Generalized Linear Classifiers in NLP

(or Discriminative Generalized Linear Feature-Based Classifiers)

Graduate School of Language Technology, Sweden 2009

Ryan McDonald

Google Inc., New York, USA
E-mail: ryanmcd@google.com

Generalized Linear Classifiers

- Go onto ACL Anthology
- Search for: “Naive Bayes”, “Maximum Entropy”, “Logistic Regression”, “SVM”, “Perceptron”
- Do the same on Google Scholar
  - “Maximum Entropy” & “NLP” 2660 hits, 141 before 2000
  - “SVM” & “NLP” 2210 hits, 16 before 2000
  - “Perceptron” & “NLP”, 947 hits, 118 before 2000
- All are examples of linear classifiers
- All have become important tools in any NLP/CL researchers tool-box in past 10 years

Attitudes

1. Treat classifiers as black-box in language systems
   - Fine for many problems
   - Separates out the language research from the machine learning research
   - Practical – many off the shelf algorithms available
2. Fuse classifiers with language systems
   - Can use our understanding of classifiers to tailor them to our needs
   - Optimizations (both computational and parameters)
   - But we need to understand how they work ... at least to some degree (*)
   - Can also tell us something about how humans manage language (see Walter's talk)

(*) What this course is about

Lecture Outline

- Preliminaries: input/output, features, etc.
- Linear Classifiers
  - Perceptron
  - Large-Margin Classifiers (SVMs, MIRA)
  - Logistic Regression (Maximum Entropy)
- Issues in parallelization
- Structured Learning with Linear Classifiers
  - Structured Perceptron
  - Conditional Random Fields
- Non-linear Classifiers with Kernels

Inputs and Outputs

- Input: $\boldsymbol{x} \in \mathcal{X}$
  - e.g., document or sentence with some words $\boldsymbol{x} = w_1 \dots w_n$, or a series of previous actions
- Output: $\boldsymbol{y} \in \mathcal{Y}$
  - e.g., parse tree, document class, part-of-speech tags, word-sense

- Input/Output pair: $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$
  - e.g., a document $\boldsymbol{x}$ and its label $\boldsymbol{y}$
  - Sometimes $\boldsymbol{x}$ is explicit in $\boldsymbol{y}$, e.g., a parse tree $\boldsymbol{y}$ will contain the sentence $\boldsymbol{x}$

General Goal

When given a new input $\boldsymbol{x}$ predict the correct output $\boldsymbol{y}$

But we need to formulate this computationally!

Feature Representations

- We assume a mapping from input-output pairs $(\boldsymbol{x}, \boldsymbol{y})$ to a high dimensional feature vector
  - $\mathbf{f}(\boldsymbol{x}, \boldsymbol{y}) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m$
- For some cases, i.e., binary classification $\mathcal{Y} = \{-1, +1\}$, we can map only from the input to the feature space
  - $\mathbf{f}(\boldsymbol{x}) : \mathcal{X} \rightarrow \mathbb{R}^m$
- However, most problems in NLP require more than two classes, so we focus on the multi-class case
- For any vector $\mathbf{v} \in \mathbb{R}^m$, let $\mathbf{v}_j$ be the j^{th} value

Examples

- $\boldsymbol{x}$ is a document and $\boldsymbol{y}$ is a label

$$\mathbf{f}_j(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains the word ``interest''} \\ & \text{and } \boldsymbol{y} = \text{``financial''} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{f}_j(\boldsymbol{x},\boldsymbol{y}) = \text{\% of words in } \boldsymbol{x} \text{ with punctuation and } \boldsymbol{y} = \text{``scientific''}$$

- $\boldsymbol{x}$ is a word and $\boldsymbol{y}$ is a part-of-speech tag

$$\mathbf{f}_j(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} = \text{ ``bank'' and } \boldsymbol{y} = \text{ Verb} \\ 0 & \text{otherwise} \end{cases}$$

Example 2

- $\boldsymbol{x}$ is a name, $\boldsymbol{y}$ is a label classifying the name

$$\mathbf{f}_0(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains “George”} \\ & \text{and } \boldsymbol{y} = \text{“Person”} \\ 0 & \text{otherwise} \end{cases} \qquad \mathbf{f}_4(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains “George”} \\ & \text{and } \boldsymbol{y} = \text{“Object”} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{f}_1(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains “Washington”} \\ & \text{and } \boldsymbol{y} = \text{“Person”} \\ 0 & \text{otherwise} \end{cases} \qquad \mathbf{f}_5(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains “Washington”} \\ & \text{and } \boldsymbol{y} = \text{“Object”} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{f}_2(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains “Bridge”} \\ & \text{and } \boldsymbol{y} = \text{“Person”} \\ 0 & \text{otherwise} \end{cases} \qquad \mathbf{f}_6(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains “Bridge”} \\ & \text{and } \boldsymbol{y} = \text{“Object”} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{f}_3(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains “General”} \\ & \text{and } \boldsymbol{y} = \text{“Person”} \\ 0 & \text{otherwise} \end{cases} \qquad \mathbf{f}_7(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains “General”} \\ & \text{and } \boldsymbol{y} = \text{“Object”} \\ 0 & \text{otherwise} \end{cases}$$

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

Block Feature Vectors

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

- Each equal size block of the feature vector corresponds to one label
- 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 $\mathbf{w} \in \mathbb{R}^m$ be a high dimensional weight vector
- If we assume that $\mathbf{w}$ is known, then we our classifier as
  - **Multiclass Classification**: $\mathcal{Y} = \{0, 1, \ldots, N\}$

$$
\begin{aligned}
\boldsymbol{y} &= \arg\max_{\boldsymbol{y}} \; \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}, \boldsymbol{y}) \\
&= \arg\max_{\boldsymbol{y}} \; \sum_{j=0}^{m} \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}
$$

  - **Binary Classification** just a special case of multiclass

Linear Classifiers - Bias Terms

- Often linear classifiers presented as

$$\boldsymbol{y} = \arg\max_{\boldsymbol{y}} \sum_{j=0}^{m} \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}, \boldsymbol{y}) + b_{\boldsymbol{y}}$$

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

$\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Person $\rightarrow \mathbf{f}(\boldsymbol{x}, \boldsymbol{y}) = [1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0]$

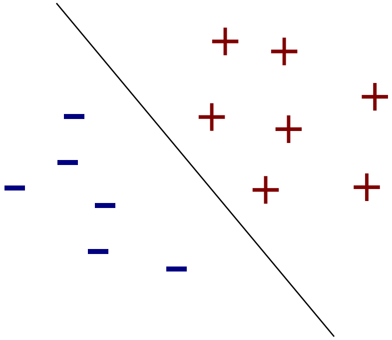
$\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Object $\rightarrow \mathbf{f}(\boldsymbol{x}, \boldsymbol{y}) = [0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1]$

$$\mathbf{f}_4(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \boldsymbol{y} = \text{“Person”} \\ 0 & \text{otherwise} \end{cases} \qquad \mathbf{f}_9(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \boldsymbol{y} = \text{“Object”} \\ 0 & \text{otherwise} \end{cases}$$

