Introduction to Data-Driven Dependency Parsing

Introductory Course, ESSLLI 2007

Ryan McDonald[1] Joakim Nivre[2]

[1]Google Inc., New York, USA
E-mail: `ryanmcd@google.com`

[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 $L = \{l_1, \ldots, l_{|L|}\}$

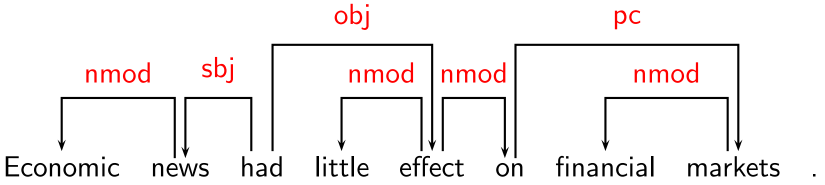
- G is (weakly) connected:
  - If $i, j \in V$, $i \leftrightarrow^* j$.
- G is acyclic:
  - If $i \rightarrow j$, then not $j \rightarrow^* i$.
- G obeys the single-head constraint:
  - If $i \rightarrow j$, then not $i' \rightarrow j$, for any $i' \neq i$.
- G is projective:
  - If $i \rightarrow j$, 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 $j' \neq j$
    - $w_0 =$ 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

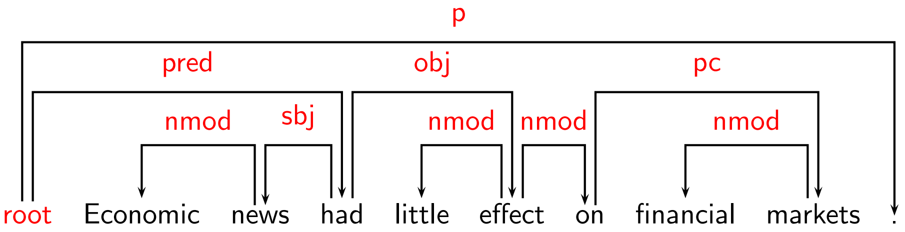
Dependency Graphs as Trees

Satisfies: connected, single-head



Dependency Graphs as Trees

Satisfies: connected, single-head, single-root



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 $\rightarrow$ 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 Σ^*

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: $\boldsymbol{x} \in \mathcal{X}$
  - 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., dependency tree, document class, part-of-speech tags, next parsing action

- We assume a mapping from $\boldsymbol{x}$ to a high dimensional feature vector
  - $\mathbf{f}(\boldsymbol{x}) : \mathcal{X} \rightarrow \mathbb{R}^m$
- But sometimes it will be easier to think of a mapping from an input/output pair to a feature vector
  - $\mathbf{f}(\boldsymbol{x}, \boldsymbol{y}) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m$
- For any vector $\mathbf{v} \in \mathbb{R}^m$, let $\mathbf{v}_j$ be the j^{th} value

Examples

- x is a document

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

$$\mathbf{f}_j(x) = \text{The percentage of words than contain punctuation}$$

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

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

Example 2

$$\mathbf{f}_0(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains the word ``John''} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{f}_1(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains the word ``Mary''} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{f}_2(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains the word ``Harry''} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{f}_3(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains the word ``likes''} \\ 0 & \text{otherwise} \end{cases}$$

- $\boldsymbol{x}$=John likes Mary $\rightarrow \mathbf{f}(\boldsymbol{x}) = [1\ 1\ 0\ 1]$
- $\boldsymbol{x}$=Mary likes John $\rightarrow \mathbf{f}(\boldsymbol{x}) = [1\ 1\ 0\ 1]$
- $\boldsymbol{x}$=Harry likes Mary $\rightarrow \mathbf{f}(\boldsymbol{x}) = [0\ 1\ 1\ 1]$
- $\boldsymbol{x}$=Harry likes Harry $\rightarrow \mathbf{f}(\boldsymbol{x}) = [0\ 0\ 1\ 1]$

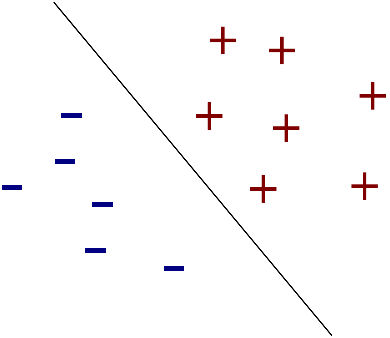
Linear Classifiers

- **Linear classifier**: score (or probability) of a particular classification is based on a linear combination of features and their weights
- Let $\mathbf{w} \in \mathbb{R}^m$ be a high dimensional weight vector
- If we assume that $\mathbf{w}$ is known, then we can define two kinds of linear classifiers
  - Reminder:
  $$\mathbf{v} \cdot \mathbf{v}' = \sum_j \mathbf{v}_j \times \mathbf{v}'_j \in \mathbb{R}$$
  - **Binary Classification**: $\mathcal{Y} = \{-1, 1\}$
  $$\boldsymbol{y} = \mathit{sign}(\mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}))$$
  - **Multiclass Classification**: $\mathcal{Y} = \{0, 1, \dots, N\}$
  $$\boldsymbol{y} = \arg\max_{\boldsymbol{y}} \ \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}, \boldsymbol{y})$$

