Machine Learning for NLP

Uppsala University
Department of Linguistics and Philology

Slides borrowed from Ryan McDonald, Google Research
Linear Classifiers

- Classifiers covered so far:
  - Decision trees
  - Nearest neighbor

- Next two lectures: Linear classifiers

- Statistics from Google Scholar (October 2009):
  - “Maximum Entropy” & “NLP” 2660 hits, 141 before 2000
  - “SVM” & “NLP” 2210 hits, 16 before 2000
  - “Perceptron” & “NLP”, 947 hits, 118 before 2000

- All are linear classifiers that have become important tools in any NLP/CL researcher’s tool-box in past 10 years
Outline

▶ Today:
  ▶ Preliminaries: input/output, features, etc.
  ▶ Linear classifiers
    ▶ Perceptron
    ▶ Large-margin classifiers (SVMs, MIRA)
    ▶ Logistic regression (Maximum Entropy)

▶ Next time:
  ▶ Structured prediction with linear classifiers
    ▶ Structured perceptron
    ▶ Structured large-margin classifiers (SVMs, MIRA)
    ▶ Conditional random fields
  ▶ Case study: Dependency parsing
Inputs and Outputs

- **Input:** \( x \in \mathcal{X} \)
  - e.g., document or sentence with some words \( x = w_1 \ldots w_n \), or a series of previous actions
- **Output:** \( y \in \mathcal{Y} \)
  - e.g., parse tree, document class, part-of-speech tags, word-sense
- **Input/output pair:** \((x, y) \in \mathcal{X} \times \mathcal{Y}\)
  - e.g., a document \( x \) and its label \( y \)
  - Sometimes \( x \) is explicit in \( y \), e.g., a parse tree \( y \) will contain the sentence \( x \)
Feature Representations

- We assume a mapping from input-output pairs \((x, y)\) to a high dimensional feature vector
  \[ f(x, y) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^m \]

- For some cases, i.e., binary classification \(\mathcal{Y} = \{-1, +1\}\), we can map only from the input to the feature space
  \[ f(x) : \mathcal{X} \to \mathbb{R}^m \]

- However, most problems in NLP require more than two classes, so we focus on the multi-class case

- For any vector \(v \in \mathbb{R}^m\), let \(v_j\) be the \(j^{th}\) value
Features and Classes

- All features must be numerical
  - Numerical features are represented directly as $f_i(x, y) \in \mathbb{R}$
  - Binary (boolean) features are represented as $f_i(x, y) \in \{0, 1\}$
- Multinomial (categorical) features must be binarized
  - Instead of: $f_i(x, y) \in \{v_0, \ldots, v_p\}$
  - We have: $f_{i+0}(x, y) \in \{0, 1\}, \ldots, f_{i+p}(x, y) \in \{0, 1\}$
  - Such that: $f_{i+j}(x, y) = 1$ iff $f_i(x, y) = v_j$
- We need distinct features for distinct output classes
  - Instead of: $f_i(x)$ ($1 \leq i \leq m$)
  - We have: $f_{i+0m}(x, y), \ldots, f_{i+Nm}(x, y)$ for $Y = \{0, \ldots, N\}$
  - Such that: $f_{i+jm}(x, y) = f_i(x)$ iff $y = y_j$
Examples

- $x$ is a document and $y$ is a label
  \[ f_j(x, y) = \begin{cases} 
  1 & \text{if } x \text{ contains the word “interest”} \\
  & \text{and } y = “financial” \\
  0 & \text{otherwise} 
\end{cases} \]

  \[ f_j(x, y) = \% \text{ of words in } x \text{ with punctuation and } y = “scientific” \]

- $x$ is a word and $y$ is a part-of-speech tag
  \[ f_j(x, y) = \begin{cases} 
  1 & \text{if } x = “bank” \text{ and } y = \text{Verb} \\
  0 & \text{otherwise} 
\end{cases} \]
Examples

