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)
### Shift-Reduce Parsers

- **Another way to derive a tree:**

```
<table>
<thead>
<tr>
<th>Stack</th>
<th>Remaining Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td></td>
</tr>
<tr>
<td>Det</td>
<td>the dog</td>
</tr>
<tr>
<td>V</td>
<td>saw</td>
</tr>
<tr>
<td>NP</td>
<td></td>
</tr>
<tr>
<td>Det</td>
<td>a man</td>
</tr>
<tr>
<td>P</td>
<td>in</td>
</tr>
<tr>
<td>NP</td>
<td>the park</td>
</tr>
</tbody>
</table>
```

- **Parsing**
  - No useful dynamic programming search
  - Can still use beam search [Ratnaparkhi 97]
Other Syntactic Models
- Lexicalized parsers can be seen as producing *dependency trees*

- Each local binary tree corresponds to an attachment in the dependency graph
### Dependency Parsing

- 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?)

- Examples:

  John ⊨ NP
  shares ⊨ NP
  buys ⊨ (S\NP)/NP
  sleeps ⊨ S\NP
  well ⊨ (S\NP)\( (S\NP)\)

\[
\begin{array}{c}
S \\
\downarrow \\
NP \quad S\NP
\end{array}
\begin{array}{c}
John \\
\downarrow \\
(S\NP)/NP \\
\downarrow \\
\downarrow \\
buys \\
\downarrow \\
shares
\end{array}
\]
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**

\( \mathbf{X}_i \) \hspace{1cm} close the ____

**CANDIDATE SET**

\( \mathcal{Y}(x) \) \hspace{1cm} \{door, table, ...\}

**CANDIDATES**

\( y \) \hspace{1cm} table

**TRUE OUTPUTS**

\( y^*_i \) \hspace{1cm} door

**FEATURE VECTORS**

\( f(x, y) \) \hspace{1cm} \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}

- \( x_{-1} = \text{“the”} \land y = \text{“door”} \)
- \( x_{-1} = \text{“the”} \land y = \text{“table”} \)
- \( \text{“close” in } x \land y = \text{“door”} \)
- \( y \text{ occurs in } x \)
Features
Feature Vectors

- Example: web page ranking (not actually classification)

\[ x_i = \text{“Apple Computers”} \]

\[ f_i(\text{Apple}) = [0.3 \ 5 \ 0 \ 0 \ \ldots] \]

\[ f_i(\text{Apple Inc.}) = [0.8 \ 4 \ 2 \ 1 \ \ldots] \]
Sometimes, we think of the input as having features, which are multiplied by outputs to form the candidates

\[
\mathbf{x} \quad \ldots \text{win the election} \ldots
\]

\[
\text{“f}(\mathbf{x})\text{”} \quad \begin{bmatrix}
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\text{“win”} \quad \text{“election”}
\]

\[
\text{f}(\text{SPORTS}) = \begin{bmatrix}
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
\text{f}(\text{POLITICS}) = \begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
\text{f}(\text{OTHER}) = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0
\end{bmatrix}
\]
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

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

- Different candidates will thus often share features.
- We’ll return to the non-block case later.
Linear Models
In a linear model, each feature gets a weight \( w \)

\[
\begin{align*}
\mathbf{f}(\textsc{Politics}) &= [0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0] \\
\mathbf{f}(\textsc{Sports}) &= [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] \\
\mathbf{w} &= [1, 1, -1, -2, 1, -1, 1, -2, -2, -1, -1, 1]
\end{align*}
\]

We score hypotheses by multiplying features and weights:

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

\[
\begin{align*}
\mathbf{f}(\textsc{Politics}) &= [0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0] \\
\mathbf{w} &= [1, 1, -1, -2, 1, -1, 1, -2, -2, -1, -1, 1]
\end{align*}
\]

\[
\text{score}(\textsc{Politics}, \mathbf{w}) = 1 \times 1 + 1 \times 1 = 2
\]
The linear decision rule:

