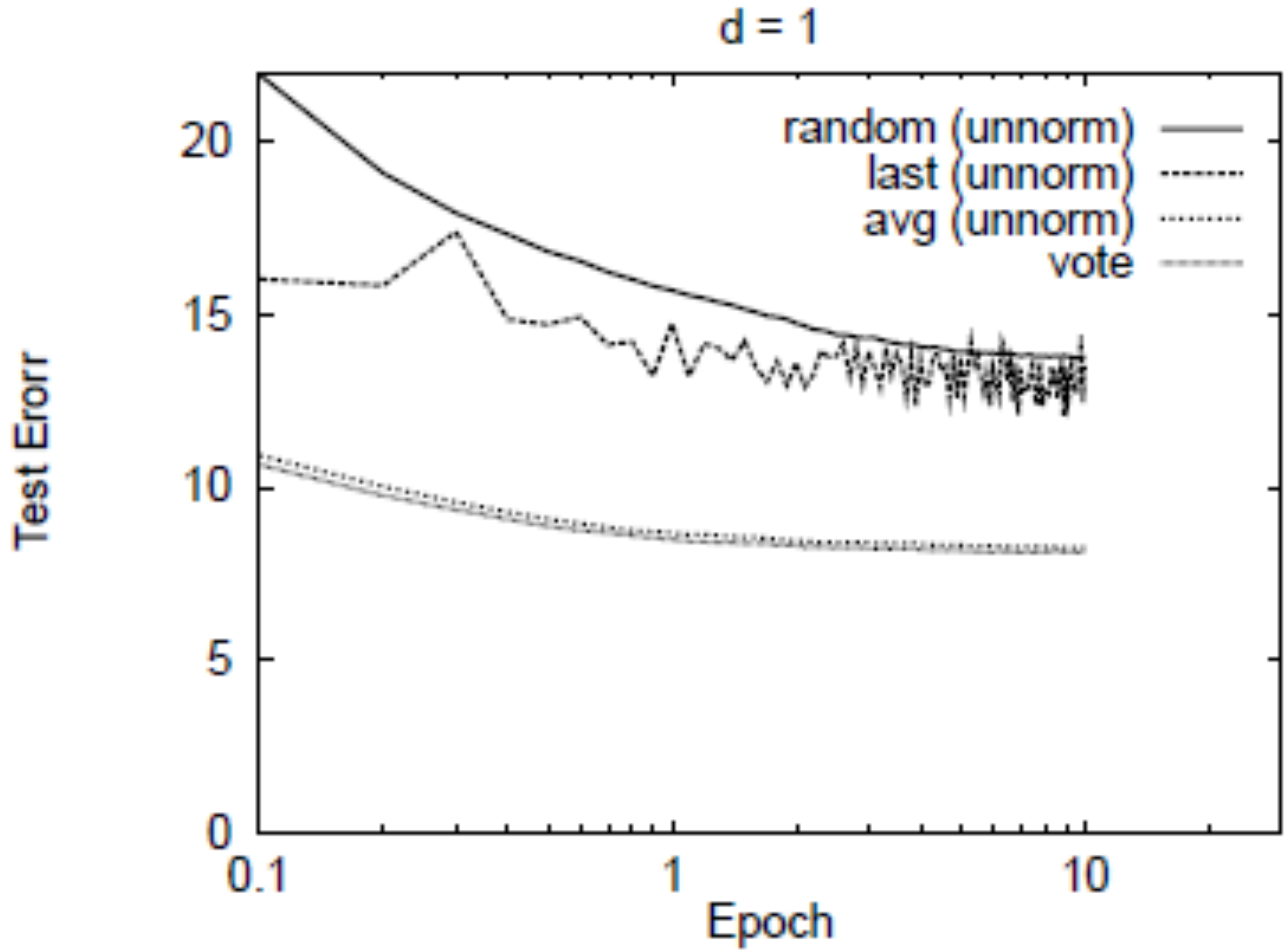
- We have shown that
  - *If* : exists a **u** with unit norm that has margin γ on examples in the seq $(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),\ldots.$
  - *Then* : the perceptron algorithm makes $< R^2/\gamma^2$ mistakes on the sequence (where $R \geq ||\mathbf{x}_i||$)
  - *Independent* of dimension of the data or classifier (!)
- We *don't* know what happens if the data's not separable
  - Unless I explain the "Δ trick" to you
- We *don't* know what classifier to use "after" training

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

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



d = 1
random (unnorm)
last (unnorm)
avg (unnorm)
vote
Test Erorr
Epoch
0
5
10
15
20
0.1
1
10

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 += $y x_i$

Complexity of *averaged* perceptron

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

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

The kernel trick

You can think of a perceptron as a weighted nearest-neighbor classifier….

Let $\mathcal{M}_k$ be the first k indices i where a mistake was made: then

$$\mathbf{v}_k = \sum_{i \in \mathcal{M}_k} y_i \mathbf{x}_i$$

so the prediction made on some test example $\mathbf{x}$ would be

$$\mathbf{v}_k \cdot \mathbf{x} = \Big(\sum_{i \in \mathcal{M}_k} y_i \mathbf{x}_i \Big) \cdot \mathbf{x} = \sum_{i \in \mathcal{M}_k} y_i (\mathbf{x}_i \cdot \mathbf{x}) = \sum_{i \in \mathcal{M}_k} y_i K(\mathbf{x}_i, \mathbf{x})$$

where K(***v***,***x***) = dot product of v and x (a similarity function)

The kernel trick

Here's another similarity function: K' (***v***,***x***)=dot product of *H'*(***v***),*H'*(***x***)) where

$$H'(\langle x_1, \ldots, x_n \rangle) = \langle x_1 x_1, x_1 x_2, \ldots, x_n x_n,\ x_1 \ldots x_n,\ 1 \rangle$$

Here's yet another similarity function: *K*(***v***,***x***) is

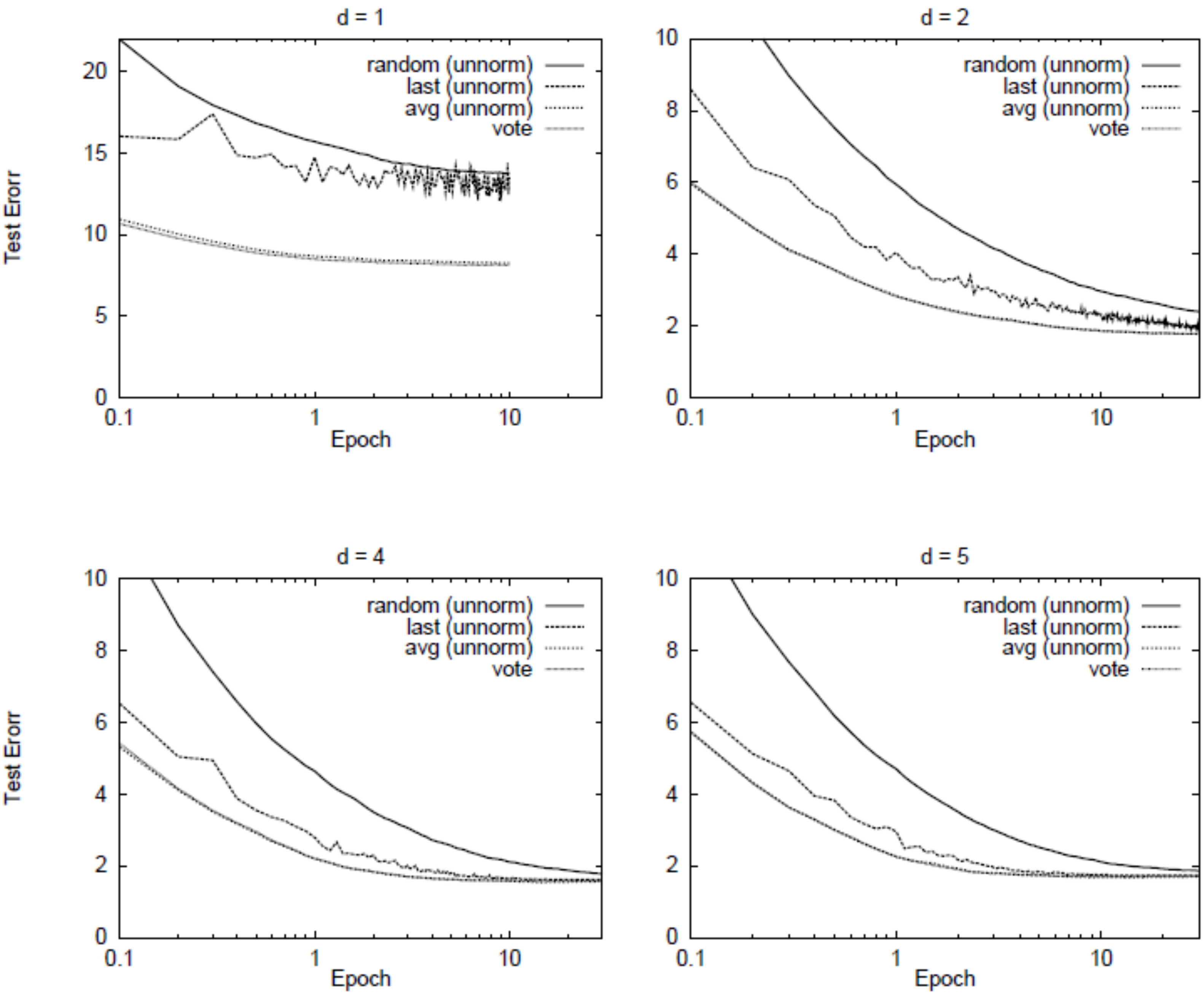
$$\begin{aligned}
K(\mathrm{v}, \mathrm{x}) &= (\mathrm{v} \cdot \mathrm{x} + 1)(\mathrm{v} \cdot \mathrm{x} + 1) \\
&= (\mathrm{vx})^2 + 2\mathrm{vx} + 1 \\
&= (v_1 x_1 + \ldots + v_n x_n)^2 + 2(v_1 x_1 + \ldots + v_n x_n) + 1 \\
&= \sum_{i,j} v_i x_i v_j x_j + 2 \sum_i v_i x_i + 1 \\
&= \sum_{i,j} v_i v_j x_i x_j + 2 \sum_i v_i x_i + 1
\end{aligned}$$

