MaxEnt: Training, Smoothing, Tagging

Advanced Statistical Methods in NLP
Ling572
February 7, 2012
Roadmap

- Maxent:
  - Training

- Smoothing

- Case study:
  - POS Tagging (redux)
  - Beam search
Training
Training

- Learn $\lambda$'s from training data
Training

- Learn $\lambda$'s from training data
- Challenge: Usually can’t solve analytically
  - Employ numerical methods
Training

- Learn $\lambda$'s from training data
- Challenge: Usually can’t solve analytically
  - Employ numerical methods
- Main different techniques:
  - Generalized Iterative Scaling (GIS, Darroch & Ratcliff, ‘72)
  - Improved Iterative Scaling (IIS, Della Pietra et al, ‘95)
  - L-BFGS,......
Generalized Iterative Scaling

- **GIS Setup:**
  - GIS required constraint:
  - \[ \forall (x, y) \in (X, Y) \sum_{j=1}^{k} f_j(x, y) = C, \] where C is a constant
Generalized Iterative Scaling

- GIS Setup:
  - GIS required constraint:
  - \( \forall (x, y) \in (X, Y) \sum_{j=1}^{k} f_j(x, y) = C \), where \( C \) is a constant
  - If not, then set
Generalized Iterative Scaling

- GIS Setup:
  - GIS required constraint:
  
  \[ \forall (x, y) \in (X, Y) \sum_{j=1}^{k} f_j(x, y) = C \], where \( C \) is a constant

- If not, then set
  
  \[ C = \max_{(x_i, y_i) \in S} \sum_{j=1}^{k} f_j(x_i, y_i) \]
Generalized Iterative Scaling

- GIS Setup:
  - GIS required constraint:
    \[ \forall (x, y) \in (X, Y) \sum_{j=1}^{k} f_j(x, y) = C, \text{ where } C \text{ is a constant} \]
  - If not, then set
    \[ C = \max_{(x_i, y_i) \in S} \sum_{j=1}^{k} f_j(x_i, y_i) \]
  - and add a correction feature function \( f_{k+1} \):
    \[ \forall (x, y) \in (X, Y) f_{k+1}(x, y) = C - \sum_{j=1}^{k} f_j(x, y) \]
Generalized Iterative Scaling

- GIS Setup:
  - GIS required constraint:

\[ \forall (x, y) \in (X, Y) \sum_{j=1}^{k} f_j(x, y) = C, \text{ where } C \text{ is a constant} \]

- If not, then set

\[ C = \max_{(x_i, y_i) \in S} \sum_{j=1}^{k} f_j(x_i, y_i) \]

- and add a correction feature function \( f_{k+1} \):

\[ \forall (x, y) \in (X, Y) f_{k+1}(x, y) = C - \sum_{j=1}^{k} f_j(x, y) \]

- GIS also requires at least one active feature for any event
  - Default feature functions solve this problem
GIS Iteration

- Compute the empirical expectation
GIS Iteration

• Compute the empirical expectation
• Initialization: $\lambda_j^{(0)}$; set to 0 or some value
GIS Iteration

- Compute the empirical expectation
- Initialization: $\lambda_j^{(0)}$; set to 0 or some value
- Iterate until convergence for each $j$: 
GIS Iteration

- Compute the empirical expectation
- Initialization: $\lambda_j^{(0)}$; set to 0 or some value
- Iterate until convergence for each $j$:
  - Compute $p(y|x)$ under the current model
GIS Iteration

- Compute the empirical expectation
- Initialization: $\lambda_j^{(0)}$; set to 0 or some value
- Iterate until convergence for each $j$:
  - Compute $p(y|x)$ under the current model
  - Compute model expectation under current model
GIS Iteration

- Compute the empirical expectation
- Initialization: $\lambda_j^{(0)}$; set to 0 or some value
- Iterate until convergence for each $j$:
  - Compute $p(y|x)$ under the current model
  - Compute model expectation under current model
  - Update model parameters by weighted ratio of empirical and model expectations
GIS Iteration

- Compute 
  \[ d_j = E_p(f_j) = \frac{1}{N} \sum_{i=1}^{N} f_j(x_i, y_i) \]
GIS Iteration

- Compute $d_j = E_p(f_j) = \frac{1}{N} \sum_{i=1}^{N} f_j(x_i, y_i)$
- Initialization: $\lambda_j^{(0)}$; set to 0 or some value
GIS Iteration

