# Maximum Entropy Model (II)

LING 572

Fei Xia

Week 6: 02/09/2010

# Outline

- Overview
- The Maximum Entropy Principle

- Modeling**
- Decoding

- Training**
- Smoothing**

- Case study: POS tagging: covered in ling570 already

# Training

# Algorithms

- Generalized Iterative Scaling (GIS): (Darroch and Ratcliff, 1972)
- Improved Iterative Scaling (IIS): (Della Pietra et al., 1995)
- L-BFGS:
- …

# GIS: setup**

Requirements for running GIS:

$$\forall (x, y) \in X \times Y \quad \sum_{j=1}^{k} f_j(x, y) = C$$

- If that's not the case,

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

Add a "correction" feature function $f_{k+1}$:

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

# GIS algorithm

- Compute $d_j = E_{\tilde{p}} f_j = \frac{1}{N} \sum_{i=1}^{N} f_j(x_i, y_i)$
- Initialize $\lambda_j^{(0)}$ (any values, e.g., 0)
- Repeat until convergence
  - For each j
    - Compute $$p^{(n)}(y \mid x) = \frac{e^{\sum_{j=1}^{k} \lambda_j^{(n)} f_j(x,y)}}{Z}$$
    - Compute $$E_{p^{(n)}} f_j = \frac{1}{N} \sum_{i=1}^{N} \sum_{y \in Y} p^{(n)}(y \mid x_i) f_j(x_i, y)$$
    - Update $$\lambda_j^{(n+1)} = \lambda_j^{(n)} + \frac{1}{C} (\log \frac{d_j}{E_{p^{(n)}} f_j})$$

# “Until convergence”

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

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

$$L(p^{(n+1)}) - L(p^{(n)}) < threshold$$

$$\frac{L(p^{(n+1)}) - L(p^{(n)})}{L(p^{(n)})} < threshold$$

# Calculating LL(p)

```
LL = 0;

for each training instance x
  let y be the true label of x
  prob = p(y | x);     # p is the current model
  LL += 1/N * log (prob);
```

# Properties of GIS

- $L(p^{(n+1)}) >= L(p^{(n)})$
- The sequence is guaranteed to converge to p*.
- The converge can be very slow.

- The running time of each iteration is O(NPA):
  - N: the training set size
  - P: the number of classes
  - A: the average number of features that are active for an instance (x, y).

# L-BFGS

- BFGS stands for Broyden-Fletcher-Goldfarb-Shanno: authors four single-authored papers published in 1970.
- Limited-memory BFGS: proposed in 1980s.
- It is a quasi-Newton method for unconstrained optimization. **
- It is especially efficient on problems involving a large number of variables.

# L-BFGS (cont)**

- J. Nocedal. Updating Quasi-Newton Matrices with Limited Storage (1980), Mathematics of Computation 35, pp. 773-782.

- D.C. Liu and J. Nocedal. On the Limited Memory Method for Large Scale Optimization (1989), Mathematical Programming B, 45, 3, pp. 503-528.

- Implementation:
  - Fortune: http://www.ece.northwestern.edu/~nocedal/lbfgs.html
  - C: http://www.chokkan.org/software/liblbfgs/index.html
  - Perl: http://search.cpan.org/~laye/Algorithm-LBFGS-0.12/lib/Algorithm/LBFGS.pm

# Outline

- Overview
- The Maximum Entropy Principle

- Modeling**
- Decoding

- Training**
- Smoothing**

- Case study: POS tagging

# Smoothing

Many slides come from

(Klein and Manning, 2003)

# Papers

- (Klein and Manning, 2003)

- Chen and Rosenfeld (1999): A Gaussian Prior for Smoothing Maximum Entropy Models, CMU Technical report (CMU-CS-99-108).