▶ \( x \) is a name, \( y \) is a label classifying the name

\[
\begin{align*}
\mathbf{f}_0(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "George"} \\
& \text{and } y = "Person" \\
0 & \text{otherwise}
\end{cases} \\
\mathbf{f}_4(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "George"} \\
& \text{and } y = "Object" \\
0 & \text{otherwise}
\end{cases} \\
\mathbf{f}_1(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "Washington"} \\
& \text{and } y = "Person" \\
0 & \text{otherwise}
\end{cases} \\
\mathbf{f}_5(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "Washington"} \\
& \text{and } y = "Object" \\
0 & \text{otherwise}
\end{cases} \\
\mathbf{f}_2(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "Bridge"} \\
& \text{and } y = "Person" \\
0 & \text{otherwise}
\end{cases} \\
\mathbf{f}_6(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "Bridge"} \\
& \text{and } y = "Object" \\
0 & \text{otherwise}
\end{cases} \\
\mathbf{f}_3(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "General"} \\
& \text{and } y = "Person" \\
0 & \text{otherwise}
\end{cases} \\
\mathbf{f}_7(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "General"} \\
& \text{and } y = "Object" \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]

▶ \( x = \text{General George Washington}, \ y = \text{Person} \rightarrow \mathbf{f}(x, y) = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0] \)
▶ \( x = \text{George Washington Bridge}, \ y = \text{Object} \rightarrow \mathbf{f}(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0] \)
▶ \( x = \text{George Washington George}, \ y = \text{Object} \rightarrow \mathbf{f}(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0] \)
Block Feature Vectors

- \( x = \text{General George Washington}, \quad y = \text{Person} \rightarrow f(x, y) = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0] \)
- \( x = \text{George Washington Bridge}, \quad y = \text{Object} \rightarrow f(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0] \)
- \( x = \text{George Washington George}, \quad y = \text{Object} \rightarrow f(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0] \)

- One equal-size block of the feature vector for each label
- Input features duplicated in each block
- Non-zero values allowed only in one block
Linear Classifiers

- **Linear classifier**: score (or probability) of a particular classification is based on a linear combination of features and their weights.

- Let $w \in \mathbb{R}^m$ be a high dimensional weight vector.

- If we assume that $w$ is known, then we define our classifier as:

  - **Multiclass Classification**: $\mathcal{Y} = \{0, 1, \ldots, N\}$
    \[
    y = \arg \max_y \quad w \cdot f(x, y)
    \]
    \[
    = \arg \max_y \quad \sum_{j=0}^{m} w_j \times f_j(x, y)
    \]

- **Binary Classification** just a special case of multiclass.
Linear Classifiers - Bias Terms

- Often linear classifiers presented as

\[ y = \arg \max_y \sum_{j=0}^{m} w_j \times f_j(x, y) + b_y \]

- Where \( b \) is a bias or offset term
- But this can be folded into \( f \)

\( x=\text{General George Washington, } y=\text{Person} \rightarrow f(x, y) = [1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0] \)
\( x=\text{General George Washington, } y=\text{Object} \rightarrow f(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1] \)

\[ f_4(x, y) = \begin{cases} 1 & y = \text{“Person”} \\ 0 & \text{otherwise} \end{cases} \]
\[ f_9(x, y) = \begin{cases} 1 & y = \text{“Object”} \\ 0 & \text{otherwise} \end{cases} \]

- \( w_4 \) and \( w_9 \) are now the bias terms for the labels
Binary Linear Classifier

Divides all points:
**Multiclass Linear Classifier**

Defines regions of space:

\[ + = \text{arg max}_y \ w \cdot f(x, y) \]

- i.e., + are all points \((x, y)\) where \(+ = \text{arg max}_y \ w \cdot f(x, y)\)
Separability

- A set of points is separable, if there exists a $w$ such that classification is perfect

- This can also be defined mathematically (and we will shortly)
Supervised Learning – how to find $w$