- Compute
  \[ d_j = E_p(f_j) = \frac{1}{N} \sum_{i=1}^{N} f_j(x_i, y_i) \]

- Initialization: \( \lambda_j^{(0)} \); set to 0 or some value

- Iterate until convergence:
  - Compute
GIS Iteration

- Compute \( d_j = E_p(f_j) = \frac{1}{N} \sum_{i=1}^{N} f_j(x_i, y_i) \)
- Initialization: \( \lambda_j^{(0)} \); set to 0 or some value
- Iterate until convergence:
  - Compute \( p^{(n)}(y|x) = \frac{\sum_j \lambda_j f_j(x, y)}{Z} \)
GIS Iteration

- Compute $d_j = E_p(f_j) = \frac{1}{N} \sum_{i=1}^{N} f_j(x_i, y_i)$
- Initialization: $\lambda_j^{(0)}$; set to 0 or some value
- Iterate until convergence: $e \sum_j \lambda_j f_j(x, y)$
  - Compute $p^{(n)}(y | x) = \frac{1}{Z}$
  - Compute $E_{p^{(n)}}(f_j) = \frac{1}{N} \sum_{i=1}^{N} \sum_{y \in Y} p^{(n)}(y | x_i) f_j(x_i, y)$
GIS Iteration

- Compute \( d_j = E_p(f_j) = \frac{1}{N} \sum_{i=1}^{N} f_j(x_i, y_i) \)

- Initialization: \( \lambda_j^{(0)} \); set to 0 or some value

- Iterate until convergence:
  - Compute \( p^{(n)}(y|\mathbf{x}) = e^{\sum_j \lambda_j f_j(x,y)} / Z \)
  - Compute \( E_{p^{(n)}}(f_j) = \frac{1}{N} \sum_{i=1}^{N} \sum_{y \in Y} p^{(n)}(y|x_i) f_j(x_i, y) \)
  - Update \( \lambda_j^{(n+1)} = \lambda_j^{(n)} + \frac{1}{C} \left( \log \frac{d_j}{E_{p^{(n)}}(f_j)} \right) \)
Convergence

- Methods have convergence guarantees
Convergence

- Methods have convergence guarantees
- However, full convergence may take very long time
Convergence

- Methods have convergence guarantees
- However, full convergence may take very long time
  - Frequently use threshold

\[ L(p) = \sum_{(x,y) \in S} \tilde{p}(x, y) \log p(y \mid x) \]

\[ L(p^{(n)}) = \sum_{(x,y) \in S} \tilde{p}(x, y) \log p^{(n)}(y \mid x) \]
Convergence

- Methods have convergence guarantees
- However, full convergence may take very long time
  - Frequently use threshold

\[
L(p) = \sum_{(x,y) \in S} \tilde{p}(x,y) \log p(y \mid x)
\]

\[
L(p^{(n)}) = \sum_{(x,y) \in S} \tilde{p}(x,y) \log p^{(n)}(y \mid x)
\]

\[
L(p^{(n+1)}) - L(p^{(n)}) < \text{threshold}
\]

\[
\frac{L(p^{(n+1)}) - L(p^{(n)})}{L(p^{(n)})} < \text{threshold}
\]
Calculating LL(p)

- LL = 0

- For each sample x in the training data
  - Let y be the true label of x
  - prob = p(y|x)
  - LL += 1/N * prob
Running Time

- For each iteration the running time is:
Running Time

- For each iteration the running time is $O(NPA)$, where:
  - $N$: number of training instances
  - $P$: number of classes
  - $A$: Average number of active features for instance $(x,y)$
L-BFGS

- Limited-memory version of
  - Broyden–Fletcher–Goldfarb–Shanno (BFGS) method
L-BFGS

- Limited-memory version of
  - Broyden–Fletcher–Goldfarb–Shanno (BFGS) method

- Quasi-Newton method for unconstrained optimization
L-BFGS

- Limited-memory version of
  - Broyden–Fletcher–Goldfarb–Shanno (BFGS) method
- Quasi-Newton method for unconstrained optimization
- Good for optimization problems with many variables
L-BFGS

- Limited-memory version of Broyden–Fletcher–Goldfarb–Shanno (BFGS) method
- Quasi-Newton method for unconstrained optimization
- Good for optimization problems with many variables
- “Algorithm of choice” for MaxEnt and related models
L-BFGS

- References:
L-BFGS

- References:

- Implementations:
  - Java, Matlab, Python via scipy, R, etc
  - See Wikipedia page
Smoothing

Based on Klein & Manning, 2003; F. Xia
Smoothing

- Problems of scale:
Smoothing

- Problems of scale:
  - Large numbers of features
  - Some NLP problems in MaxEnt ➔ 1M features
  - Storage can be a problem
Smoothing

- Problems of scale:
  - Large numbers of features
    - Some NLP problems in MaxEnt ➔ 1M features
    - Storage can be a problem
  - Sparseness problems
    - Ease of overfitting
Smoothing

- Problems of scale:
  - Large numbers of features
    - Some NLP problems in MaxEnt → 1M features
      - Storage can be a problem
  
- Sparseness problems
  - Ease of overfitting

- Optimization problems
  - Features can be near infinite, take long time to converge
Smoothing

- Consider the coin flipping problem
- Three empirical distributions
- Models

From K&M ‘03
Need for Smoothing

- Two problems
Need for Smoothing

- Two problems
  - Optimization:
    - Optimal value of $\lambda$? $\infty$
    - Slow to optimize

From K&M ‘03
Need for Smoothing

- Two problems
  - Optimization:
    - Optimal value of $\lambda$?
    - Slow to optimize
  - No smoothing
    - Learned distribution just as spiky (K&M’03)

From K&M ‘03
Possible Solutions
Possible Solutions

- Early stopping
- Feature selection
- Regularization
Early Stopping

- Prior use of early stopping
Early Stopping

- Prior use of early stopping
  - Decision tree heuristics
Early Stopping

- Prior use of early stopping
  - Decision tree heuristics

- Similarly here
  - Stop training after a few iterations
  - $\lambda$ will have increased
  - Guarantees bounded, finite training time
Feature Selection

- Approaches:
Feature Selection

- Approaches:
  - Heuristic: Drop features based on fixed thresholds
    - i.e. number of occurrences
Feature Selection

- Approaches:
  - Heuristic: Drop features based on fixed thresholds
    - i.e. number of occurrences
  - Wrapper methods:
    - Add feature selection to training loop
Feature Selection

- Approaches:
  - Heuristic: Drop features based on fixed thresholds
    - i.e. number of occurrences
  - Wrapper methods:
    - Add feature selection to training loop

- Heuristic approaches:
  - Simple, reduce features, but could harm performance
Regularization

- In statistics and machine learning, regularization is any method of preventing overfitting of data by a model.
Regularization

- In statistics and machine learning, regularization is any method of preventing overfitting of data by a model.

- Typical examples of regularization in statistical machine learning include ridge regression, lasso, and L2-norm minimization support vector machines.

From K&M ’03, F. Xia
Regularization

- In statistics and machine learning, regularization is any method of preventing overfitting of data by a model.

- Typical examples of regularization in statistical machine learning include ridge regression, lasso, and L2-norm in support vector machines.

- In this case, we change the objective function:
  \[ \log P(Y, \lambda | X) = \log P(\lambda) + \log P(Y|X, \lambda) \]
Prior

- Possible prior distributions: uniform, exponential
Prior

- Possible prior distributions: uniform, exponential
- Gaussian prior:

\[ P(\lambda_i) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(\lambda_i - \mu_i)^2}{2\sigma^2}\right) \]
Prior

- Possible prior distributions: uniform, exponential
- Gaussian prior:

\[
P(\lambda_i) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(\lambda_i - \mu_i)^2}{2\sigma^2}\right)
\]

- \(\log P(Y, \lambda \mid X) = \log P(\lambda) + \log P(Y \mid X, \lambda)\)

\[
= \sum_{i=1}^{k} \log P(\lambda_i) + \log P(Y \mid X, \lambda)
\]

\[
= -k \log \sqrt{2\pi\sigma} - \sum_{i=1}^{k} \frac{(\lambda_i - \mu)^2}{2\sigma^2} + \log P(Y \mid X, \lambda)
\]
• Maximize $P(Y|X, \lambda )$

$$E_p(f_j) = E_{\tilde{p}}(f_j)$$

• Maximize $P(Y, \lambda |X)$

$$E_p(f_j) = E_{\tilde{p}}(f_j) - \frac{\lambda_j - \mu}{\sigma^2}$$

• In practice, $\mu = 0; 2\sigma^2 = 1$
L1 and L2 Regularization

\[ L_1 = \sum_i \log P(y_i, \lambda | x_i) - \frac{\|\lambda\|}{\sigma} \]

\[ L_2 = \sum_i \log P(y_i, \lambda | x_i) - \frac{\|\lambda\|^2}{\sigma} \]
Smoothing: POS Example

