Algorithms for NLP

Classification I

Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein – UC Berkeley
Efficient Parsing for Lexical Grammars
Lexicalized Trees

- Add “head words” to each phrasal node
  - Syntactic vs. semantic heads
  - Headship not in (most) treebanks
  - Usually use head rules, e.g.:
    - **NP:**
      - Take leftmost NP
      - Take rightmost N*
      - Take rightmost JJ
      - Take right child
    - **VP:**
      - Take leftmost VB*
      - Take leftmost VP
      - Take left child
Lexicalized PCFGs?

- Problem: we now have to estimate probabilities like

\[ VP(\text{saw}) \rightarrow VBD(\text{saw}) \, NP-C(\text{her}) \, NP(\text{today}) \]

- Never going to get these atomically off of a treebank

- Solution: break up derivation into smaller steps
Lexical Derivation Steps

- A derivation of a local tree [Collins 99]

Choose a head tag and word

Choose a complement bag

Generate children (incl. adjuncts)

Recursively derive children
Lexicalized CKY

\[
\text{bestScore}(X, i, j, h) = \begin{cases} 
\text{tagScore}(X, s[i]) & \text{if } (j = i+1) \\
\max \max_{k, h', X \toYZ} \left[ \text{score}(X[h] \to Y[h] Z[h']) \right. & \left. \times \text{bestScore}(Y, i, k, h) \times \text{bestScore}(Z, k, j, h') \right]
\end{cases}
\]

\[
O(n^3)
\]
Quartic Parsing

- Turns out, you can do (a little) better [Eisner 99]

- Gives an $O(n^4)$ algorithm
- Still prohibitive in practice if not pruned
Pruning with Beams

- The Collins parser prunes with per-cell beams [Collins 99]
  - Essentially, run the \(O(n^5)\) CKY
  - Remember only a few hypotheses for each span \(<i,j>\).
  - If we keep \(K\) hypotheses at each span, then we do at most \(O(nK^2)\) work per span (why?)
  - Keeps things more or less cubic (and in practice is more like linear!)

- Also: certain spans are forbidden entirely on the basis of punctuation (crucial for speed)
Pruning with a PCFG

- The Charniak parser prunes using a two-pass, coarse-to-fine approach [Charniak 97+]
  - First, parse with the base grammar
  - For each $X:[i,j]$ calculate $P(X|i,j,s)$
    - This isn’t trivial, and there are clever speed ups
  - Second, do the full $O(n^5)$ CKY
    - Skip any $X:[i,j]$ which had low (say, < 0.0001) posterior
    - Avoids almost all work in the second phase!

- Charniak et al 06: can use more passes
- Petrov et al 07: can use many more passes
Results

- Stanford Parser – 86.3 (unlex / struct annotation)
- Collins 99 – 88.6 F1 (lexical)
- Charniak and Johnson 05 – 89.7 / 91.3 F1 (lexical + rerank)
- McClosky et al 06 – 92.1 F1 (lexical + rerank + self-train)
- Petrov et al 06 – 90.7 F1 (unlex / latent vars)
- Petrov et al 10 – 91.8 (unlex / latent vars + ensemble)
- Socher et al 13 – 90.4 (unlex + neural rerank)
- Vinyals et al 15 – 90.5 / 92.1 (neural sequence + self-train)
- Dyer et al 16 – 92.4 (neural shift-reduce)

...many more that are really cool (e.g. Hall and Klein 12,14)
Latent Variable PCFGs
The Game of Designing a Grammar

- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Parent annotation [Johnson ’98]
The Game of Designing a Grammar

- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Parent annotation [Johnson ’98]
  - Head lexicalization [Collins ’99, Charniak ’00]
Annotation refines base treebank symbols to improve statistical fit of the grammar

- Parent annotation [Johnson ’98]
- Head lexicalization [Collins ’99, Charniak ’00]
- Automatic clustering?
Latent Variable Grammars

Parse Tree

Sentence $T$

Derivations $\omega$:

Grammar $G$

Parameters $\theta$

Lexicon

$S_0 \rightarrow NP_0 \ VP_0$

$S_0 \rightarrow NP_1 \ VP_0$

$S_0 \rightarrow NP_0 \ VP_1$

$S_0 \rightarrow NP_1 \ VP_1$

$S_1 \rightarrow NP_0 \ VP_0$

$S_1 \rightarrow NP_1 \ VP_1$

$NP_0 \rightarrow PRP_0$

$NP_0 \rightarrow PRP_1$

$PRP_0 \rightarrow She$

$PRP_1 \rightarrow She$

$VBD_0 \rightarrow was$

$VBD_1 \rightarrow was$

$VBD_2 \rightarrow was$
Learning Latent Annotations

EM algorithm:
- Brackets are known
- Base categories are known
- Only induce subcategories

\[ \theta \rightarrow \mathbb{E}\{x \mid T, \omega\}, \mathbb{E}\{x \mid T, \omega\} \rightarrow \theta \]

Just like Forward-Backward for HMMs.

Backward Learning

Base categories are known

Forward Learning
Refinement of the DT tag
Hierarchical refinement

- the (0.50)
  - a (0.24)
  - The (0.08)
- the (0.54)
  - a (0.25)
  - The (0.09)
- that (0.15)
  - this (0.14)
  - some (0.11)
- a (0.61)
  - the (0.19)
  - an (0.11)
- the (0.80)
  - The (0.15)
  - a (0.01)
- this (0.39)
  - that (0.28)
  - That (0.11)
- some (0.20)
  - all (0.19)
  - those (0.12)
Hierarchical Estimation Results

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat Training</td>
<td>87.3</td>
</tr>
<tr>
<td>Hierarchical Training</td>
<td>88.4</td>
</tr>
</tbody>
</table>
Refinement of the , tag

- Splitting all categories equally is wasteful:
Adaptive Splitting

- Want to split complex categories more
- Idea: split everything, roll back splits which were least useful
Adaptive Splitting Results

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>88.4</td>
</tr>
<tr>
<td>With 50% Merging</td>
<td>89.5</td>
</tr>
</tbody>
</table>
Number of Phrasal Subcategories
Number of Lexical Subcategories
Learned Splits

- **Proper Nouns (NNP):**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NNP-12</td>
<td>John</td>
<td>Robert</td>
<td>James</td>
</tr>
<tr>
<td>NNP-2</td>
<td>J.</td>
<td>E.</td>
<td>L.</td>
</tr>
<tr>
<td>NNP-1</td>
<td>Bush</td>
<td>Noriega</td>
<td>Peters</td>
</tr>
<tr>
<td>NNP-15</td>
<td>New</td>
<td>San</td>
<td>Wall</td>
</tr>
<tr>
<td>NNP-3</td>
<td>York</td>
<td>Francisco</td>
<td>Street</td>
</tr>
</tbody>
</table>

- **Personal pronouns (PRP):**

<table>
<thead>
<tr>
<th>PRP-0</th>
<th>it</th>
<th>He</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRP-1</td>
<td>it</td>
<td>he</td>
<td>they</td>
</tr>
<tr>
<td>PRP-2</td>
<td>it</td>
<td>them</td>
<td>him</td>
</tr>
</tbody>
</table>
## Learned Splits

- **Relative adverbs (RBR):**

<table>
<thead>
<tr>
<th>RBR</th>
<th>Further</th>
<th>Lower</th>
<th>Higher</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBR-0</td>
<td>further</td>
<td>lower</td>
<td>higher</td>
</tr>
<tr>
<td>RBR-1</td>
<td>more</td>
<td>less</td>
<td>More</td>
</tr>
<tr>
<td>RBR-2</td>
<td>earlier</td>
<td>Earlier</td>
<td>later</td>
</tr>
</tbody>
</table>