- Input: training examples $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{||\mathcal{T}||}$
- Input: feature representation $f$
- Output: $w$ that maximizes/minimizes some important function on the training set
  - minimize error (Perceptron, SVMs, Boosting)
  - maximize likelihood of data (Logistic Regression, Naive Bayes)
- Assumption: The training data is separable
  - Not necessary, just makes life easier
  - There is a lot of good work in machine learning to tackle the non-separable case
Perceptron

- Choose a $w$ that minimizes error

$$w = \arg \min_w \sum_t 1 - 1[y_t = \arg \max_y w \cdot f(x_t, y)]$$

$$1[p] = \begin{cases} 
1 & p \text{ is true} \\
0 & \text{otherwise} 
\end{cases}$$

- This is a 0-1 loss function
  - Aside: when minimizing error people tend to use hinge-loss or other smoother loss functions
Perceptron Learning Algorithm

Training data: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\mathcal{T}}$

1. $w^{(0)} = 0$; $i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. Let $y' = \arg \max_y w^{(i)} \cdot f(x_t, y)$
5. if $y' \neq y_t$
6. $w^{(i+1)} = w^{(i)} + f(x_t, y_t) - f(x_t, y')$
7. $i = i + 1$
8. return $w^i$
Linear Classifiers

Perceptron: Separability and Margin

- Given an training instance \((x_t, y_t)\), define:
  - \(\tilde{Y}_t = \mathcal{Y} - \{y_t\}\)
  - i.e., \(\tilde{Y}_t\) is the set of incorrect labels for \(x_t\)

- A training set \(\mathcal{T}\) is separable with margin \(\gamma > 0\) if there exists a vector \(u\) with \(\|u\| = 1\) such that:

  \[
  u \cdot f(x_t, y_t) - u \cdot f(x_t, y') \geq \gamma
  \]
  
  for all \(y' \in \tilde{Y}_t\) and \(\|u\| = \sqrt{\sum_j u_j^2}\) (Euclidean or \(L^2\) norm)

- **Assumption**: the training set is separable with margin \(\gamma\)
Perceptron: Main Theorem

- **Theorem**: For any training set separable with a margin of $\gamma$, the following holds for the perceptron algorithm:

  \[
  \text{mistakes made during training} \leq \frac{R^2}{\gamma^2}
  \]

  where $R \geq ||f(x_t, y_t) - f(x_t, y')||$ for all $(x_t, y_t) \in \mathcal{T}$ and $y' \in \bar{\mathcal{Y}}_t$

- Thus, after a finite number of training iterations, the error on the training set will converge to zero

- For proof, see Collins (2002)
Perceptron Summary

- Learns a linear classifier that minimizes error
- Guaranteed to find a $\mathbf{w}$ in a finite amount of time
- Perceptron is an example of an online learning algorithm
  - $\mathbf{w}$ is updated based on a single training instance in isolation
    \[
    \mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + f(x_t, y_t) - f(x_t, y')
    \]
- Compare decision trees that perform batch learning
  - All training instances are used to find best split
Margin

Denote the value of the margin by $\gamma$
Maximizing Margin

For a training set $\mathcal{T}$, the margin of a weight vector $\mathbf{w}$ is the smallest $\gamma$ such that

$$\mathbf{w} \cdot \mathbf{f}(x_t, y_t) - \mathbf{w} \cdot \mathbf{f}(x_t, y') \geq \gamma$$

for every training instance $(x_t, y_t) \in \mathcal{T}$, $y' \in \bar{Y}_t$.
Maximizing Margin

▶ Intuitively maximizing margin makes sense
▶ More importantly, generalization error to unseen test data is proportional to the inverse of the margin

\[ \epsilon \propto \frac{R^2}{\gamma^2 \times |T|} \]

▶ Perceptron: we have shown that:
  ▶ If a training set is separable by some margin, the perceptron will find a \( w \) that separates the data
  ▶ However, the perceptron does not pick \( w \) to maximize the margin!
Maximizing Margin