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

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

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

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

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

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
    \[ w^T f(x) = 0 \]

\[ \begin{array}{l}
\text{BIAS} : -3 \\
\text{free} : 4 \\
\text{money} : 2 \\
\end{array} \]

\[ w^T f = 0 \]
If more than two classes:
- Highest score wins
- Boundaries are more complex
- Harder to visualize

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

There are other ways: e.g. reconcile pairwise decisions
Learning Classifier Weights

- 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
Linear Models: Perceptron

- **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
      \[
      \hat{y} = \arg \max_{y \in \mathcal{Y}(x)} w^T f(y)
      \]
    - If correct, no change!
    - If wrong: adjust weights
      \[
      w \leftarrow w + f(y_i^*)
      \]
      \[
      w \leftarrow w - f(\hat{y})
      \]
**Example: “Best” Web Page**

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

\[ x_i = “Apple Computers” \]

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

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

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

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

- Separable Case
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

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

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?
- 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 $w$

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

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

- Can stick this into Matlab and (slowly) get an SVM
- Won’t work (well) if non-separable
Why Max Margin?

- 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
Max Margin / Small Norm

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

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

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

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

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

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

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

$$\forall i, y \quad w^T f_i(y_i^*) \geq w^T f_i(y) + 1 \ell_i(y)$$
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.

---

**Diagram:**

- Red circles: Data points.
- Blue circles: Support vectors.
- Dashed lines: Non-linear boundaries.
- Red circles with blue circles: Slack variables $\xi_i$. 

---

Note: The diagram illustrates various instances of the soft margin classifier, showing how slack variables are used to account for non-linear separability.
Maximum Margin

- **Non-separable SVMs**
  - Add slack to the constraints
  - Make objective pay (linearly) for slack:
    \[
    \min_{\mathbf{w}, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i
    \]
    \[
    \forall i, y, \quad \mathbf{w}^T f_i(y^*_i) + \xi_i \geq \mathbf{w}^T f_i(y) + \ell_i(y)
    \]
  - 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
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
    \[
    L(w) = \log \prod_i P(y_i^*|x_i, w) = \sum_i \log \left( \frac{\exp(w^T f_i(y_i^*))}{\sum_y \exp(w^T f_i(y))} \right)
    \]
    \[
    = \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right)
    \]
  - Make positive
  - Normalize
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

\[
\min_{w} \frac{1}{2} ||w||^2 + C \sum_i \left( \max_y \left( w^T f_i(y) + \ell_i(y) \right) - w^T f_i(y^*_i) \right)
\]
Consider the per-instance objective:

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

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)
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**
  \[ \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp \left( w^T f_i(y) \right) \right) \]
Separators: Comparison
Conditional vs Joint Likelihood
Example: Sensors

**Reality**

<table>
<thead>
<tr>
<th>Raining</th>
<th>Sunny</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Cylinder" /></td>
<td><img src="image2.png" alt="Cylinder" /></td>
</tr>
<tr>
<td><img src="image3.png" alt="Cylinder" /></td>
<td><img src="image4.png" alt="Cylinder" /></td>
</tr>
</tbody>
</table>

- \( P(+,+,r) = \frac{3}{8} \)
- \( P(-,-,r) = \frac{1}{8} \)
- \( P(+,+,s) = \frac{1}{8} \)
- \( P(-,-,s) = \frac{3}{8} \)

**NB Model**

- **Raining?**
  - M1
  - M2

**NB FACTORS:**
- \( P(s) = \frac{1}{2} \)
- \( P(+|s) = \frac{1}{4} \)
- \( P(+|r) = \frac{3}{4} \)

**PREDICTIONS:**
- \( P(r,+,+) = \left(\frac{1}{2}\right)\left(\frac{3}{4}\right)\left(\frac{3}{4}\right) \)
- \( P(s,+,+) = \left(\frac{1}{2}\right)\left(\frac{1}{4}\right)\left(\frac{1}{4}\right) \)
- \( P(r|+,+) = \frac{9}{10} \)
- \( P(s|+,+) = \frac{1}{10} \)
Example: Stoplights