- $\mathbf{w}_4$ and $\mathbf{w}_9$ are now the bias terms for the labels

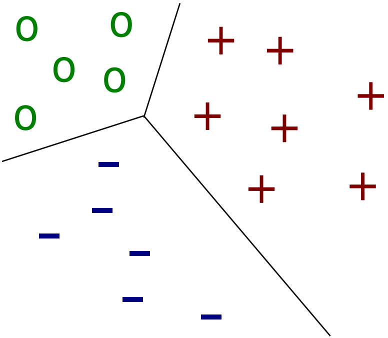
Binary Linear Classifier

Divides all points:



Multiclass Linear Classifier

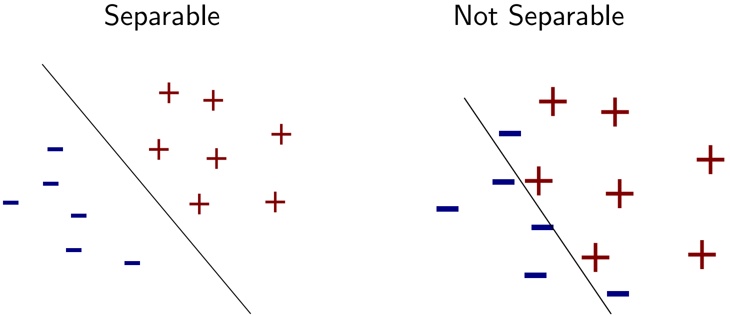
Defines regions of space:



- i.e., + are all points $(\boldsymbol{x}, \boldsymbol{y})$ where $+ = \arg\max_{\boldsymbol{y}} \; \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}, \boldsymbol{y})$

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 $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$
- 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

$$\mathbf{w} = \arg\min_{\mathbf{w}} \sum_t 1 - \mathbb{1}[\boldsymbol{y}_t = \arg\max_{\boldsymbol{y}} \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y})]$$

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

- 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: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

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

Perceptron: Separability and Margin

- Given an training instance $(\boldsymbol{x}_t, \boldsymbol{y}_t)$, define:
  - $\bar{\mathcal{Y}}_t = \mathcal{Y} - \{\boldsymbol{y}_t\}$
  - i.e., $\bar{\mathcal{Y}}_t$ is the set of incorrect labels for $\boldsymbol{x}_t$
- A training set $\mathcal{T}$ is separable with margin $\gamma > 0$ if there exists a vector $\mathbf{u}$ with $\|\mathbf{u}\| = 1$ such that:

$$\mathbf{u} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{u} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

  for all $\boldsymbol{y}' \in \bar{\mathcal{Y}}_t$ and $||\mathbf{u}|| = \sqrt{\sum_j \mathbf{u}_j^2}$
- **Assumption**: the training set is separable with margin γ

Perceptron: Main Theorem

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

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

  where $R \geq ||\mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')||$ for all $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$ and $\boldsymbol{y}' \in \bar{\mathcal{Y}}_t$
- Thus, after a finite number of training iterations, the error on the training set will converge to zero
- **Let's prove it!** (proof taken from Collins '02)

Perceptron Learning Algorithm

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

```
w^(0) = 0; i = 0
for n : 1..N
  for t : 1..T
    Let y' = arg max_{y'} w^(i) · f(x_t, y')
    if y' ≠ y_t
      w^(i+1) = w^(i) + f(x_t, y_t) − f(x_t, y')
      i = i + 1
return w^i
```

- $w^{(k-1)}$ are the weights before k^{th} mistake
- Suppose k^{th} mistake made at the t^{th} example, $(\boldsymbol{x}_t, \boldsymbol{y}_t)$
- $\boldsymbol{y}' = \arg\max_{\boldsymbol{y}'} \mathbf{w}^{(k-1)} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')$
- $\boldsymbol{y}' \neq \boldsymbol{y}_t$
- $w^{(k)} = \mathbf{w}^{(k-1)} + \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')$

- Now: $\mathbf{u} \cdot \mathbf{w}^{(k)} = \mathbf{u} \cdot \mathbf{w}^{(k-1)} + \mathbf{u} \cdot (\mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')) \geq \mathbf{u} \cdot \mathbf{w}^{(k-1)} + \gamma$
- Now: $\mathbf{w}^{(0)} = 0$ and $\mathbf{u} \cdot \mathbf{w}^{(0)} = 0$, by induction on k, $\mathbf{u} \cdot \mathbf{w}^{(k)} \geq k\gamma$
- Now: since $\mathbf{u} \cdot \mathbf{w}^{(k)} \leq ||\mathbf{u}|| \times ||\mathbf{w}^{(k)}||$ and $||\mathbf{u}|| = 1$ then $||\mathbf{w}^{(k)}|| \geq k\gamma$
- Now:

$$
\begin{aligned}
||\mathbf{w}^{(k)}||^2 &= ||\mathbf{w}^{(k-1)}||^2 + ||\mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')||^2 + 2\mathbf{w}^{(k-1)} \cdot (\mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')) \\
||\mathbf{w}^{(k)}||^2 &\leq ||\mathbf{w}^{(k-1)}||^2 + R^2 \\
&\quad (\text{since } R \geq ||\mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')|| \\
&\quad \text{and } \mathbf{w}^{(k-1)} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{w}^{(k-1)} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}') \leq 0)
\end{aligned}
$$

Perceptron Learning Algorithm

- We have just shown that $||\mathbf{w}^{(k)}|| \geq k\gamma$ and $||\mathbf{w}^{(k)}||^2 \leq ||\mathbf{w}^{(k-1)}||^2 + R^2$
- By induction on k and since $\mathbf{w}^{(0)} = 0$ and $||\mathbf{w}^{(0)}||^2 = 0$

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

- Therefore,

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

- and solving for k

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

- Therefore the number of errors is bounded!

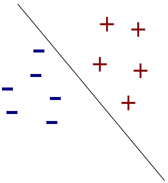
Perceptron Summary

- Learns a linear classifier that minimizes error
- Guaranteed to find a **w** in a finite amount of time
- Perceptron is an example of an Online Learning Algorithm
  - **w** is updated based on a single training instance in isolation

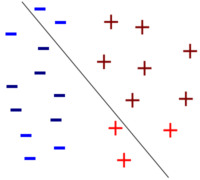
$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')$$

Margin

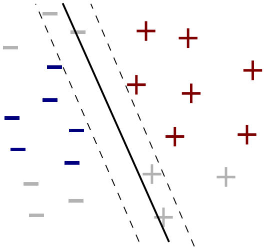
Training

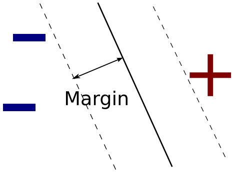


Testing



Denote the value of the margin by γ





Maximizing Margin

- For a training set $\mathcal{T}$
- Margin of a weight vector **w** is smallest γ such that

$$\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

- for every training instance $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$, $\boldsymbol{y}' \in \bar{\mathcal{Y}}_t$

Maximizing Margin

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

$$\epsilon \propto \frac{R^2}{\gamma^2 \times |\mathcal{T}|}$$

- **Perceptron**: we have shown that:
  - If a training set is separable by some margin, the perceptron will find a $\mathbf{w}$ that separates the data
  - However, the perceptron does not pick $\mathbf{w}$ to maximize the margin!