Let $\gamma > 0$

$$\max_{||w|| \leq 1} \gamma$$

such that:

$$w \cdot f(x_t, y_t) - w \cdot f(x_t, y') \geq \gamma$$

$\forall (x_t, y_t) \in T$

and $y' \in \bar{Y}_t$

- Note: algorithm still minimizes error
- $||w||$ is bound since scaling trivially produces larger margin

$$\beta(w \cdot f(x_t, y_t) - w \cdot f(x_t, y')) \geq \beta \gamma$$, for some $\beta \geq 1$
Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

$$\max_{||w|| \leq 1} \gamma$$

such that:

$$w \cdot f(x_t, y_t) - w \cdot f(x_t, y') \geq \gamma$$

$\forall (x_t, y_t) \in T$

and $y' \in \bar{Y}_t$

Min Norm:

$$\min_w \frac{1}{2} ||w||^2$$

such that:

$$w \cdot f(x_t, y_t) - w \cdot f(x_t, y') \geq 1$$

$\forall (x_t, y_t) \in T$

and $y' \in \bar{Y}_t$

- Instead of fixing $||w||$ we fix the margin $\gamma = 1$
- Technically $\gamma \propto 1/||w||$
Support Vector Machines

\[ \min \frac{1}{2} ||w||^2 \]

such that:

\[ w \cdot f(x_t, y_t) - w \cdot f(x_t, y') \geq 1 \]

\[ \forall (x_t, y_t) \in \mathcal{T} \]

and \( y' \in \bar{Y}_t \)

- Quadratic programming problem – a well known convex optimization problem
- Can be solved with out-of-the-box algorithms
- **Batch learning algorithm** – \( w \) set w.r.t. all training points
Support Vector Machines

- Problem: Sometimes $|\mathcal{T}|$ is far too large
- Thus the number of constraints might make solving the quadratic programming problem very difficult
- Common technique: Sequential Minimal Optimization (SMO)
- Sparse: solution depends only on features in support vectors
Margin Infused Relaxed Algorithm (MIRA)

- Another option – maximize margin using an online algorithm
- Batch vs. Online
  - Batch – update parameters based on entire training set (SVM)
  - Online – update parameters based on a single training instance at a time (Perceptron)
- MIRA can be thought of as a *max-margin perceptron* or an *online SVM*
**MIRA**

Batch (SVMs):

\[
\min \frac{1}{2} ||w||^2
\]

such that:

\[w \cdot f(x_t, y_t) - w \cdot f(x_t, y') \geq 1\]

\(\forall(x_t, y_t) \in \mathcal{T} \text{ and } y' \in \bar{Y}_t\)

Online (MIRA):

Training data: \(\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\mathcal{T}}\)

1. \(w^{(0)} = 0; i = 0\)
2. for \(n : 1..N\)
3. for \(t : 1..\mathcal{T}\)
4. \(w^{(i+1)} = \arg \min_{w^*} ||w^* - w^{(i)}||\)
   such that:
   \[w \cdot f(x_t, y_t) - w \cdot f(x_t, y') \geq 1\]
   \(\forall y' \in \bar{Y}_t\)
5. \(i = i + 1\)
6. return \(w^i\)

- MIRA has much smaller optimizations with only \(|\bar{Y}_t|\)
  constraints
- Cost: sub-optimal optimization
Interim Summary

What we have covered

▶ Linear classifiers:
  ▶ Perceptron
  ▶ SVMs
  ▶ MIRA
▶ All are trained to minimize error
  ▶ With or without maximizing margin
  ▶ Online or batch

What is next

▶ Logistic Regression / Maximum Entropy
▶ Train linear classifiers to maximize likelihood
Logistic Regression / Maximum Entropy

Define a conditional probability:

\[ P(y|x) = \frac{e^{w \cdot f(x, y)}}{Z_x}, \quad \text{where} \quad Z_x = \sum_{y' \in Y} e^{w \cdot f(x, y')} \]