# Smoothing

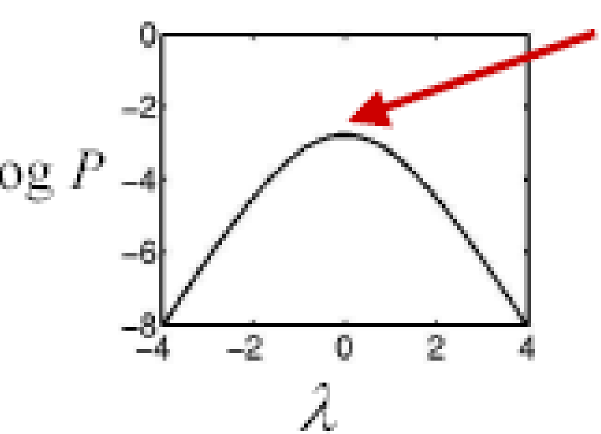
- MaxEnt models for NLP tasks can have millions of features.
- Overfitting is a problem.
- Feature weights can be infinite, and the iterative trainers can take a long time to reach those values.

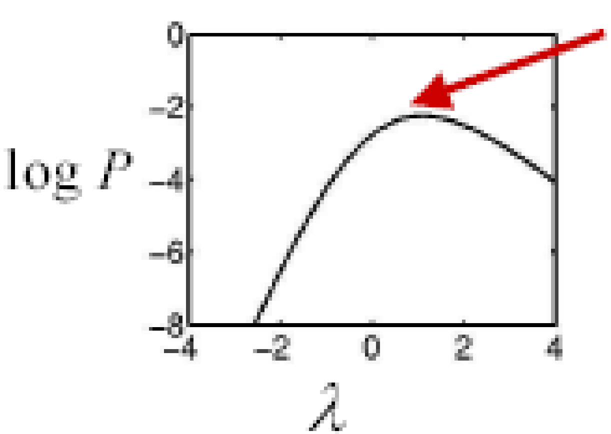
# An example

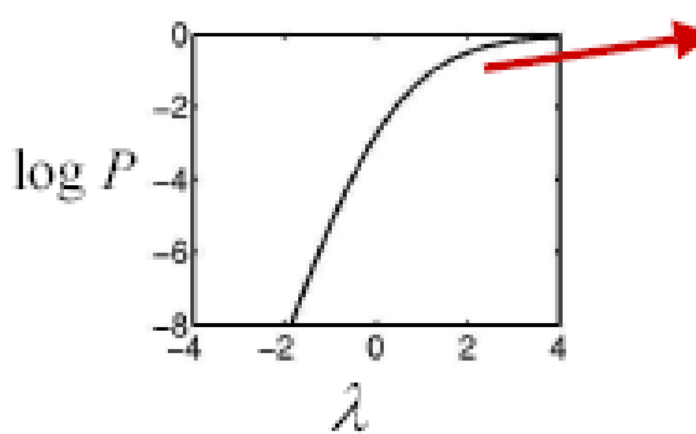
| Heads | Tails |
|---|---|
| 2 | 2 |

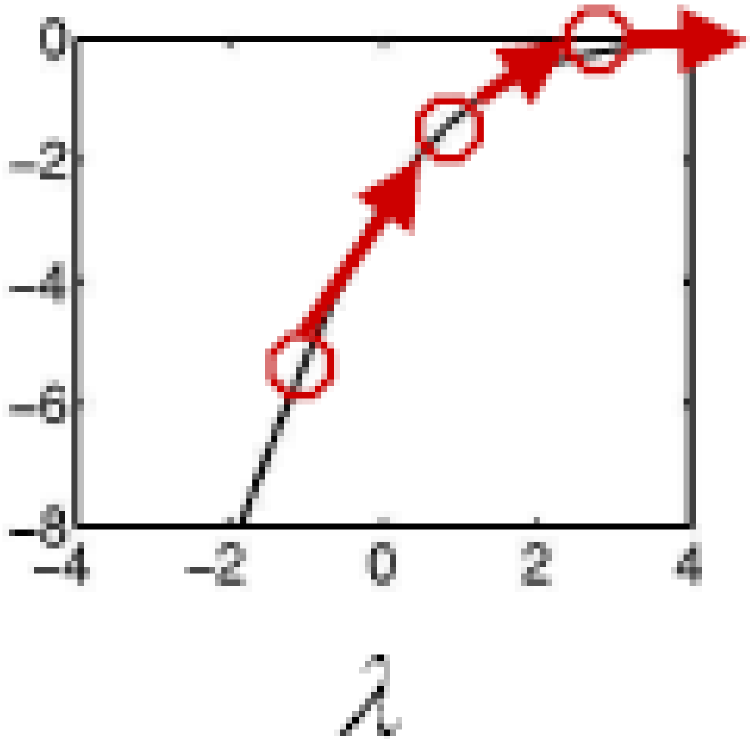
| Heads | Tails |
|---|---|
| 3 | 1 |

| Heads | Tails |
|---|---|
| 4 | 0 |






| Heads | Tails |
|---|---|
| 4 | 0 |

Input

| Heads | Tails |
|---|---|
| 1 | 0 |

Output

In the 4/0 case, there were two problems:

- The optimal value of $\lambda$ was $\infty$, which is a long trip for an optimization procedure.
- The learned distribution is just as spiked as the empirical one – no smoothing.

