Introduction to Data-Driven Dependency Parsing

Introductory Course, ESSLLI 2007

Ryan McDonald\textsuperscript{1} Joakim Nivre\textsuperscript{2}

\textsuperscript{1}Google Inc., New York, USA
E-mail: ryanmcd@google.com

\textsuperscript{2}Uppsala University and Växjö University, Sweden
E-mail: nivre@msi.vxu.se
Formal Conditions on Dependency Graphs

Last Lecture

- For a dependency graph $G = (V, A)$
- With label set $L = \{l_1, \ldots, l_{|L|}\}$

- $G$ is (weakly) connected:
  - If $i, j \in V$, $i \leftrightarrow^* j$.

- $G$ is acyclic:
  - If $i \rightarrow j$, then not $j \rightarrow^* i$.

- $G$ obeys the single-head constraint:
  - If $i \rightarrow j$, then not $i' \rightarrow j$, for any $i' \neq i$.

- $G$ is projective:
  - If $i \rightarrow j$, then $i \rightarrow^* i'$, for any $i'$ such that $i < i' < j$ or $j < i' < i$. 
Dependency Graphs as Trees

Consider a dependency graph $G = (V, A)$ satisfying:

- $G$ is (weakly) connected:
  - If $i, j \in V$, $i \leftrightarrow^* j$.
- $G$ obeys the single-head constraint:
  - If $i \rightarrow j$, then not $i' \rightarrow j$, for any $i' \neq i$.
- $G$ obeys the single-root constraint:
  - If $\nexists i$ such that $i \rightarrow j$, then $\exists i$ such that $i \rightarrow j'$, for any $j' \neq j$.
  - $w_0 = \text{root}$ is always this node.

This dependency graph is by definition a tree.

For the rest of the course we assume that all dependency graphs are trees.
Dependency Graphs as Trees

Satisfies: connected, single-head

Economic news had little effect on financial markets.
Dependency Graphs as Trees

Satisfies: connected, single-head, single-root

```
root     Economic news had little effect on financial markets
  |          
  v          
  pred      obj      pc
  |          |          |  
  v          v          v
  nmod      nmod      nmod
  v          v          v
  subj      obj
  v          v
  nmod      pc
```

Economic news had little effect on financial markets.
Overview of the Course

- Dependency parsing (Joakim)
- **Machine learning methods** (Ryan)
- Transition-based models (Joakim)
- Graph-based models (Ryan)
- Loose ends (Joakim, Ryan):
  - Other approaches
  - Empirical results
  - Available software
Data-Driven Parsing

- Data-Driven → Machine Learning
- Parameterize a model
- **Supervised**: Learn parameters from annotated data
- Unsupervised: Induce parameters from a large corpora
- Data-Driven vs. Grammar-driven
  - Can parse all sentences vs. generate specific language
  - Data-driven = grammar of $\Sigma^*$
Lecture 2: Outline

- Feature Representations
- Linear Classifiers
  - Perceptron
  - Large-Margin Classifiers (SVMs, MIRA)
  - Others
- Non-linear Classifiers
  - K-NNs and Memory-based Learning
  - Kernels
- Structured Learning
  - Structured Perceptron
  - Large-Margin Perceptron
  - Others
Important Message

- This lecture contains a lot of details
- Not important if you do not follow all proofs and maths
- What is important
  - Understand basic representation of data – features
  - Understand basic goal and structure of classifiers
  - Understand important distinctions: linear vs. non-linear, binary vs. multiclass, multiclass vs. structured, etc.
- Interested in ML for NLP
  - Check out afternoon course “Machine learning methods for NLP”
Feature Representations

- 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., dependency tree, document class, part-of-speech tags, next parsing action

- We assume a mapping from \( x \) to a high dimensional feature vector
  \[ f(x) : \mathcal{X} \rightarrow \mathbb{R}^m \]

- But sometimes it will be easier to think of a mapping from an input/output pair to a feature vector
  \[ f(x, y) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m \]

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

▶ $x$ is a document

$$f_j(x) = \begin{cases} 
1 & \text{if } x \text{ contains the word "interest"} \\
0 & \text{otherwise}
\end{cases}$$