Maximizing Margin

Let $\gamma > 0$

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

such that:

$$\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

$$\text{and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

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

$$\beta(\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')) \geq \beta\gamma, \text{ for some } \beta \geq 1$$

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

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

such that:

$$\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

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

=

Min Norm:

$$\min_{\mathbf{w}} \quad \frac{1}{2} ||\mathbf{w}||^2$$

such that:

$$\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}') \geq 1$$

$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

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

- Instead of fixing $||\mathbf{w}||$ we fix the margin $\gamma = 1$
- Technically $\gamma \propto 1/||\mathbf{w}||$

Support Vector Machines

$$\min \frac{1}{2}||\mathbf{w}||^2$$

such that:

$$\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}') \geq 1$$

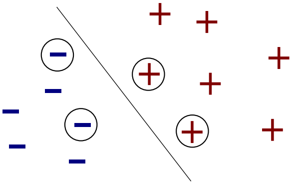
$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$$

$$\text{and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

- 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
- Most 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 (e.g., SVMs)
  - Online – update parameters based on a single training instance at a time (e.g., Perceptron)
- MIRA can be thought of as a *max-margin perceptron* or an *online SVM*

MIRA

Batch (SVMs):

$$\min \frac{1}{2}||\mathbf{w}||^2$$

such that:

$$\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{w}\cdot\mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}') \geq 1$$

$$\forall(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T} \text{ and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

Online (MIRA):

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

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

- MIRA has much smaller optimizations with only $|\bar{\mathcal{Y}}_t|$ constraints
- Cost: sub-optimal optimization

Summary

What we have covered

- Feature-based representations
- Linear Classifiers
  - Perceptron
  - Large-Margin – SVMs (batch) and MIRA (online)

What is next

- Logistic Regression / Maximum Entropy
- Issues in parallelization
- Structured Learning
- Non-linear classifiers

Logistic Regression / Maximum Entropy

Define a conditional probability:

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y})}}{Z_{\boldsymbol{x}}}, \qquad \text{where } Z_{\boldsymbol{x}} = \sum_{\boldsymbol{y}'\in\mathcal{Y}} e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y}')}$$

Note: still a linear classifier

$$\begin{aligned}\arg\max_{\boldsymbol{y}} \; P(\boldsymbol{y}|\boldsymbol{x}) &= \arg\max_{\boldsymbol{y}} \; \frac{e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y})}}{Z_{\boldsymbol{x}}} \\ &= \arg\max_{\boldsymbol{y}} \; e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y})} \\ &= \arg\max_{\boldsymbol{y}} \; \mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y})\end{aligned}$$

Logistic Regression / Maximum Entropy

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y})}}{Z_{\boldsymbol{x}}}$$

- Q: How do we learn weights **w**
- A: Set weights to maximize log-likelihood of training data:

$$\mathbf{w} = \arg\max_{\mathbf{w}} \prod_t P(\boldsymbol{y}_t|\boldsymbol{x}_t) = \arg\max_{\mathbf{w}} \sum_t \log P(\boldsymbol{y}_t|\boldsymbol{x}_t)$$

- In a nut shell we set the weights **w** so that we assign as much probability to the correct label $\boldsymbol{y}$ for each $\boldsymbol{x}$ 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

$$1000 \times (\boldsymbol{x}_i, \boldsymbol{y} = 0) = [-1, 1, 0, 0] \quad \text{for} \quad i = 1 \ldots 1000$$
$$1 \times (\boldsymbol{x}_{1001}, \boldsymbol{y} = 1) = [0, 0, 3, 1]$$

- Now consider $\mathbf{w} = [-1, 0, 1, 0]$
- Error in this case is 0 – so $\mathbf{w}$ 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

$$1000 \times (\boldsymbol{x}_i, \boldsymbol{y} = 0) = [-1, 1, 0, 0] \quad \text{for} \quad i = 1 \ldots 1000$$
$$1 \times (\boldsymbol{x}_{1001}, \boldsymbol{y} = 1) = [0, 0, 3, 1]$$

- Now consider $\mathbf{w} = [-1, 7, 1, 0]$
- Error in this case is 1 – so **w** does not minimizes error

$$[-1, 7, 1, 0] \cdot [-1, 1, 0, 0] = 8 > [-1, 7, 1, 0] \cdot [-1, 1, 0, 0] = -1$$

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

- 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

$$
\begin{aligned}
\mathbf{w} &= \arg\max_{\mathbf{w}} \sum_t \log P(\boldsymbol{y}_t|\boldsymbol{x}_t) \\
&= \arg\max_{\mathbf{w}} \sum_t \log \frac{e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y})}}{\sum_{\boldsymbol{y}'\in\mathcal{Y}} e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y}')}} \\
&= \arg\max_{\mathbf{w}} \sum_t \mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y}) - \log \sum_{\boldsymbol{y}'\in\mathcal{Y}} e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y}')}
\end{aligned}
$$

- Pick **w** to maximize the score difference between the correct labeling and every possible labeling
- Margin: maximize the difference between the correct and all incorrect
- The above formulation is often referred to as the soft-margin

Logistic Regression

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y})}}{Z_{\boldsymbol{x}}}, \qquad \text{where } Z_{\boldsymbol{x}} = \sum_{\boldsymbol{y}'\in\mathcal{Y}} e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y}')}$$

$$\mathbf{w} = \arg\max_{\mathbf{w}} \sum_t \log P(\boldsymbol{y}_t|\boldsymbol{x}_t) \ (*)$$

- The objective function (*) is concave (take the 2nd derivative)
- Therefore there is a global maximum
- No closed form solution, but lots of numerical techniques
  - Gradient methods (gradient ascent, conjugate gradient, iterative scaling)
  - Newton methods (limited-memory quasi-newton)

Gradient Ascent

- Let $F(\mathbf{w}) = \sum_t \log \frac{e^{\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z_{\boldsymbol{x}}}$
- Want to find $\arg\max_{\mathbf{w}} F(\mathbf{w})$
  - Set $\mathbf{w}^0 = O^m$
  - Iterate until convergence

$$\mathbf{w}^i = \mathbf{w}^{i-1} + \alpha \nabla F(\mathbf{w}^{i-1})$$

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

The partial derivatives

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

$$
\begin{aligned}
F(\mathbf{w}) &= \sum_t \log P(\boldsymbol{y}_t | \boldsymbol{x}_t) \\
&= \sum_t \log \frac{e^{\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{\sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')}} \\
&= \sum_t \log \frac{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{\sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}')}}
\end{aligned}
$$

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

The partial derivatives

$$
\begin{aligned}
\frac{\partial}{\partial w_i} F(\mathbf{w}) &= \frac{\partial}{\partial w_i} \sum_t \log \frac{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{\sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}')}} \\
&= \sum_t \frac{\partial}{\partial w_i} \log \frac{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{\sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}')}} \\
&= \sum_t \left(\frac{\sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}')}}{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}} \right) \left(\frac{\partial}{\partial w_i} \frac{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{\sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_{w_j} w_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}')}} \right) \\
&= \sum_t \left(\frac{Z_{\boldsymbol{x}_t}}{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}} \right) \left(\frac{\partial}{\partial w_i} \frac{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}} \right)
\end{aligned}
$$

