Perceptrons
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 do 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)
On-line learning

Train Data

\[ \text{instance } x_i \quad \xrightarrow{\text{Compute: } y_i = \text{sign}(v \cdot x_i)} \quad B \]

+1,-1: label \( y_i \)

Get \( y_i \) and make update to \( v \)

To detect interactions:
- increase/decrease \( v_k \) 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
On-line learning

Train Data

Instance $x_i$ ➔ $B$

Compute: $y_i = \text{sign}(v \cdot x_i)$

If mistake: $v_{k+1} = v_k + \text{correction}$

To detect interactions:
• increase/decrease $v_k$ 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 \( t=1, \ldots, \)
    • Player A picks \( x=(x_1, \ldots, x_n) \) 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, \( \hat{y} \), and sends it to A
    • A sends B the true label \( y=c(x) \)
    • 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: $y_i = v_k \cdot x_i$

If mistake: $v_{k+1} = v_k + y_i x_i$

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

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

and furthermore, $\|u\| = 1$.

Radius $R$. $A$ must provide examples “near the origin”, ie

$$\forall x_i \text{ given by } A, \|x\|^2 < R$$
(1) A target $u$

(2) The guess $v_1$ after one positive example.

(3a) The guess $v_2$ after the two positive examples: $v_2 = v_1 + x_2$

(3b) The guess $v_2$ after the one positive and one negative example: $v_2 = v_1 - x_2$

If mistake: $v_{k+1} = v_k + y_i x_i$
The guess $v_2$ after the two positive examples: $v_2 = v_1 + x_2$.

The guess $v_2$ after the one positive and one negative example: $v_2 = v_1 - x_2$.

Lemma 1 $\forall k, v_k \cdot u \geq k\gamma$. In other words, the dot product between $v_k$ and $u$ increases with each mistake, at a rate depending on the margin $\gamma$.

Proof:

\[
v_{k+1} \cdot u = (v_k + y_i x_i) \cdot u
\]

\[
\Rightarrow v_{k+1} \cdot u = (v_k \cdot u) + y_i (x_i \cdot u)
\]

\[
\Rightarrow v_{k+1} \cdot u \geq v_k \cdot u + \gamma
\]

\[
\Rightarrow v_k \cdot u \geq k\gamma
\]
The guess $v_2$ after the two positive examples: $v_2 = v_1 + x_2$  

The guess $v_2$ after the one positive and one negative example: $v_2 = v_1 - x_2$

**Lemma 2** $\forall k, \|v_k\|^2 \leq kR^2$. In other words, the norm of $v_k$ grows “slowly”, at a rate depending on $R^2$.

**Proof:**

$$v_{k+1} \cdot v_{k+1} = (v_k + y_i x_i) \cdot (v_k + y_i x_i)$$

$$\Rightarrow \|v_{k+1}\|^2 = \|v_k\|^2 + 2y_i x_i \cdot v_k + y_i^2 \|x_i\|^2$$

$$\Rightarrow \|v_{k+1}\|^2 = \|v_k\|^2 + [\text{something negative}] + 1\|x_i\|^2$$

$$\Rightarrow \|v_{k+1}\|^2 \leq \|v_k\|^2 + \|x\|^2$$

$$\Rightarrow \|v_{k+1}\|^2 \leq \|v_k\|^2 + R^2$$

$$\Rightarrow \|v_k\|^2 \leq kR^2$$
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 \( \gamma \).

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 \( R \).

\[
(k\gamma)^2 \leq (\mathbf{v}_k \cdot \mathbf{u})^2
\]

\[
\Rightarrow \quad k^2\gamma^2 \leq \| \mathbf{v}_k \|^2 \| \mathbf{u} \|^2
\]

\[
\Rightarrow \quad k^2\gamma^2 \leq \| \mathbf{v}_k \|^2
\]

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

\[
\Rightarrow \quad k\gamma^2 \leq R^2
\]

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

Radius \( R \). A must provide examples “near the origin”, i.e.

\[
\forall x_i \text{ given by } A, \| x \|^2 < R^2
\]
Summary

• We have shown that
  – \textit{If} : exists a \( \mathbf{u} \) with unit norm that has margin \( \gamma \) on examples in
    the seq \((x_1,y_1),(x_2,y_2),\ldots\),
  – \textit{Then} : the perceptron algorithm makes \(< R^2/ \gamma^2\) mistakes on the
    sequence (where \( R \geq ||x_i|| \))
  – \textit{Independent} of dimension of the data or classifier (!)
  – This doesn’t follow from \( M(C) \leq VCDim(C) \)