Note: still a linear classifier

\[
\arg\max_y P(y|x) = \arg\max_y \frac{e^{w \cdot f(x, y)}}{Z_x} = \arg\max_y e^{w \cdot f(x, y)} = \arg\max_y w \cdot f(x, y)
\]
Logistic Regression / Maximum Entropy

\[ P(y|x) = \frac{e^{w \cdot f(x,y)}}{Z_x} \]

- Q: How do we learn weights \( w \)?
- A: Set weights to maximize log-likelihood of training data:

\[
\begin{align*}
    w &= \arg \max_w \prod_t P(y_t|x_t) = \arg \max_w \sum_t \log P(y_t|x_t)
\end{align*}
\]

- In a nutshell we set the weights \( w \) so that we assign as much probability to the correct label \( y \) for each \( x \) in the training set.
Aside: Min error versus max log-likelihood

- Highly related but not identical

- Example: consider a training set $\mathcal{T}$ with 1001 points

  \[ 1000 \times (x_i, y = 0) = [-1, 1, 0, 0] \quad \text{for} \quad i = 1 \ldots 1000 \]
  \[ 1 \times (x_{1001}, y = 1) = [0, 0, 3, 1] \]

- Now consider $w = [-1, 0, 1, 0]$

- Error in this case is 0 – so $w$ minimizes error

  \[ [-1, 0, 1, 0] \cdot [-1, 1, 0, 0] = 1 > [-1, 0, 1, 0] \cdot [0, 0, -1, 1] = -1 \]
  \[ [-1, 0, 1, 0] \cdot [0, 0, 3, 1] = 3 > [-1, 0, 1, 0] \cdot [3, 1, 0, 0] = -3 \]

- However, log-likelihood = $-126.9$ (omit calculation)
Aside: Min error versus max log-likelihood

Highly related but not identical

Example: consider a training set $\mathcal{T}$ with 1001 points

\[
1000 \times (x_i, y = 0) = [-1, 1, 0, 0] \quad \text{for} \quad i = 1 \ldots 1000
\]
\[
1 \times (x_{1001}, y = 1) = [0, 0, 3, 1]
\]

Now consider $w = [-1, 7, 1, 0]$

Error in this case is 1 – so $w$ does not minimizes error

\[
[-1, 7, 1, 0] \cdot [-1, 1, 0, 0] = 8 > [-1, 7, 1, 0] \cdot [0, 0, -1, 1] = -1
\]
\[
[-1, 7, 1, 0] \cdot [0, 0, 3, 1] = 3 < [-1, 7, 1, 0] \cdot [3, 1, 0, 0] = 4
\]

However, log-likelihood = -1.4

Better log-likelihood and worse error
Aside: Min error versus max log-likelihood

- Max likelihood $\neq$ min error
- Max likelihood pushes as much probability on correct labeling of training instance
  - Even at the cost of mislabeling a few examples
- Min error forces all training instances to be correctly classified
- SVMs with slack variables – allows some examples to be classified wrong if resulting margin is improved on other examples
Aside: Max margin versus max log-likelihood

Let’s re-write the max likelihood objective function

\[
\mathbf{w} = \arg \max_{\mathbf{w}} \sum_t \log P(y_t|x_t)
\]

\[
= \arg \max_{\mathbf{w}} \sum_t \log \frac{e^{\mathbf{w} \cdot f(x, y)}}{\sum_{y' \in Y} e^{\mathbf{w} \cdot f(x, y')}}
\]

\[
= \arg \max_{\mathbf{w}} \sum_t \mathbf{w} \cdot f(x, y) - \log \sum_{y' \in Y} e^{\mathbf{w} \cdot f(x, y')}
\]

Pick \( \mathbf{w} \) to maximize score difference between correct labeling and every possible labeling.

