A Discriminative Approach

- View WSD as a discrimination task (regression, really)

\[ P(\text{sense} \mid \text{context: jail, context: county, context: feeding, ...}) \]
- Have to estimate multinomial (over senses) where there are a huge number of things to condition on
  - History is too complex to think about this as a smoothing / back-off problem
- Many feature-based classification techniques out there
  - We tend to need ones that output distributions over classes (why?)

Feature Representations

- Features are indicator functions \( f_i \) which count the occurrences of certain patterns in the input
- We map each input to a vector of feature predicate counts

\[
\begin{align*}
\text{context: jail} & = 1 \\
\text{context: county} & = 1 \\
\text{context: feeding} & = 1 \\
\text{context: game} & = 0 \\
\text{local-context: jail} & = 1 \\
\text{local-context: meals} & = 1 \\
\text{subcat: NP} & = 1 \\
\text{subcat: PP} & = 0 \\
\text{object-head: meals} & = 1 \\
\text{object-head: ball} & = 0
\end{align*}
\]

Example: Text Classification

- We want to classify documents into categories
  - Document length
  - Average word length
  - Document's source
  - Document layout

\[
\begin{array}{ll}
\text{DOCUMENT} & \text{CATEGORY} \\
\ldots \text{win the election} \ldots & \text{POLITICS} \\
\ldots \text{win the game} \ldots & \text{SPORTS} \\
\ldots \text{see a movie} \ldots & \text{OTHER}
\end{array}
\]

Some Definitions

- Sometimes, we want \( Y \) to depend on \( x \)

Block Feature Vectors

- Sometimes, we think of the input as having features, which are multiplied by outputs to form the candidates

\[
\begin{align*}
X & \ldots \text{win the election} \ldots \\
& \text{sports} \quad \text{politics} \quad \text{other} \\
& \text{DOCLEN} \\
& \text{AVGLEN} \\
& \text{SOURCE} \\
& \text{LAYOUT}
\end{align*}
\]

\[
\begin{align*}
f_i(x) & = [1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\
& \text{sports} = \text{"win"} \\
& \text{politics} = \text{"election"}
\end{align*}
\]

\[
\begin{align*}
f_i(\text{SPORTS}) & = [0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0] \\
f_i(\text{POLITICS}) & = [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0] \\
f_i(\text{OTHER}) & = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0]
\end{align*}
\]
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.

\[ f_1(\text{NP VP}) = [1 1 0 0] \]
\[ f_2(\text{NP VP}) = [1 0 1 0] \]

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

Linear Models: Scoring

- In a linear model, each feature gets a weight \( w \).

\[ f_1(\text{POLITICS}) = [0 0 0 0 1 0 1 0 0 0 0] \]
\[ f_2(\text{SPORTS}) = [1 0 1 0 0 0 0 0 0 0] \]
\[ w = [1 1 -1 -2 1 -1 1 -2 -2 -1 -1] \]

- We compare hypotheses on the basis of their linear scores:

\[ score(x^i, y, w) = w^T f_i(y) \]

\[ f_1(\text{POLITICS}) = [0 0 0 0 1 0 1 0 0 0 0] \]
\[ w = [1 1 -1 -2 1 -1 1 -2 -2 -1 -1] \]

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

Linear Models: Prediction Rule

- The linear prediction rule:

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

\[ score(x^i, \text{SPORTS}, w) = 1 \times 1 + (-1) \times 1 = 0 \]
\[ score(x^i, \text{POLITICS}, w) = 1 \times 1 + 1 \times 1 = 2 \]
\[ score(x^i, \text{OTHER}, w) = (-2) \times 1 + (-1) \times 1 = -3 \]

\[ prediction(x^i, w) = \text{POLITICS} \]

- We’ve said nothing about where weights come from!