- From (Toutanova et al., 2003):
  
<table>
<thead>
<tr>
<th></th>
<th>Overall Accuracy</th>
<th>Unknown Word Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Smoothing</td>
<td>96.54</td>
<td>85.20</td>
</tr>
<tr>
<td>With Smoothing</td>
<td>97.10</td>
<td>88.20</td>
</tr>
</tbody>
</table>

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

- Smooths distributions
Advantages of Smoothing

- Smooths distributions
- Moves weight onto more informative features
Advantages of Smoothing

- Smooths distributions
- Moves weight onto more informative features
- Enables effective use of larger numbers of features
Advantages of Smoothing

- Smooths distributions
- Moves weight onto more informative features
- Enables effective use of larger numbers of features
- Can speed up convergence
Summary: Training

- Many training methods:
  - Generalized Iterative Scaling (GIS)

- Smoothing:
  - Early stopping, feature selection, regularization

- Regularization:
  - Change objective function – add prior
  - Common prior: Gaussian prior
  - Maximizing posterior not equivalent to max ent
MaxEnt POS Tagging
Notation

- (Ratnaparkhi, 1996)
- $h$: history $\Rightarrow x$
  - Word and tag history
- $t$: tag $\Rightarrow y$
POS Tagging Model

- \( P(t_1, \ldots, t_n | w_1, \ldots, w_n) \)
  \[
    = \prod_{i=1}^{n} P(t_i | w_1^n, t_i^{i-1})
  \]
  \[
    \approx \prod_{i=1}^{n} P(t_i | h_i)
  \]
  \[
    p(t | h) = \frac{p(t, h)}{\sum_{t' \in T} p(t', h)}
  \]

- where \( h_i = \{w_i, w_{i-1}, w_{i-2}, w_{i+1}, w_{i+2}, t_{i-1}, t_{i-2}\} \)
MaxEnt Feature Set

<table>
<thead>
<tr>
<th>Condition</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_i$ is not rare</td>
<td>$w_i = X$ &amp; $t_i = T$</td>
</tr>
<tr>
<td>$w_i$ is rare</td>
<td>$X$ is prefix of $w_i$, $</td>
</tr>
<tr>
<td></td>
<td>$X$ is suffix of $w_i$, $</td>
</tr>
<tr>
<td></td>
<td>$w_i$ contains number &amp; $t_i = T$</td>
</tr>
<tr>
<td></td>
<td>$w_i$ contains uppercase character &amp; $t_i = T$</td>
</tr>
<tr>
<td></td>
<td>$w_i$ contains hyphen &amp; $t_i = T$</td>
</tr>
<tr>
<td>$\forall w_i$</td>
<td>$t_{i-1} = X$ &amp; $t_i = T$</td>
</tr>
<tr>
<td></td>
<td>$t_{i-2}t_{i-1} = XY$ &amp; $t_i = T$</td>
</tr>
<tr>
<td></td>
<td>$w_{i-1} = X$ &amp; $t_i = T$</td>
</tr>
<tr>
<td></td>
<td>$w_{i-2} = X$ &amp; $t_i = T$</td>
</tr>
<tr>
<td></td>
<td>$w_{i+1} = X$ &amp; $t_i = T$</td>
</tr>
<tr>
<td></td>
<td>$w_{i+2} = X$ &amp; $t_i = T$</td>
</tr>
</tbody>
</table>
Example

- Feature for ‘about’

\[
\begin{align*}
    w_i &= \text{about} & t_i &= \text{IN} \\
    w_{i-1} &= \text{stories} & t_i &= \text{IN} \\
    w_{i-2} &= \text{the} & t_i &= \text{IN} \\
    w_{i+1} &= \text{well-heeled} & t_i &= \text{IN} \\
    w_{i+2} &= \text{communities} & t_i &= \text{IN} \\
    t_{i-1} &= \text{NNS} & t_i &= \text{IN} \\
    t_{i-2} t_{i-1} &= \text{DT NNS} & t_i &= \text{IN}
\end{align*}
\]

Exclude features seen < 10 times
Training

- GIS

- Training time: $O(NTA)$
  - N: training set size
  - T: number of tags
  - A: average number of features active for event $(h,t)$

- 24 hours on a ‘96 machine
Finding Features

- In training, where do features come from?
- Where do features come from in testing?