Margin: maximize difference between correct and all incorrect

The above formulation is often referred to as the soft-margin
Logistic Regression

\[ P(y|x) = \frac{e^{w \cdot f(x,y)}}{Z_x}, \quad \text{where } Z_x = \sum_{y' \in \mathcal{Y}} e^{w \cdot f(x,y')} \]

\[ w = \arg \max_w \sum_t \log P(y_t|x_t) \]  (*)

The objective function (*) is concave
Therefore there is a global maximum
No closed form solution, but lots of numerical techniques
  - Gradient methods (gradient ascent, iterative scaling)
  - Newton methods (limited-memory quasi-newton)
Logistic Regression Summary

- Define conditional probability

\[ P(y|x) = \frac{e^{w \cdot f(x, y)}}{Z_x} \]

- Set weights to maximize log-likelihood of training data:

\[ w = \operatorname{arg\ max}_w \sum_t \log P(y_t|x_t) \]

- Can find the gradient and run gradient ascent (or any gradient-based optimization algorithm)

\[ \nabla F(w) = \left( \frac{\partial}{\partial w_0} F(w), \frac{\partial}{\partial w_1} F(w), \ldots, \frac{\partial}{\partial w_m} F(w) \right) \]

\[ \frac{\partial}{\partial w_i} F(w) = \sum_t f_i(x_t, y_t) - \sum_t \sum_{y' \in Y} P(y'|x_t) f_i(x_t, y') \]
Logistic Regression = Maximum Entropy

- Well known equivalence
- Max Ent: maximize entropy subject to constraints on features
  - Empirical feature counts must equal expected counts
- Quick intuition
  - Partial derivative in logistic regression

\[
\frac{\partial}{\partial w_i} F(w) = \sum_t f_i(x_t, y_t) - \sum_t \sum_{y' \in Y} P(y'|x_t) f_i(x_t, y')
\]

- First term is empirical feature counts and second term is expected counts
- Derivative set to zero maximizes function
- Therefore when both counts are equivalent, we optimize the logistic regression objective!
Linear Classification

- Basic form of (multiclass) classifier:

\[
y = \arg \max_y w \cdot f(x, y)
\]

- Different learning methods:
  - Perceptron – separate data (0-1 loss, online)
  - Support vector machine – maximize margin (hinge loss, batch)
  - Logistic regression – maximize likelihood (log loss, batch)

- All three methods are widely used in NLP
Appendix

Proofs and Derivations
**Perceptron Learning Algorithm**

Training data: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\mathcal{T}}$

1. $w^{(0)} = 0; \ i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. Let $y' = \arg \max_y w^{(i)} \cdot f(x_t, y)$
5. if $y' \neq y_t$
6. $w^{(i+1)} = w^{(i)} + f(x_t, y_t) - f(x_t, y')$
7. $i = i + 1$
8. return $w^i$

- $w^{(k-1)}$ are the weights before $k^{th}$ mistake
- Suppose $k^{th}$ mistake made at the $t^{th}$ example, $(x_t, y_t)$
- $y' = \arg \max_y w^{(k-1)} \cdot f(x_t, y)$
- $y' \neq y_t$
- $w^{(k)} = w^{(k-1)} + f(x_t, y_t) - f(x_t, y')$

Now: $u \cdot w^{(k)} = u \cdot w^{(k-1)} + u \cdot (f(x_t, y_t) - f(x_t, y')) \geq u \cdot w^{(k-1)} + \gamma$

Now: $w^{(0)} = 0$ and $u \cdot w^{(0)} = 0$, by induction on $k$, $u \cdot w^{(k)} \geq k \gamma$

Now: since $u \cdot w^{(k)} \leq \|u\| \times \|w^{(k)}\|$ and $\|u\| = 1$ then $\|w^{(k)}\| \geq k \gamma$

Now:

$$\|w^{(k)}\|^2 = \|w^{(k-1)}\|^2 + \|f(x_t, y_t) - f(x_t, y')\|^2 + 2w^{(k-1)} \cdot (f(x_t, y_t) - f(x_t, y'))$$

$$\|w^{(k)}\|^2 \leq \|w^{(k-1)}\|^2 + R^2$$

(since $R \geq \|f(x_t, y_t) - f(x_t, y')\|$)

and $w^{(k-1)} \cdot f(x_t, y_t) - w^{(k-1)} \cdot f(x_t, y') \leq 0$
Perceptron Learning Algorithm