The kernel trick

$$\begin{aligned} K(\mathrm{v},\mathrm{x}) &= (\mathrm{v}\cdot\mathrm{x}+1)(\mathrm{v}\cdot\mathrm{x}+1) \\ &= (\mathrm{vx})^2 + 2\mathrm{vx} + 1 \\ &= (v_1x_1 + \ldots + v_nx_n)^2 + 2(v_1x_1 + \ldots + v_nx_n) + 1 \\ &= \sum_{i,j} v_ix_iv_jx_j + 2\sum_i v_ix_i + 1 \\ &= \sum_{i,j} v_iv_jx_ix_j + 2\sum_i v_ix_i + 1 \end{aligned}$$

Claim: K(**v,x**)=dot product of H**(x),**H**(v)** for this H:

$$H(\mathbf{x}) \equiv \left\langle x_1^2, x_1x_2, \ldots, x_{n-1}x_n, x_n^2, \sqrt{2}x_1, \ldots, \sqrt{2}x_n, 1 \right\rangle$$

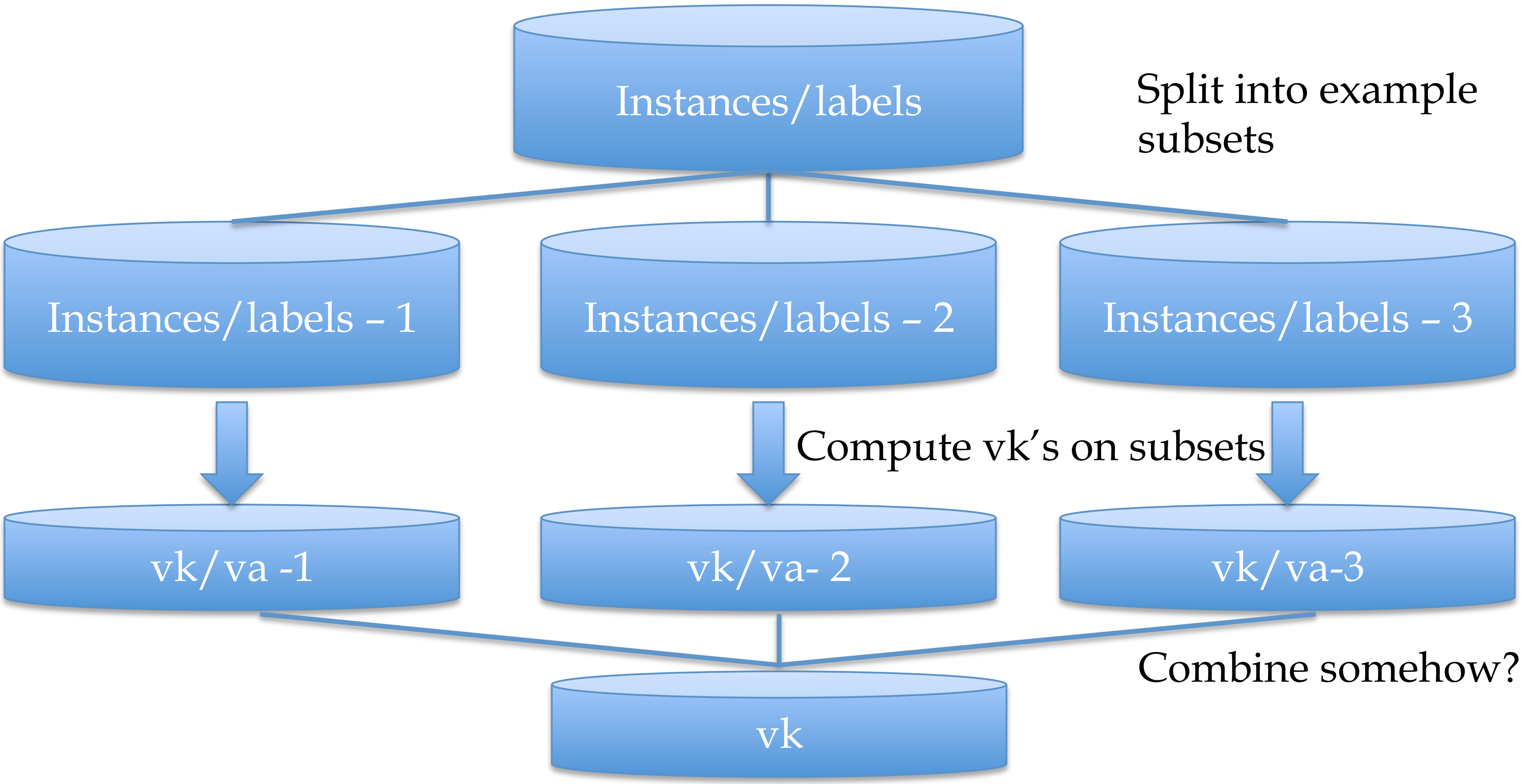


d = 1
d = 2
d = 4
d = 5
random (unnorm)
last (unnorm)
avg (unnorm)
vote
Test Erorr
Epoch
0.1
1
10
0
5
10
15
20
2
4
6
8

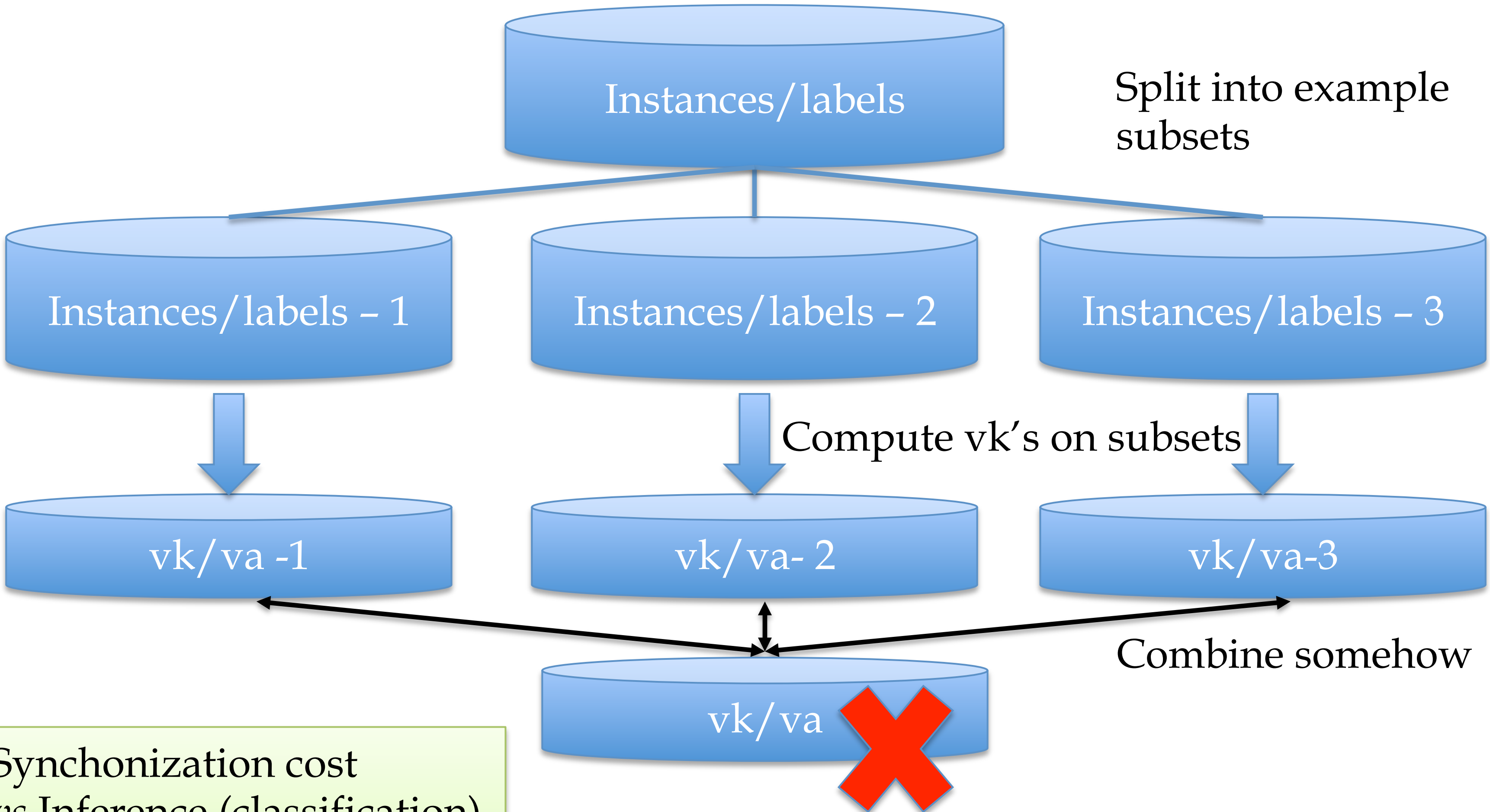
Table 1. Results of experiments on NIST 10-class OCR data with $d = 1, 2, 3$. The rows marked SupVec and Mistake give average number of support vectors and average number of mistakes. All other rows give test error rate in percent for the various methods.