**Reality**

- **Lights Working**
  - $P(g,r,w) = \frac{3}{7}$
  - $P(r,g,w) = \frac{3}{7}$
- **Lights Broken**
  - $P(r,r,b) = \frac{1}{7}$

**NB Model**

- **Working?**
  - **NS**
  - **EW**

**NB FACTORS:**

- $P(w) = \frac{6}{7}$
- $P(r|w) = \frac{1}{2}$
- $P(g|w) = \frac{1}{2}$
- $P(b) = \frac{1}{7}$
- $P(r|b) = 1$
- $P(g|b) = 0$
Example: Stoplights

- What does the model say when both lights are red?
  - \( P(b, r, r) = (1/7)(1)(1) = 1/7 = 4/28 \)
  - \( P(w, r, r) = (6/7)(1/2)(1/2) = 6/28 = 6/28 \)
  - \( P(w | r, r) = 6/10! \)

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

- Imagine if \(P(b)\) were boosted higher, to 1/2:
  - \( P(b, r, r) = (1/2)(1)(1) = 1/2 = 4/8 \)
  - \( P(w, r, r) = (1/2)(1/2)(1/2) = 1/8 = 1/8 \)
  - \( P(w | r, r) = 1/5! \)

- Changing the parameters bought accuracy at the expense of data likelihood
Duals and Kernels
Nearest-Neighbor Classification

- Nearest neighbor, e.g. for digits:
  - Take new example
  - Compare to all training examples
  - Assign based on closest example

- Encoding: image is vector of intensities:
  \[ 1 = \langle 0.0 \ 0.0 \ 0.3 \ 0.8 \ 0.7 \ 0.1 \ldots 0.0 \rangle \]

- Similarity function:
  - E.g. dot product of two images’ vectors
  \[ \text{sim}(x, y) = x^\top y = \sum_i x_i y_i \]
Non-Parametric Classification

- Non-parametric: more examples means (potentially) more complex classifiers

- How about K-Nearest Neighbor?
  - We can be a little more sophisticated, averaging several neighbors
  - But, it’s still not really error-driven learning
  - The magic is in the distance function

- Overall: we can exploit rich similarity functions, but not objective-driven learning
A Tale of Two Approaches...

- Nearest neighbor-like approaches
  - Work with data through similarity functions
  - No explicit “learning”

- Linear approaches
  - Explicit training to reduce empirical error
  - Represent data through features

- Kernelized linear models
  - Explicit training, but driven by similarity!
  - Flexible, powerful, very very slow
The Perceptron, Again

- Start with zero weights
- Visit training instances one by one
  - Try to classify

\[ \hat{y} = \arg \max_{y \in \mathcal{Y}(x)} w^T f_i(y) \]

- If correct, no change!
- If wrong: adjust weights

\[ w \leftarrow w + f_i(y^*_i) \]
\[ w \leftarrow w - f_i(\hat{y}) \]
\[ w \leftarrow w + (f_i(y^*_i) - f_i(\hat{y})) \]
\[ w \leftarrow w + \Delta_i(\hat{y}) \]

*mistake vectors*
Perceptron Weights

- What is the final value of $w$?
  - Can it be an arbitrary real vector?
  - No! It’s built by adding up feature vectors (mistake vectors).

\[
\begin{align*}
  w &= \Delta_i(y) + \Delta_{i'}(y') + \cdots \\
  w &= \sum_{i,y} \alpha_i(y) \Delta_i(y) & \text{mistake counts}
\end{align*}
\]