The partial derivatives

Now,

$$
\begin{aligned}
\frac{\partial}{\partial w_i} \frac{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}} &= \frac{Z_{\boldsymbol{x}_t} \frac{\partial}{\partial w_i} e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)} - e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)} \frac{\partial}{\partial w_i} Z_{\boldsymbol{x}_t}}{Z^2_{\boldsymbol{x}_t}} \\
&= \frac{Z_{\boldsymbol{x}_t} e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)} \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)} \frac{\partial}{\partial w_i} Z_{\boldsymbol{x}_t}}{Z^2_{\boldsymbol{x}_t}} \\
&= \frac{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z^2_{\boldsymbol{x}_t}} \left(Z_{\boldsymbol{x}_t} \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \frac{\partial}{\partial w_i} Z_{\boldsymbol{x}_t}\right) \\
&= \frac{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z^2_{\boldsymbol{x}_t}} \Big(Z_{\boldsymbol{x}_t} \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) \\
&\qquad - \sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}')} \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}')\Big)
\end{aligned}
$$

because

$$
\frac{\partial}{\partial w_i} Z_{\boldsymbol{x}_t} = \frac{\partial}{\partial w_i} \sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}')} = \sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}')} \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}')
$$

The partial derivatives

From before,

$$\frac{\partial}{\partial w_i} \frac{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}} = \frac{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z^2_{\boldsymbol{x}_t}} (Z_{\boldsymbol{x}_t} \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}')} \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}'))$$

Sub this in,

$$\begin{aligned}
\frac{\partial}{\partial w_i} F(\mathbf{w}) &= \sum_t \left(\frac{Z_{\boldsymbol{x}_t}}{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}} \right) \left(\frac{\partial}{\partial w_i} \frac{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}} \right) \\
&= \sum_t \frac{1}{Z_{x_t}} (Z_{\boldsymbol{x}_t} \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}')} \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}'))) \\
&= \sum_t \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_t \sum_{\boldsymbol{y}' \in \mathcal{Y}} \frac{e^{\sum_j \mathbf{w}_j \times \mathbf{f}_j(\boldsymbol{x}_t, \boldsymbol{y}')}}{Z_{x_t}} \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}') \\
&= \sum_t \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_t \sum_{\boldsymbol{y}' \in \mathcal{Y}} P(\boldsymbol{y}' | \boldsymbol{x}_t) \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}')
\end{aligned}$$

FINALLY!!!

- After all that,

$$\frac{\partial}{\partial w_i} F(\mathbf{w}) \quad = \quad \sum_t \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_t \sum_{\boldsymbol{y}' \in \mathcal{Y}} P(\boldsymbol{y}' | \boldsymbol{x}_t) \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}')$$

- And the gradient is:

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

- So we can now use gradient assent to find **w**!!

Logistic Regression Summary

- Define conditional probability

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y})}}{Z_{\boldsymbol{x}}}$$

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

$$\mathbf{w} = \arg\max_{\mathbf{w}} \sum_t \log P(\boldsymbol{y}_t|\boldsymbol{x}_t)$$

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

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

$$\frac{\partial}{\partial w_i}F(\mathbf{w}) = \sum_t \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_t \sum_{\boldsymbol{y}'\in\mathcal{Y}} P(\boldsymbol{y}'|\boldsymbol{x}_t)\mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}')$$

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

$$\frac{\partial}{\partial w_i} F(\mathbf{w}) = \sum_t \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_t \sum_{\boldsymbol{y}' \in \mathcal{Y}} P(\boldsymbol{y}'|\boldsymbol{x}_t)\mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}')$$

  - 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!

Online Logistic Regression??

- Stochastic Gradient Descent (SGD)
  - Set $\mathbf{w}^0 = O^m$
  - Iterate until convergence
    - Randomly select $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$ // often sequential

$$\mathbf{w}^i = \mathbf{w}^{i-1} + \alpha \nabla F_t(\mathbf{w}^{i-1})$$

- ... well in our case it is an ascent (could just negate things)
- $\nabla F_t(\mathbf{w}^{i-1})$ is the gradient with respect to $(\boldsymbol{x}_t, \boldsymbol{y}_t)$

$$\frac{\partial}{\partial w_i} F_t(\mathbf{w}) = \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_{\boldsymbol{y}' \in \mathcal{Y}} P(\boldsymbol{y}'|\boldsymbol{x}_t)\mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}')$$

- Guaranteed to converge and is fast in practice [Zhang 2004]

Aside: Discriminative versus Generative

- Logistic Regression, Perceptron, MIRA, and SVMs are all discriminative models
- A discriminative model sets it parameters to optimize some notion of prediction
  - Perceptron/SVMs – min error
  - Logistic Regression – max likelihood of conditional distribution
    - The conditional distribution is used for prediction
- Generative models attempt to explain the input as well
  - e.g., Naive Bayes maximizes the likelihood of the joint distribution $P(\boldsymbol{x}, \boldsymbol{y})$
- This course is really about discriminative linear classifiers

Issues in Parallelization

- What if $\mathcal{T}$ is enormous? Can't even fit it into memory?
- Examples:
  - All pages/images on the web
  - Query logs of a search engine
  - All patient records in a health-care system
- Can use online algorithms
  - It may take long to see all interesting examples
  - All examples may not exist in same location physically
- Can we parallelize learning? Yes!

Parallel Logistic Regression / Gradient Ascent

- Core computation for gradient ascent

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

$$\frac{\partial}{\partial w_i} F(\mathbf{w}) = \sum_t \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_t \sum_{\boldsymbol{y}' \in \mathcal{Y}} P(\boldsymbol{y}'|\boldsymbol{x}_t)\mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}')$$

- Note that each $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ independently contributes to the calculation
- If we have P machines, put $|\mathcal{T}|/P$ training instances on each machine so that $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \ldots \cup \mathcal{T}_P$
- Compute above values on each machine and send to master
- On master machine, sum up gradients and do gradient ascent update

Parallel Logistic Regression / Gradient Ascent

- Algorithm:
  - Set $\mathbf{w}^0 = O^m$
  - Iterate until convergence
    - Compute $\nabla F_p(\mathbf{w}^{i-1})$ in parallel on P machines
    - $\nabla F(\mathbf{w}^{i-1}) = \sum_p \nabla F_p(\mathbf{w}^{i-1})$
    - $\mathbf{w}^i = \mathbf{w}^{i-1} + \alpha \nabla F(\mathbf{w}^{i-1})$
- Where $\nabla F_p(\mathbf{w}^{i-1})$ is the gradient of the training instances on machine p, e.g.,

$$\frac{\partial}{\partial w_i} F_p(\mathbf{w}) = \sum_{t \in \mathcal{T}_p} \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_{t \in \mathcal{T}_p} \sum_{\boldsymbol{y}' \in \mathcal{Y}} P(\boldsymbol{y}' | \boldsymbol{x}_t) \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}')$$

Parallelization through Averaging

- Again, we have P machines and $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \ldots \cup \mathcal{T}_P$
  - Let $\mathbf{w}_p$ be the weight vector if we just trained on $\mathcal{T}_p$
  - Let $\mathbf{w} = \frac{1}{P} \sum_p \mathbf{w}_p$
- This is called parameter/weight averaging
- Advantages: simple and very resource efficient (wrt network bandwidth – no passing around gradients)
- Disadvantages: sub optimal, unlike parallel gradient ascent
- Does it work?

