Objectives

- Understand how the “vote” from the last lecture can be used in a high-powered classifier
- Introduce Maximum Entropy (MaxEnt) classifiers
- Lay the groundwork for the idea of “Feature Engineering”

Start with our Vote

- For a word \( w \), we take a weighted vote for each class (word sense):
  \[
  \text{vote}(c | "serve") \text{ in context of document } d = \sum_{i} \lambda_i(c) f_i(d)
  \]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Food</th>
<th>Law</th>
<th>Stere</th>
</tr>
</thead>
<tbody>
<tr>
<td>context jail</td>
<td>-0.5</td>
<td>+1.2</td>
<td>+0</td>
</tr>
<tr>
<td>subject NP</td>
<td>+0.0</td>
<td>+1.0</td>
<td>+0.8</td>
</tr>
<tr>
<td>object noun mark</td>
<td>+0.0</td>
<td>+1.0</td>
<td>+0.1</td>
</tr>
<tr>
<td>object head names</td>
<td>+1.8</td>
<td>+2.1</td>
<td>+0.0</td>
</tr>
<tr>
<td>TOTAL</td>
<td>+2.5</td>
<td>+0.7</td>
<td>+2.6</td>
</tr>
</tbody>
</table>

Truth about Vote

- Rather than:
  \[
  \text{vote}(c | "serve") \text{ in context of document } d = \sum_{i} \lambda_i(c) f_i(d)
  \]
- Actually:
  \[
  \text{vote}(c | "serve") \text{ in context of document } d = \sum_{i} \lambda_i(c) f_i(d, c)
  \]
How to Train?

vote("serve" in context of document $d$) = $\sum_{c=1}^{R} \lambda_c f_c(d, c)$

A perfectly useful classifier:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serve</td>
<td>2.5</td>
</tr>
<tr>
<td>Fork</td>
<td>1.0</td>
</tr>
<tr>
<td>Food</td>
<td>1.5</td>
</tr>
<tr>
<td>Tennis</td>
<td>0.7</td>
</tr>
</tbody>
</table>

But how do we train the weights $\lambda_c$?

We need to get back to our probabilistic foundations:
- Cast as a probabilistic model
- Then formulate a likelihood function we can maximize.

Assume Exponential Form

- Turn the votes into a probability distribution:
  $P(c | d, \lambda) = \frac{\exp \sum \lambda_c f_c(d, c)}{\sum_c \exp \sum \lambda_c f_c(d, c)}$  
  \[\text{Makes votes positive.}\]
- Not an arbitrary choice
  - For any weights $\lambda$, we get a conditional probability distribution $P(c | d, \lambda)$
  - But we’re looking for the best one!
- This distribution is known by various names:
  - Exponential, log-linear, MaxEnt, logistic regression, Gibbs

Choose the Parameters

- We want to choose parameters that maximize the (log) conditional likelihood of the labeled data:
  $L(\lambda) = \log P(\text{all labels}|\text{all documents}, \lambda)$
  \[\sum_{c \in C} \log \prod_{d \in D} P(c_d | d, \lambda) \]
- Assume the data instances are i.i.d. (independent and identically distributed)
  $L(\lambda) = \log \prod_{d \in D} P(c_d | d, \lambda)$

The Likelihood Function

- Assuming fixed data, the (log) conditional likelihood is a function of the parameters $\lambda$:
  $L(\lambda) = \log \prod_{d \in D} P(c_d | d, \lambda) = \sum_{d \in D} \log P(c_d | d, \lambda)$
- If there aren’t many values of class $c$, it’s easy to calculate:
  $L(\lambda) = \sum_{d \in D} \log \sum_c \exp \sum \lambda_c f_c(d, c)$
- We can separate this into two components:
  $L(\lambda) = \sum_{d \in D} \sum_c \lambda_c f_c(d, c) - \sum_{d \in D} \log \sum_c \exp \sum \lambda_c f_c(d, c)$
  $L(\lambda) = N(\lambda) - M(\lambda)$

Maximizing the Likelihood

- How does the likelihood $L(\lambda)$ change when we tweak $\lambda_c$ (infinitesimally)?
  - Use differential calculus to find out.
- Take the derivative of the objective function $L(\lambda)$ with respect to $\lambda_c$:
  $\frac{\partial}{\partial \lambda_c} L(\lambda) = \frac{\partial}{\partial \lambda_c} \sum_{d \in D} \log \prod_{c \in C} P(c_d | d, \lambda) = \sum_{d \in D} \frac{\partial}{\partial \lambda_c} \sum_{c \in C} \log P(c_d | d, \lambda)$
- To maximize, set the derivative to 0.
- Will lead us to data-driven constraints!