- Can reconstruct weight vectors (the primal representation) from update counts (the dual representation) for each $i$

\[
\alpha_i = \langle \alpha_i(y_1), \alpha_i(y_2), \ldots, \alpha_i(y_n) \rangle
\]
Dual Perceptron

- Track mistake counts rather than weights
- Start with zero counts ($\alpha$)
- For each instance $x$
  - Try to classify
  
  $$\hat{y} = \arg \max_{y \in \mathcal{Y}(x_i)} \sum_{i', y'} \alpha_{i'}(y') \Delta_{i'}(y')^\top f_i(y)$$

- If correct, no change!
- If wrong: raise the mistake count for this example and prediction

$$\alpha_i(\hat{y}) \leftarrow \alpha_i(\hat{y}) + 1$$

$$w \leftarrow w + \Delta_i(\hat{y})$$
Dual / Kernelized Perceptron

- How to classify an example $x$?

$$score(y) = w^\top f_i(y) = \left( \sum_{i',y'} \alpha_{i'}(y') \Delta_{i'}(y') \right)^\top f_i(y)$$

$$= \sum_{i',y'} \alpha_{i'}(y') \left( \Delta_{i'}(y')^\top f_i(y) \right)$$

$$= \sum_{i',y'} \alpha_{i'}(y') \left( f_{i'}(y^*_i)^\top f_i(y) - f_{i'}(y')^\top f_i(y) \right)$$

$$= \sum_{i',y'} \alpha_{i'}(y') \left( K(y^*_i, y) - K(y', y) \right)$$

- If someone tells us the value of $K$ for each pair of candidates, never need to build the weight vectors
Issues with Dual Perceptron

- Problem: to score each candidate, we may have to compare to all training candidates

\[ \text{score}(y) = \sum_{i', y'} \alpha_{i'}(y') \left( K(y^*_i, y) - K(y', y) \right) \]

- Very, very slow compared to primal dot product!
- One bright spot: for perceptron, only need to consider candidates we made mistakes on during training
- Slightly better for SVMs where the alphas are (in theory) sparse

- This problem is serious: fully dual methods (including kernel methods) tend to be extraordinarily slow
- Of course, we can (so far) also accumulate our weights as we go...
Kernels: Who Cares?

- So far: a very strange way of doing a very simple calculation

- “Kernel trick”: we can substitute any* similarity function in place of the dot product

- Lets us learn new kinds of hypotheses

* Fine print: if your kernel doesn’t satisfy certain technical requirements, lots of proofs break. E.g. convergence, mistake bounds. In practice, illegal kernels sometimes work (but not always).
Some Kernels

- Kernels **implicitly** map original vectors to higher dimensional spaces, take the dot product there, and hand the result back.

- Linear kernel:
  \[ K(x, x') = x' \cdot x' = \sum_{i} x_i x'_i \]

- Quadratic kernel:
  \[ K(x, x') = (x \cdot x' + 1)^2 \]
  \[ = \sum_{i,j} x_i x_j x'_i x'_j + 2 \sum_{i} x_i x'_i + 1 \]

- RBF: infinite dimensional representation
  \[ K(x, x') = \exp(-||x - x'||^2) \]

- Discrete kernels: e.g. string kernels, tree kernels
Want to compute number of common subtrees between $T, T'$

Add up counts of all pairs of nodes $n, n'$

- Base: if $n, n'$ have different root productions, or are depth 0:
  \[
  C(n_1, n_2) = 0
  \]

- Base: if $n, n'$ are share the same root production:
  \[
  C(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} \left(1 + C(ch(n_1, j), ch(n_2, j))\right)
  \]
Dual Formulation for SVMs

- We want to optimize: (separable case for now)

\[
\min_w \quad \frac{1}{2}||w||^2
\]