[Mann et al. 2009]

- Let $\mathbf{w}$ be the weight vector learned using gradient ascent
- Let $\mathbf{w}_{\text{avg}}$ be the weight vector learned by averaging
- If algorithm is stable with respect to $\mathbf{w}$, then with high probability:

$$\|\mathbf{w} - \mathbf{w}_{\text{avg}}\| \leq O(\frac{1}{\sqrt{|\mathcal{T}|}})$$

- I.e., difference shrinks as training data increases
- Stable algorithms: Logistic regression, SVMs, others??
- Stability is beyond scope of course
- See [Mann et al. 2009], which also has experimental study

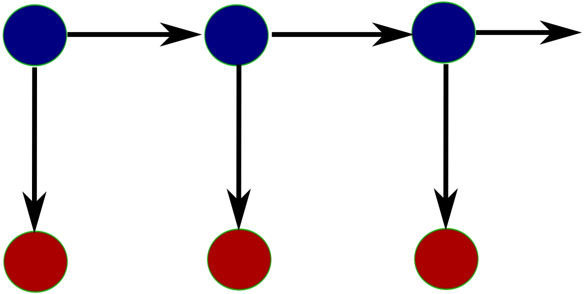
Parallel Wrap-up

- Many learning algorithms can be parallelized
- Logistic Regression and other gradient-based algorithms are naturally paralleled without any heuristics
- Parameter averaging an easy solution that is efficient and works for all algorithms
  - *Stable* algorithms have some optimal bound guarantees
- See [Chu et al. 2007] for a nice overview of parallel ML

Structured Learning

- Sometimes our output space $\mathcal{Y}$ is not simply a category
- Examples:
  - **Parsing**: for a sentence x, $\mathcal{Y}$ is the set of possible parse trees
  - **Sequence tagging**: for a sentence x, $\mathcal{Y}$ is the set of possible tag sequences, e.g., part-of-speech tags, named-entity tags
  - **Machine translation**: for a source sentence x, $\mathcal{Y}$ is the set of possible target language sentences
- Can't we just use our multiclass learning algorithms?
- In all the cases, the size of the set $\mathcal{Y}$ is exponential in the length of the input x
- It is often non-trivial to run learning algorithms in such cases

Hidden Markov Models



- Generative Model – maximizes likelihood of $P(\boldsymbol{x}, \boldsymbol{y})$
- We are looking at discriminative version of these
  - Not just for sequences, though that will be the running example

Structured Learning

- Sometimes our output space $\mathcal{Y}$ is not simply a category
- Can't we just use our multiclass learning algorithms?
- In all the cases, the size of the set $\mathcal{Y}$ is exponential in the length of the input $\boldsymbol{x}$
- It is often non-trivial to solve our learning algorithms in such cases

Perceptron

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

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

(**) Solving the argmax requires a search over an exponential space of outputs!

Large-Margin Classifiers

Batch (SVMs):

$$\min \frac{1}{2}||\mathbf{w}||^2$$

such that:

$$\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x}_t,\boldsymbol{y}_t)-\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x}_t,\boldsymbol{y}') \geq 1$$

$\forall(\boldsymbol{x}_t,\boldsymbol{y}_t) \in \mathcal{T}$ and $\boldsymbol{y}' \in \bar{\mathcal{Y}}_t$ (**)

Online (MIRA):

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t,\boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

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

(**) There are exponential constraints in the size of each input!!

Factor the Feature Representations

- We can make an assumption that our feature representations factor relative to the output
- Example:
  - Context Free Parsing:

$$\mathbf{f}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{A \rightarrow BC \in \boldsymbol{y}} \mathbf{f}(\boldsymbol{x}, A \rightarrow BC)$$

  - Sequence Analysis – Markov Assumptions:

$$\mathbf{f}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{|\boldsymbol{y}|} \mathbf{f}(\boldsymbol{x}, y_{i-1}, y_i)$$

- These kinds of factorizations allow us to run algorithms like CKY and Viterbi to compute the argmax function

Example – Sequence Labeling

- Many NLP problems can be cast in this light
  - Part-of-speech tagging
  - Named-entity extraction
  - Semantic role labeling
  - ...
- Input: $\boldsymbol{x} = x_0 x_1 \ldots x_n$
- Output: $\boldsymbol{y} = y_0 y_1 \ldots y_n$
- Each $y_i \in \mathcal{Y}_{\text{atom}}$ – which is small
- Each $\boldsymbol{y} \in \mathcal{Y} = \mathcal{Y}_{\text{atom}}^n$ – which is large
- Example: part-of-speech tagging – $\mathcal{Y}_{\text{atom}}$ is set of tags

$\boldsymbol{x}$	$=$	John	saw	Mary	with	the	telescope
$\boldsymbol{y}$	$=$	noun	verb	noun	preposition	article	noun

Sequence Labeling – Output Interaction

$\boldsymbol{x}$	$=$	John	saw	Mary	with	the	telescope
$\boldsymbol{y}$	$=$	noun	verb	noun	preposition	article	noun

- Why not just break up sequence into a set of multi-class predictions?
- Because there are interactions between neighbouring tags
  - What tag does "saw" have?
  - What if I told you the previous tag was *article*?
  - What if it was *noun*?

Sequence Labeling – Markov Factorization

$\boldsymbol{x}$	=	John	saw	Mary	with	the	telescope
$\boldsymbol{y}$	=	noun	verb	noun	preposition	article	noun

- Markov factorization – factor by adjacent labels
- First-order (like HMMs)

$$\mathbf{f}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{|\boldsymbol{y}|} \mathbf{f}(\boldsymbol{x}, y_{i-1}, y_i)$$

- kth-order

$$\mathbf{f}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=k}^{|\boldsymbol{y}|} \mathbf{f}(\boldsymbol{x}, y_{i-k}, \ldots, y_{i-1}, y_i)$$

Sequence Labeling – Features

$\boldsymbol{x}$	=	John	saw	Mary	with	the	telescope
$\boldsymbol{y}$	=	noun	verb	noun	preposition	article	noun

- First-order

$$\mathbf{f}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{|\boldsymbol{y}|} \mathbf{f}(\boldsymbol{x}, y_{i-1}, y_i)$$

- $\mathbf{f}(\boldsymbol{x}, y_{i-1}, y_i)$ is any feature of the input & two adjacent labels

$$\mathbf{f}_j(\boldsymbol{x}, y_{i-1}, y_i) = \begin{cases} 1 & \text{if } x_i = \text{“saw”} \\ & \text{and } y_{i-1} = \text{noun and } y_i = \text{verb} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{f}_{j'}(\boldsymbol{x}, y_{i-1}, y_i) = \begin{cases} 1 & \text{if } x_i = \text{“saw”} \\ & \text{and } y_{i-1} = \text{article and } y_i = \text{verb} \\ 0 & \text{otherwise} \end{cases}$$

- $\mathbf{w}_j$ should get high weight and $\mathbf{w}_{j'}$ should get low weight

Sequence Labeling - Inference

- How does factorization effect inference?

