Introduction

• So far we’ve mainly looked at “generative models”
  - Language models, IBM alignment models, PCFGs
  - But there is much use of conditional or discriminative models in NLP, Speech, IR, and ML generally
  - Because:
    - They give high accuracy performance
    - They make it easy to incorporate lots of linguistically important features
    - They allow easy building of language independent, retargetable NLP modules

Joint vs. Conditional Models

• We have some data \((d, c)\) of paired observations \(d\) and hidden classes \(c\).
• Joint (generative) models place probabilities over both observed data and the hidden stuff:
  - They generate the observed data from the hidden stuff
  - At the classic 1990s StaNLP models:
    - n-gram language models, Naive Bayes classifiers, hidden Markov models, probabilistic context-free grammars, IBM machine translation alignment models

Today’s plan

1. Generative vs. discriminative models [15 mins]
2. Optimizing softmax/maxent model parameters [20 mins]
3. Named Entity Recognition [10 mins]
4. Maximum entropy sequence models [10 mins]
Conditional vs. Joint Likelihood

- A joint model gives probabilities $P(d,c) = P(c)P(d|c)$ and tries to maximize this joint likelihood.
  - It tends up trivial to choose weights; just count!
  - Relative frequencies give maximum joint likelihood on categorical data
- A conditional model gives probabilities $P(c|d)$. It models only the conditional probability of the class.
  - We seek to maximize conditional likelihood.
  - Harder to do (as we’ll see…)
  - More closely related to classification error.

Conditional models work well: Word Sense Disambiguation

<table>
<thead>
<tr>
<th>Training Set</th>
<th>Objective</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint Like.</td>
<td>86.8</td>
<td></td>
</tr>
<tr>
<td>Cond. Like.</td>
<td>98.5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Set</th>
<th>Objective</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint Like.</td>
<td>73.6</td>
<td></td>
</tr>
<tr>
<td>Cond. Like.</td>
<td>76.1</td>
<td></td>
</tr>
</tbody>
</table>

(Klein and Manning 2002, using Senseval 1 Data)

PCFGs Maximize Joint, not Conditional Likelihood

1. What parse for eat rice with chopsticks?
2. How can you get the other parse?

<table>
<thead>
<tr>
<th>46</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Based on an example by Mark Johnson

Maxent/Softmax Model Likelihood

- Maximum (Conditional) Likelihood Models
  - Given a model form, we choose values of parameters $\lambda$ to maximize the (conditional) likelihood of the data.
  - For any given feature weights, we can calculate:
    - Conditional likelihood of training data
      $\log P(C|D,\lambda) = \log \prod_{(d,c) \in D} P(c|d,\lambda) - \sum_{(d,c) \in D} \log P(c|d,\lambda)$
    - Derivative of the likelihood wrt each feature weight
The Likelihood Value

- The (log) conditional likelihood of iid* data \((C, D)\) according to a maxent model is a function of the data and the parameters \(\lambda\):
  \[
  \log P(C \mid D, \lambda) = \log \prod_{c \in C} P(c \mid d, \lambda) = \sum_{c \in C} \log P(c \mid d, \lambda)
  \]
  - If there aren’t many values of \(c\), it’s easy to calculate:
    \[
    \log P(C \mid D, \lambda) = \sum_{c \in C} \log \sum_{d \in D} \exp \sum \lambda_i f_i(c, d)
    \]

\*A fancy statistics term meaning “independent and identically distributed.” This normally needs to assume this for anything beyond the maxent models, even though it’s never quite true in practice.

The Derivative I: Numerator

\[
\frac{\partial N(\lambda)}{\partial \lambda_i} = \sum_{(c, d) \in C \times D} \frac{\partial}{\partial \lambda_i} \log \sum \lambda_i f_i(c, d) = \sum_{(c, d) \in C \times D} \lambda_i f_i(c, d)
\]

Derivative of the numerator is: the empirical count \((c, d)\)

The Derivative II: Denominator

\[
\frac{\partial M(\lambda)}{\partial \lambda_i} = \sum_{(c, d) \in C \times D} \frac{\partial}{\partial \lambda_i} \sum \lambda_i f_i(c, d) = \sum_{(c, d) \in C \times D} \lambda_i f_i(c, d)
\]