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

- This is hard because of the constraints
- Solution: method of Lagrange multipliers
- The *Lagrangian* representation of this problem is:

\[
\min_w \max_{\alpha \geq 0} \quad \Lambda(w, \alpha) = \frac{1}{2}||w||^2 - \sum_{i, y} \alpha_i(y) \left( w^T f_i(y_i^*) - w^T f_i(y) - \ell_i(y) \right)
\]

- All we’ve done is express the constraints as an adversary which leaves our objective alone if we obey the constraints but ruins our objective if we violate any of them
Lagrange Duality

- We start out with a constrained optimization problem:
  \[ f(w^*) = \min_w f(w) \]
  \[ g(w) \geq 0 \]

- We form the Lagrangian:
  \[ \Lambda(w, \alpha) = f(w) - \alpha g(w) \]

- This is useful because the constrained solution is a saddle point of \( \Lambda \) (this is a general property):
  \[ f(w^*) = \min_w \max_{\alpha \geq 0} \Lambda(w, \alpha) = \max_{\alpha \geq 0} \min_w \Lambda(w, \alpha) \]

- **Primal problem in** \( w \)
- **Dual problem in** \( \alpha \)
Duality tells us that

\[
\min_w \max_{\alpha \geq 0} \quad \frac{1}{2} \|w\|^2 - \sum_{i,y} \alpha_i(y) \left( w^T f_i(y_i) - w^T f_i(y) - \ell_i(y) \right)
\]

has the same value as

\[
Z(\alpha)
\]

\[
\max_{\alpha \geq 0} \quad \min_w \quad \frac{1}{2} \|w\|^2 - \sum_{i,y} \alpha_i(y) \left( w^T f_i(y_i) - w^T f_i(y) - \ell_i(y) \right)
\]

This is useful because if we think of the \(\alpha\)'s as constants, we have an unconstrained min in \(w\) that we can solve analytically.

Then we end up with an optimization over \(\alpha\) instead of \(w\) (easier).
Minimize the Lagrangian for fixed $\alpha$’s:

\[
\Lambda(w, \alpha) = \frac{1}{2}||w||^2 - \sum_{i,y} \alpha_i(y) \left( w^T f_i(y_i^*) - w^T f_i(y) - \ell_i(y) \right)
\]

\[
\frac{\partial \Lambda(w, \alpha)}{\partial w} = w - \sum_{i,y} \alpha_i(y) \left( f_i(y_i^*) - f_i(y) \right)
\]

\[
\frac{\partial \Lambda(w, \alpha)}{\partial w} = 0 \quad \Rightarrow \quad w = \sum_{i,y} \alpha_i(y) \left( f_i(y_i^*) - f_i(y) \right)
\]

So we have the Lagrangian as a function of only $\alpha$’s:

\[
\min_{\alpha \geq 0} Z(\alpha) = \frac{1}{2} \left\| \sum_{i,y} \alpha_i(y) \left( f_i(y_i^*) - f_i(y) \right) \right\|^2 - \sum_{i,y} \alpha_i(y) \ell_i(y)
\]
Back to Learning SVMs

- We want to find $\alpha$ which minimize

$$\min_{\alpha \geq 0} \lambda(\alpha) = \frac{1}{2} \left\| \sum_{i,y} \alpha_i(y) \left( f_i(y^i) - f_i(y) \right) \right\|^2 - \sum_{i,y} \alpha_i(y) \ell_i(y)$$

$$\forall i, \sum_y \alpha_i(y) = C$$

- This is a quadratic program:
  - Can be solved with general QP or convex optimizers
  - But they don’t scale well to large problems
  - Cf. maxent models work fine with general optimizers (e.g. CG, L-BFGS)

- How would a special purpose optimizer work?
Coordinate Descent I

\[
\min_{\alpha \geq 0} Z(\alpha) = \min_{\alpha \geq 0} \frac{1}{2} \left\| \sum_{i,y} \alpha_i(y) (f_i(y_i^*) - f_i(y)) \right\|^2 - \sum_{i,y} \alpha_i(y) \ell_i(y)
\]