• We \textit{don’t} know if this algorithm could be better
  – There are many variants that rely on similar analysis (ROMMA, Passive-Aggressive, MIRA, …)

• We \textit{don’t} know what happens if the data’s not separable
  – Unless I explain the “\( \Delta \) trick” to you

• We \textit{don’t} know what classifier to use “after” training
The $\Delta$ 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

\[
\begin{align*}
\mathbf{x}^1 &= (x_1^1, x_2^1, \ldots, x_m^1) \rightarrow (x_1^1, x_2^1, \ldots, x_m^1, \Delta, 0, \ldots, 0) \\
\mathbf{x}^2 &= (x_1^2, x_2^2, \ldots, x_m^2) \rightarrow (x_1^2, x_2^2, \ldots, x_m^2, 0, \Delta, \ldots, 0) \\
\ldots \\
\mathbf{x}^n &= (x_1^n, x_2^n, \ldots, x_m^n) \rightarrow (x_1^n, x_2^n, \ldots, x_m^n, 0, 0, \ldots, \Delta)
\end{align*}
\]
The $\Delta$ Trick

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

• We have shown that
  – *If*: exists a $\mathbf{u}$ with unit norm that has margin $\gamma$ on examples in the seq $(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),\ldots$.
  – *Then*: the perceptron algorithm makes $< \frac{R^2}{\gamma^2}$ 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 “$\Delta$ trick” to you

• We *don’t* know what classifier to use “after” training
On-line to batch learning

1. Pick a $v_k$ at random according to $m_k/m$, the fraction of examples it was used for.

2. Predict using the $v_k$ you just picked.

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

---

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

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

---

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

<table>
<thead>
<tr>
<th>$i$</th>
<th>guess</th>
<th>input</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$v_0$</td>
<td>$x_1$</td>
<td>X (a mistake)</td>
</tr>
<tr>
<td>2</td>
<td>$v_1$</td>
<td>$x_2$</td>
<td>✓ (correct!)</td>
</tr>
<tr>
<td>3</td>
<td>$v_1$</td>
<td>$x_3$</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>$v_1$</td>
<td>$x_4$</td>
<td>X (a mistake)</td>
</tr>
<tr>
<td>5</td>
<td>$v_2$</td>
<td>$x_5$</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>$v_2$</td>
<td>$x_6$</td>
<td>✓</td>
</tr>
<tr>
<td>7</td>
<td>$v_2$</td>
<td>$x_7$</td>
<td>✓</td>
</tr>
<tr>
<td>8</td>
<td>$v_2$</td>
<td>$x_8$</td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>$v_3$</td>
<td>$x_9$</td>
<td>✓</td>
</tr>
<tr>
<td>10</td>
<td>$v_3$</td>
<td>$x_{10}$</td>
<td>X</td>
</tr>
</tbody>
</table>
STOPPED - TUES
Complexity of perceptron learning

- Algorithm: $O(n)$
- $v = 0$
- for each example $x, y$:
  - if $\text{sign}(v \cdot x) \neq y$
    - $v = v + yx$
      - $O(|x|) = O(|d|)$
- init hashtable
- for $x_i \neq 0$, $v_i += yx_i$
Complexity of *averaged* perceptron

- Algorithm: \( O(n) \) \( O(n|V|) \)
- \( v_k = 0 \)
- \( v_a = 0 \)
- for each example \( x,y \):
  - if \( \text{sign}(v_k.x) \neq y \) \( O(|V|) \)
    - \( v_a = v_a + v_k \)
    - \( v_k = v_k + yx \)
    - \( m_k = 1 \) \( O(|x|) = O(|d|) \)
  - else
    - \( n_k++ \)
- init hash tables
- for \( v_{k_i} \neq 0 \), \( v_{a_i} += v_{k_i} \)
- for \( x_{i} \neq 0 \), \( v_{i} += yx_{i} \)
SPARSIFYING THE AVERAGED PERCEPTRON UPDATE
Complexity of perceptron learning