Binary Decision Rule

- Heavily studied case: binary classification.

\[ prediction(x^i, w) = (w^T f_i > 0) \]

- Decision rule is a hyperplane.
- One side will be class 1.
- Other side will be class 0.

```
<table>
<thead>
<tr>
<th>CLASS</th>
<th>BIAS</th>
<th>free</th>
<th>money</th>
<th>the</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPAM</td>
<td>-3</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>HAM</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Multiclass Decision Rule

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

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

- 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.
Example: Stoplights

**Reality**

<table>
<thead>
<tr>
<th>Lights Working</th>
<th>Lights Broken</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Lights Working" /></td>
<td><img src="image2" alt="Lights Broken" /></td>
</tr>
</tbody>
</table>

P(g, r, w) = 3/7
P(r, g, w) = 3/7
P(r, r, b) = 1/7

**NB Model**

**Working?**

- P(w) = 6/7
- P(r) = 1/2
- P(g) = 1/2
- P(∅) = 0

**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) = (1/7)(1/1) = 1/7 = 4/28
  - P(w, r) = (6/7)(1/2)(1/2) = 6/28 = 6/28
  - P(w|b) = 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) = (1/2)(1/1) = 1/2 = 4/8
  - P(w, r) = (1/2)(1/2)(1/2) = 1/8 = 1/8
  - P(w|b) = 1/5!

- Non-generative values can give better classification

**Linear Models: Naïve-Bayes**

- (Multinomial) Naïve-Bayes is a linear model, where:

  \[ x^j = d_1, d_2, \ldots, d_n \]
  \[ w = [ \cdots, 0, \cdot, \log P(y), \log P(\eta_1 | y), \log P(\eta_2 | y), \ldots, \log P(\eta_m | y), \cdot, \cdots ] \]

  \[ \text{score}(x^j, y, w) = \log P(x^j | y) \]

  \[ = \log \left( \prod_{d \in x^j} P(d | y) \right) \]
  \[ = \log \prod_{d \in x^j} P(d | y)^{\#_d} \]
  \[ = \log P(y) + \sum_{d} \#_d \log P(d | y) \]
  \[ = w^T f(y) \]

**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

**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
  - Visit training instances one by one
  - Try to classify
    \[ y^* = \arg \max_y w^T f(y) \]
  - If correct, no change!
  - If wrong: adjust weights
    \[ w \leftarrow w + f(y^*) \]
    \[ w \leftarrow w - f(y^*) \]
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 quite as bad as overfitting, but is similar
- 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
  \[ \forall i, y_i \neq y^i \quad w^T f_i(y^i) \geq w^T f_i(y) \]
  
  1. This may be an entire feasible space
  2. Or it may be impossible

Objective Functions

- What do we want from our weights?
  - Depends!
  - So far: minimize (training) errors:
    \[ \sum \delta^{(i)} (w^T f_i(y^i) - \max_{y \neq y^i} w^T f_i(y)) \]
  - 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?

Linear Models: Maximum Entropy

- Maximum entropy (logistic regression)
  - Use the scores as probabilities:
    \[ P(y | x, w) = \frac{\exp(w^T f_i(y))}{\sum_y \exp(w^T f_i(y))} \]
  - Make positive
  - Normalize
  - Maximize the (log) conditional likelihood of training data
    \[ L(w) = \log \prod \left( \frac{\exp(w^T f_i(y))}{\sum_y \exp(w^T f_i(y))} \right) \]
    \[ = \sum \left( w^T f_i(y^i) - \log \sum_y \exp(w^T f_i(y)) \right) \]

Derivative for Maximum Entropy

\[ L(w) = \sum_i \left( w^T f_i(y^i) - \log \sum_y \exp(w^T f_i(y)) \right) \]
\[ \frac{\partial L(w)}{\partial w_n} = \sum_i \left( f_i(y^i)_n - \sum_y P(y | x, w)_n \right) \]

Expected Counts