$$
\begin{aligned}
\boldsymbol{y} &= \arg\max_{\boldsymbol{y}} \ \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}, \boldsymbol{y}) \\
&= \arg\max_{\boldsymbol{y}} \ \mathbf{w} \cdot \sum_{i=1}^{|\boldsymbol{y}|} \mathbf{f}(\boldsymbol{x}, y_{i-1}, y_i) \\
&= \arg\max_{\boldsymbol{y}} \ \sum_{i=1}^{|\boldsymbol{y}|} \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}, y_{i-1}, y_i)
\end{aligned}
$$

- Can use the Viterbi algorithm

Sequence Labeling – Viterbi Algorithm

- Let $\alpha_{y,i}$ be the score of the best labeling
  - Of the sequence $x_0 x_1 \ldots x_i$
  - Where $y_i = y$
- Let's say we know α, then
  - $\max_y \alpha_{y,n}$ is the score of the best labeling of the sequence
- $\alpha_{y,i}$ can be calculate with the following recursion

$$\alpha_{y,0} = 0.0 \quad \forall y \in \mathcal{Y}_{\text{atom}}$$

$$\alpha_{y,i} = \max_{y*} \ \alpha_{y*,i-1} + \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}, y*, y)$$

Sequence Labeling - Back-pointers

- But that only tells us what the best score is
- Let $\beta_{y,i}$ be the i-1st label in the best labeling
  - Of the sequence $x_0 x_1 \ldots x_i$
  - Where $y_i = y$
- $\beta_{y,i}$ can be calculate with the following recursion

$$\beta_{y,0} = \text{nil} \quad \forall y \in \mathcal{Y}_{\text{atom}}$$

$$\beta_{y,i} = \arg\max_{y*} \; \alpha_{y*,i-1} + \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}, y*, y)$$

- The last label in the best sequence is $y_n = \arg\max_y \; \beta_{y,n}$
- And the second-to-last label is $y_{n-1} \arg\max_y \; \beta_{y_n,n-1}$ ...
- ... $y_0 = \arg\max_y \; \beta_{y_1,1}$

Structured Learning

- We know we can solve the inference problem
  - At least for sequence labeling
  - But for many other problems where one can factor features appropriately
- How does this change learning ..
  - for the perceptron algorithm?
  - for SVMs?
  - for Logistic Regression?

Structured Perceptron

- Exactly like original perceptron
- Except now the argmax function uses factored features
  - Which we can solve with algorithms like the Viterbi algorithm
- All of the original analysis carries over!!

$\mathbf{w}^{(0)} = 0;\ i = 0$
for $n : 1..N$
  for $t : 1..T$
    Let $\boldsymbol{y}' = \arg\max_{\boldsymbol{y}'} \mathbf{w}^{(i)} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')$ (**)
    if $\boldsymbol{y}' \neq \boldsymbol{y}_t$
      $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}')$
      $i = i + 1$
return $\mathbf{w}^{i}$

(**) Solve the argmax with Viterbi for sequence problems!

Structured SVMs

$$\min \frac{1}{2}||\mathbf{w}||^2$$

such that:

$$\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \mathcal{L}(y_t, y')$$

$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T} \text{ and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t \text{ (**)}$$

- Still have an exponential # of constraints
- Feature factorizations also allow for solutions
  - Maximum Margin Markov Networks (Taskar et al. '03)
  - Structured SVMs (Tsochantaridis et al. '04)
- **Note:** Old fixed margin of 1 is now a fixed loss $\mathcal{L}(y_t, y')$ between two structured outputs

Conditional Random Fields

- What about a structured logistic regression / maximum entropy
- Such a thing exists – Conditional Random Fields (CRFs)
- Let's again consider the sequential case with 1^{st} order factorization
- Inference is identical to the structured perceptron – use Viterbi

$$
\begin{aligned}
\arg\max_{\boldsymbol{y}} \; P(\boldsymbol{y}|\boldsymbol{x}) &= \arg\max_{\boldsymbol{y}} \frac{e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y})}}{Z_{\boldsymbol{x}}} \\
&= \arg\max_{\boldsymbol{y}} e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y})} \\
&= \arg\max_{\boldsymbol{y}} \mathbf{w}\cdot\mathbf{f}(\boldsymbol{x},\boldsymbol{y}) \\
&= \arg\max_{\boldsymbol{y}} \sum_{i=1} \mathbf{w}\cdot\mathbf{f}(\boldsymbol{x}, y_{i-1}, y_i)
\end{aligned}
$$

Conditional Random Fields

- However, learning does change
- Reminder: pick $\mathbf{w}$ to maximize log-likelihood of training data:

$$\mathbf{w} = \arg\max_{\mathbf{w}} \sum_t \log P(\boldsymbol{y}_t | \boldsymbol{x}_t)$$

- Take gradient and use gradient ascent

$$\frac{\partial}{\partial w_i} F(\mathbf{w}) \quad = \quad \sum_t \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_t \sum_{\boldsymbol{y}' \in \mathcal{Y}} P(\boldsymbol{y}' | \boldsymbol{x}_t) \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}')$$

- And the gradient is:

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

Conditional Random Fields

- Problem: sum over output space $\mathcal{Y}$

$$
\begin{aligned}
\frac{\partial}{\partial w_i} F(\mathbf{w}) &= \sum_t \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_t \sum_{\boldsymbol{y}' \in \mathcal{Y}} P(\boldsymbol{y}'|\boldsymbol{x}_t) \mathbf{f}_i(\boldsymbol{x}_t, \boldsymbol{y}') \\
&= \sum_t \sum_{j=1} \mathbf{f}_i(\boldsymbol{x}_t, y_{t,j-1}, y_{t,j}) - \sum_t \sum_{\boldsymbol{y}' \in \mathcal{Y}} \sum_{j=1} P(\boldsymbol{y}'|\boldsymbol{x}_t) \mathbf{f}_i(\boldsymbol{x}_t, y'_{j-1}, y'_j)
\end{aligned}
$$

- Can easily calculate first term – just empirical counts
- What about the second term?

Conditional Random Fields

- Problem: sum over output space $\mathcal{Y}$

$$\sum_{t} \sum_{\boldsymbol{y}' \in \mathcal{Y}} \sum_{j=1} P(\boldsymbol{y}' | \boldsymbol{x}_t) \mathbf{f}_i(\boldsymbol{x}_t, y'_{j-1}, y'_j)$$

- We need to show we can compute it for arbitrary $\boldsymbol{x}_t$

$$\sum_{\boldsymbol{y}' \in \mathcal{Y}} \sum_{j=1} P(\boldsymbol{y}' | \boldsymbol{x}_t) \mathbf{f}_i(\boldsymbol{x}_t, y'_{j-1}, y'_j)$$

- Solution: the forward-backward algorithm

Forward Algorithm

- Let α_u^m be the forward scores
- Let $|\boldsymbol{x}_t| = n$
- α_u^m is the sum over all labelings of $x_0 \dots x_m$ such that $y'_m = u$

$$\alpha_u^m = \sum_{|\boldsymbol{y}'|=m,\ y'_m=u} e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x}_t,\boldsymbol{y}')}$$

$$= \sum_{|\boldsymbol{y}'|=m\ y'_m=u} e^{\sum_{j=1} \mathbf{w}\cdot\mathbf{f}(\boldsymbol{x}_t,y_{j-1},y_j)}$$