- **Cardinal Numbers (CD):**

<table>
<thead>
<tr>
<th>CD</th>
<th>One</th>
<th>Two</th>
<th>Three</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD-7</td>
<td>one</td>
<td>two</td>
<td>Three</td>
</tr>
<tr>
<td>CD-4</td>
<td>1989</td>
<td>1990</td>
<td>1988</td>
</tr>
<tr>
<td>CD-11</td>
<td>million</td>
<td>billion</td>
<td>trillion</td>
</tr>
<tr>
<td>CD-0</td>
<td>1</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>CD-3</td>
<td>1</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>CD-9</td>
<td>78</td>
<td>58</td>
<td>34</td>
</tr>
<tr>
<td>Language</td>
<td>Method</td>
<td>≤ 40 words F1</td>
<td>all F1</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------</td>
<td>---------------</td>
<td>--------</td>
</tr>
<tr>
<td>ENG</td>
<td>Charniak&amp;Johnson ‘05 (generative)</td>
<td>90.1</td>
<td>89.6</td>
</tr>
<tr>
<td></td>
<td>Split / Merge</td>
<td><strong>90.6</strong></td>
<td><strong>90.1</strong></td>
</tr>
<tr>
<td>GER</td>
<td>Dubey ‘05</td>
<td>76.3</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Split / Merge</td>
<td><strong>80.8</strong></td>
<td><strong>80.1</strong></td>
</tr>
<tr>
<td>CHN</td>
<td>Chiang et al. ‘02</td>
<td>80.0</td>
<td>76.6</td>
</tr>
<tr>
<td></td>
<td>Split / Merge</td>
<td><strong>86.3</strong></td>
<td><strong>83.4</strong></td>
</tr>
</tbody>
</table>

Still higher numbers from reranking / self-training methods
Efficient Parsing for Hierarchical Grammars
Coarse-to-Fine Inference

Example: PP attachment
Hierarchical Pruning

coarse:

split in two:

split in four:

split in eight:

- QP
- NP
- VP
- QP1
- QP2
- NP1
- NP2
- VP1
- VP2
- QP3
- QP4
- NP3
- NP4
- VP3
- VP4
- ...
Results

- Stanford Parser – 86.3 (unlex / struct annotation)
- Collins 99 – 88.6 F1 (lexical)
- Charniak and Johnson 05 – 89.7 / 91.3 F1 (lexical + rerank)
- McClosky et al 06 – 92.1 F1 (lexical + rerank + self-train)
- Petrov et al 06 – 90.7 F1 (unlex / latent vars)
- Petrov et al 10 – 91.8 (unlex / latent vars + ensemble)
- Socher et al 13 – 90.4 (unlex + neural rerank)
- Vinyals et al 15 – 90.5 / 92.1 (neural sequence + self-train)
- Dyer et al 16 – 92.4 (neural shift-reduce)

...many more that are really cool (e.g. Hall and Klein 12,14)
Assume the number of parses is very small

- We can represent each parse $T$ as a feature vector $\varphi(T)$
  - Typically, all local rules are features
  - Also non-local features, like how right-branching the overall tree is
  - [Charniak and Johnson 05] gives a rich set of features
Results

- Stanford Parser – 86.3 (unlex / struct annotation)
- Collins 99 – 88.6 F1 (lexical)
- Charniak and Johnson 05 – 89.7 / 91.3 F1 (lexical + rerank)
- McClosky et al 06 – 92.1 F1 (lexical + rerank + self-train)

- Petrov et al 06 – 90.7 F1 (unlex / latent vars)
- Petrov et al 10 – 91.8 (unlex / latent vars + ensemble)

- Socher et al 13 – 90.4 (unlex + neural rerank)
- Vinyals et al 15 – 90.5 / 92.1 (neural sequence + self-train)
- Dyer et al 16 – 92.4 (neural shift-reduce)

...many more that are really cool (e.g. Hall and Klein 12,14)
Another way to derive a tree:

Parsing
- No useful dynamic programming search
- Can still use beam search [Ratnaparkhi 97]
Results