<table>
<thead>
<tr>
<th></th>
<th>(w_{-1})</th>
<th>(w_0)</th>
<th>(w_{-1}w_0)</th>
<th>(w_{+1})</th>
<th>(t_{-1})</th>
<th>(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1) (Time)</td>
<td>(&lt;s&gt;)</td>
<td>Time</td>
<td>(&lt;s&gt;Time)</td>
<td>flies</td>
<td>BOS</td>
<td>N</td>
</tr>
<tr>
<td>(x_2) (flies)</td>
<td>Time</td>
<td>flies</td>
<td>Time flies</td>
<td>like</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>(x_3) (like)</td>
<td>flies</td>
<td>like</td>
<td>flies like</td>
<td>an</td>
<td>N</td>
<td>V</td>
</tr>
</tbody>
</table>
Finding Features

- In training, where do features come from?
- Where do features come from in testing?
  - tag features come from classification of prior word

<table>
<thead>
<tr>
<th></th>
<th>(w_{-1})</th>
<th>(w_0)</th>
<th>(w_{-1}w_0)</th>
<th>(w_{+1})</th>
<th>(t_{-1})</th>
<th>(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1) (Time)</td>
<td>(&lt;s&gt;)</td>
<td>Time</td>
<td>(&lt;s&gt;)Time</td>
<td>flies</td>
<td>BOS</td>
<td>N</td>
</tr>
<tr>
<td>(x_2) (flies)</td>
<td>Time</td>
<td>flies</td>
<td>Time flies</td>
<td>like</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>(x_3) (like)</td>
<td>flies</td>
<td>like</td>
<td>flies like</td>
<td>an</td>
<td>N</td>
<td>V</td>
</tr>
</tbody>
</table>
Decoding

- Goal: Identify highest probability tag sequence
Decoding

- **Goal:** Identify highest probability tag sequence
- **Issues:**
  - Features include tags from previous words
    - Not immediately available
Decoding

- Goal: Identify highest probability tag sequence
- Issues:
  - Features include tags from previous words
    - Not immediately available
  - Uses tag **history**
    - Just knowing highest probability preceding tag insufficient
Beam Search

- **Intuition:**
  - Breadth-first search explores all paths
  - Lots of paths are (pretty obviously) bad
  - Why explore bad paths?
  - Restrict to (apparently best) paths

- **Approach:**
  - Perform breadth-first search, *but*
Beam Search

- Intuition:
  - Breadth-first search explores all paths
  - Lots of paths are (pretty obviously) bad
  - Why explore bad paths?
  - Restrict to (apparently best) paths

- Approach:
  - Perform breadth-first search, *but*
  - Retain only $k$ ‘best’ paths thus far
  - $k$: beam width
Beam Search, k=3

<s> time flies like an arrow
Beam Search, $k=3$

<s> time flies like an arrow
Beam Search, $k=3$

<s> time flies like an arrow
Beam Search, $k=3$
Beam Search, $k=3$

$\langle s \rangle \quad \text{time} \quad \text{flies} \quad \text{like} \quad \text{an} \quad \text{arrow}$
Beam Search

- $W = \{w_1, w_2, \ldots, w_n\}$: test sentence
Beam Search

- $W=\{w_1, w_2, \ldots, w_n\}$: test sentence
- $s_{ij}$: $j^{th}$ highest prob. sequence up to & inc. word $w_i$
Beam Search

- $W = \{w_1, w_2, \ldots, w_n\}$: test sentence
- $s_{ij}$: $j^{th}$ highest prob. sequence up to & inc. word $w_i$
- Generate tags for $w_1$, keep top $k$, set $s_{1j}$ accordingly
Beam Search

- \( W = \{w_1, w_2, \ldots, w_n\} \): test sentence
- \( s_{ij} \): \( j^{th} \) highest prob. sequence up to & inc. word \( w_i \)
- Generate tags for \( w_1 \), keep top \( k \), set \( s_{1j} \) accordingly
- for \( i = 2 \) to \( n \):
Beam Search

- $W=\{w_1, w_2, ..., w_n\}$: test sentence
- $s_{ij}$: $j^{th}$ highest prob. sequence up to & inc. word $w_i$
- Generate tags for $w_1$, keep top $k$, set $s_{1j}$ accordingly
- for $i=2$ to $n$:
  - Extension: add tags for $w_i$ to each $s_{(i-1)j}$
Beam Search