- Despite all the mess, \( Z \) is just a quadratic in each \( \alpha_i(y) \)
- Coordinate descent: optimize one variable at a time

- If the unconstrained argmin on a coordinate is negative, just clip to zero...
Ordinarily, treating coordinates independently is a bad idea, but here the update is very fast and simple

\[
\alpha_i(y) \leftarrow \max \left( 0, \alpha_i(y) + \frac{\ell_i(y) - w^T (f_i(y^*_i) - f_i(y))}{\| (f_i(y^*_i) - f_i(y)) \|^2} \right)
\]

So we visit each axis many times, but each visit is quick

This approach works fine for the separable case

For the non-separable case, we just gain a simplex constraint and so we need slightly more complex methods (SMO, exponentiated gradient)

\[
\forall i, \sum_y \alpha_i(y) = C
\]
What are the Alphas?

- Each candidate corresponds to a primal constraint

\[
\min_{\mathbf{w}, \xi} \frac{1}{2}||\mathbf{w}||^2 + C \sum_i \xi_i \\
\forall i, y \quad \mathbf{w}^\top \mathbf{f}_i(y^*) \geq \mathbf{w}^\top \mathbf{f}_i(y) + \ell_i(y) - \xi_i
\]

- In the solution, an \( \alpha_i(y) \) will be:
  - Zero if that constraint is inactive
  - Positive if that constraint is active
  - i.e. positive on the support vectors

- Support vectors contribute to weights:

\[
\mathbf{w} = \sum_{i, y} \alpha_i(y) \left( \mathbf{f}_i(y^*) - \mathbf{f}_i(y) \right)
\]
Structure
Handwriting recognition

Sequential structure

[Slides: Taskar and Klein 05]
The screen was a sea of red
What is the anticipated cost of collecting fees under the new proposal?

En vertu de nouvelle propositions, quel est le coût prévu de perception de les droits?

Combinatorial structure
Structured Models

\[ \text{prediction}(x, w) = \arg \max_{y \in \mathcal{Y}(x)} \text{score}(y, w) \]

Assumption:

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

Score is a sum of local “part” scores
Parts = nodes, edges, productions
CFG Parsing

\[ P(y \mid x) \propto \prod_{A \rightarrow \alpha \in (x,y)} \phi(A \rightarrow \alpha) \]

\[ \prod_{A \rightarrow \alpha \in (x,y)} \exp \left\{ w^\top f(A \rightarrow \alpha) \right\} = \exp \left\{ w^\top f(x, y) \right\} \]
Bilingual word alignment

\[ \sum_{y_{jk} \in y} w^\top f(x_{jk}) = w^\top f(x, y) \]

- What is the anticipated cost of collecting fees under the new proposal?

- En vertu de les nouvelles propositions, quel est le coût prévu de perception de le droits?

- association
- position
- orthography
Option 0: Reranking

Input

N-Best List
(e.g. n=100)

Output

\[ x = \text{"The screen was a sea of red."} \]

Baseline Parser

Non-Structured Classification

[e.g. Charniak and Johnson 05]
Reranking

- **Advantages:**
  - Directly reduce to non-structured case
  - No locality restriction on features

- **Disadvantages:**
  - Stuck with errors of baseline parser
  - Baseline system must produce n-best lists
  - But, feedback is possible [McCloskey, Charniak, Johnson 2006]
Efficient Primal Decoding

- Common case: you have a black box which computes

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

at least approximately, and you want to learn \( w \)

- Many learning methods require more (expectations, dual representations, k-best lists), but the most commonly used options do not

- Easiest option is the structured perceptron [Collins 01]
  - Structure enters here in that the search for the best \( y \) is typically a combinatorial algorithm (dynamic programming, matchings, ILPs, A*...)
  - Prediction is structured, learning update is not
Structured Margin