		$T =$	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	
$d = 2$	Vote		6.0	2.8	2.4	2.2	2.1	1.8	1.8
	Avg.	(unnorm)	6.0	2.8	2.4	2.2	2.1	1.9	1.8
		(norm)	6.2	3.0	2.5	2.3	2.2	1.9	1.8
	Last	(unnorm)	8.6	4.0	3.4	3.0	2.7	2.3	2.0
		(norm)	8.4	3.9	3.3	3.0	2.7	2.3	1.9
	Rand.	(unnorm)	13.4	5.9	4.7	4.1	3.8	2.9	2.4
		(norm)	13.2	5.9	4.7	4.1	3.8	2.9	2.3
	SupVec		1,639	8,190	9,888	10,818	11,424	12,963	13,861
	Mistake		2,150	10,201	15,290	19,093	22,100	32,451	41,614
$d = 3$	Vote		5.4	2.3	1.9	1.8	1.7	1.6	1.6
	Avg.	(unnorm)	5.3	2.3	1.9	1.8	1.7	1.6	1.5
		(norm)	5.5	2.5	2.0	1.8	1.8	1.6	1.5
	Last	(unnorm)	6.9	3.1	2.5	2.2	2.0	1.7	1.6
		(norm)	6.8	3.1	2.5	2.2	2.0	1.7	1.6
	Rand.	(unnorm)	11.6	4.9	3.7	3.2	2.9	2.2	1.8
		(norm)	11.5	4.8	3.7	3.2	2.9	2.2	1.8
	SupVec		1,460	6,774	8,073	8,715	9,102	9,883	10,094
	Mistake		1,937	8,475	11,739	13,757	15,129	18,422	19,473

Parallelizing perceptrons



Parallelizing perceptrons



Synchonization cost
vs Inference (classification) cost

Review/outline

- How to implement Naïve Bayes
  - Time is linear in size of data (one scan!)
  - We need to count C(*X=word ^ Y=label)*
- Can you parallelize Naïve Bayes?
  - Trivial solution 1
    1. Split the data up into multiple subsets
    2. Count and total each subset independently
    3. Add up the counts
  - Result should be the same
- This is unusual for streaming learning algorithms
  - Why? **no interaction between feature weight updates**
  - For perceptron that's not the case

A hidden agenda

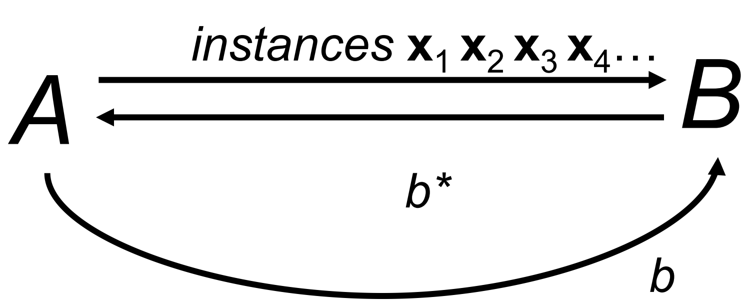
- Part of machine learning is good grasp of theory
- Part of ML is a good grasp of what hacks tend to work
- These are not always the same
  - Especially in big-data situations

- 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

William Cohen

The voted perceptron *for ranking*



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

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

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

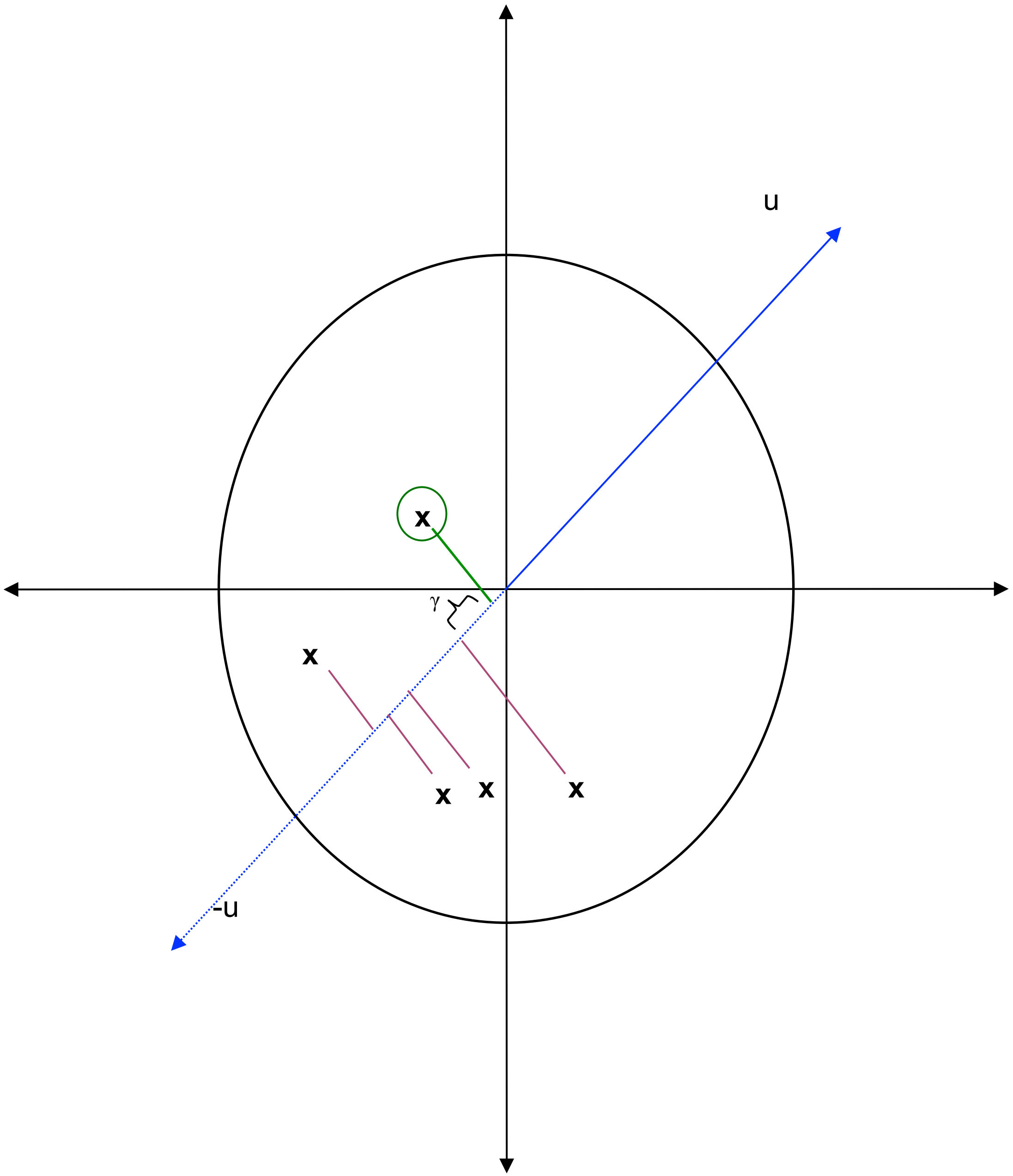
Margin γ**.** A must provide examples that can be correctly ranked with some vector $\mathbf{u}$ with margin $\gamma > 0$, ie