- We have just shown that $\|w^{(k)}\| \geq k\gamma$ and 
  $\|w^{(k)}\|^2 \leq \|w^{(k-1)}\|^2 + R^2$

- By induction on $k$ and since $w^{(0)} = 0$ and $\|w^{(0)}\|^2 = 0$

  $\|w^{(k)}\|^2 \leq kR^2$

- Therefore,

  $k^2\gamma^2 \leq \|w^{(k)}\|^2 \leq kR^2$

- and solving for $k$

  $k \leq \frac{R^2}{\gamma^2}$

- Therefore the number of errors is bounded!
Gradient Ascent

- Let $F(w) = \sum_t \log \frac{e^{w \cdot f(x_t, y_t)}}{Z_x}$
- Want to find $\arg\max_w F(w)$
  - Set $w^0 = O^m$
  - Iterate until convergence

\[ w^i = w^{i-1} + \alpha \nabla F(w^{i-1}) \]

- $\alpha > 0$ and set so that $F(w^i) > F(w^{i-1})$
- $\nabla F(w)$ is gradient of $F$ w.r.t. $w$
  - A gradient is all partial derivatives over variables $w_i$
  - i.e., $\nabla F(w) = (\frac{\partial}{\partial w_0} F(w), \frac{\partial}{\partial w_1} F(w), \ldots, \frac{\partial}{\partial w_m} F(w))$

- Gradient ascent will always find $w$ to maximize $F$
The partial derivatives

- Need to find all partial derivatives \( \frac{\partial}{\partial w_i} F(w) \)

\[
F(w) = \sum_t \log P(y_t|x_t)
= \sum_t \log \frac{e^{w \cdot f(x_t, y_t)}}{\sum_{y' \in Y} e^{w \cdot f(x_t, y')}}
= \sum_t \log \frac{e^{\sum_j w_j \times f_j(x_t, y_t)}}{\sum_{y' \in Y} e^{\sum_j w_j \times f_j(x_t, y')}}
\]
Partial derivatives - some reminders

1. $\frac{\partial}{\partial x} \log F = \frac{1}{F} \frac{\partial}{\partial x} F$
   ▶ We always assume log is the natural logarithm $\log_e$

2. $\frac{\partial}{\partial x} e^F = e^F \frac{\partial}{\partial x} F$

3. $\frac{\partial}{\partial x} \sum_t F_t = \sum_t \frac{\partial}{\partial x} F_t$

4. $\frac{\partial}{\partial x} \frac{F}{G} = \frac{G \frac{\partial}{\partial x} F - F \frac{\partial}{\partial x} G}{G^2}$
Gradient Ascent for Logistic Regression

The partial derivatives

\[
\frac{\partial}{\partial w_i} F(w) = \frac{\partial}{\partial w_i} \sum_t \log \frac{e^{\sum_j w_j \times f_j(x_t, y_t)}}{\sum_{y' \in Y} e^{\sum_j w_j \times f_j(x_t, y')}}
\]

\[
= \sum_t \frac{\partial}{\partial w_i} \log \frac{e^{\sum_j w_j \times f_j(x_t, y_t)}}{\sum_{y' \in Y} e^{\sum_j w_j \times f_j(x_t, y')}}
\]

\[
= \sum_t \left( \frac{\sum_{y' \in Y} e^{\sum_j w_j \times f_j(x_t, y')}}{e^{\sum_j w_j \times f_j(x_t, y')}} \right) \left( \frac{\partial}{\partial w_i} \frac{e^{\sum_j w_j \times f_j(x_t, y_t)}}{\sum_{y' \in Y} e^{\sum_j w_j \times f_j(x_t, y')}} \right)
\]

\[
= \sum_t \left( \frac{Z_{x_t}}{e^{\sum_j w_j \times f_j(x_t, y_t)}} \right) \left( \frac{\partial}{\partial w_i} \frac{e^{\sum_j w_j \times f_j(x_t, y_t)}}{Z_{x_t}} \right)
\]
The partial derivatives