$f_j(x)$ = The percentage of words than contain punctuation

▶ $x$ is a word and $y$ is a part-of-speech tag

$$f_j(x, y) = \begin{cases} 
1 & \text{if } x = "\text{bank}" \text{ and } y = \text{Verb} \\
0 & \text{otherwise}
\end{cases}$$
Example 2

\[ f_0(x) = \begin{cases} 1 & \text{if } x \text{ contains the word “John”} \\ 0 & \text{otherwise} \end{cases} \]

\[ f_1(x) = \begin{cases} 1 & \text{if } x \text{ contains the word “Mary”} \\ 0 & \text{otherwise} \end{cases} \]

\[ f_2(x) = \begin{cases} 1 & \text{if } x \text{ contains the word “Harry”} \\ 0 & \text{otherwise} \end{cases} \]

\[ f_3(x) = \begin{cases} 1 & \text{if } x \text{ contains the word “likes”} \\ 0 & \text{otherwise} \end{cases} \]

- \( x = \text{John likes Mary} \rightarrow f(x) = [1 \ 1 \ 0 \ 1] \)
- \( x = \text{Mary likes John} \rightarrow f(x) = [1 \ 1 \ 0 \ 1] \)
- \( x = \text{Harry likes Mary} \rightarrow f(x) = [0 \ 1 \ 1 \ 1] \)
- \( x = \text{Harry likes Harry} \rightarrow f(x) = [0 \ 0 \ 1 \ 1] \)
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 can define two kinds of linear classifiers.
  - **Reminder**: $\mathbf{v} \cdot \mathbf{v}' = \sum_j v_j \times v'_j \in \mathbb{R}$
  - **Binary Classification**: $\mathcal{Y} = \{-1, 1\}$
    $$y = \text{sign}(w \cdot f(x))$$
  - **Multiclass Classification**: $\mathcal{Y} = \{0, 1, \ldots, N\}$
    $$y = \arg\max_y w \cdot f(x, y)$$
Binary Linear Classifier

Divides all points:

\[ y = \text{sign}(w \cdot f(x)) \]
Multiclass Linear Classifier

Defines regions of space:

$$y = \arg \max_y w \cdot f(x, y)$$
A set of points is separable, if there exists a $w$ such that classification is perfect.
Supervised Learning

- 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, CRFs)
- 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

- Minimize error
  - Binary classification: $\mathcal{Y} = \{-1, 1\}$
    \[
    \mathbf{w} = \arg\min_{\mathbf{w}} \sum_t 1 - 1[y_t = \text{sign}(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}_t))]
    \]
  - Multiclass classification: $\mathcal{Y} = \{0, 1, \ldots, N\}$
    \[
    \mathbf{w} = \arg\min_{\mathbf{w}} \sum_t 1 - 1[y_t = \arg\max_y \mathbf{w} \cdot \mathbf{f}(\mathbf{x}_t, y)]
    \]

1[p] = \begin{cases} 1 & p \text{ is true} \\ 0 & \text{otherwise} \end{cases}
Perceptron Learning Algorithm (multiclass)

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

1. $w^{(0)} = 0; \ i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. Let $y' = \text{arg max}_{y'} w^{(i)} \cdot f(x_t, y')$
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 $w^i$
Perceptron Learning Algorithm (multiclass)

- Given an training instance \((x_t, y_t)\), define:
  - \(\mathcal{Y}_t = \mathcal{Y} - \{y_t\}\)

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

\[
\begin{align*}
    u \cdot f(x_t, y_t) - u \cdot f(x_t, y') &\geq \gamma
\end{align*}
\]

for all \(y' \in \mathcal{Y}_t\) and \(\|u\| = \sqrt{\sum_j u_j^2}\)

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

\[
\text{Number of training errors} \leq \frac{R^2}{\gamma^2}
\]

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

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)
Perception Learning Algorithm (multiclass)

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

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 \( k \), \( u \cdot w^{(k)} \geq (k - 1)\gamma \)
- Now: since \( u \cdot w^{(k)} \leq ||u|| \times ||w^{(k)}|| \) and \( ||u|| = 1 \) then \( ||w^{(k)}|| \geq (k - 1)\gamma \)
- 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 \( w^{(k-1)} \cdot f(x_t, y_t) - w^{(k-1)} \cdot f(x_t, y') \leq 0 \)
Perception Learning Algorithm (multiclass)