Finding the optimal parameters

- We want to choose parameters \(\lambda_0, \lambda_1, \lambda_2, \ldots\) that maximize the conditional log-likelihood of the training data:
  \[
  CLogLik(D) = \sum_{d \in D} \log P(c_i \mid d_i)
  \]
  - To be able to do that, we’ve worked out how to calculate the function value and its partial derivatives (its gradient)
Finding the optimal parameters

- Use your favorite numerical optimization package...
  - Commonly (and in our code), you minimize the negative of CLogLik
  1. Gradient descent (GD); Stochastic gradient descent (SGD)
  2. Iterative proportional fitting methods: Generalized Iterative Scaling (GIS) and Improved Iterative Scaling (IIS)
  3. Conjugate gradient (CG), perhaps with preconditioning
  4. Quasi-Newton methods – limited memory variable metric (LMVM) methods, in particular, L-BFGS

Named Entity Recognition (NER)

- A very important NLP sub-task: find and classify names in text, for example:
  - The decision by the independent MP Andrew Wilkie to withdraw his support for the minority Labor government sounded dramatic but it should not further threaten its stability. When, after the 2010 election, Wilkie, Rob Oakeshott, Tony Windsor and the Greens agreed to support Labor, they gave just two guarantees: confidence and supply.
**Named Entity Recognition (NER)**

- **The uses:**
  - Named entities can be indexed, linked off, etc.
  - Sentiment can be attributed to companies or products
  - A lot of relations (employs, won, born-in) are between named entities
  - For question answering, answers are often named entities.

- **Concretely:**
  - Many web pages tag various entities, with links to bio or topic pages, etc.
  - Reuters’ OpenCabinet, Evri, AlchemyAPI, Yahoo’s Term Extraction, ...
  - Apple/Google/Microsoft/… smart recognizers for document content

---

**The Named Entity Recognition Task**

We

should

show

Neha

Eric

King

assignment

<table>
<thead>
<tr>
<th>We</th>
<th>ORG</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>should</td>
<td>ORG</td>
<td>O</td>
</tr>
<tr>
<td>show</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Neha</td>
<td>PER</td>
<td>B-PER</td>
</tr>
<tr>
<td>Eric</td>
<td>PER</td>
<td>B-PER</td>
</tr>
<tr>
<td>King</td>
<td>PER</td>
<td>I-PER</td>
</tr>
<tr>
<td>%</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>assignment</td>
<td>ORG</td>
<td>O</td>
</tr>
</tbody>
</table>

---

**Precision/Recall/F1 for NER**

- Recall and precision are straightforward for tasks like IR and text categorization, where there is only one grain size (documents)
- The measure behaves a bit peculiarly for IE/NER when there are boundary errors (which are common):
  - First Bank of Chicago announced earnings …
  - This counts as both a false positive and a false negative
  - Selecting nothing would have been better
  - Some other metrics (e.g., MUC scorer) give partial credit (according to complex rules)

---

**Sequence problems**

- Many problems in NLP have data which is a sequence of characters, words, phrases, lines, or sentences …
- We can think of our task as one of labeling each item

---

**Maximum entropy sequence models**

- Maximum entropy Markov models (MEMMs) a.k.a. Conditional Markov models

---

**Named Entity Recognition Evaluation**

Task: Predict entities in a text

<table>
<thead>
<tr>
<th>Foreign</th>
<th>ORG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ministry</td>
<td>ORG</td>
</tr>
<tr>
<td>spokesman</td>
<td>O</td>
</tr>
<tr>
<td>Shen</td>
<td>PER</td>
</tr>
<tr>
<td>Guofang</td>
<td>PER</td>
</tr>
<tr>
<td>told</td>
<td>O</td>
</tr>
<tr>
<td>Reuters</td>
<td>ORG</td>
</tr>
</tbody>
</table>

- Standard evaluation is per entity, not per token
MEMM inference in systems

- For a Conditional Markov Model (CMM) a.k.a. a Maximum Entropy Markov Model (MEMM), the classifier makes a single decision at a time, conditioned on evidence from observations and previous decisions.
- A larger space of sequences is usually explored via search.