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

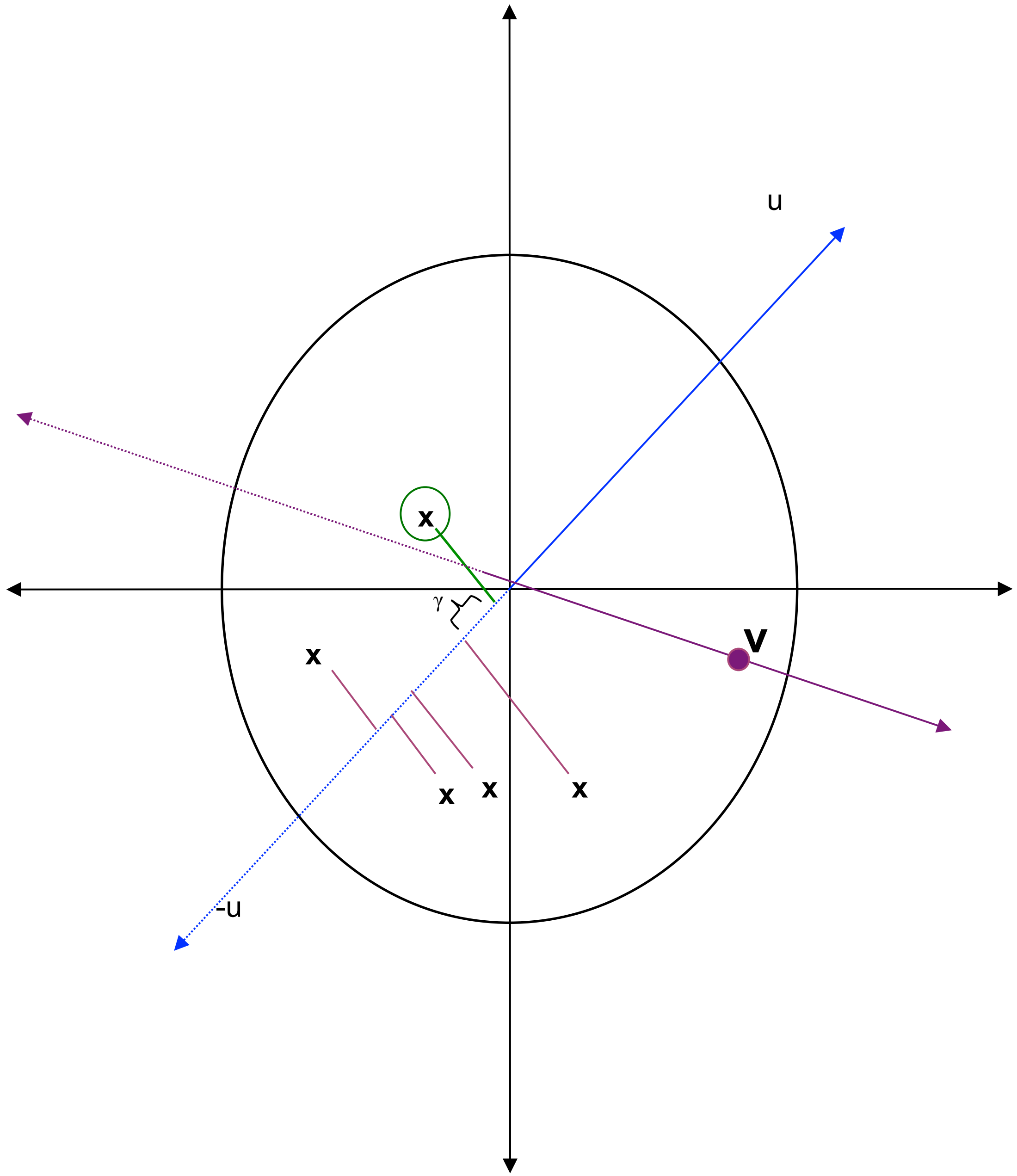
and furthermore, $\|\mathbf{u}\|^2 = 1$.

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

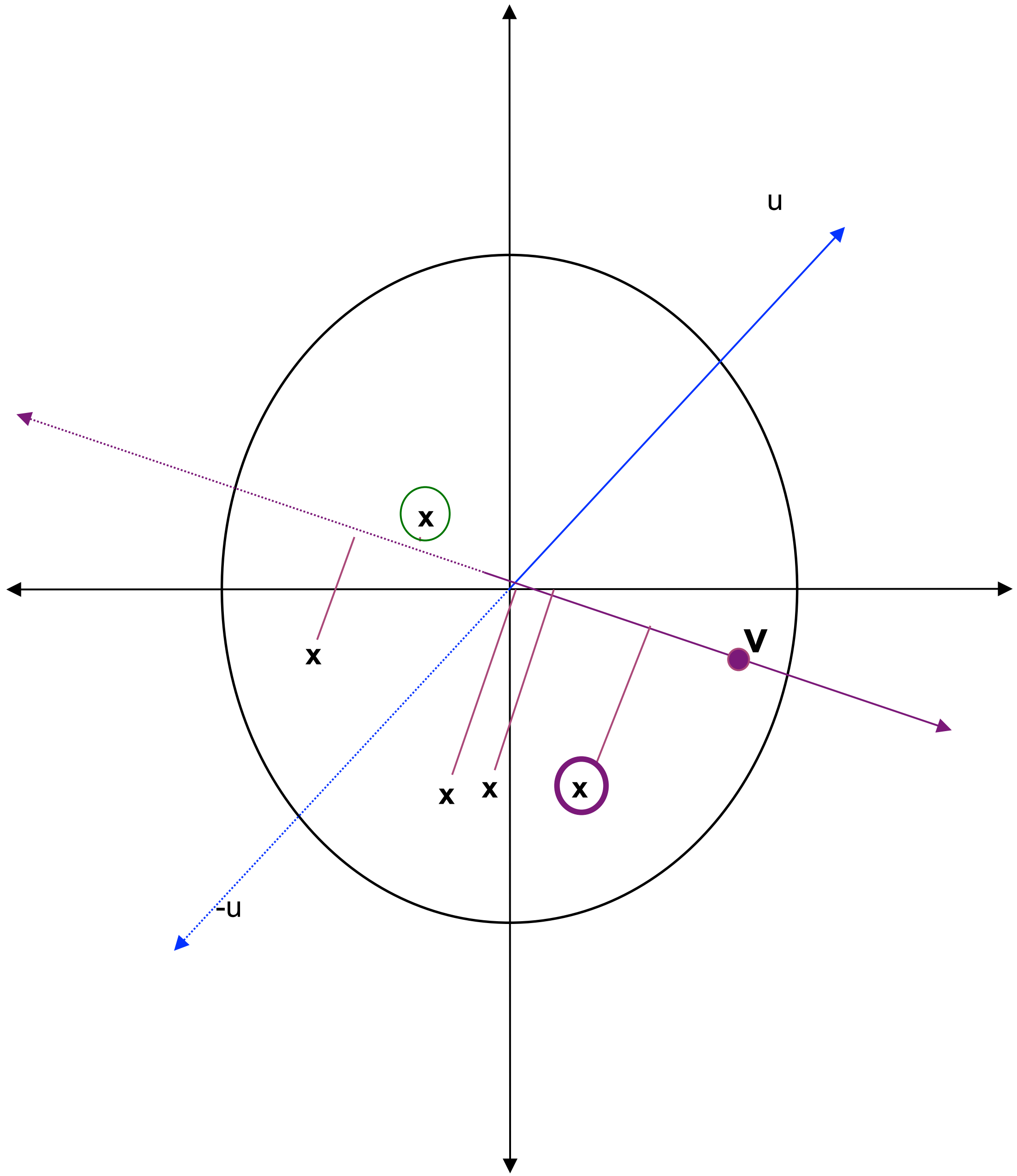
$$\forall \mathbf{x}_i \text{ given by } A, \|\mathbf{x}\|^2 < R^2$$