- i.e., the sum of all labelings of length m, ending at position m with label u
- Note then that

$$Z_{\boldsymbol{x}_t} = \sum_{\boldsymbol{y}'} e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x}_t,\boldsymbol{y}')} = \sum_u \alpha_u^n$$

Forward Algorithm

- We can fill in α as follows:

$$\begin{aligned} \alpha_u^0 &= 1.0 \quad \forall u \\ \alpha_u^m &= \sum_v \alpha_v^{m-1} \times e^{\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, v, u)} \end{aligned}$$

Backward Algorithm

- Let β_u^m be the symmetric backward scores
- i.e., the sum over all labelings of $x_m \ldots x_n$ such that $x_m = u$
- We can fill in β as follows:

$$\begin{aligned} \beta_u^n &= 1.0 \quad \forall u \\ \beta_u^m &= \sum_v \beta_v^{m+1} \times e^{\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}_t, u, v)} \end{aligned}$$

- Note: β is overloaded – different from back-pointers

Conditional Random Fields

- Let's show we can compute it for arbitrary $\boldsymbol{x}_t$

$$\sum_{\boldsymbol{y}' \in \mathcal{Y}} \sum_{j=1} P(\boldsymbol{y}'|\boldsymbol{x}_t)\mathbf{f}_i(\boldsymbol{x}_t, y'_{j-1}, y'_j)$$

- So we can re-write it as:

$$\sum_{j=1} \frac{\alpha^{j-1}_{y'_{j-1}} \times e^{\mathbf{w}\cdot\mathbf{f}(\boldsymbol{x}_t, y'_{j-1}, y'_j)} \times \beta^{j}_{y_j}}{Z_{\boldsymbol{x}_t}} \mathbf{f}_i(\boldsymbol{x}_t, y'_{j-1}, y'_j)$$

- Forward-backward can calculate partial derivatives efficiently

Conditional Random Fields Summary

- Inference: Viterbi
- Learning: Use the forward-backward algorithm
- What about not sequential problems
  - Context-Free parsing – can use inside-outside algorithm
  - General problems – message passing & belief propagation
- Great tutorial by [Sutton and McCallum 2006]

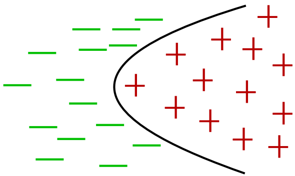
Structured Learning Summary

- Can't use multiclass algorithms – search space too large
- Solution: factor representations
- Can allow for efficient inference and learning
  - Showed for sequence learning: Viterbi + forward-backward
  - But also true for other structures
    - CFG parsing: CKY + inside-outside
    - Dependency Parsing: Spanning tree / Eisner algorithm
    - General graphs: junction-tree and message passing

- End of linear classifiers!!
- Brief look at non-linear classification ...

Non-Linear Classifiers

- Some data sets require more than a linear classifier to be correctly modeled
- A lot of models out there
  - **K-Nearest Neighbours** (see Walter's lecture)
  - Decision Trees
  - **Kernels**
  - Neural Networks



Kernels

- A kernel is a similarity function between two points that is symmetric and positive semi-definite, which we denote by:

$$\phi(\boldsymbol{x}_t, \boldsymbol{x}_r) \in \mathbb{R}$$

- Let M be a $n \times n$ matrix such that ...

$$M_{t,r} = \phi(\boldsymbol{x}_t, \boldsymbol{x}_r)$$

- ... for any n points. Called the Gram matrix.
- Symmetric:

$$\phi(\boldsymbol{x}_t, \boldsymbol{x}_r) = \phi(\boldsymbol{x}_r, \boldsymbol{x}_t)$$

- Positive definite: for all non-zero $\mathbf{v}$

$$\mathbf{v} M \mathbf{v}^T \geq 0$$

Kernels

- **Mercer's Theorem**: for any kernal ϕ, there exists an **f**, such that:

$$\phi(\boldsymbol{x}_t, \boldsymbol{x}_r) = \mathbf{f}(\boldsymbol{x}_t) \cdot \mathbf{f}(\boldsymbol{x}_r)$$

- Since our features are over pairs $(\boldsymbol{x}, \boldsymbol{y})$, we will write kernels over pairs

$$\phi((\boldsymbol{x}_t, \boldsymbol{y}_t), (\boldsymbol{x}_r, \boldsymbol{y}_r)) = \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) \cdot \mathbf{f}(\boldsymbol{x}_r, \boldsymbol{y}_r)$$

Kernel Trick – Perceptron Algorithm

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

1. $\mathbf{w}^{(0)} = 0;\ i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. Let $\boldsymbol{y} = \arg\max_{\boldsymbol{y}} \mathbf{w}^{(i)} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y})$
5. if $\boldsymbol{y} \neq \boldsymbol{y}_t$
6. $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y})$
7. $i = i + 1$
8. return $\mathbf{w}^i$

- Each feature function $\mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t)$ is added and $\mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y})$ is subtracted to $\mathbf{w}$ say $\alpha_{\boldsymbol{y},t}$ times
  - $\alpha_{\boldsymbol{y},t}$ is the # of times during learning label $\boldsymbol{y}$ is predicted for example t
- Thus,

$$\mathbf{w} = \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t} [\mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y})]$$

Kernel Trick – Perceptron Algorithm

- We can re-write the argmax function as:

$$
\begin{aligned}
\boldsymbol{y}* &= \arg\max_{\boldsymbol{y}^*} \mathbf{w}^{(i)} \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}^*) \\
&= \arg\max_{\boldsymbol{y}^*} \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t} [\mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y})] \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}^*) \\
&= \arg\max_{\boldsymbol{y}^*} \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t} [\mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}_t) \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}^*) - \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}) \cdot \mathbf{f}(\boldsymbol{x}_t, \boldsymbol{y}^*)] \\
&= \arg\max_{\boldsymbol{y}^*} \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t} [\phi((\boldsymbol{x}_t, \boldsymbol{y}_t), (\boldsymbol{x}_t, \boldsymbol{y}^*)) - \phi((\boldsymbol{x}_t, \boldsymbol{y}), (\boldsymbol{x}_t, \boldsymbol{y}^*))]
\end{aligned}
$$

- We can then re-write the perceptron algorithm strictly with kernels

Kernel Trick – Perceptron Algorithm

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

1. $\forall \boldsymbol{y}, t$ set $\alpha_{\boldsymbol{y},t} = 0$
2. for $n : 1..N$
3.   for $t : 1..T$
4.   Let $\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}^*} \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t}[\phi((\boldsymbol{x}_t, \boldsymbol{y}_t), (\boldsymbol{x}_t, \boldsymbol{y}^*)) - \phi((\boldsymbol{x}_t, \boldsymbol{y}), (\boldsymbol{x}_t, \boldsymbol{y}^*))]$
5.   if $\boldsymbol{y}^* \neq \boldsymbol{y}_t$
6. $\alpha_{\boldsymbol{y}^*,t} = \alpha_{\boldsymbol{y}^*,t} + 1$