- Remember the margin objective:

\[
\min_w \frac{1}{2} \|w\|^2 \\
\forall i, y \quad w^\top f_i(y_i^*) \geq w^\top f_i(y) + \ell_i(y)
\]

- This is still defined, but lots of constraints
We want:

\[ \arg \max_y w^T f(\text{brace}, y) = \text{“brace”} \]

Equivalently:

\[
\begin{align*}
  w^T f(\text{brace}, \text{“brace”}) & > w^T f(\text{brace}, \text{“aaaaa”}) \\
  w^T f(\text{brace}, \text{“brace”}) & > w^T f(\text{brace}, \text{“aaaab”}) \\
  \vdots \\
  w^T f(\text{brace}, \text{“brace”}) & > w^T f(\text{brace}, \text{“zzzzz”})
\end{align*}
\]
We want:

\[ \arg \max_y w^\top f('It was red', y) = \hat{S}_{ABCD} \]

Equivalently:

\[ w^\top f('It was red', \hat{S}_{ABCD}) > w^\top f('It was red', \hat{S}_{ABDF}) \]
\[ w^\top f('It was red', \hat{S}_{ABCD}) > w^\top f('It was red', \hat{S}_{ABC}) \]
\[ \ldots \]
\[ w^\top f('It was red', \hat{S}_{ABCD}) > w^\top f('It was red', \hat{S}_{EFGH}) \]
Alignment example

- We want:

\[
\arg \max_y \ w^\top f(\text{'What is the'}, y) = \begin{array}{c}
1 \\
2 \\
3
\end{array}
\]

- Equivalently:

\[
\begin{align*}
& w^\top f(\text{'What is the'}, \begin{array}{c}1 \\
2 \\
3\end{array}) > w^\top f(\text{'Quel est le'}, \begin{array}{c}1 \\
2 \\
3\end{array}) \\
& w^\top f(\text{'What is the'}, \begin{array}{c}1 \\
2 \\
3\end{array}) > w^\top f(\text{'Quel est le'}, \begin{array}{c}1 \\
2 \times \\
3\end{array}) \\
& \ldots \\
& w^\top f(\text{'What is the'}, \begin{array}{c}1 \\
2 \\
3\end{array}) > w^\top f(\text{'Quel est le'}, \begin{array}{c}1 \\
2 \times \\
3\end{array})
\end{align*}
\]

\{a lot!\}
A constraint induction method [Joachims et al 09]
- Exploits that the number of constraints you actually need per instance is typically very small
- Requires (loss-augmented) primal-decode only

Repeat:
- Find the most violated constraint for an instance:
  \[ \forall y \quad w^T f_i(y^*) \geq w^T f_i(y) + \ell_i(y) \]
  \[ \arg \max_y w^T f_i(y) + \ell_i(y) \]
- Add this constraint and resolve the (non-structured) QP (e.g. with SMO or other QP solver)
Some issues:

- Can easily spend too much time solving QPs
- Doesn’t exploit shared constraint structure
- In practice, works pretty well; fast like MIRA, more stable, no averaging
M3Ns

- **Another option:** express all constraints in a packed form
  - Maximum margin Markov networks [Taskar et al 03]
  - Integrates solution structure deeply into the problem structure

- **Steps**
  - Express inference over constraints as an LP
  - Use duality to transform minimax formulation into min-min
  - Constraints factor in the dual along the same structure as the primal; alphas essentially act as a dual “distribution”
  - Various optimization possibilities in the dual
Likelihood, Structured

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

\[ \frac{\partial L(w)}{\partial w} = -2kw + \sum_i \left( f_i(y_i^*) - \sum_y P(y|x_i)f_i(y) \right) \]

- **Structure needed to compute:**
  - Log-normalizer
  - Expected feature counts
    - E.g. if a feature is an indicator of DT-NN then we need to compute posterior marginals \( P(DT-NN|\text{sentence}) \) for each position and sum

- Also works with latent variables (more later)