- Stanford Parser – 86.3 (unlex / struct annotation)
- Collins 99 – 88.6 F1 (lexical)
- Charniak and Johnson 05 – 89.7 / 91.3 F1 (lexical + rerank)
- McClosky et al 06 – 92.1 F1 (lexical + rerank + self-train)
- Petrov et al 06 – 90.7 F1 (unlex / latent vars)
- Petrov et al 10 – 91.8 (unlex / latent vars + ensemble)
- Socher et al 13 – 90.4 (unlex + neural rerank)
- Vinyals et al 15 – 90.5 / 92.1 (neural sequence + self-train)
- Dyer et al 16 – 92.4 (neural shift-reduce)

...many more that are really cool (e.g. Hall and Klein 12,14)
Other Syntactic Models
- Lexicalized parsers can be seen as producing *dependency trees*

- Each local binary tree corresponds to an attachment in the dependency graph
- Pure dependency parsing is only cubic [Eisner 99]

- Some work on non-projective dependencies
  - Common in, e.g. Czech parsing
  - Can do with MST algorithms [McDonald and Pereira 05]
Tree Insertion Grammars

- Rewrite large (possibly lexicalized) subtrees in a single step

- Formally, a tree-insertion grammar
- Derivational ambiguity whether subtrees were generated atomically or compositionally
- Most probable parse is NP-complete
Tree-adjoining grammars

- Start with local trees
- Can insert structure with *adjunction* operators
- Mildly context-sensitive
- Models long-distance dependencies naturally
- ... as well as other weird stuff that CFGs don’t capture well (e.g. cross-serial dependencies)
CCG Parsing

- **Combinatory Categorial Grammar**
  - Fully (mono-) lexicalized grammar
  - Categories encode argument sequences
  - Very closely related to the lambda calculus (more later)
  - Can have spurious ambiguities (why?)

\[
\begin{align*}
  John & \rightarrow \text{NP} \\
  shares & \rightarrow \text{NP} \\
  buys & \rightarrow (S \backslash NP) /\text{NP} \\
  sleeps & \rightarrow S \backslash NP \\
  well & \rightarrow (S \backslash NP) \backslash (S \backslash NP)
\end{align*}
\]

```
    S
   / \  \  \\
  NP  S\backslash NP
     /   /  \  \\
    John buys shares
```

```
Classification
Classification

- **Automatically make a decision about inputs**
  - Example: document → category
  - Example: image of digit → digit
  - Example: image of object → object type
  - Example: query + webpages → best match
  - Example: symptoms → diagnosis
  - ...

- **Three main ideas**
  - Representation as feature vectors / kernel functions
  - Scoring by linear functions
  - Learning by optimization
Some Definitions

**INPUTS**

\[ X_i \quad \text{close the} \quad ____ \]

**CANDIDATE SET**

\[ \mathcal{Y}(x) \quad \{ \text{door, table, ...} \} \]

**CANDIDATES**

\[ y \quad \text{table} \]

**TRUE OUTPUTS**

\[ y^* \quad \text{door} \]

**FEATURE VECTORS**

\[ f(x, y) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \]