• Algorithm: \( O(n) \)
• \( v=0 \)
• for each example \( x,y \):
  – if \( \text{sign}(v.x) \neq y \)
    • \( v = v + yx \) \( O(|x|)=O(|d|) \)
• init hashtable
• for \( x_i \neq 0 \), \( v_i \mathbf{+}= yx_i \)
Complexity of *averaged* perceptron

- Algorithm: \( O(n) \quad O(n |V|) \)

- \( v_k = 0 \)

- \( v_a = 0 \)

- for each example \( x,y \):
  - if \( \text{sign}(v_k.x) \neq y \)
    - \( v_a = v_a + v_k \)
    - \( v_k = v_k + yx \)
    - \( m_k = 1 \)
    - \( O(|x|) = O(|d|) \)
  - else
    - \( n_k ++ \)

- init hashtables

- for \( v_{k_i} \neq 0 \), \( v_{a_i} += v_{k_i} \)

- for \( x_{i} \neq 0 \), \( v_{i} += yx_{i} \)
Alternative averaged perceptron

• Algorithm:
  • \( v_k = 0 \)
  • \( v_a = 0 \)
  • for each example \( x, y \):
    – \( v_a = v_a + v_k \)
    – \( t = t + 1 \)
    – if \( \text{sign}(v_k \cdot x) \neq y \)
      • \( v_k = v_k + y^*x \)
  • Return \( v_a / t \)

Observe:

\[ v_k = \sum_{j \in S_k} y_j x_j \]

\( S_k \) is the set of examples including the first \( k \) mistakes
**Alternative averaged perceptron**

- **Algorithm:**
  - $v_k = 0$
  - $v_a = 0$
  - for each example $x, y$:
    - $v_a = v_a + \sum_{j \in S_k} y_j x_j$
    - $t = t + 1$
    - if $\text{sign}(v_k . x) \neq y$
      - $v_k = v_k + y^* x$
  - Return $v_a / t$

So when there’s a mistake at time $t$ on $x, y$:

$y^* x$ is added to $v_a$ on every subsequent iteration

Suppose you know $T$, the total number of examples in the stream...
Alternative averaged perceptron

• Algorithm:
  • $v_k = 0$
  • $v_a = 0$
  • for each example $x, y$:
    - $v_a = v_a + \sum_{j \in S_k} y_j x_j$
    - $t = t+1$
    - if $\text{sign}(v_k.x) \neq y$
      • $v_k = v_k + y^* x$
      • $v_a = v_a + (T-t) y^* x$
  • Return $v_a / T$

$T = \text{the total number of examples in the stream…}$

Unpublished? I figured this out recently, Leon Bottou knows it too.
KERNELS AND PERCEPTRONS
The kernel perceptron

Compute: $\hat{y}_i = v_k \cdot x_i$

If mistake: $v_{k+1} = v_k + y_i x_i$

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

Compute: \( y_i = v_k \cdot x_i \)

If mistake: \( v_{k+1} = v_k + y_i x_i \)

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.

\[
K(x, x_k) \equiv x \cdot x_k
\]

\[
\hat{y} = \sum_{x_{k+} \in FN} K(x_i, x_{k+}) - \sum_{x_{k-} \in FP} K(x_i, x_{k-})
\]
Some common kernels

• Linear kernel: \[ K(x, x') \equiv x \cdot x' \]

• Polynomial kernel: \[ K(x, x') \equiv (x \cdot x' + 1)^d \]

• Gaussian kernel: \[ K(x, x') \equiv e^{-\|x-x'|^2 / \sigma} \]

• More later…. 
Kernels 101

• Duality
  – and computational properties
  – Reproducing Kernel Hilbert Space (RKHS)
• Gram matrix
• Positive semi-definite
• Closure properties
Kernels 101

• Duality: two ways to look at this

\[ \hat{y} = x \cdot w = K(x, w) \]

\[ w = \sum_{x_{k^+} \in FN} x_{k^+} - \sum_{x_{k^-} \in FP} x_{k^-} \]

\[ \hat{y} = \sum_{x_{k^+} \in FN} K(x_i, x_{k^+}) - \sum_{x_{k^-} \in FP} K(x_i, x_{k^-}) \]

\[ K(x, x_k) \equiv \phi(x) \cdot \phi(x_k) \]

\[ \hat{y} = \sum_{x_{k^+} \in FN} K(x_i, x_{k^+}) - \sum_{x_{k^-} \in FP} K(x_i, x_{k^-}) \]

\[ K(x, x_k) \equiv \phi(x') \cdot \phi(x'_k) \]