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

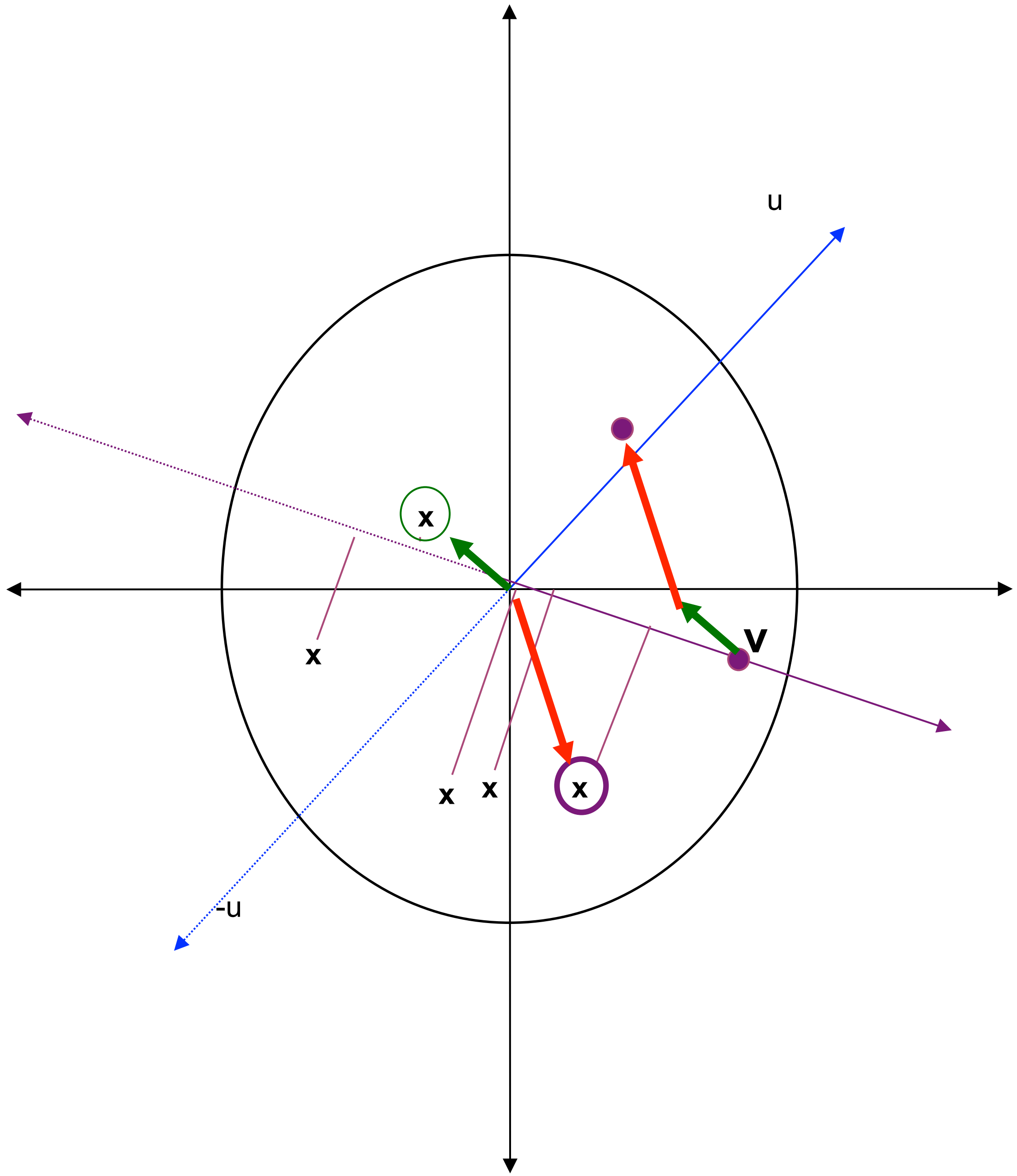


Ranking some x' s with some guess vector **v** – part 1

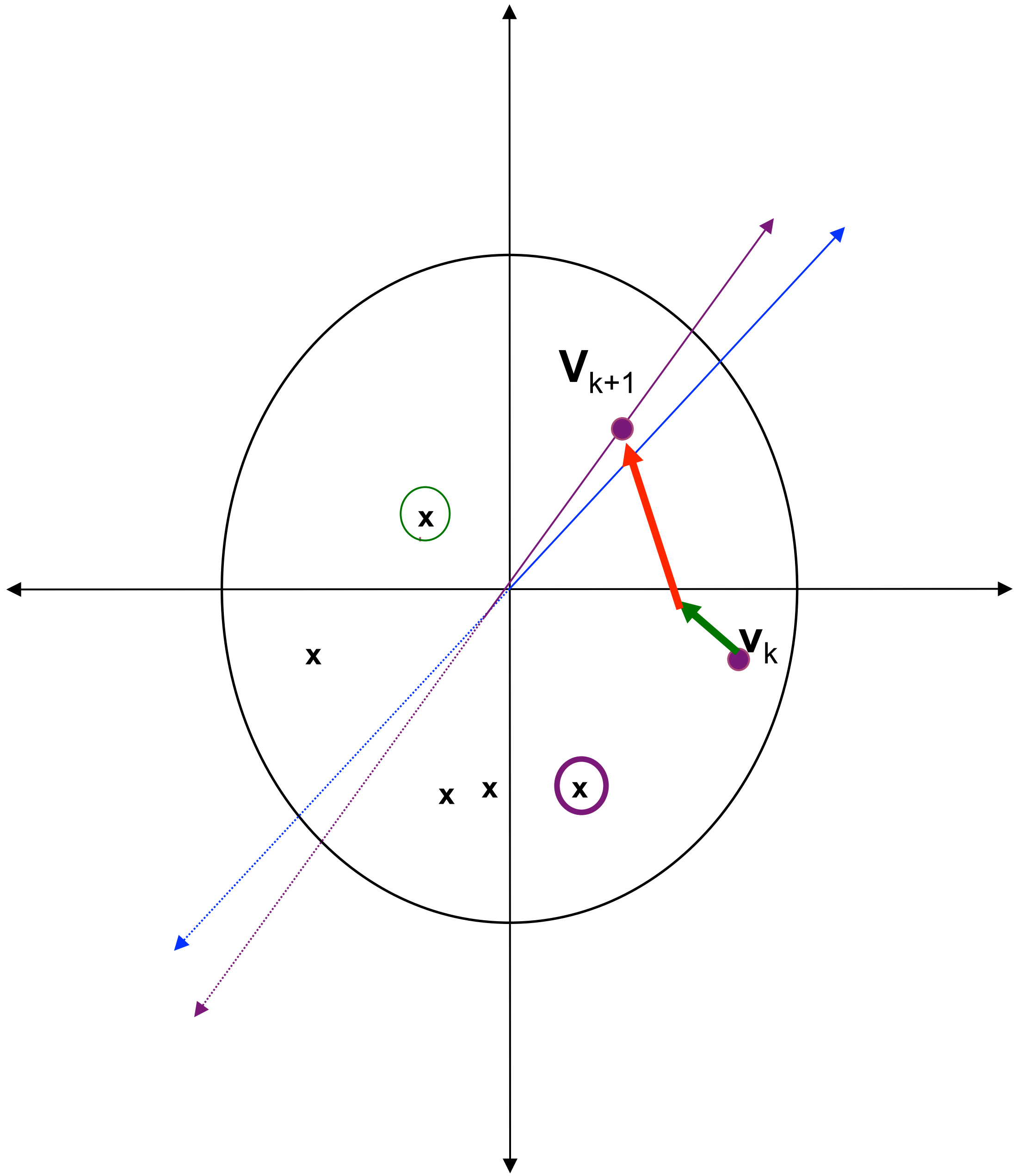


Ranking some x' s with some guess vector **v** – part 2.

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.



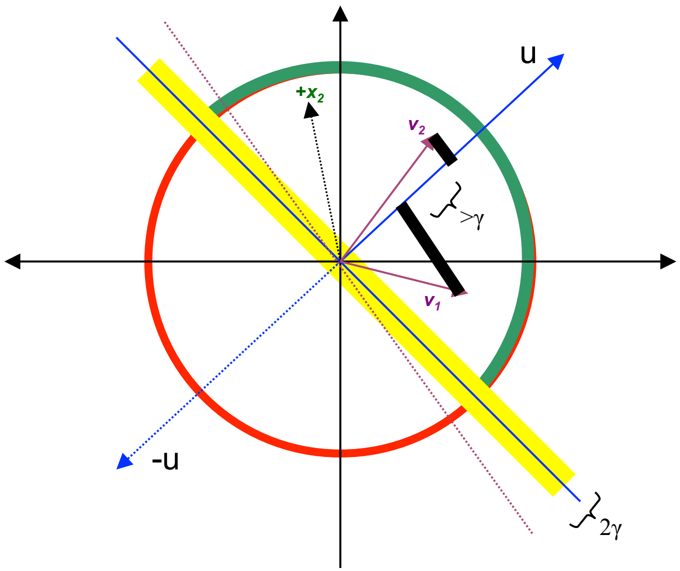
Correcting **v** by adding $x_b - x_{b^*}$

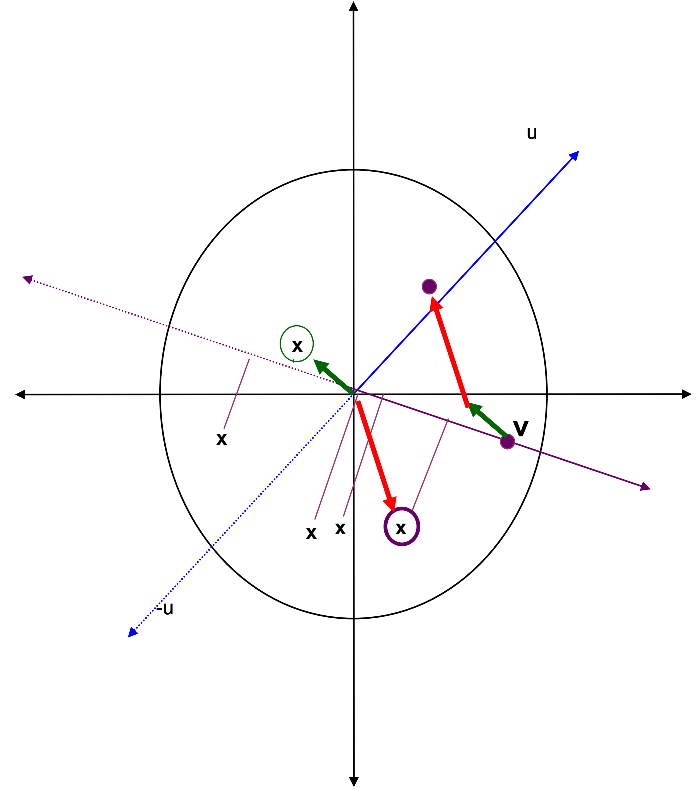


Correcting **v** by adding $x_b - x_{b^*}$

(part 2)