The Likelihood I: Numerator

For some feature $j$ and some class $c$,

$$\frac{\partial}{\partial \lambda_j} N(\lambda) = \sum_{d \in D} \lambda_j f_j(d, c)$$

$$= \sum_{d \in D} \sum_{c \in C} \frac{\partial}{\partial \lambda_j} \log P(c_d | d, \lambda)$$

$$= \sum_{d \in D} \sum_{c \in C} \frac{\partial}{\partial \lambda_j} \log P(c_d | d, \lambda)$$

E.g., we actually observed the word “fork” ($f$) near the “food” sense (c) of “serve” 3 times (say, twice in one example and once in another).
The Derivative II: Denominator

For some feature $j$ and some class $c$, the derivative of the log-likelihood with respect to the weight $w_j$ is given by:

$$\frac{\partial L(\theta)}{\partial w_j} = \frac{1}{N} \sum_{i=1}^{N} \sum_{d=1}^{D} \sum_{k} \lambda_k \frac{f_j(d,c)}{\sum_k \exp(\lambda_k f_j(d,c))}$$

The Gradient

$$\nabla L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial \lambda_1} & \cdots & \frac{\partial L(\theta)}{\partial \lambda_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial L(\theta)}{\partial \lambda_m} & \cdots & \frac{\partial L(\theta)}{\partial \lambda_k} \end{bmatrix}$$

Convex Optimization

- This likelihood function is provably convex!

Where does this lead us?

$$\frac{\partial \log P(c,d)}{\partial \lambda_j} = \text{empirical count}(f_j,c) - \text{predicted count}(f_j,c)$$

$$= \mathbb{E}[f_j(d,c)] - \frac{P(c|d)}{P(c)} f_j(d,c)$$

How to Train

- Initialize weights to starting value: $\hat{\lambda} = [1]$.
- Iterate:
  - Given current weights $\hat{\lambda}$, calculate the conditional likelihood of the data (the function to optimize):
    $$L(\theta) = \sum c \log P(c|x)$$
  - Compute the gradient of the likelihood $\nabla L(\theta)$ at current weights $\hat{\lambda}$.
  - Feed this into a numerical optimizer (L-BFGS) to compute new weight vector $\hat{\lambda}$.
- This is a quasi-Newton method that can be used for solving unconstrained nonlinear optimization problems.
- If stopping criterion is met (e.g., $\Delta L(\theta)$ is small), then stop.

How to Classify with MaxEnt

$$\hat{c} = \arg \max_c P(c|d,\hat{\lambda})$$

$$= \arg \max_c \frac{\exp \sum \lambda_k f_j(d,c)}{\sum_c \exp \sum \lambda_k f_j(d,c)}$$

$$= \arg \max_c \sum \lambda_k f_j(d,c)$$

$$= \arg \max_c \log \frac{\sum \exp \lambda_k f_j(d,c)}{\sum_c \exp \lambda_k f_j(d,c)}$$
Primary Differences

Naive-Bayes
- Trained to maximize joint likelihood of data & classes: $P(C, D)$
- Features assumed to supply independent evidence.
- Feature weights can be set independently, usually by counting and smoothing. Quickly.
- Not much knowledge required.

Maxent
- Trained to maximize the conditional likelihood of classes given data: $P(C | D)$
- Feature weights take feature dependence into account.
- Feature weights must be mutually estimated. Takes time!
- Good compromise between empiricism and rationalism.
- High bias, low variance, esp. when data is sparse.

Why Conditional Models?
Revisited

Consider this metaphor:
Which would you rather take: a multiple-choice exam or an essay exam?
- Joint, generative models are like essay questions.
- Conditional models are like multiple-choice questions.
- Discuss!

Next
- More insight into MaxEnt
- Discussion of Feature Engineering!