Two different computational ways of getting the same behavior
**Kernels 101**

- **Duality**
- **Gram matrix**: \( k_{ij} = K(x_i, x_j) \)

\[ K(x, x') = K(x', x) \Rightarrow \text{Gram matrix is symmetric} \]

\[ K(x, x) > 0 \Rightarrow \text{diagonal of } K \text{ is positive} \Rightarrow K \text{ is “positive semi-definite”} \Rightarrow z^T K z \geq 0 \text{ for all } z \]

<table>
<thead>
<tr>
<th></th>
<th>K(1,1)</th>
<th>K(1,2)</th>
<th>K(1,3)</th>
<th>...</th>
<th>K(1,m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>K(2,1)</td>
<td>K(2,2)</td>
<td>K(2,3)</td>
<td>...</td>
<td>K(2,m)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>K(m,1)</td>
<td>K(m,2)</td>
<td>K(m,3)</td>
<td>...</td>
<td>K(m,m)</td>
<td></td>
</tr>
</tbody>
</table>
Kernels 101

• Duality
• Gram matrix: $K_{ij} = K(x_i, x_j)$

\[
K(x, x') = K(x', x) \Rightarrow \text{Gram matrix is symmetric}
\]

\[
K(x, x) > 0 \Rightarrow \text{diagonal of } K \text{ is positive } \Leftrightarrow K \text{ is “positive semi-definite” } \Leftrightarrow \ldots \Leftrightarrow z^T K z \geq 0 \text{ for all } z
\]

Fun fact: Gram matrix positive semi-definite $\Leftrightarrow$

\[
K(x_i, x_j) = \phi(x_i), \phi(x_j) \text{ for some } \phi
\]

Proof: $\phi(x)$ uses the eigenvectors of $K$ to represent $x$
HASH KERNELS AND "THE HASH TRICK"
Question

• Most of the weights in a classifier are small and not important
Hash Kernels

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

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

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

Vishy Vishwanathan  
Department of Statistics  
Purdue University, IN, USA
Some details

Slightly different hash to avoid systematic bias

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

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

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

Slightly different hash to avoid systematic bias

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

Lemma 2 The hash kernel is unbiased, that is

\[ E_{\varphi}[\langle x, x' \rangle_{\varphi}] = \langle x, x' \rangle. \] 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 \n
\[ \|x\|_2 = \|x'\|_2 = 1, \quad \sigma^2_{x,x'} = O \left( \frac{1}{m} \right). \]

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

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

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

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

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
Corollary 5  Denote by $X = \{x_1, \ldots, x_n\}$ a set of vectors which satisfy $\|x_i - x_j\|_\infty \leq \eta \|x_i - x_j\|_2$ for all pairs $i, j$. In this case with probability $1 - \delta$ we have for all $i, j$

$$\left| \frac{\|x_i - x_j\|_\phi^2 - \|x_i - x_j\|_2^2}{\|x_i - x_j\|_2^2} \right| \leq \sqrt{\frac{2}{m}} + 64\eta^2 \log^2 \frac{n}{2\delta}.$$ 

This means that the number of observations $n$ (or correspondingly the size of the un-hashed kernel matrix) only enters logarithmically in the analysis.
The hash kernel: implementation

• One problem: debugging is harder
  – Features are no longer meaningful
  – There’s a new way to ruin a classifier
    • Change the hash function 🙁

• You can separately compute the set of all words that hash to $h$ and guess what features mean
  – Build an inverted index $h \rightarrow w_1, w_2, \ldots$,
Parallelizing perceptrons

Instances/labels

Instances/labels – 1
Instances/labels – 2
Instances/labels – 3

vk/va -1
vk/va- 2
vk/va-3

vk

Split into example subsets
Compute vk’s on subsets
Combine somehow?
Parallelizing perceptrons

Instances/labels

Instances/labels – 1
Instances/labels – 2
Instances/labels – 3

vk/va -1
vk/va - 2
vk/va-3

Compute vk’s on subsets

Split into example subsets

Combine somehow

Synchronization 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

William Cohen
The voted perceptron for ranking

Compute: \( y_i = \hat{v}_k \cdot x_i \)
Return: the index \( b^* \) of the "best" \( x_i \)
If mistake: \( v_{k+1} = v_k + x_b - x_{b^*} \)

Margin \( \gamma \). A must provide examples that can be correctly ranked with some vector \( u \) with margin \( \gamma > 0 \), ie

\[ \exists u : \forall x_{i,1}, \ldots, x_{i,n}, \ell \text{ given by } A, \forall j \neq \ell, \ u \cdot x_{\ell} - u \cdot x_j > \gamma \]
and furthermore, \( \|u\|^2 = 1 \).