# Approaches

- Early stopping
- Feature selection
- Regularization**

# Early stopping

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 maxent work.

# Feature selection

- Methods:
  - Using predefined functions: e.g., Dropping features with low counts
  - Wrapping approach: Feature selection during training

- It is equivalent to setting the removed features’ weights to be zero.

- It reduces model size, but the performance could suffer.

# 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:

$$logP(Y,\lambda|X) = logP(\lambda) + logP(Y|X,\lambda)$$

Posterior Prior Likelihood

# MAP estimate**

- ML: Maximum likelihood

$$P(X, Y|\lambda)$$

$$P(Y|X, \lambda)$$

- MAP: Maximum A Posterior

$$P(\lambda|X, Y)$$

$$P(Y, \lambda|X)$$

$$logP(Y, \lambda|X) = logP(\lambda) + logP(Y|X, \lambda)$$

# The prior**

- Uniform distribution, Exponential prior, …
- Gaussian prior:

$$P(\lambda_i) = \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(-\frac{(\lambda_i - \mu)^2}{2\sigma^2}\right)$$

$$logP(Y, \lambda | X) = logP(\lambda) + logP(Y|X, \lambda)$$

$$= \sum_{i=1}^{k} logP(\lambda_i) + logP(Y|X, \lambda)$$

$$= -k\, log\sqrt{2\pi}\sigma - \sum_{i=1}^{k} \frac{(\lambda_i - \mu)^2}{2\sigma^2} + logP(Y|X, \lambda)$$

- 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: $\mu = 0 \quad 2\sigma^2 = 1$

# L1 or L2**

$$L_1 = \sum_i logP(y_i, \lambda | x_i) - \frac{||\lambda||}{\sigma}$$

Orthant-Wise limited-memory Quasi-Newton (OW-LQN) method (Andrew and Gao, 2007)

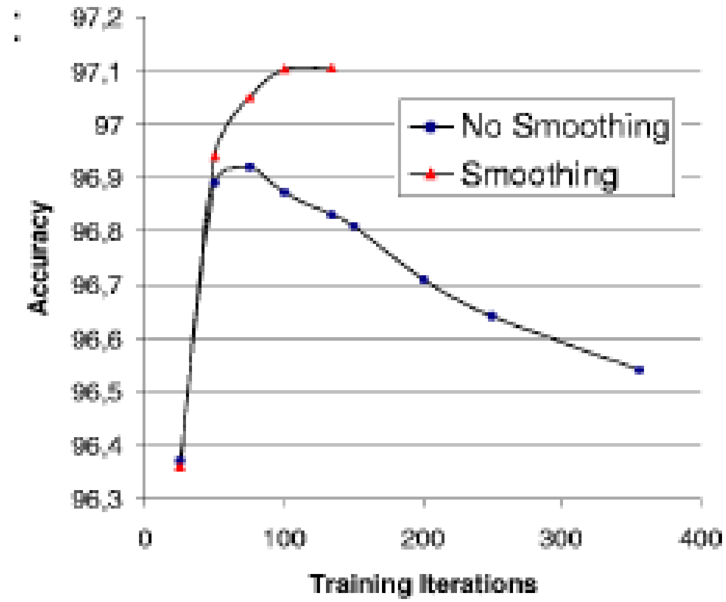
$$L_2 = \sum_i logP(y_i, \lambda | x_i) - \frac{||\lambda||^2}{2\sigma^2}$$

L-BFGS method (Nocedal, 1980)

# Example: POS tagging

- From (Toutanova et al., 2003):

| | Overall Accuracy | Unknown Word Acc |
|---|---|---|
| Without Smoothing | 96.54 | 85.20 |
| With Smoothing | 97.10 | 88.20 |



# Benefits of smoothing**

- Softens distributions
- Pushes weights onto more explanatory features
- Allows many features to be used safely
- Speed up convergence (if both are allowed to converge)

# Summary: training and smoothing

- Training: many methods (e.g., GIS, IIS, L-BFGS).