- The optimum parameters are the ones for which each feature's predicted expectation equals its empirical expectation. The optimum distribution is:
  - Always unique (but parameters may not be unique)
  - Always exists (if features counts are from actual data).
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)

\[
\begin{align*}
\max_w & \quad \sum_y \left( w^T f(y') - \log \sum_y \exp(w^T f(y)) \right) - \lambda \|w\|^2 \\
\min_w & \quad \|w\|^2 - \sum_y \left( w^T f(y') - \log \sum_y \exp(w^T f(y)) \right)
\end{align*}
\]

Example: NER Smoothing

Because of smoothing, the more common prefixes have larger weights, even though entire-word features are more specific.

<table>
<thead>
<tr>
<th>Local Context</th>
<th>Feature Type</th>
<th>Feature</th>
<th>PERS</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Previous</td>
<td>at</td>
<td>0.85</td>
<td>0.84</td>
</tr>
<tr>
<td></td>
<td>Current</td>
<td>Grace</td>
<td>0.80</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>Beginning</td>
<td>&lt;G</td>
<td>0.45</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>Current POS</td>
<td>IN</td>
<td>0.47</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>Previous tags</td>
<td>NNP</td>
<td>0.10</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>Other</td>
<td>0.70</td>
<td>0.92</td>
</tr>
<tr>
<td></td>
<td>Current sign</td>
<td>Xr</td>
<td>0.80</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>Prev state</td>
<td>O-Xr</td>
<td>0.68</td>
<td>0.37</td>
</tr>
<tr>
<td></td>
<td>current sig</td>
<td>Xr-Xr</td>
<td>0.69</td>
<td>0.37</td>
</tr>
<tr>
<td></td>
<td>P. state</td>
<td>O-Xr-Xr</td>
<td>0.69</td>
<td>0.37</td>
</tr>
<tr>
<td></td>
<td>P. - current</td>
<td>O-Xr-Xr</td>
<td>0.20</td>
<td>0.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total:</td>
<td>-0.58</td>
<td>2.68</td>
</tr>
</tbody>
</table>

Log-Loss

- If we view maxent as a minimization problem:

\[
\min_w \quad \|w\|^2 - \sum_y \left( w^T f(y') - \log \sum_y \exp(w^T f(y)) \right)
\]

- This minimizes the "log loss" on each example

\[
- \left( w^T f(y') - \log \sum_y \exp(w^T f(y)) \right) = - \log P(y'|x, w)
\]

- One view: Log loss is an upper bound on zero-one loss
Derivative for Maximum Entropy

\[ L(w) = -\frac{1}{2}||w||^2 + \sum_{y} \left( w^T f(y) - \log \sum_{y'} \exp(w^T f(y')) \right) \]

\[ \frac{\partial L(w)}{\partial w_n} = -2w_n + \sum_{y} \left( f(y)_n - \sum_{y'} p(y|x) f(y')_n \right) \]

Big weights are bad

Expected count of feature \( n \) in predicted candidates

Total count of feature \( n \) in correct candidates

Unconstrained Optimization

- The maxent objective is an unconstrained optimization problem

\[ \nabla L(w) = 0 \]

- Basic idea: move uphill from current guess
- Gradient ascent / descent follows the gradient incrementally
- At local optimum, derivative vector is zero
- Will converge if step sizes are small enough, but not efficient
- All we need is to be able to evaluate the function and its derivative

Convexity

- The maxent objective is nicely behaved:
  - Differentiable (so many ways to optimize)
  - Convex (so no local optima)

\[ f(\lambda a + (1 - \lambda)b) \geq \lambda f(a) + (1 - \lambda)f(b) \]

Convex

Non-Convex

Convexity guarantees a single, global maximum value because any higher points are greedily reachable

Classification Margin (Binary)

- Distance of \( x_i \) to separator is its margin, \( w_i \)
- Examples closest to the hyperplane are support vectors
- Margin \( \gamma \) of the separator is the minimum \( w \)