- \( x_{-1} = \text{“the”} \land y = \text{“door”} \)
- \( \text{“close” in } x \land y = \text{“door”} \)
- \( x_{-1} = \text{“the”} \land y = \text{“table”} \)
- \( y \text{ occurs in } x \)
Features
Sometimes, we think of the input as having features, which are multiplied by outputs to form the candidates.

\[
\begin{align*}
x & \quad \ldots \text{win the election} \ldots \\
\text{``}f(x)\text{''} & \quad [1 \ 0 \ 1 \ 0] \\
\text{``win''} & \quad \text{``election''}
\end{align*}
\]

\[
\begin{align*}
f(SPORTS) & = [1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\
f(POLITICS) & = [0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0] \\
f(OTHER) & = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0]
\end{align*}
\]
Non-Block Feature Vectors

- Sometimes the features of candidates cannot be decomposed in this regular way.
- Example: a parse tree’s features may be the productions present in the tree.

\[ f(\text{NP N NP V VP N}) = [1, 0, 1, 0, 1] \]

\[ f(\text{NP N V NP V N}) = [1, 1, 0, 1, 0] \]

- Different candidates will thus often share features.
- We’ll return to the non-block case later.
Linear Models
Linear Models: Scoring

- In a linear model, each feature gets a weight $w$

\[
\begin{align*}
\text{... win the election ...} & \\
f(\text{POLITICS}) & = [0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0] \\
\text{... win the election ...} & \\
f(\text{SPORTS}) & = [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] \\
\text{... win the election ...} & \\
w & = [1, 1, -1, -2, 1, -1, 1, -2, -2, -1, -1, 1] \\
\end{align*}
\]

- We score hypotheses by multiplying features and weights:

\[
\text{score}(y, w) = w^\top f(y)
\]

\[
\begin{align*}
\text{... win the election ...} & \\
f(\text{POLITICS}) & = [0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0] \\
\text{... win the election ...} & \\
w & = [1, 1, -1, -2, 1, -1, 1, -2, -2, -1, -1, 1] \\
\text{... win the election ...} & \\
\text{score}(\text{POLITICS}, w) & = 1 \times 1 + 1 \times 1 = 2
\end{align*}
\]
Linear Models: Decision Rule

- The linear decision rule:

\[
prediction(\ldots \text{win the election} \ldots, w) = \arg \max_{y \in \mathcal{Y}(x)} w^\top f(y)
\]

\[
score(\text{SPORTS}, w) = 1 \times 1 + (-1) \times 1 = 0
\]

\[
score(\text{POLITICS}, w) = 1 \times 1 + 1 \times 1 = 2
\]

\[
score(\text{OTHER}, w) = (-2) \times 1 + (-1) \times 1 = -3
\]

\[
prediction(\ldots \text{win the election} \ldots, w) = \text{POLITICS}
\]

- We’ve said nothing about where weights come from
Binary Classification

- Important special case: binary classification
  - Classes are $y=+1/-1$
    - $f(x, -1) = -f(x, +1)$
    - $f(x) = 2f(x, +1)$
  - Decision boundary is a hyperplane
    - $w^T f(x) = 0$