- Smoothing:
  - Early stopping
  - Feature selection
  - Regularization

- Regularization:
  - Changing the objective function by adding the prior
  - A common prior: Gaussian distribution
  - Maximizing posterior is no longer the same as maximizing entropy.

# Outline

- Overview
- The Maximum Entropy Principle

- Modeling**
- Decoding

- Training**
- Smoothing**

- Case study: POS tagging

Covered in ling570 already

# Case study

# POS tagging (Ratnaparkhi, 1996)

- Notation variation:
  - $f_j(x, y)$: x: input, y: output
  - $f_j(h, t)$: h: history, t: tag for the word
- History:

$$h_i = \{w_i, w_{i-1}, w_{i-2}, w_{i+1}, w_{i+2}, t_{i-1}, t_{i-2}\}$$

- Training data:
  - Treat a sentence as a set of $(h_i, t_i)$ pairs.
  - How many pairs are there for a sentence?

# Using a MaxEnt Model

- Modeling:

- Training:
  - Define features templates
  - Create the feature set
  - Determine the optimum feature weights via GIS or IIS

- Decoding:

# Modeling

$$P(t_1,\ldots,t_n \mid w_1,\ldots,w_n)$$

$$= \prod_{i=1}^{n} p(t_i \mid w_1^n, t_1^{i-1})$$

$$\approx \prod_{i=1}^{n} p(t_i \mid h_i)$$

$$p(t \mid h) = \frac{p(h,t)}{\sum_{t' \in T} p(h,t')}$$

# Training step 1: define feature templates

| Condition | Features | |
|---|---|---|
| $w_i$ is not rare | $w_i = X$ | $\& \ t_i = T$ |
| $w_i$ is rare | $X$ is prefix of $w_i$, $\lvert X \rvert \leq 4$ | $\& \ t_i = T$ |
| | $X$ is suffix of $w_i$, $\lvert X \rvert \leq 4$ | $\& \ t_i = T$ |
| | $w_i$ contains number | $\& \ t_i = T$ |
| | $w_i$ contains uppercase character | $\& \ t_i = T$ |
| | $w_i$ contains hyphen | $\& \ t_i = T$ |
| $\forall \ w_i$ | $t_{i-1} = X$ | $\& \ t_i = T$ |
| | $t_{i-2}t_{i-1} = XY$ | $\& \ t_i = T$ |
| | $w_{i-1} = X$ | $\& \ t_i = T$ |
| | $w_{i-2} = X$ | $\& \ t_i = T$ |
| | $w_{i+1} = X$ | $\& \ t_i = T$ |
| | $w_{i+2} = X$ | $\& \ t_i = T$ |

History $h_i$

Tag $t_i$

# Step 2: Create feature set

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Word:* | the | stories | about | well-heeled | communities | and | developers |
| *Tag:* | DT | NNS | IN | JJ | NNS | CC | NNS |
| *Position:* | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

$$
\begin{array}{ll}
w_i = \texttt{about} & \& \; t_i = \texttt{IN} \\
w_{i-1} = \texttt{stories} & \& \; t_i = \texttt{IN} \\
w_{i-2} = \texttt{the} & \& \; t_i = \texttt{IN} \\
w_{i+1} = \texttt{well-heeled} & \& \; t_i = \texttt{IN} \\
w_{i+2} = \texttt{communities} & \& \; t_i = \texttt{IN} \\
t_{i-1} = \texttt{NNS} & \& \; t_i = \texttt{IN} \\
t_{i-2}t_{i-1} = \texttt{DT NNS} & \& \; t_i = \texttt{IN}
\end{array}
$$

➔Collect all the features from the training data

➔Throw away features that appear less than 10 times

# The thresholds

- Raw words: words that occur < 5 in the training data.

- Features (not feature functions):
  - All curWord features will be kept.
  - For the rest of features, keep them if they occur >= 10 in the training data.