Now,

\[
\frac{\partial}{\partial w_i} \frac{e^{\sum_j w_j \times f_j(x_t, y_t)}}{Z_{x_t}} = \frac{Z_{x_t} \frac{\partial}{\partial w_i} e^{\sum_j w_j \times f_j(x_t, y_t)} - e^{\sum_j w_j \times f_j(x_t, y_t)} \frac{\partial}{\partial w_i} Z_{x_t}}{Z_{x_t}^2} \]

\[
= \frac{Z_{x_t} e^{\sum_j w_j \times f_j(x_t, y_t)} f_i(x_t, y_t) - e^{\sum_j w_j \times f_j(x_t, y_t)} \frac{\partial}{\partial w_i} Z_{x_t}}{Z_{x_t}^2} \]

\[
= \frac{\sum_j w_j \times f_j(x_t, y_t)}{Z_{x_t}^2} (Z_{x_t} f_i(x_t, y_t) - \frac{\partial}{\partial w_i} Z_{x_t}) \]

\[
= \frac{\sum_j w_j \times f_j(x_t, y_t)}{Z_{x_t}^2} (Z_{x_t} f_i(x_t, y_t) - \sum_{y' \in Y} e^{\sum_j w_j \times f_j(x_t, y')} f_i(x_t, y')) \]

because

\[
\frac{\partial}{\partial w_i} Z_{x_t} = \sum_{y' \in Y} e^{\sum_j w_j \times f_j(x_t, y')} = \sum_{y' \in Y} e^{\sum_j w_j \times f_j(x_t, y')} f_i(x_t, y') \]
The partial derivatives

From before,

\[ \frac{\partial}{\partial w_i} e^{\sum_j w_j \times f_j(x_t, y_t)} \frac{Z_{x_t}}{Z_{x_t}} = \frac{e^{\sum_j w_j \times f_j(x_t, y_t)}}{Z_{x_t}^2} \left( Z_{x_t} f_i(x_t, y_t) \right) \]

\[ - \sum_{y' \in \mathcal{Y}} e^{\sum_j w_j \times f_j(x_t, y')} f_i(x_t, y') \]

Sub this in,

\[ \frac{\partial}{\partial w_i} F(w) = \sum_t \left( \frac{Z_{x_t}}{e^{\sum_j w_j \times f_j(x_t, y_t)}} \right) \left( \frac{e^{\sum_j w_j \times f_j(x_t, y_t)}}{Z_{x_t}} \right) \left( \frac{\partial}{\partial w_i} \frac{e^{\sum_j w_j \times f_j(x_t, y_t)}}{Z_{x_t}} \right) \]

\[ = \sum_t \frac{1}{Z_{x_t}} \left( Z_{x_t} f_i(x_t, y_t) - \sum_{y' \in \mathcal{Y}} e^{\sum_j w_j \times f_j(x_t, y')} f_i(x_t, y') \right) \]

\[ = \sum_t f_i(x_t, y_t) - \sum_t \sum_{y' \in \mathcal{Y}} \frac{e^{\sum_j w_j \times f_j(x_t, y')}}{Z_{x_t}} f_i(x_t, y') \]

\[ = \sum_t f_i(x_t, y_t) - \sum_t \sum_{y' \in \mathcal{Y}} P(y'|x_t) f_i(x_t, y') \]
FINALLY!!!

After all that,

$$\frac{\partial}{\partial w_i} F(w) = \sum_t f_i(x_t, y_t) - \sum_t \sum_{y' \in \mathcal{Y}} P(y'|x_t)f_i(x_t, y')$$

And the gradient is:

$$\nabla F(w) = \left( \frac{\partial}{\partial w_0} F(w), \frac{\partial}{\partial w_1} F(w), \ldots, \frac{\partial}{\partial w_m} F(w) \right)$$

So we can now use gradient ascent to find $w$!!