Local Context | Features
---|---
1 2 3 0 | 22.6
ORG ORG | 22.6
hasDigit? | true

(Bothwick 1999, Klein et al. 2003, etc.)

Example: NER

- Scoring individual labeling decisions is no more complex than standard classification decisions.
  - We have some assumed labels to use for prior positions.
  - We use features of those and the observed data (which can include current, previous, and next words) to predict the current label.
- Local context features:
  - \( W_0 \) and \( W_{+1} \)
  - \( W_{-1} \)
  - \( C_{-1} \)
  - \( C_{-2} \)
  - \( C_{-3} \)
  - \( O \)
  - \( ORG \)
  - \( ORG \)
  - \( ORG \)
  - \( ORG \)

Local Context | Decision Point
---|---
3 2 1 0 | 22.6
ORG ORG | 22.6
hasDigit? | true

Inference in Systems

- Sequence Level
  - Sequence Model
  - Inference
    - Features
  - Best Sequence
- Local Level
  - Sequence Model
  - Inference
    - Features
  - Best Sequence

Greedy Inference

- We just start at the left, and use our classifier at each position to assign a label.
- The classifier can depend on previous labeling decisions as well as observed data.
- Advantages:
  - Fast, no extra memory requirements.
  - Easy to implement.
  - With rich features including observations to the right, it can perform quite well.
- Disadvantages:
  - Greedy. We make commit errors we cannot recover from.

Beam Inference

- Beam inference:
  - At each position keep the top \( k \) complete sequences.
  - Extend each sequence in each local way.
  - The extensions compete for the \( k \) slots at the next position.
- Advantages:
  - Fast; beam sizes of 3–5 are almost as good as exact inference in many cases.
  - Easy to implement (no dynamic programming required).
- Disadvantages:
  - Inexact; the globally best sequence can fall off the beam.

Viterbi Inference

- Viterbi inference:
  - Dynamic programming or memorization.
  - Requires small window of state influence (e.g., past two states are relevant).
- Advantages:
  - Exact; the global best sequence is returned.
- Disadvantages:
  - Harder to implement long-distance state-state interactions (but beam inference tends not to allow long-distance resurrection of sequences anyway).
CRFs [Lafferty, Pereira, and McCallum 2001]

- Another sequence model: Conditional Random Fields (CRFs)
- A whole-sequence conditional model rather than a chaining of local models.

\[ P(c \mid d, \lambda) = \frac{\exp \left\{ \sum \lambda_f (c, d) \right\}}{Z(d)} \]

- The space of \( c \)'s is now the space of sequences.
  - But if the features remain local, the conditional sequence likelihood can be calculated exactly using dynamic programming.
- Training is slower, but CRFs avoid causal competition biases.
- These (or a variant using a max margin criterion) are seen as the state-of-the-art these days … but in practice they usually work much the same as MEMMs.

CoNLL 2003 NER shared task

Results on English Devset

<table>
<thead>
<tr>
<th>Overall</th>
<th>Loc</th>
<th>Misc</th>
<th>Org</th>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMM</td>
<td>83</td>
<td>85</td>
<td>87</td>
<td>93</td>
</tr>
<tr>
<td>CRF</td>
<td>83</td>
<td>86</td>
<td>89</td>
<td>94</td>
</tr>
<tr>
<td>MMNN</td>
<td>82</td>
<td>84</td>
<td>86</td>
<td>92</td>
</tr>
</tbody>
</table>

Smoothing/Priors/Regularization for Maxent Models

- Assume the following empirical distribution:
  - Heads: | Tails: | 1.0 | 0.0 |
- Features: [Heads], [Tails]
- We'll have the following softmax model distribution:
  \[
  p(\text{Heads}) = \frac{e^{\lambda \cdot \text{Heads}}}{e^{\lambda \cdot \text{Heads}} + e^{\lambda \cdot \text{Tails}}}
  \]
  \[
  p(\text{Tails}) = \frac{e^{\lambda \cdot \text{Tails}}}{e^{\lambda \cdot \text{Heads}} + e^{\lambda \cdot \text{Tails}}}
  \]
- Really, only one degree of freedom (\( \lambda = \lambda_{\text{Heads}} - \lambda_{\text{Tails}} \))

Smoothing: Issues of Scale

- Lots of features:
  - NLP maxent models can have ten million features.
  - Even storing a single array of parameter values can have a substantial memory cost.
- Lots of sparsity:
  - Overfitting very easy – we need smoothing!
  - Many features seen in training unlikely occur again at test time.
- Optimization problems:
  - Feature weights can become, and iterative solvers can take a long time to get to those minima.