```
<table>
<thead>
<tr>
<th>BIAS</th>
<th>free</th>
<th>money</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
```

Graph:
- $w^T f = 0$
- $+1 = \text{SPAM}$
- $-1 = \text{HAM}$
### Multiclass Decision Rule

- **If more than two classes:**
  - Highest score wins
  - Boundaries are more complex
  - Harder to visualize

\[
prediction(x_i, w) = \arg \max_{y \in Y} w^T f_i(y)
\]

- There are other ways: e.g. reconcile pairwise decisions
Learning
Two broad approaches to learning weights

Generative: work with a probabilistic model of the data, weights are (log) local conditional probabilities
- Advantages: learning weights is easy, smoothing is well-understood, backed by understanding of modeling

Discriminative: set weights based on some error-related criterion
- Advantages: error-driven, often weights which are good for classification aren’t the ones which best describe the data

We’ll mainly talk about the latter for now
How to pick weights?

- **Goal:** choose “best” vector \( w \) given training data
  - For now, we mean “best for classification”

- **The ideal:** the weights which have greatest test set accuracy / F1 / whatever
  - But, don’t have the test set
  - Must compute weights from training set

- **Maybe we want weights which give best training set accuracy?**
  - Hard discontinuous optimization problem
  - May not (does not) generalize to test set
  - Easy to overfit

*Though, min-error training for MT does exactly this.*
Minimize Training Error?

- A loss function declares how costly each mistake is

\[ \ell_i(y) = \ell(y, y_i^*) \]

- E.g. 0 loss for correct label, 1 loss for wrong label
- Can weight mistakes differently (e.g. false positives worse than false negatives or Hamming distance over structured labels)

- We could, in principle, minimize training loss:

\[ \min_w \sum_i \ell_i \left( \arg \max_y w^T f_i(y) \right) \]

- This is a hard, discontinuous optimization problem
The perceptron algorithm
- Iteratively processes the training set, reacting to training errors
- Can be thought of as trying to drive down training error

The (online) perceptron algorithm:
- Start with zero weights $w$
- Visit training instances one by one
  - Try to classify
    $$\hat{y} = \arg \max_{y \in \mathcal{Y}(x)} w^\top f(y)$$
  - If correct, no change!
  - If wrong: adjust weights
    $$w \leftarrow w + f(y_i^*)$$
    $$w \leftarrow w - f(\hat{y})$$
Example: “Best” Web Page

\[ w = [1 \ 2 \ 0 \ 0 \ ...] \]

\[ x_i = “Apple Computers” \]

\[ f_i( \ ) = [0.3 \ 5 \ 0 \ 0 \ ...] \quad w^\top f = 10.3 \quad \hat{y} \]

\[ f_i( \ ) = [0.8 \ 4 \ 2 \ 1 \ ...] \quad w^\top f = 8.8 \quad y_i^* \]

\[ w \leftarrow w + f(y_i^*) - f(\hat{y}) \]

\[ w = [1.5 \ 1 \ 2 \ 1 \ ...] \]
Examples: Perceptron

- Separable Case
Perceptrons and Separability

- A data set is separable if some parameters classify it perfectly.
- Convergence: if training data separable, perceptron will separate (binary case).
- Mistake Bound: the maximum number of mistakes (binary case) related to the margin or degree of separability.
Examples: Perceptron

- Non-Separable Case
Issues with Perceptrons

- **Overtraining:** test / held-out accuracy usually rises, then falls
  - Overtraining isn’t the typically discussed source of overfitting, but it can be important

- **Regularization:** if the data isn’t separable, weights often thrash around
  - Averaging weight vectors over time can help (averaged perceptron)
  - [Freund & Schapire 99, Collins 02]

- **Mediocre generalization:** finds a “barely” separating solution
Problems with Perceptrons

- Perceptron “goal”: separate the training data

\[ \forall i, \forall y \neq y^i \quad w^T f_i(y^i) \geq w^T f_i(y) \]

1. This may be an entire feasible space
2. Or it may be impossible
Margin
Objective Functions

- What do we want from our weights?
  - Depends!
  - So far: minimize (training) errors:
    \[ \sum_i \text{step} \left( w^T f_i(y_i^*) - \max_{y \neq y_i^*} w^T f_i(y) \right) \]