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



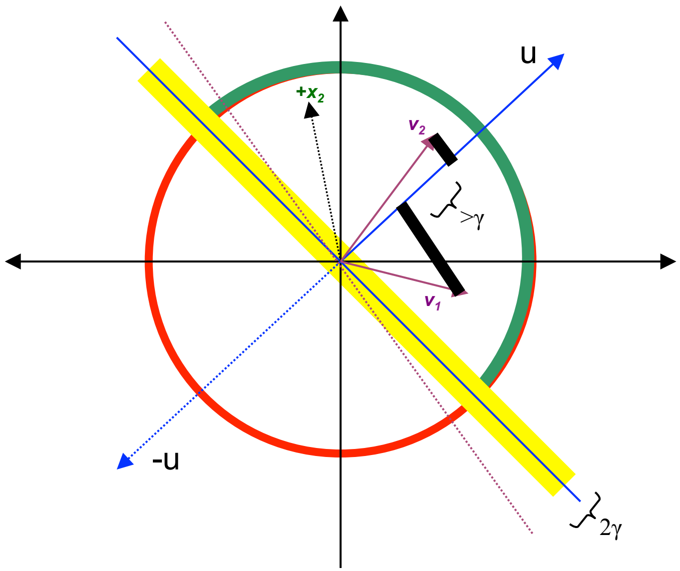


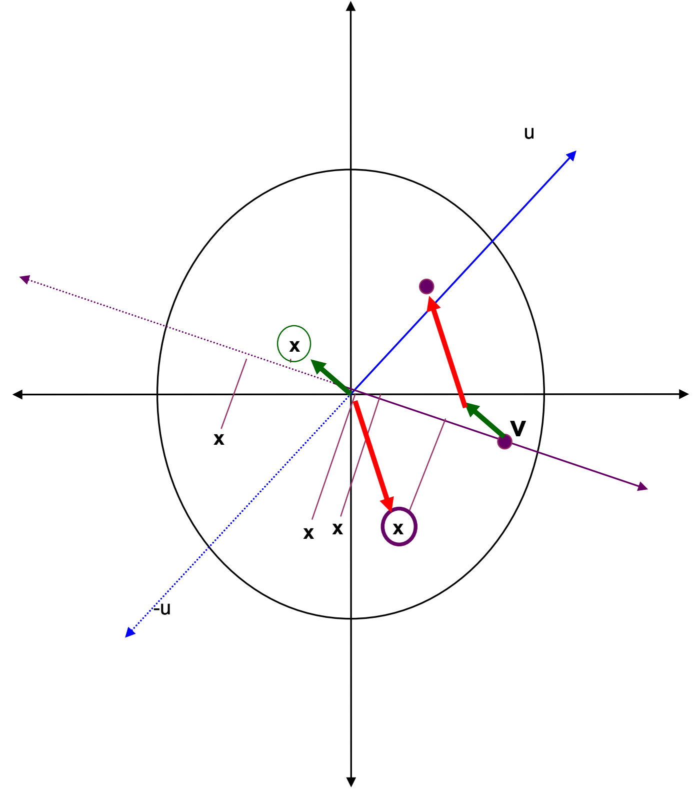
Lemma 1 $\forall k$, $\mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$. *In other words, the dot product between* $\mathbf{v}_k$ *and* $\mathbf{u}$ *increases with each mistake, at a rate depending on the margin* γ*.*

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



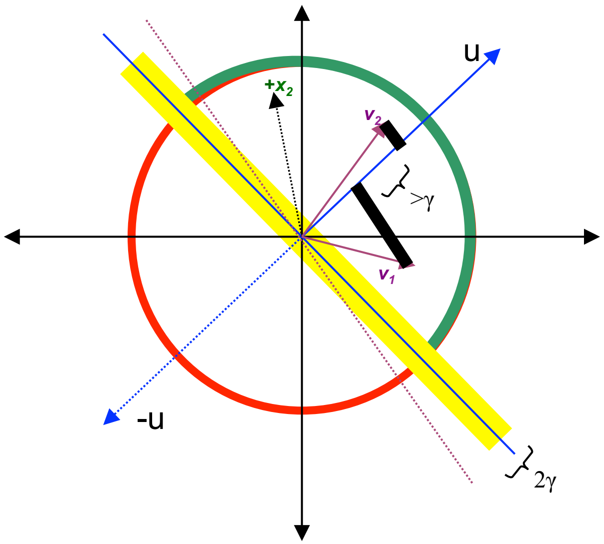


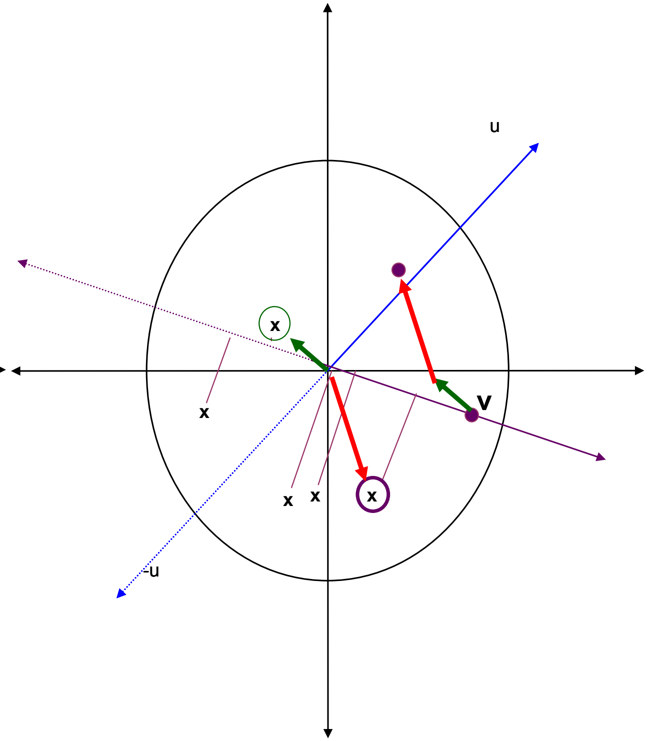
Lemma ~~1~~ 3 $\forall k$, $\mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$. *In other words, the dot product between* $\mathbf{v}_k$ *and* $\mathbf{u}$ *increases with each mistake, at a rate depending on the margin* γ.

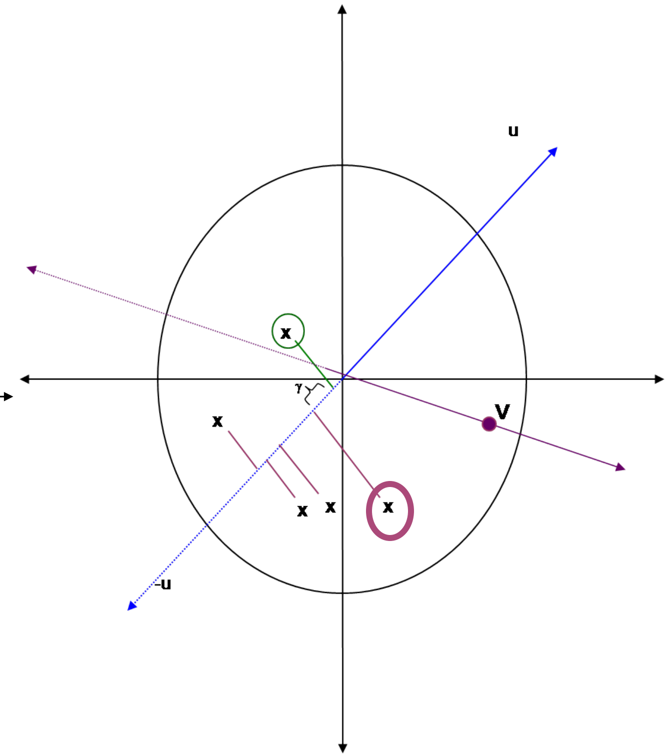
$$
\begin{aligned}
& \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k + y_i \mathbf{x}_i) \cdot \mathbf{u} \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k \cdot \mathbf{u}) + y_i(\mathbf{x}_i \cdot \mathbf{u}) \\
\Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} \geq \mathbf{v}_k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

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

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







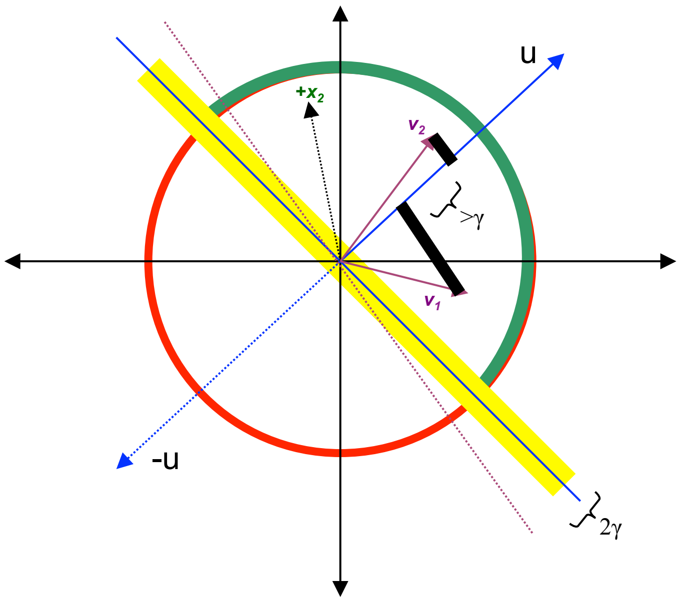
Lemma ~~1~~ 3 *$\forall k$, $\mathbf{v}_k \cdot \mathbf{u} \geq k\gamma$. In other words, the dot product between $\mathbf{v}_k$ and $\mathbf{u}$ increases with each mistake, at a rate depending on the margin γ.*