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

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

  \[ ||w^{(k)}||^2 \leq (k - 1)R^2 \]

- Therefore,

  \[ (k - 1)^2\gamma^2 \leq ||w^{(k)}||^2 \leq (k - 1)R^2 \]

- and solving for $k$

  \[ k - 1 \leq \frac{R^2}{\gamma^2} \]

- Therefore the number of errors is bounded!
**Margin**

**Training**

```
- - + + +
- - + + +
- - + + +
- - + + +
```

**Testing**

```
- - - - -
- - - - -
- - - - -
- - - - -
```

Denote the value of the margin by $\gamma$.
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, it does not pick a \( w \) to maximize the margin!**
Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

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

such that:

$$y_t(w \cdot f(x_t)) \geq \gamma$$

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

$\triangleright$ $||w||$ is bound since scaling trivially produces larger margin

$$y_t([\beta w] \cdot f(x_t)) \geq \beta \gamma,$$ for some $\beta \geq 1$

$\triangleright$ Instead of fixing $||w||$ we fix the margin $\gamma = 1$

Min Norm:

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

such that:

$$y_t(w \cdot f(x_t)) \geq 1$$

$\forall (x_t, y_t) \in T$
Support Vector Machines

Binary:

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

such that:

\[
y_t(w \cdot f(x_t)) \geq 1
\]
\[
\forall (x_t, y_t) \in \mathcal{T}
\]

Multiclass:

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

such that:

\[
w \cdot f(x_t, y_t) - w \cdot f(x_t, y') \geq 1
\]
\[
\forall (x_t, y_t) \in \mathcal{T} \text{ and } y' \in \bar{\mathcal{Y}}_t
\]

Both are \textit{quadratic programming problems} – a well known convex optimization problem

Can be solved with out-of-the-box algorithms
Support Vector Machines

Binary:

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

\[
y_t (w \cdot f(x_t)) \geq 1
\]
\[
\forall (x_t, y_t) \in \mathcal{T}
\]

- 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 only depends on 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 (multiclass)

Batch (SVMs):

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

such that:

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

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

Online (MIRA):

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

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

▶ MIRA has much smaller optimizations with only \( |\bar{Y}_t| \) constraints

▶ Cost: sub-optimal optimization
Summary

What we have covered

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

What is next

- Non-linear classifiers
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
  - Decision Trees
  - Kernels
  - Neural Networks
- Will only discuss a couple due to time constraints
K-Nearest Neighbours

- Simplest form: for a given test point $x$, find $k$-nearest neighbours in training set
- Neighbours vote for classification
- Distance is Euclidean distance

$$d(x_t, x_r) = \sqrt{\sum_j (f_j(x_t) - f_j(x_r))^2}$$

No linear classifier can correctly label data set. But 3-nearest neighbours does.
K-Nearest Neighbours

- A training set $\mathcal{T}$, distance function $d$, and value $K$ define a non-linear classification boundary
K-Nearest Neighbours

- K-NN is often called a lazy learning algorithm or memory based learning (MBL)
- K-NN generalized in the Tilburg Memory Based Learning Package
  - Different distance functions
  - Different voting schemes for classification
  - Tie-breaking
  - Memory representations
Kernels

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

$$\phi(x_t, x_r) \in \mathbb{R}$$

Mercer’s Theorem: for any kernel $\phi$, there exists an $f$, such that:

$$\phi(x_t, x_r) = f(x_t) \cdot f(x_r)$$
Kernel Trick – Perceptron Algorithm

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

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

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

- Thus,
  \[
  \mathbf{w} = \sum_{t, y} \alpha_{y, t} [f(x_t, y_t) - f(x_t, y)]
  \]
Kernel Trick – Perceptron Algorithm

We can re-write the argmax function as:

\[
\begin{align*}
y^* &= \arg \max_{y^*} \mathbf{w}^{(i)} \cdot f(x_t, y^*) \\
&= \arg \max_{y^*} \sum_{t, y} \alpha_{y,t} [f(x_t, y_t) - f(x_t, y)] \cdot f(x_t, y^*) \\
&= \arg \max_{y^*} \sum_{t, y} \alpha_{y,t} [f(x_t, y_t) \cdot f(x_t, y^*) - f(x_t, y) \cdot f(x_t, y^*)] \\
&= \arg \max_{y^*} \sum_{t, y} \alpha_{y,t} [\phi((x_t, y_t), (x_t, y^*)) - \phi((x_t, y), (x_t, y^*))]
\end{align*}
\]