  - This is the “zero-one loss”
    - Discontinuous, minimizing is NP-complete
    - Not really what we want anyway
  - Maximum entropy and SVMs have other objectives related to zero-one loss
Linear Separators

- Which of these linear separators is optimal?
Classification Margin (Binary)

- Distance of $x_i$ to separator is its margin, $m_i$
- Examples closest to the hyperplane are support vectors
- Margin $\gamma$ of the separator is the minimum $m$
Classification Margin

- For each example $x_i$ and possible mistaken candidate $y$, we avoid that mistake by a margin $m_i(y)$ (with zero-one loss)
  
  \[ m_i(y) = w^T f_i(y_i^*) - w^T f_i(y) \]

- Margin $\gamma$ of the entire separator is the minimum $m$

  \[ \gamma = \min_i \left( w^T f_i(y_i^*) - \max_{y \neq y_i^*} w^T f_i(y) \right) \]

- It is also the largest $\gamma$ for which the following constraints hold

  \[ \forall i, \forall y \quad w^T f_i(y_i^*) \geq w^T f_i(y) + \gamma \ell_i(y) \]
Separable SVMs: find the max-margin $w$

\[
\max_{\|w\| = 1} \gamma \\
\forall i, \forall y \quad w^T f_i(y_i^*) \geq w^T f_i(y) + \gamma \ell_i(y)
\]

\[
\ell_i(y) = \begin{cases} 
0 & \text{if } y = y_i^* \\
1 & \text{if } y \neq y_i^*
\end{cases}
\]

- Can stick this into Matlab and (slowly) get an SVM
- Won’t work (well) if non-separable
Why do this? Various arguments:

- Solution depends only on the boundary cases, or support vectors (but remember how this diagram is broken!)
- Solution robust to movement of support vectors
- Sparse solutions (features not in support vectors get zero weight)
- Generalization bound arguments
- Works well in practice for many problems

Support vectors
Max Margin / Small Norm

- Reformulation: find the smallest $w$ which separates data

$$\forall i, y \quad w^T f_i(y_i^*) \geq w^T f_i(y) + \gamma \ell_i(y)$$

- $\gamma$ scales linearly in $w$, so if $||w||$ isn’t constrained, we can take any separating $w$ and scale up our margin

$$\gamma = \min_{i, y \neq y_i^*} \frac{[w^T f_i(y_i^*) - w^T f_i(y)]}{\ell_i(y)}$$

- Instead of fixing the scale of $w$, we can fix $\gamma = 1$

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

$$\forall i, y \quad w^T f_i(y_i^*) \geq w^T f_i(y) + \ell_i(y)$$
Soft Margin Classification

- What if the training set is not linearly separable?
- **Slack variables** $\xi_i$ can be added to allow misclassification of difficult or noisy examples, resulting in a *soft margin* classifier.
**Maximum Margin**

- **Non-separable SVMs**
  - Add slack to the constraints
  - Make objective pay (linearly) for slack:
    \[
    \min \frac{1}{2} \|w\|^2 + C \sum \xi_i
    \]
    \[
    \forall i, y, \quad w^\top f_i(y^*_i) + \xi_i \geq w^\top f_i(y) + \ell_i(y)
    \]
  - C is called the *capacity* of the SVM – the smoothing knob

- **Learning:**
  - Can still stick this into Matlab if you want
  - Constrained optimization is hard; better methods!
  - We’ll come back to this later

*Note: exist other choices of how to penalize slacks!*
Maximum Margin

![Graph showing Maximum Margin](image)
Likelihood
Linear Models: Maximum Entropy

- **Maximum entropy (logistic regression)**
  - Use the scores as probabilities:

\[
P(y|x, w) = \frac{\exp(w^T f(y))}{\sum_{y'} \exp(w^T f(y'))}
\]

- Maximize the (log) conditional likelihood of training data

\[
L(w) = \log \prod_i P(y_i^* | x_i, w) = \sum_i \log \left( \frac{\exp(w^T f_i(y_i^*))}{\sum_y \exp(w^T f_i(y))} \right)
\]

\[
= \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right)
\]
Maximum Entropy II

- **Motivation for maximum entropy:**
  - Connection to maximum entropy principle (sort of)
  - Might want to do a good job of being uncertain on noisy cases...
  - ... in practice, though, posteriors are pretty peaked

- **Regularization (smoothing)**

\[
\max_w \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right) - k ||w||^2
\]

\[
\min_w k ||w||^2 - \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right)