# Step 3: determine the weights of feature functions

- GIS

- Training time:
  - Each iteration: O(NTA):
    - N: the training set size
    - T: the number of allowable tags
    - A: average number of features that are active for a (h, t).
  - About 24 hours on an IBM RS/6000 Model 380.

# Beam search

# Why do we need beam search?

- Features refer to tags of previous words, which are not available for the TEST data.

- Knowing only the best tag of the previous word is not good enough.

- So let's keep multiple tag sequences available during the decoding.

# Beam search

Parameters: topN, topK, beam_size

(1) Get topN tags for $w_1$ and form nodes $s_{1,j}$

(2) For $i$=2 to n (n is the sentence length)

For each surviving node $s_{i-1,j}$

form the vector for $w_i$

get topN tags for $w_i$ and

form new nodes

Prune nodes at position $i$

(3) Pick the node at position $n$ with highest prob

# Pruning at Position i

Each node at Position $i$ should store a tag for $w_i$ and a prob, where the prob is $\prod_{k=1}^{i} P(t_k|h_k)$.

Let $max\_prob$ be the highest prob among the nodes at Position $i$

For each node $s_{i,j}$ at Position $i$

    Let $prob_{i,j}$ be the probability stored at the node

    keep the node iff $prob_{i,j}$ is among the topK of the nodes

        and $lg(prob_{i,j})$ + beam_size $\geq lg(max\_prob)$

# Beam search

- 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; and beam sizes of 3–5 are as good or almost as good as exact inference in many cases.
  - Easy to implement (no dynamic programming required).
- Disadvantage:
  - Inexact: the globally best sequence can fall off the beam.

# Decoding (cont)

- Tags for words:
  - Known words: use tag dictionary
  - Unknown words: try all possible tags

- Ex: “time flies like an arrow”

- Running time: O(NTAB)
  - N: sentence length
  - B: beam size
  - T: tagset size
  - A: average number of features that are active for a given event

# Experiment results

| | | | |
|---|---|---|---|
| MF tag | O | 7.66 | |
| Markov 1-gram | B | 6.74 | |
| Markov 3-gram | W | 3.7 | |
| Markov 3-gram | B | 3.64 | |
| Decision tree | M | 3.5 | |
| Transformation | B | 3.39 | |
| Maxent | R | 3.37 | |
| Maxent | O | 3.11 | ±.07 |
| Multi-tagger Voting | B | 2.84 | ±.03 |

# Comparison with other learners

- HMM: MaxEnt can use more context
- DT:  MaxEnt does not split data
- Naïve Bayes: MaxEnt does not assume that features are independent given the class.

# Hw6

# Hw6: Beam search

- format: beamsearch_maxent.sh test_data boundary_file model_file syst_output beam_size topN topK
- test_data: instanceName goldClass f1 v1 …

  This includes words from all the test sentences.
- boundary_file: length of each sentence
- model_file: MaxEnt model in text format
- sys_output: instanceName goldClass sysClass prob
- beam_size, topN, and topK: see slide 41-42

- Remember to add prevT=tag and prevTwoTags=tag+tag features:
  - For the list of tags, see prevT=tag in the model file
  - If a tag bigram is not in the model, do not add the prevTwoTag=tag1+tag2 feature as the model does not have the weights for it.
  - For different hypotheses for the same word, the features will be different.

# Additional slides

# The “correction” feature function for GIS

$$f_{k+1}(x,y) = C - \sum_{j=1}^{k} f_j(x,y)$$

$$f_{k+1}(x,c_1) = f_{k+1}(x,c_2) = \ldots$$

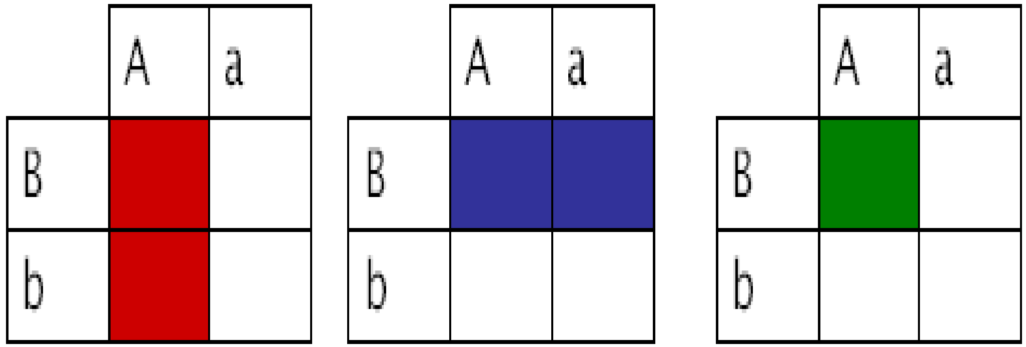
The weight of $f_{k+1}$ will not affect P(y | x).