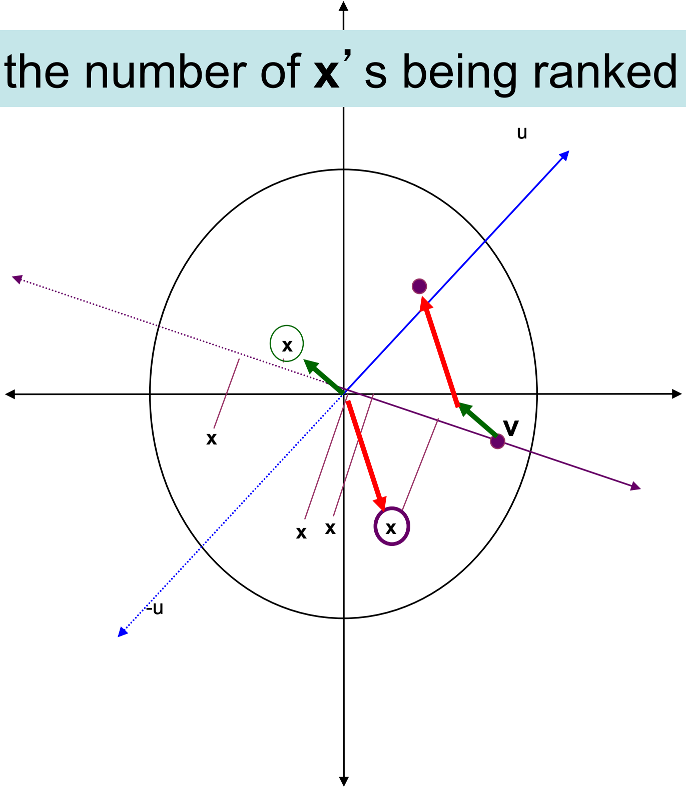
$$\boxed{\begin{aligned} & \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k + y_i\mathbf{x}_i) \cdot \mathbf{u} \\ \Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} = (\mathbf{v}_k \cdot \mathbf{u}) + y_i(\mathbf{x}_i \cdot \mathbf{u}) \\ \Rightarrow \quad & \mathbf{v}_{k+1} \cdot \mathbf{u} \geq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

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

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 $\mathbf{x}$

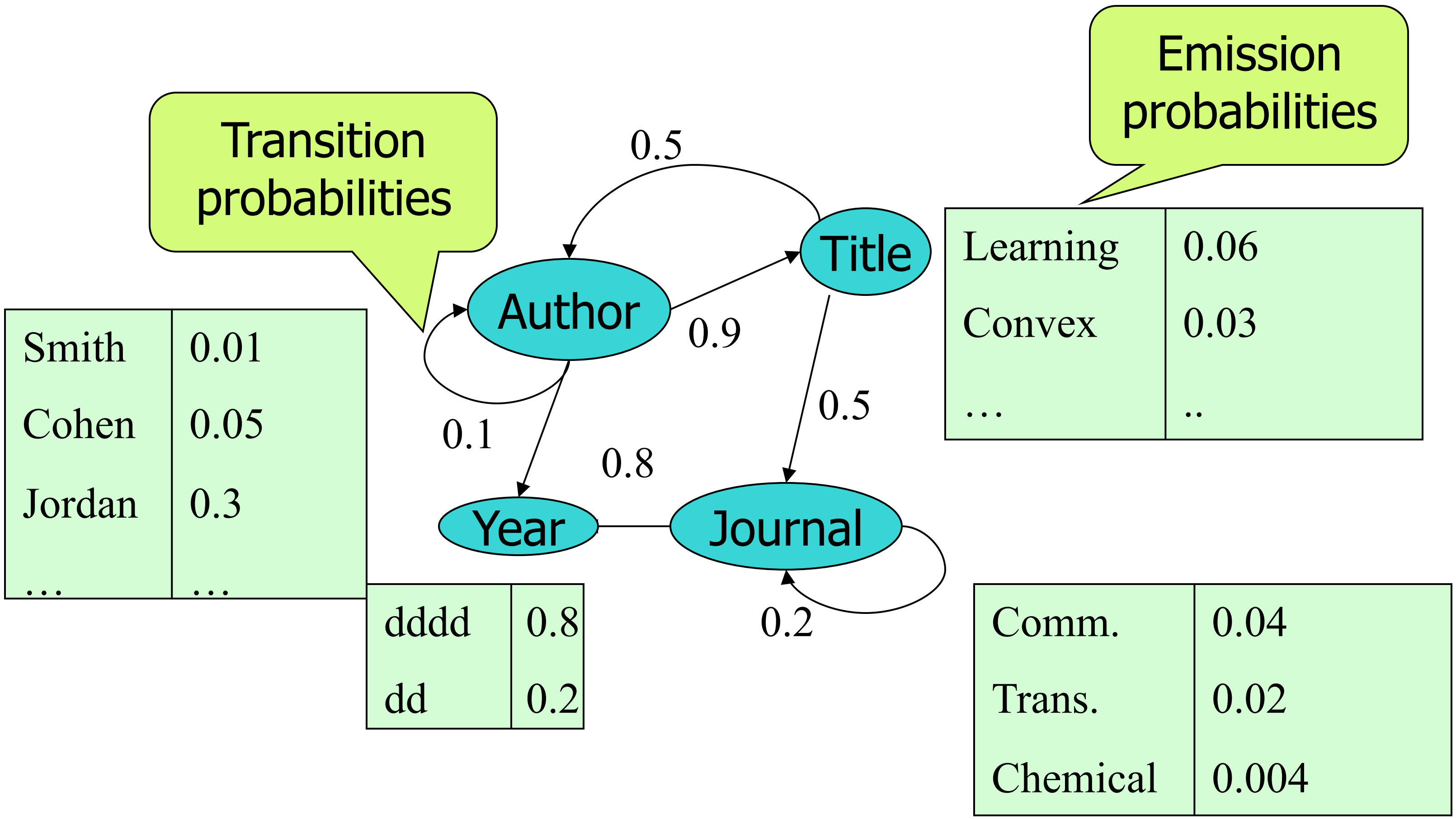
Borkar et al's: HMMs for segmentation

– 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



Smith	0.01
Cohen	0.05
Jordan	0.3
…	…

dddd	0.8
dd	0.2

Learning	0.06
Convex	0.03
…	..

Comm.	0.04
Trans.	0.02
Chemical	0.004

Inference for linear-chain MRFs

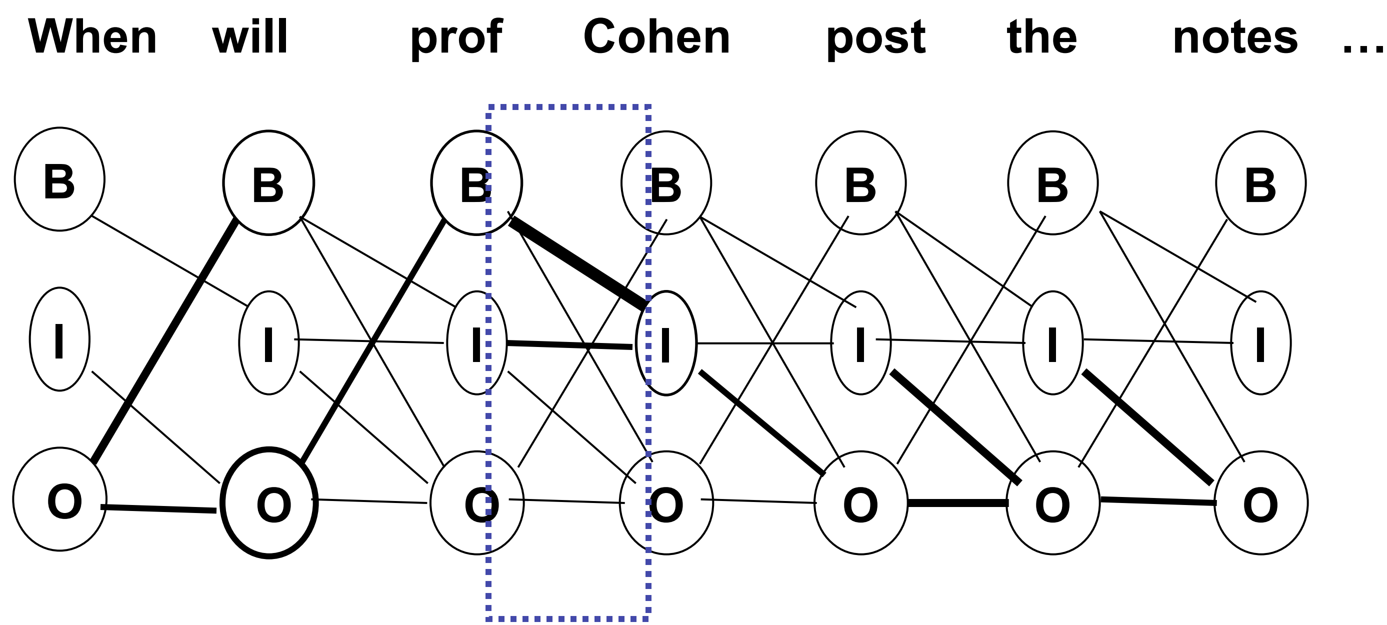
When will prof Cohen post the notes …

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 Ziv William is on the way”

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

Inference for a linear-chain MRF



•Inference: find the highest-weight path

•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 $\mathbf{x}$

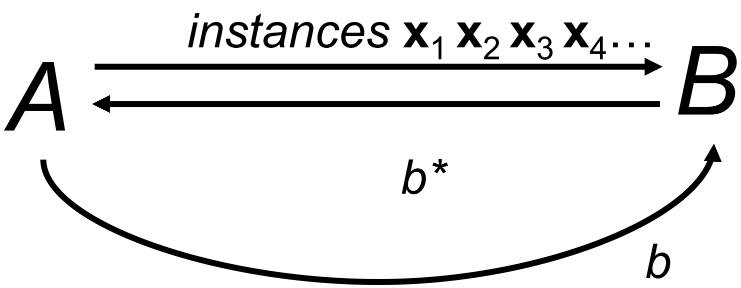
Ranking perceptrons → structured perceptrons

- 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*



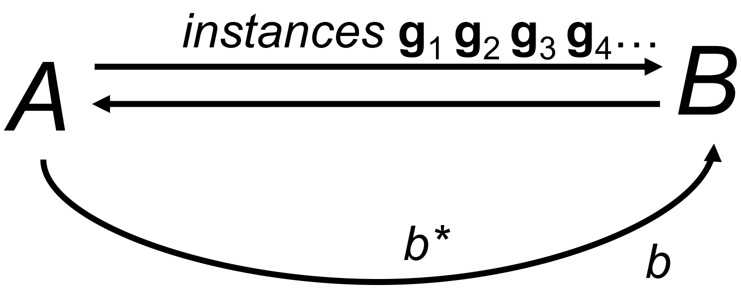
Compute: $y_i = \mathbf{v}_k^{\wedge} . \mathbf{x}_i$

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 NER*



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

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 **y*** that gives the best score for the dot product $\mathbf{v}_k \,.\, \mathbf{F}(\mathbf{x}_i,\mathbf{y}^*)$ 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.