\]
Maximum Entropy
Loss Comparison
Log-Loss

- If we view maxent as a minimization problem:

\[
\min_w \ k||w||^2 + \sum_i - \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right)
\]

- This minimizes the “log loss” on each example

\[
- \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right) = -\log P(y_i^*|x_i, w)
\]

- One view: log loss is an upper bound on zero-one loss
Remember SVMs...

- We had a **constrained** minimization

\[
\min_{w, \xi} \frac{1}{2} ||w||^2 + C \sum_{i} \xi_i \\
\forall i, y, \quad w^T f_i (y_i^*) + \xi_i \geq w^T f_i (y) + \ell_i (y)
\]

- ...but we can solve for \( \xi_i \)

\[
\forall i, y, \quad \xi_i \geq w^T f_i (y) + \ell_i (y) - w^T f_i (y_i^*) \\
\forall i, \quad \xi_i = \max_y \left( w^T f_i (y) + \ell_i (y) \right) - w^T f_i (y_i^*)
\]

- Giving

\[
\min_w \frac{1}{2} ||w||^2 + C \sum_i \left( \max_y \left( w^T f_i (y) + \ell_i (y) \right) - w^T f_i (y_i^*) \right)
\]
Hinge Loss

- Consider the per-instance objective:

\[
\min_w k||w||^2 + \sum_i \left( \max_y (w^T f_i(y) + \ell_i(y)) - w^T f_i(y^*_i) \right)
\]

- This is called the “hinge loss”
  - Unlike maxent / log loss, you stop gaining objective once the true label wins by enough
  - You can start from here and derive the SVM objective
  - Can solve directly with sub-gradient decent (e.g. Pegasos: Shalev-Shwartz et al 07)

Plot really only right in binary case
Max vs “Soft-Max” Margin

- **SVMs:**

\[
\min_w k||w||^2 - \sum_i \left( w^T f_i(y_i^*) - \max_y \left( w^T f_i(y) + \ell_i(y) \right) \right)
\]

You can make this zero

- **Maxent:**

\[
\min_w k||w||^2 - \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp \left( w^T f_i(y) \right) \right)
\]

... but not this one

- Very similar! Both try to make the true score better than a function of the other scores
  - The SVM tries to beat the augmented runner-up
  - The Maxent classifier tries to beat the “soft-max”
Loss Functions: Comparison

- **Zero-One Loss**
  \[
  \sum_{i} \text{step} \left( w^T f_i(y_i^*) - \max_{y \neq y_i^*} w^T f_i(y) \right)
  \]

- **Hinge**
  \[
  \sum_{i} \left( w^T f_i(y_i^*) - \max_{y} (w^T f_i(y) + \ell_i(y)) \right)
  \]

- **Log**
  \[
  \sum_{i} \left( w^T f_i(y_i^*) - \log \sum_{y} \exp \left( w^T f_i(y) \right) \right)
  \]

\[
\text{w}^T f_i(y_i^*) - \max_{y \neq y_i^*} \left( w^T f_i(y) \right)
\]
Separators: Comparison
Conditional vs Joint Likelihood
### Example: Sensors

#### Reality

<table>
<thead>
<tr>
<th></th>
<th>Raining</th>
<th>Sunny</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(+,+,r)$</td>
<td>$3/8$</td>
<td></td>
</tr>
<tr>
<td>$P(-,-,r)$</td>
<td>$1/8$</td>
<td></td>
</tr>
<tr>
<td>$P(+,+,s)$</td>
<td>$1/8$</td>
<td></td>
</tr>
<tr>
<td>$P(-,-,s)$</td>
<td>$3/8$</td>
<td></td>
</tr>
</tbody>
</table>

#### NB Model

- **Raining?**
  - M1
  - M2

#### NB FACTORS:

- $P(s) = 1/2$
- $P(+|s) = 1/4$
- $P(+|r) = 3/4$

#### PREDICTIONS:

- $P(r,+,+) = (1/2)(3/4)(3/4)$
- $P(s,+,+) = (1/2)(1/4)(1/4)$
- $P(r|+,+) = 9/10$
- $P(s|+,+) = 1/10$
**Example: Stoplights**

**Reality**

- Lights Working
  - $P(g, r, w) = 3/7$
  - $P(r, g, w) = 3/7$
- Lights Broken
  - $P(r, r, b) = 1/7$

**NB Model**

- Working?
  - NS
  - EW

**NB FACTORS:**

- $P(w) = 6/7$
- $P(r | w) = 1/2$
- $P(g | w) = 1/2$
- $P(b) = 1/7$
- $P(r | b) = 1$
- $P(g | b) = 0$
Example: Stoplights

- What does the model say when both lights are red?
  - \( P(b, r, r) = \frac{1}{7}(1)(1) = \frac{1}{7} \)
  - \( P(w, r, r) = \frac{6}{7}(\frac{1}{2})(\frac{1}{2}) = \frac{6}{28} \)
  - \( P(w | r, r) = \frac{6}{10}! \)

- We’ll guess that \((r, r)\) indicates lights are working!

- Imagine if \(P(b)\) were boosted higher, to \(1/2\):
  - \( P(b, r, r) = \frac{1}{2}(1)(1) = \frac{1}{2} \)
  - \( P(w, r, r) = \frac{1}{2}(\frac{1}{2})(\frac{1}{2}) = \frac{1}{8} \)
  - \( P(w | r, r) = \frac{1}{5}! \)

- Changing the parameters bought accuracy at the expense of data likelihood