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

\[ \forall x_i \text{ given by } A, \|x\|^2 < R^2 \]
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 \( \mathbf{v} \) by adding \( x_b - x_{b^*} \)
Correcting $\mathbf{v}$ by adding $x_b - x_{b*}$

(part 2)
Lemma 1 \( \forall k, v_k \cdot u \geq k \gamma \). In other words, the dot product between \( v_k \) and \( u \) increases with each mistake, at a rate depending on the margin \( \gamma \).

Proof:

\[
\begin{align*}
v_{k+1} \cdot u &= (v_k + y_i x_i) \cdot u \\
\Rightarrow v_{k+1} \cdot u &= (v_k \cdot u) + y_i (x_i \cdot u) \\
\Rightarrow v_{k+1} \cdot u &\geq v_k \cdot u + \gamma \\
\Rightarrow v_k \cdot u &\geq k \gamma
\end{align*}
\]
The guess $v_2$ after the two positive examples: $v_2 = v_1 + x_2$

Lemma 3 \( \forall k, v_k \cdot u \geq k\gamma \). In other words, the dot product between $v_k$ and $u$ increases with each mistake, at a rate depending on the margin $\gamma$.

\[
\begin{align*}
    v_{k+1} \cdot u &= (v_k + y_i x_i) \cdot u \\
    \Rightarrow & \quad v_{k+1} \cdot u = (v_k \cdot u) + y_i (x_i \cdot u) \\
    \Rightarrow & \quad v_{k+1} \cdot u \geq v_k \cdot u + \gamma \\
\end{align*}
\]

\[
\begin{align*}
    v_{k+1} \cdot u &= (v_k + x_{i,\ell} - x_{i,\hat{\ell}}) \cdot u \\
    \Rightarrow & \quad v_{k+1} \cdot u = v_k \cdot u + x_{i,\ell} \cdot u - x_{i,\hat{\ell}} \cdot u \\
    \Rightarrow & \quad v_{k+1} \cdot u \geq v_k \cdot u + \gamma \\
\end{align*}
\]
The guess $v_2$ after the two positive examples: $v_2 = v_1 + x_2$

Lemma 3

$\forall k$, $v_k \cdot u \geq k\gamma$. In other words, the dot product between $v_k$ and $u$ increases with each mistake, at a rate depending on the margin $\gamma$.

\[
\begin{align*}
\mathbf{v}_{k+1} \cdot \mathbf{u} &= (\mathbf{v}_k + y_i \mathbf{x}_i) \cdot \mathbf{u} \\
\Rightarrow \quad \mathbf{v}_{k+1} \cdot \mathbf{u} &= (\mathbf{v}_k \cdot \mathbf{u}) + y_i (\mathbf{x}_i \cdot \mathbf{u}) \\
\Rightarrow \quad \mathbf{v}_{k+1} \cdot \mathbf{u} &\geq \mathbf{v}_k \cdot \mathbf{u} + \gamma \\
\Rightarrow \quad \mathbf{v}_k \cdot \mathbf{u} &\geq k\gamma
\end{align*}
\]
Notice this doesn’t depend at all on the number of $x$’s being ranked.

$\text{(3a)}$ The guess $v_2$ after the two positive examples: $v_2 = v_1 + x_2$

Lemma 4 $\forall k, \|v_k\|^2 \leq 2kR$.