Binary Linear Classifier

Divides all points:

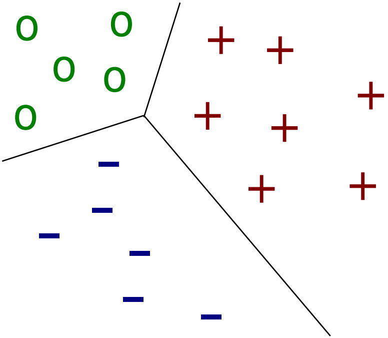
$$y = sign(\mathbf{w} \cdot \mathbf{f}(x))$$



Multiclass Linear Classifier

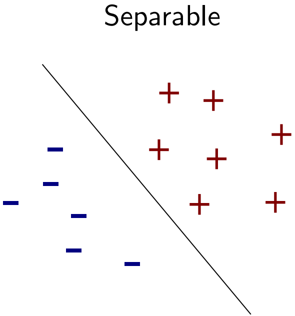
Defines regions of space:

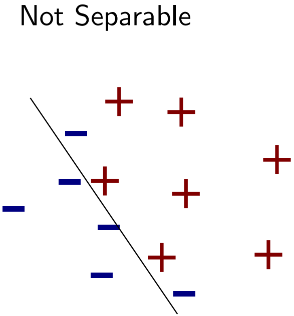
$$\boldsymbol{y} = \arg\max_{\boldsymbol{y}} \; \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}, \boldsymbol{y})$$



Separability

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





Supervised Learning

- 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, 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: $\mathcal{Y} = \{-1, 1\}$

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

  - Multiclass classification: $\mathcal{Y} = \{0, 1, \ldots, N\}$

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

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

Perceptron Learning Algorithm (multiclass)

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 Learning Algorithm (multiclass)

- Given an training instance $(\boldsymbol{x}_t, \boldsymbol{y}_t)$, define:
  - $\bar{\mathcal{Y}}_t = \mathcal{Y} - \{\boldsymbol{y}_t\}$
- 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 Learning Algorithm (multiclass)

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

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

  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)

Perception Learning Algorithm (multiclass)

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 $\mathbf{w}^i$

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

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

Perception Learning Algorithm (multiclass)

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

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

- Therefore,

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

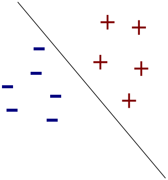
- and solving for k

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

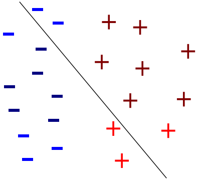
- Therefore the number of errors is bounded!

Margin

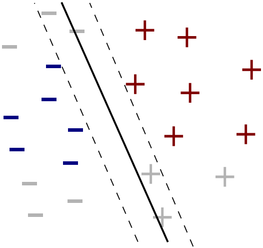
Training

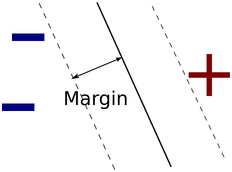


Testing



Denote the value of the margin by γ





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 **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:

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

such that:

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

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

=

Min Norm:

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

such that:

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

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

- $||\mathbf{w}||$ is bound since scaling trivially produces larger margin

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

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

Support Vector Machines

Binary:

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

such that:

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

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

Multiclass:

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

such that:

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

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

Both are quadratic programming problems – a well known convex optimization problem

Can be solved with out-of-the-box algorithms

Support Vector Machines

Binary:

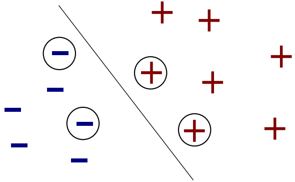
$$\min \frac{1}{2}||\mathbf{w}||^2$$

such that:

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

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

- 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)

Online (MIRA):

Batch (SVMs):

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

such that:

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

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

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

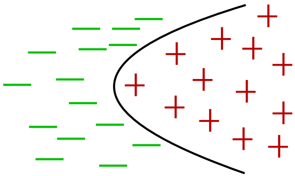
- Feature-based representations
- Linear Classifiers
  - Perceptron
  - Large-Margin – SVMs (batch) and MIRA (online)