Smoothing: Issues

- The data likelihood in this model is:
  \[
  \log P(h, t \mid \lambda) = h \lambda_{\text{Heads}} + t \lambda_{\text{Tails}}
  \]
  \[
  \log P(h, t \mid \lambda) = h \lambda - (t + h) \log(1 + e^\lambda)
  \]
Gaussian, or quadratic, or L1 priors:
- Intuition: parameters shouldn’t be large.
- Formalization: prior expectation that each parameter will be distributed according to a gaussian with mean $\mu$ and variance $\sigma^2$.
- $P(\lambda) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\lambda - \mu)^2}{2\sigma^2}\right)$
- Penalizes parameters for drifting too far from their mean prior value (usually $\mu=0$).
- $2\sigma^2=1$ works surprisingly well.

If we use gaussian priors / L2 regularization:
- Trade off some expectation matching for smaller parameters.
- When multiple features can be recruited to explain a datapoint, the more common ones generally receive more weight.
- Accuracy generally goes up!
- Change the objective:
  $\log P(C,\lambda | D) = \log P(C | D,\lambda) + \log P(\lambda)$
  $\log P(C,\lambda | D) = \sum_{i=1}^{N} P(C_i | D,\lambda) \sum_{j=1}^{M} \frac{\lambda_j^2}{2\sigma^2} + \lambda$  
- Change the derivative:
  $\delta \log P(C,\lambda | D) / \delta \lambda_i = \text{actual}(f_i,C) - \text{predicted}(f_i,\lambda) - \lambda_i / \sigma^2$

Gaussian, or quadratic, or L1 priors:
- Intuition: parameters shouldn’t be large.
- Formalization: prior expectation that each parameter will be distributed according to a gaussian with mean $\mu$ and variance $\sigma^2$.
- $P(\lambda) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\lambda - \mu)^2}{2\sigma^2}\right)$
- Penalizes parameters for drifting too far from their mean prior value (usually $\mu=0$).
- $2\sigma^2=1$ works surprisingly well.

If we use gaussian priors / L2 regularization:
- Trade off some expectation matching for smaller parameters.
- When multiple features can be recruited to explain a datapoint, the more common ones generally receive more weight.
- Accuracy generally goes up!
- Change the objective:
  $\log P(C,\lambda | D) = \log P(C | D,\lambda) + \log P(\lambda)$
  $\log P(C,\lambda | D) = \sum_{i=1}^{N} P(C_i | D,\lambda) \sum_{j=1}^{M} \frac{\lambda_j^2}{2\sigma^2} + \lambda$  
- Change the derivative:
  $\delta \log P(C,\lambda | D) / \delta \lambda_i = \text{actual}(f_i,C) - \text{predicted}(f_i,\lambda) - \lambda_i / \sigma^2$

Smoothing: Early Stopping
- In the 0/1 case, there were two problems:
  - The optimal value of $\lambda$ was 0, which is a huge trip for an optimization procedure.
  - The learned distribution is just as spiked as the empirical one – no smoothing.
- One way to solve both issues is to just stop the optimization early, after a few iterations:
  - The value of $\lambda$ will be finite (but presumably big)
  - The optimization won’t take forever (clearly)
  - Commonly used in early wasn’t work
  - Has been viewed in deep learning

Smoothing: Early Stopping
- In the 0/1 case, there were two problems:
  - The optimal value of $\lambda$ was 0, which is a huge trip for an optimization procedure.
  - The learned distribution is just as spiked as the empirical one – no smoothing.
- One way to solve both issues is to just stop the optimization early, after a few iterations:
  - The value of $\lambda$ will be finite (but presumably big)
  - The optimization won’t take forever (clearly)
  - Commonly used in early wasn’t work
  - Has been viewed in deep learning