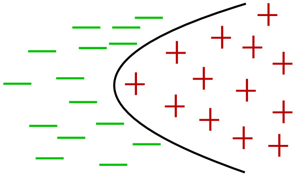
- Given a new instance $\boldsymbol{x}$

$$\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}^*} \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t}[\phi((\boldsymbol{x}_t, \boldsymbol{y}_t), (\boldsymbol{x}, \boldsymbol{y}^*)) - \phi((\boldsymbol{x}_t, \boldsymbol{y}), (\boldsymbol{x}, \boldsymbol{y}^*))]$$

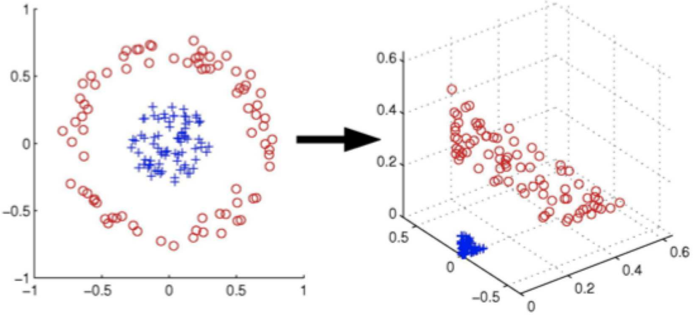
- But it seems like we have just complicated things???

Kernels = Tractable Non-Linearity

- A linear classifier in a higher dimensional feature space is a non-linear classifier in the original space
- Computing a non-linear kernel is often better computationally than calculating the corresponding dot product in the high dimension feature space
- Thus, kernels allow us to efficiently learn non-linear classifiers



Linear Classifiers in High Dimension



$$\begin{array}{ccc} \Re^2 & \longrightarrow & \Re^3 \\ (x_1, x_2) & \longmapsto & (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \end{array}$$

Example: Polynomial Kernel

- $\mathbf{f}(\boldsymbol{x}) \in \mathbb{R}^M$, $d \geq 2$
- $\phi(\boldsymbol{x}_t, \boldsymbol{x}_s) = (\mathbf{f}(\boldsymbol{x}_t) \cdot \mathbf{f}(\boldsymbol{x}_s) + 1)^d$
  - $O(M)$ to calculate for any d!!
- But in the original feature space (primal space)
  - Consider $d = 2$, $M = 2$, and $\mathbf{f}(\boldsymbol{x}_t) = [x_{t,1}, x_{t,2}]$

$$
\begin{aligned}
(\mathbf{f}(\boldsymbol{x}_t) \cdot \mathbf{f}(\boldsymbol{x}_s) + 1)^2 &= ([x_{t,1}, x_{t,2}] \cdot [x_{s,1}, x_{s,2}] + 1)^2 \\
&= (x_{t,1}x_{s,1} + x_{t,2}x_{s,2} + 1)^2 \\
&= (x_{t,1}x_{s,1})^2 + (x_{t,2}x_{s,2})^2 + 2(x_{t,1}x_{s,1}) + 2(x_{t,2}x_{s,2}) \\
&\quad +2(x_{t,1}x_{t,2}x_{s,1}x_{s,2}) + (1)^2
\end{aligned}
$$

which equals:

$$
[(x_{t,1})^2, (x_{t,2})^2, \sqrt{2}x_{t,1}, \sqrt{2}x_{t,2}, \sqrt{2}x_{t,1}x_{t,2}, 1] \cdot [(x_{s,1})^2, (x_{s,2})^2, \sqrt{2}x_{s,1}, \sqrt{2}x_{s,2}, \sqrt{2}x_{s,1}x_{s,2}, 1]
$$

Popular Kernels

- Polynomial kernel

$$\phi(\boldsymbol{x}_t, \boldsymbol{x}_s) = (\mathbf{f}(\boldsymbol{x}_t) \cdot \mathbf{f}(\boldsymbol{x}_s) + 1)^d$$

- Gaussian radial basis kernel (infinite feature space representation!)

$$\phi(\boldsymbol{x}_t, \boldsymbol{x}_s) = exp(\frac{-||\mathbf{f}(\boldsymbol{x}_t) - \mathbf{f}(\boldsymbol{x}_s)||^2}{2\sigma})$$

- String kernels [Lodhi et al. 2002, Collins and Duffy 2002]
- Tree kernels [Collins and Duffy 2002]

Kernels Summary

- Can turn a linear classifier into a non-linear classifier
- Kernels project feature space to higher dimensions
  - Sometimes exponentially larger
  - Sometimes an infinite space!
- Can "kernalize" algorithms to make them non-linear

Main Points of Lecture

- Feature representations
- Choose feature weights, $\mathbf{w}$, to maximize some function (min error, max margin)
- Batch learning (SVMs, Logistic Regression) versus online learning (perceptron, MIRA, SGD)
- The right way to parallelize
- Structured Learning
- Linear versus Non-linear classifiers

References and Further Reading

- A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra. 1996.
  A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1).

- C.T. Chu, S.K. Kim, Y.A. Lin, Y.Y. Yu, G. Bradski, A.Y. Ng, and K. Olukotun. 2007.
  Map-Reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems*.

- M. Collins and N. Duffy. 2002.
  New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proc. ACL*.

- M. Collins. 2002.
  Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. EMNLP*.

- K. Crammer and Y. Singer. 2001.
  On the algorithmic implementation of multiclass kernel based vector machines. *JMLR*.

- K. Crammer and Y. Singer. 2003.
  Ultraconservative online algorithms for multiclass problems. *JMLR*.

- K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer. 2003.

Online passive aggressive algorithms. In *Proc. NIPS*.

- K. Crammer, O. Dekel, J. Keshat, S. Shalev-Shwartz, and Y. Singer. 2006. Online passive aggressive algorithms. *JMLR*.
- Y. Freund and R.E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.
- T. Joachims. 2002. *Learning to Classify Text using Support Vector Machines*. Kluwer.
- J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*.
- H. Lodhi, C. Saunders, J. Shawe-Taylor, and N. Cristianini. 2002. Classification with string kernels. *Journal of Machine Learning Research*.
- G. Mann, R. McDonald, M. Mohri, N. Silberman, and D. Walker. 2009. Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems*.
- A. McCallum, D. Freitag, and F. Pereira. 2000. Maximum entropy Markov models for information extraction and segmentation. In *Proc. ICML*.

- R. McDonald, K. Crammer, and F. Pereira. 2005.
  Online large-margin training of dependency parsers. In *Proc. ACL*.
- K.R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. 2001.
  An introduction to kernel-based learning algorithms. *IEEE Neural Networks*, 12(2):181–201.
- F. Sha and F. Pereira. 2003.
  Shallow parsing with conditional random fields. In *Proc. HLT/NAACL*, pages 213–220.
- C. Sutton and A. McCallum. 2006.
  An introduction to conditional random fields for relational learning. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press.
- B. Taskar, C. Guestrin, and D. Koller. 2003.
  Max-margin Markov networks. In *Proc. NIPS*.
- B. Taskar. 2004.
  Learning Structured Prediction Models: A Large Margin Approach. Ph.D. thesis, Stanford.
- I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. 2004.
  Support vector learning for interdependent and structured output spaces. In *Proc. ICML*.
- T. Zhang. 2004.

Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*.