\[ \gamma_i(y) = \min \{ w^T f(y') - w^T f(y) \} \]

\[ \gamma = \min_i \left( \min_{y'} w^T f(y') - w^T f(y) \right) \]

\[ \forall i, \forall y \quad w^T f(y') \geq w^T f(y) + \gamma_i(y) \]

Classification Margin

- For each example \( x_i \), and possible mistaken candidate \( y_i \), we avoid that mistake by a margin \( w_i(y) \) (with zero-one loss)

\[ m_i(y) = w_i^T f(y') - w_i^T f(y) \]

- Margin \( \gamma \) of the entire separator is the minimum \( w \)

\[ \gamma = \min \left( \min_{y} w^T f(y') - \min_{y'} w^T f(y) \right) \]

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

\[ \forall i, \forall y \quad w^T f(y') \geq w^T f(y) + \gamma_i(y) \]
Maximum Margin

- Separable SVMs: find the max-margin w
  \[
  \max_{|w|=1} \gamma \quad \text{s.t.} \quad \forall i, y_i = +1, w^T f_i(x_i) \geq \gamma + \xi_i
  \]
  \[
  \max_{|w|=1} \gamma \quad \text{s.t.} \quad \forall i, y_i = -1, w^T f_i(x_i) \geq -\gamma + \xi_i
  \]
- 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
  \[
  \max_{|w|=1} \gamma
  \]
  \[
  \forall i, y_i, w^T f_i(x_i) \geq \gamma + \xi_i
  \]
- \( \gamma \) scales linearly in w, so if |w| isn’t constrained, we can take any separating w and scale up our margin
  \[
  \gamma = \min_{i, y \neq y_i} (\|w\|^2 - w^T f_i(x_i))/\xi_i
  \]
- Instead of fixing the scale of w, we can fix \( \gamma = 1 \)
  \[
  \min \frac{1}{2} \|w\|^2
  \]
  \[
  \forall i, y \quad w^T f_i(x_i) \geq \gamma + \xi_i
  \]

Gamma to w

\[
\forall i, y\quad w^T f_i(x_i) \geq \gamma w^T f_i(x_i) + \xi_i
\]
\[
\gamma = 1/\|w\|^2
\]
\[
\max \frac{1}{2} \|w\|^2
\]
\[
\forall i, y\quad w^T f_i(x_i) \geq \gamma w^T f_i(x_i) + \xi_i
\]

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:
    \[
    \min_{w, \xi} \frac{1}{2} \|w\|^2 + C \sum_i \xi_i
    \]
    \[
    \forall i, y_i = +1, w^T f_i(x_i) + \xi_i \geq w^T f_i(x_i) + \xi_i
    \]
    \[
    \forall i, y_i = -1, w^T f_i(x_i) + \xi_i \geq w^T f_i(x_i) + \xi_i
    \]
  - 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 latter
Maximum Margin

We had a constrained minimization
\[ \min_{\mathbf{w}} \frac{1}{2} ||\mathbf{w}||^2 + C \sum_i \xi_i \]
\[ \forall i, y_i \mathbf{w}^T f_i(y_i) + \xi_i \geq \mathbf{w}^T f_i(y) + \xi_i \]
\[ \mathbf{w} \]...but we can solve for \( \xi_i \)
\[ \forall i, \xi_i \geq \mathbf{w}^T f_i(y) + \xi_i - \mathbf{w}^T f_i(y_i) \]
\[ \forall i, \xi_i = \max(\mathbf{w}^T f_i(y) + \xi_i) - \mathbf{w}^T f_i(y_i) \]

Giving
\[ \min_{\mathbf{w}} \frac{1}{2} ||\mathbf{w}||^2 - C \sum_i (\mathbf{w}^T f_i(y) - \max(\mathbf{w}^T f_i(y) + \xi_i)) \]

Hinge Loss

- Consider the per-instance objective:
  \[ \min \mathbf{w}||\mathbf{w}||^2 - \sum_i (\mathbf{w}^T f_i(y) - \max(\mathbf{w}^T f_i(y) + \xi_i)) \]
- 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

Max vs “Soft-Max” Margin

- SVMs:
  \[ \min \mathbf{w}||\mathbf{w}||^2 - \sum_i (\mathbf{w}^T f_i(y) - \max(\mathbf{w}^T f_i(y) + \xi_i)) \]
- Maxent:
  \[ \min \mathbf{w}||\mathbf{w}||^2 - \sum_i \mathbf{w}^T f_i(y) - \log \sum \exp(\mathbf{w}^T f_i(y)) \]
  You can make this zero

Loss Functions: Comparison

- Zero-One Loss
  \[ \sum_i \text{step}(\mathbf{w}^T f_i(y) - \max(\mathbf{w}^T f_i(y))) \]
- Hinge
  \[ \sum_i (\mathbf{w}^T f_i(y) - \max(\mathbf{w}^T f_i(y) + \xi_i)) \]
- Log
  \[ \sum_i (\mathbf{w}^T f_i(y) - \log \sum \exp(\mathbf{w}^T f_i(y))) \]

Separators: Comparison

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