What is next

- 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

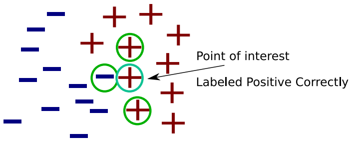


K-Nearest Neighbours

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

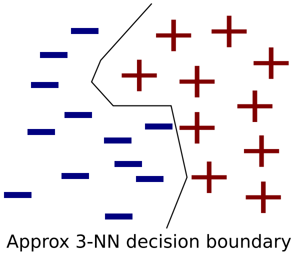
$$d(\boldsymbol{x}_t, \boldsymbol{x}_r) = \sqrt{\sum_j (\mathbf{f}_j(\boldsymbol{x}_t) - \mathbf{f}_j(\boldsymbol{x}_r))^2}$$

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:

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

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

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

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. $\alpha_{\boldsymbol{y}^*,t} = \alpha_{\boldsymbol{y}^*,t} + 1$

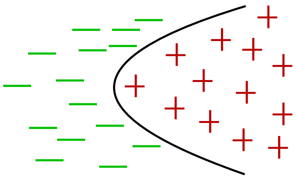
- Given a new instance $\boldsymbol{x}$

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

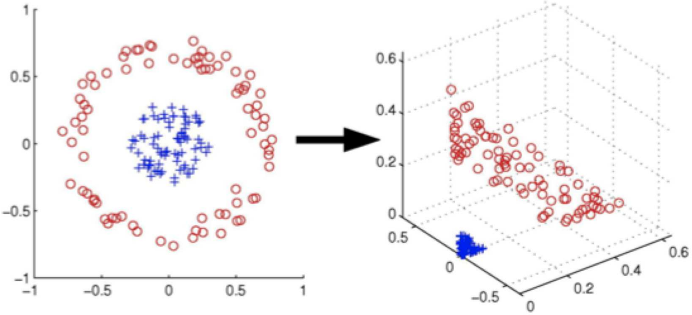
- But it seems like we have just complicated things???

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]

Structured Learning

- Sometimes our output space $\mathcal{Y}$ is not simply a category
- Examples:
  - **Parsing**: for a sentence $\boldsymbol{x}$, $\mathcal{Y}$ is the set of possible parse trees
  - **Sequence tagging**: for a sentence $\boldsymbol{x}$, $\mathcal{Y}$ is the set of possible tag sequences, e.g., part-of-speech tags, named-entity tags
  - **Machine translation**: for a source sentence $\boldsymbol{x}$, $\mathcal{Y}$ 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 $\boldsymbol{x}$
- 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:

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

  - 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

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:

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

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

- 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

Online Structured SVMs (or 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:

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

   $\forall \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$ and $\boldsymbol{y}' \in$ k-best$(\boldsymbol{x}_t, \mathbf{w}^{(i)})$ (**)
5. $i = i + 1$
6. return $\mathbf{w}^i$

- k-best$(\boldsymbol{x}_t)$ is set of outputs with highest scores using weight vector $\mathbf{w}^{(i)}$
- Simple Solution – only consider outputs $\boldsymbol{y}' \in \bar{\mathcal{Y}}_t$ 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

- 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).
- P. M. Camerini, L. Fratta, and F. Maffioli. 1980. The k best spanning arborescences of a network. *Networks*, 10(2):91–110.
- Y.J. Chu and T.H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.
- 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*.
- J. Edmonds. 1967.
  Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.
- J. Eisner. 1996.
  Three new probabilistic models for dependency parsing: An exploration. In *Proc. COLING*.
- 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.
- D. Klein and C. Manning. 2004.
  Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proc. ACL*.
- T. Koo, A. Globerson, X. Carreras, and M. Collins. 2007.
  Structured prediction models via the matrix-tree theorem. In *Proc. EMNLP*.

- 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*.
- A. McCallum, D. Freitag, and F. Pereira. 2000.
  Maximum entropy Markov models for information extraction and segmentation. In *Proc. ICML*.
- R. McDonald and F. Pereira. 2006.
  Online learning of approximate dependency parsing algorithms. In *Proc EACL*.
- R. McDonald and G. Satta. 2007.
  On the complexity of non-projective data-driven dependency parsing. In *Proc. IWPT*.
- 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.
- M.A. Paskin. 2001.

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

- K. Sagae and A. Lavie. 2006.
  Parser combination by reparsing. In *Proc. HLT/NAACL*.

- F. Sha and F. Pereira. 2003.
  Shallow parsing with conditional random fields. In *Proc. HLT/NAACL*, pages 213–220.

- N. Smith and J. Eisner. 2005.
  Guiding unsupervised grammar induction using contrastive estimation. In *Working Notes of the International Joint Conference on Artificial Intelligence Workshop on Grammatical Inference Applications*.

- D.A. Smith and N.A. Smith. 2007.
  Probabilistic models of nonprojective dependency trees. In *Proc. EMNLP*.

- 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.

- R.E. Tarjan. 1977.
  Finding optimum branchings. *Networks*, 7:25–35.

- 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*.
- W.T. Tutte. 1984.
  Graph Theory. Cambridge University Press.
- D. Yuret. 1998.
  Discovery of linguistic relations using lexical attraction. Ph.D. thesis, MIT.