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:
    \[ x = (w_1, \ldots, w_n) \]
  – Output: list of labels:
    \[ y = (y_1, \ldots, y_n) \]
  – If there are K classes, there are \( K^n \) labels possible for \( x \)
Borkar et al’s: HMMs for segmentation

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

IE with Hidden Markov Models

Transition probabilities

<table>
<thead>
<tr>
<th>Author</th>
<th>Title</th>
<th>Year</th>
<th>Journal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>0.01</td>
<td>0.1</td>
<td>0.9</td>
</tr>
<tr>
<td>Cohen</td>
<td>0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jordan</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Emission probabilities

<table>
<thead>
<tr>
<th>Emission</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning</td>
<td>0.06</td>
</tr>
<tr>
<td>Convex</td>
<td>0.03</td>
</tr>
<tr>
<td>…</td>
<td>..</td>
</tr>
<tr>
<td>Comm.</td>
<td>0.04</td>
</tr>
<tr>
<td>Trans.</td>
<td>0.02</td>
</tr>
<tr>
<td>Chemical</td>
<td>0.004</td>
</tr>
</tbody>
</table>
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: \( x=(w_1,\ldots,w_n) \)
  – Output: list of labels: \( y=(y_1,\ldots,y_n) \)
  – If there are K classes, there are \( K^n \) labels possible for \( x \)
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: \( x = (w_1, \ldots, w_n) \)
  - Output: list of labels: \( y = (y_1, \ldots, y_n) \)
  - If there are \( K \) classes, there are \( K^n \) labels possible for \( x \)
• 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: $x=(w_1,\ldots,w_n)$
  – 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 ranking

Compute: \( y_i = \hat{v}_k \cdot x_i \)

Return: the index \( b^* \) of the “best” \( x_i \)

If mistake: \( v_{k+1} = v_k + x_b - x_{b^*} \)

Change number one is notation: replace \( x \) with \( g \)
The voted perceptron for NER

A \quad \leftrightarrow \quad B

\begin{align*}
\text{Compute: } y_i &= \hat{v}_k \cdot g_i \\
\text{Return: the index } b^* \text{ of the “best” } g_i \\
\text{If mistake: } v_{k+1} &= v_k + g_b - g_{b^*}
\end{align*}

1. A sends B feature functions, and instructions for creating the instances $g$:
   - A sends a word vector $x_i$. Then B could create the instances $g_1 = F(x_i, y_1), g_2 = F(x_i, y_2), \ldots$
   - but instead B just returns the $y^*$ that gives the best score for the dot product $v_k \cdot F(x_i, y^*)$ by using Viterbi.

2. A sends B the correct label sequence $y_i$.

3. On errors, B sets $v_{k+1} = v_k + g_b - g_{b^*} = v_k + F(x_i, y) - F(x_i, y^*)$

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

EMNLP 2002
Some background...

- Collins’ parser: generative model...
  - 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…

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>R (±)</th>
<th>F (±)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max-Ent</td>
<td>84.4</td>
<td>86.3</td>
<td>85.3</td>
</tr>
<tr>
<td>Boosting</td>
<td>87.3(18.6)</td>
<td>87.9(11.6)</td>
<td>87.6(15.6)</td>
</tr>
<tr>
<td>Voted Perceptron</td>
<td>87.3(18.6)</td>
<td>88.6(16.8)</td>
<td>87.9(17.7)</td>
</tr>
</tbody>
</table>

Figure 5: Results for the three tagging methods. $P = \text{precision}$, $R = \text{recall}$, $F = \text{F-measure}$. Figures in parentheses 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

<table>
<thead>
<tr>
<th>NP Chunking Results</th>
<th>POS Tagging Results</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Method</strong></td>
<td><strong>F-Measure</strong></td>
</tr>
<tr>
<td>Perc, avg, cc=0</td>
<td>93.53</td>
</tr>
<tr>
<td>Perc, noavg, cc=0</td>
<td>93.04</td>
</tr>
<tr>
<td>Perc, avg, cc=5</td>
<td>93.33</td>
</tr>
<tr>
<td>Perc, noavg, cc=5</td>
<td>91.88</td>
</tr>
<tr>
<td>ME, cc=0</td>
<td>92.34</td>
</tr>
<tr>
<td>ME, cc=5</td>
<td>92.65</td>
</tr>
</tbody>
</table>

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

Instances/labels

Instances/labels – 1
vk/va -1

Instances/labels – 2
vk/va-2

Instances/labels – 3
vk/va-3

Split into example subsets

Compute vk’s on subsets

vk

Combine somehow?
Parallelizing perceptrons

Instances/labels

Instances/labels – 1

Instances/labels – 2

Instances/labels – 3

vk/va -1

vk/va - 2

vk/va-3

Compute vk’s on subsets

Combine somehow

Split into example subsets

Synchronization cost vs Inference (classification) cost