Smoothing: Priors (MAP)
- What if we had a prior expectation that parameter values wouldn’t be very large?
- We could then balance evidence suggesting large parameters (or infinite) against our prior.
- The evidence would never totally defeat the prior, and parameters would be smoothed (and kept finite!).
- We can do this explicitly by changing the optimization objective to maximum posterior likelihood:
  $\log P(C,\lambda | D) = \log P(\lambda) + \log P(C | D,\lambda)$
  $P(C,\lambda | D)$
  Posterior
  Prior
  Evidence

Smoothing: Priors (MAP)
- What if we had a prior expectation that parameter values wouldn’t be very large?
- We could then balance evidence suggesting large parameters (or infinite) against our prior.
- The evidence would never totally defeat the prior, and parameters would be smoothed (and kept finite!).
- We can do this explicitly by changing the optimization objective to maximum posterior likelihood:
  $\log P(C,\lambda | D) = \log P(\lambda) + \log P(C | D,\lambda)$
  $P(C,\lambda | D)$
  Posterior
  Prior
  Evidence

Smoothing: Priors
- Gaussian, or quadratic, or L1 priors:
- Intuition: parameters shouldn’t be large.
- Formalization: prior expectation that each parameter will be distributed according to a gaussian with mean $\mu$ and variance $\sigma^2$.
- $P(\lambda) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\lambda - \mu)^2}{2\sigma^2}\right)$
- Penalizes parameters for drifting too far from their mean prior value (usually $\mu=0$).
- $2\sigma^2=1$ works surprisingly well.

Smoothing: Priors
- Gaussian, or quadratic, or L1 priors:
- Intuition: parameters shouldn’t be large.
- Formalization: prior expectation that each parameter will be distributed according to a gaussian with mean $\mu$ and variance $\sigma^2$.
- $P(\lambda) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\lambda - \mu)^2}{2\sigma^2}\right)$
- Penalizes parameters for drifting too far from their mean prior value (usually $\mu=0$).
- $2\sigma^2=1$ works surprisingly well.

Example: NER Smoothing
- Because of smoothing, the more common prefix and single tag features have larger weights even though entire word and tag pair features are more specific.

Local Context

<table>
<thead>
<tr>
<th>Feature Type</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>0.72</td>
<td>0.89</td>
<td></td>
</tr>
<tr>
<td>Prems, word</td>
<td>0.81</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td>Shape</td>
<td>0.84</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td>Previous state</td>
<td>0.70</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td>Current state</td>
<td>0.47</td>
<td>0.47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>0.14</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>0.71</td>
<td>0.82</td>
<td></td>
</tr>
</tbody>
</table>
Example: POS Tagging

- From (Toutanova et al., 2003):
  - Without smoothing:
    | Overall Accuracy | Unknown Word Acc |
    |------------------|------------------|
    | 96.54            | 85.20            |
  - With smoothing:
    | Overall Accuracy | Unknown Word Acc |
    |------------------|------------------|
    | 97.10            | 88.20            |

- Smoothing helps:
  - Smooths distributions.
  - Pushes weight onto more explanatory features.
  - Allows many features to be dropped safely into the mix.
  - Speeds up convergence (if both are allowed to converge)!

Smoothing / Regularization

- Talking of "priors" and "MAP estimation" is Bayesian language
- In frequentist statistics, people will instead talk about using "regularization", and in particular, a gaussian prior is "L2 regularization"
- The choice of names makes no difference to the math
- Recently, L1 regularization is also very popular
  - Gives sparse solutions — most parameters become zero (Yay!)
  - Harder optimization problem (non-continuous derivative)

Smoothing: Virtual Data

- Another option: smooth the data, not the parameters.
- Example:
  - Equivalent to adding two extra data points.
  - Similar to add-one smoothing for generative models.
  - For feature-based models, hard to know what artificial data to create.

Smoothing: Count Cutoffs

- In NLP, features with low empirical counts are often dropped.
  - Very weak and indirect smoothing method.
  - Equivalent to locking their weight to be zero.
  - Equivalent to assigning them gaussian priors with mean zero and variance zero.
  - Dropping low counts does remove the features which were most in need of smoothing...
  - ...and speeds up the estimation by reducing model size ...
  - ...but count cutoffs generally hurt accuracy in the presence of proper smoothing.
  - Don’t use count cutoffs unless necessary for memory usage reasons. Prefer L1 regularization for finding features to drop.