And back to the 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 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

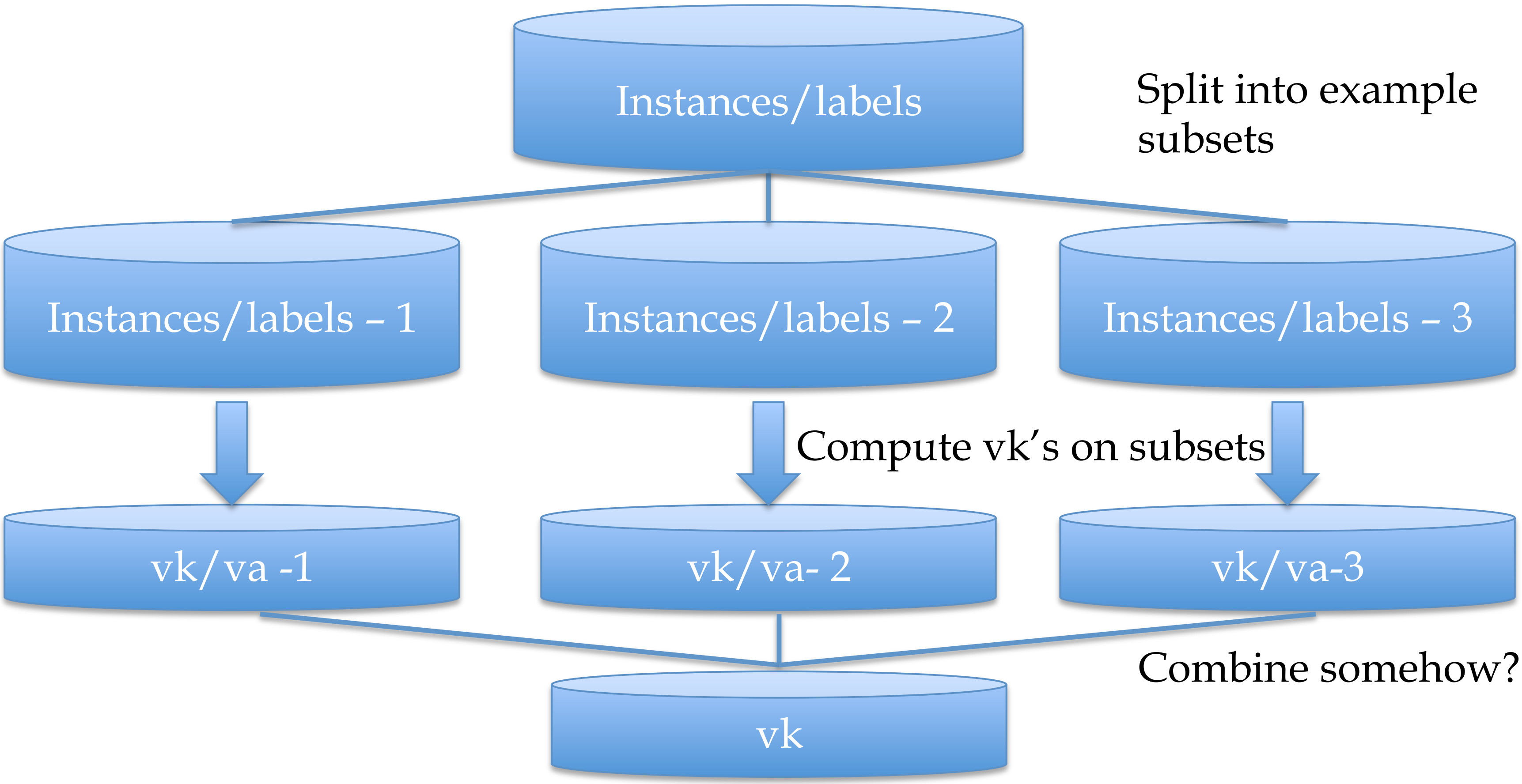
Method	F-Measure	Numits
Perc, avg, cc=0	93.53	13
Perc, noavg, cc=0	93.04	35
Perc, avg, cc=5	93.33	9
Perc, noavg, cc=5	91.88	39
ME, cc=0	92.34	900
ME, cc=5	92.65	200

POS Tagging Results

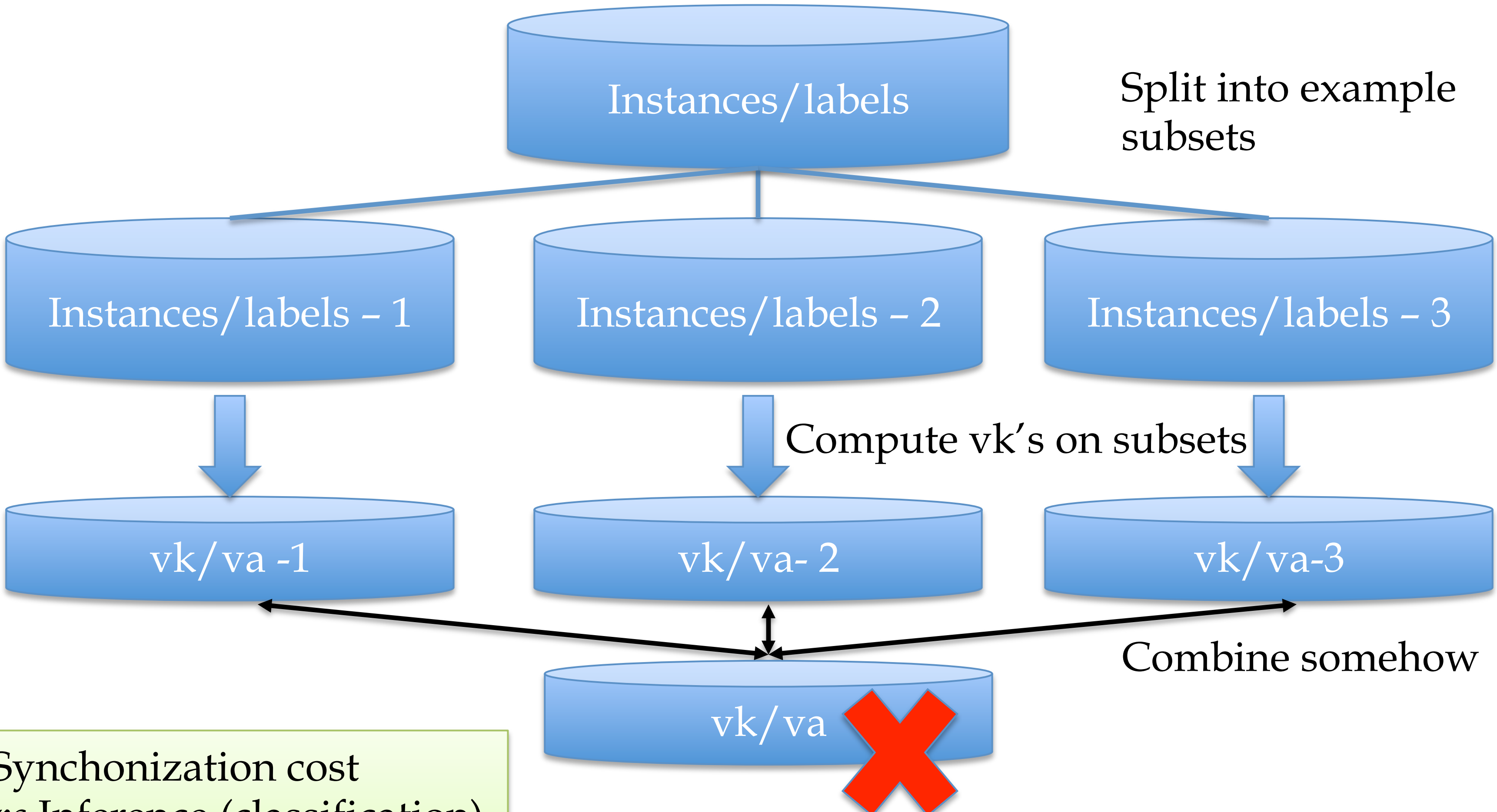
Method	Error rate/%	Numits
Perc, avg, cc=0	2.93	10
Perc, noavg, cc=0	3.68	20
Perc, avg, cc=5	3.03	6
Perc, noavg, cc=5	4.04	17
ME, cc=0	3.4	100
ME, cc=5	3.28	200

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.

Parallelizing perceptrons



Parallelizing perceptrons



Synchonization cost
vs Inference (classification) cost