We can then re-write the perceptron algorithm strictly with kernels.
Kernel Trick – Perceptron Algorithm

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

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

Given a new instance \( x \)

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

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

\[ \mathbb{R}^2 \rightarrow \mathbb{R}^3 \]

\[(x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)\]
Example: Polynomial Kernel

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

\[
(f(x_t) \cdot f(x_s) + 1)^2 = ([x_{t,1}, x_{t,2}] \cdot [x_{s,1}, x_{s,2}] + 1)^2
= (x_{t,1}x_{s,1} + x_{t,2}x_{s,2} + 1)^2
= (x_{t,1}x_{s,1})^2 + (x_{t,2}x_{s,2})^2 + 2(x_{t,1}x_{s,1}) + 2(x_{t,2}x_{s,2})
+ 2(x_{t,1}x_{t,2}x_{s,1}x_{s,2}) + (1)^2
\]

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
  \[ \phi(x_t, x_s) = (f(x_t) \cdot f(x_s) + 1)^d \]

- Gaussian radial basis kernel (infinite feature space representation!)
  \[ \phi(x_t, x_s) = \exp\left(\frac{-\|f(x_t) - f(x_s)\|^2}{2\sigma}\right) \]

- String kernels [Lodhi et al. 2002, Collins and Duffy 2002]
- Tree kernels [Collins and Duffy 2002]
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 solve our learning algorithms in such cases
Perceptron

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

1. $w^{(0)} = 0; \quad i = 0$
2. for $n: 1..N$
3. for $t: 1..T$
4. Let $y' = \text{arg max}_{y'} w^{(i)} \cdot f(x_t, y')$ (**)
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 $w^i$

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

Batch (SVMs):

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

such that:

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

\(\forall (x_t, y_t) \in T\) and \(y' \in \tilde{Y}_t\) (**)

Online (MIRA):

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

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

(**) 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:
    \[
    f(x, y) = \sum_{A\rightarrow BC \in y} f(x, A \rightarrow BC)
    \]
  - Sequence Analysis – Markov Assumptions:
    \[
    f(x, y) = \sum_{i=1}^{y} f(x, y_{i-1}, y_{i})
    \]
- These kinds of factorizations allow us to run algorithms like CKY and Viterbi to compute the argmax function
Structured Perceptron

- Exactly like original perceptron
- Except now the argmax function uses a factored feature representation
- All of the original analysis for the multiclass perceptron carries over!!
Structured SVMs

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

such that:

\[
\mathbf{w} \cdot \mathbf{f}(x_t, y_t) - \mathbf{w} \cdot \mathbf{f}(x_t, y') \geq \mathcal{L}(y_t, y')
\]

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

- 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
Online Structured SVMs (or Online MIRA)

Training data: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{T}$
1. $w^{(0)} = 0; \ i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. $w^{(i+1)} = \text{arg min}_{w^*} \|w^* - w^{(i)}\|$
   such that:
   $$ w \cdot f(x_t, y_t) - w \cdot f(x_t, y') \geq \mathcal{L}(y_t, y') $$
   $\forall y' \in \tilde{\mathcal{Y}}_t$ and $y' \in \text{k-best}(x_t, w^{(i)})$ (**)
5. $i = i + 1$
6. return $w^i$

- k-best($x_t$) is set of outputs with highest scores using weight vector $w^{(i)}$
- Simple Solution – only consider outputs $y' \in \tilde{\mathcal{Y}}_t$ that currently have highest score
Main Points of Lecture

- Feature representations
- Choose feature weights, $w$, to maximize some function (min error, max margin)
- Batch learning (SVMs) versus online learning (perceptron, MIRA)
- Linear versus Non-linear classifiers
- Structured Learning
References and Further Reading


References and Further Reading


Cubic-time parsing and learning algorithms for grammatical bigram models. Technical Report UCB/CSD-01-1148, Computer Science Division, University of California Berkeley.


  Support vector learning for interdependent and structured output spaces. In *Proc. ICML*.

  *Graph Theory*. Cambridge University Press.

  *Discovery of linguistic relations using lexical attraction*. Ph.D. thesis, MIT.