Therefore, there is no need to estimate the weight.

# Ex4 (cont)

Empirical

| | A | a |
|---|---|---|
| B | 1 | 1 |
| b | 1 | 0 |



A = 2/3

B = 2/3

AB = 1/3

| | A | a |
|---|---|---|
| B | 1/3 | 1/6 |
| b | 1/3 | 1/6 |

| | A | a |
|---|---|---|
| B | 4/9 | 2/9 |
| b | 2/9 | 1/9 |

| | A | a |
|---|---|---|
| B | 1/3 | 1/3 |
| b | 1/3 | 0 |

??

# Training

# IIS algorithm

- Compute $d_j$, j=1, …, k+1 and $f^{\#}(x,y) = \sum_{j=1}^{k} f_j(x,y)$
- Initialize $\lambda_j^{(1)}$ (any values, e.g., 0)
- Repeat until converge
  - For each j
    - Let $\Delta\lambda_j$ be the solution to

$$\sum_{x \in \varepsilon} p^{(n)}(x,y) f_j(x,y) e^{\Delta\lambda_j f^{\#}(x,y)} = d_j$$

    - Update $\lambda_j^{(n+1)} = \lambda_j^{(n)} + \Delta\lambda_j$

# Calculating $\Delta\lambda_j$

If $\forall x \in \varepsilon \quad \sum_{j=1}^{k} f_j(x) = C$

Then $\Delta\lambda_j = \frac{1}{C}(\log \frac{d_i}{E_{p^{(n)}} f_j})$

GIS is the same as IIS

Else $\Delta\lambda_j$ must be calcuated numerically.

# Feature selection

# Feature selection

- Throw in many features and let the machine select the weights
  - Manually specify feature templates

- Problem: too many features

- An alternative: greedy algorithm
  - Start with an empty set S
  - Add a feature at each iteration

# Two scenarios

Scenario #1: no feature selection during training

- Define features templates
- Create the feature set
- Determine the optimum feature weights via GIS or IIS

Scenario #2: with feature selection during training

- Define feature templates
- Create a candidate feature set F
- At every iteration, choose the feature from F (with max gain) and determine its weight (or choose top-n features and their weights).

# Notation

With the feature set S:

$$\mathcal{C}(\mathcal{S}) \equiv \{p \in \mathcal{P} \mid p(f) = \tilde{p}(f) \quad \text{for all } f \in \mathcal{S}\}$$

$$p_{\mathcal{S}} \equiv \operatorname*{argmax}_{p \in \mathcal{C}(\mathcal{S})} H(p)$$

After adding a feature:

$$\mathcal{C}(\mathcal{S} \cup \hat{f}) \equiv \{p \in \mathcal{P} \mid p(f) = \tilde{p}(f) \quad \text{for all } f \in \mathcal{S} \cup \hat{f}\}$$

$$p_{\mathcal{S} \cup \hat{f}} \equiv \operatorname*{argmax}_{p \in \mathcal{C}(\mathcal{S} \cup \hat{f})} H(p)$$

The gain in the log-likelihood of the training data:

$$\Delta L(\mathcal{S}, \hat{f}) \equiv L(p_{\mathcal{S} \cup \hat{f}}) - L(p_{\mathcal{S}})$$

# Feature selection algorithm (Berger et al., 1996)

- Start with S being empty; thus $p_s$ is uniform.

- Repeat until the gain is small enough
  - For each candidate feature f
    - Computer the model $p_{S \cup f}$ using IIS
    - Calculate the log-likelihood gain
  - Choose the feature with maximal gain, and add it to S

➔ Problem: too expensive

# Approximating gains (Berger et. al., 1996)

- Instead of recalculating all the weights, calculate only the weight of the new feature.