- \( W = \{w_1, w_2, \ldots, w_n\} \): test sentence
- \( s_{ij} \): \( j^{\text{th}} \) highest prob. sequence up to & incl. word \( w_i \)
- Generate tags for \( w_1 \), keep top \( k \), set \( s_{1j} \) accordingly
- for \( i=2 \) to \( n \):
  - Extension: add tags for \( w_i \) to each \( s_{(i-1)j} \)
  - Beam selection:
    - Sort sequences by probability
    - Keep only top \( k \) sequences
Beam Search

- $W = \{w_1, w_2, \ldots, w_n\}$: test sentence
- $s_{ij}$: $j^{th}$ highest prob. sequence up to & inc. word $w_i$
- Generate tags for $w_1$, keep topN, set $s_{1j}$ accordingly
- for $i=2$ to $n$:
  - For each $s_{(i-1)j}$
  - for $w_i$ form vector, keep topN tags for $w_i$
  - Beam selection:
    - Sort sequences by probability
    - Keep only top sequences, using pruning on next slide
- Return highest probability sequence $s_{n1}$
Beam Search

- Pruning and storage:
  - $W = \text{beam width}$
  - For each node, store:
    - Tag for $w_i$
    - Probability of sequence so far, $\text{prob}_{i,j} = \prod_{j=1}^{t} p(t_j | h_j)$
  - For each candidate $j$, $s_{i,j}$
    - Keep the node if $\text{prob}_{i,j}$ in topK, and
    - $\text{prob}_{i,j}$ is sufficiently high
      - e.g. $\log(\text{prob}_{i,j}) + W \geq \log(\text{max_prob})$
Decoding

- Tag dictionary:
  - known word: returns tags seen with word in training
  - unknown word: returns all tags

- Beam width = 5

- Running time: $O(NTAB)$
  - $N, T, A$ as before
  - $B$: beam width
POS Tagging

- Overall accuracy: 96.3+%
- Unseen word accuracy: 86.2%
- Comparable to HMM tagging accuracy or TBL
- Provides
  - Probabilistic framework
  - Better able to model different info sources
- Topline accuracy 96-97%
  - Consistency issues
Beam Search

- Beam search decoding:
  - Variant of breadth first search
  - At each layer, keep only top $k$ sequences

- Advantages:
Beam Search

- Beam search decoding:
  - Variant of breadth first search
  - At each layer, keep only top $k$ sequences

- Advantages:
  - Efficient in practice: beam 3-5 near optimal
    - Empirically, beam 5-10% of search space; prunes 90-95%
Beam Search

- Beam search decoding:
  - Variant of breadth first search
  - At each layer, keep only top $k$ sequences

- Advantages:
  - Efficient in practice: beam 3-5 near optimal
    - Empirically, beam 5-10% of search space; prunes 90-95%
  - Simple to implement
    - Just extensions + sorting, no dynamic programming
Beam Search

- Beam search decoding:
  - Variant of breadth first search
  - At each layer, keep only top $k$ sequences

- Advantages:
  - Efficient in practice: beam 3-5 near optimal
    - Empirically, beam 5-10% of search space; prunes 90-95%
  - Simple to implement
    - Just extensions + sorting, no dynamic programming
  - Running time:
Beam Search

- Beam search decoding:
  - Variant of breadth first search
  - At each layer, keep only top sequences

- Advantages:
  - Efficient in practice: beam 3-5 near optimal
    - Empirically, beam 5-10% of search space; prunes 90-95%
  - Simple to implement
    - Just extensions + sorting, no dynamic programming

- Disadvantage: Not guaranteed optimal (or complete)
MaxEnt POS Tagging

- Part of speech tagging by classification:
  - Feature design
    - word and tag context features
    - orthographic features for rare words
MaxEnt POS Tagging

- Part of speech tagging by classification:
  - Feature design
    - word and tag context features
    - orthographic features for rare words

- Sequence classification problems:
  - Tag features depend on prior classification
MaxEnt POS Tagging

- Part of speech tagging by classification:
  - Feature design
    - word and tag context features
    - orthographic features for rare words

- Sequence classification problems:
  - Tag features depend on prior classification

- Beam search decoding
  - Efficient, but inexact
    - Near optimal in practice