# Maximum Entropy Model (III): training and smoothing

LING 572
Advanced Statistical Methods for NLP
February 4, 2020

# 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:

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

- If that's not the case, let

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

- Add "correction" feature function:

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

# GIS algorithm

- Compute empirical expectation: $d_j = E_{\tilde{p}} f_j = \frac{1}{N} \sum_{i=1}^{N} f_j(x_i, y_i)$
- Initialize $\lambda_j^{(0)}$ to 0 or some other value
- Repeat until convergence for each j:
  - Calculate p(y | x) under the current model: $p^{(n)}(y \mid x) = \frac{e^{\sum_{j=1}^{k} \lambda_j^{(n)} f_j(x,y)}}{Z}$
  - Calculate model expectation under current model: $E_{p^{(n)}} f_j = \frac{1}{N} \sum_{i=1}^{N} \sum_{y \in Y} p^{(n)}(y \mid x_i) f_j(x_i, y)$
  - Update model parameters: $\lambda_j^{(n+1)} = \lambda_j^{(n)} + \frac{1}{C}(\log \frac{d_j}{E_{p^{(n)}} f_j})$

# “Until convergence”

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

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

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

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

# Calculating LL(p)

```
LL = 0;

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

# Properties of GIS

- $L(p^{(n+1)}) >= L(p^{(n)})$
- The sequence is guaranteed to converge to p*.
- The convergence 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 of four single-authored papers published in 1970.
- L-BFGS: Limited-memory BFGS, proposed in 1980s.
- Quasi-Newton method for unconstrained optimization. **
- Especially efficient on problems involving a large number of variables.

# L-BFGS (cont)**

- References:
  - 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
  - Scipy: https://docs.scipy.org/doc/scipy/reference/optimize.minimize-lbfgsb.html

# 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

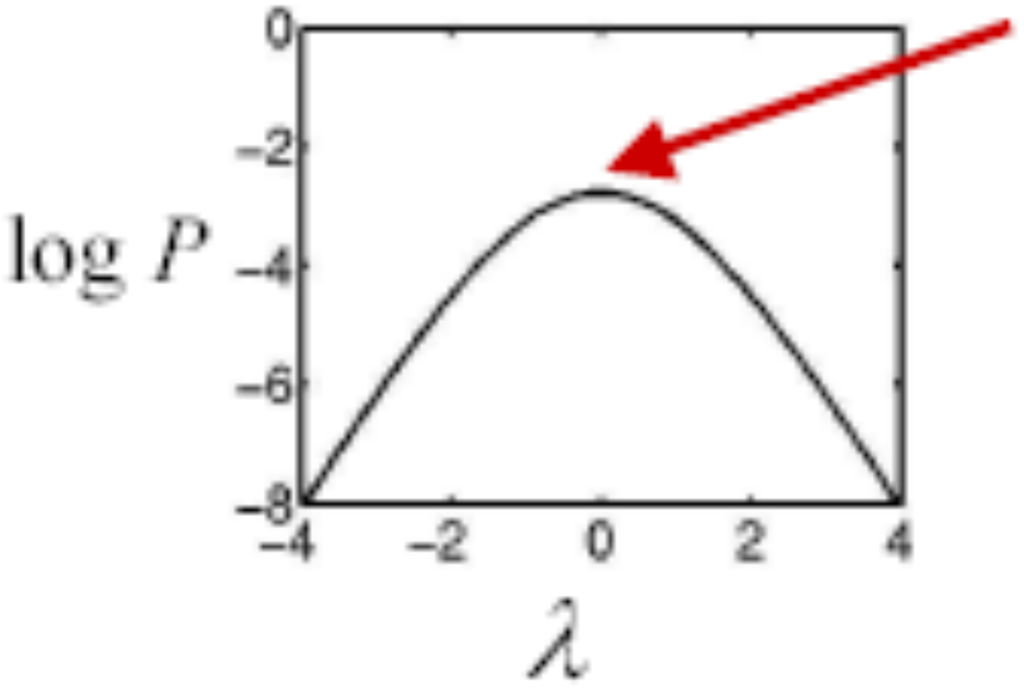
- MaxEnt models for NLP tasks can have millions of features.
- Overfitting is a problem.
- Feature weights can be infinite, and the iterative trainers can take a long time to reach those values.

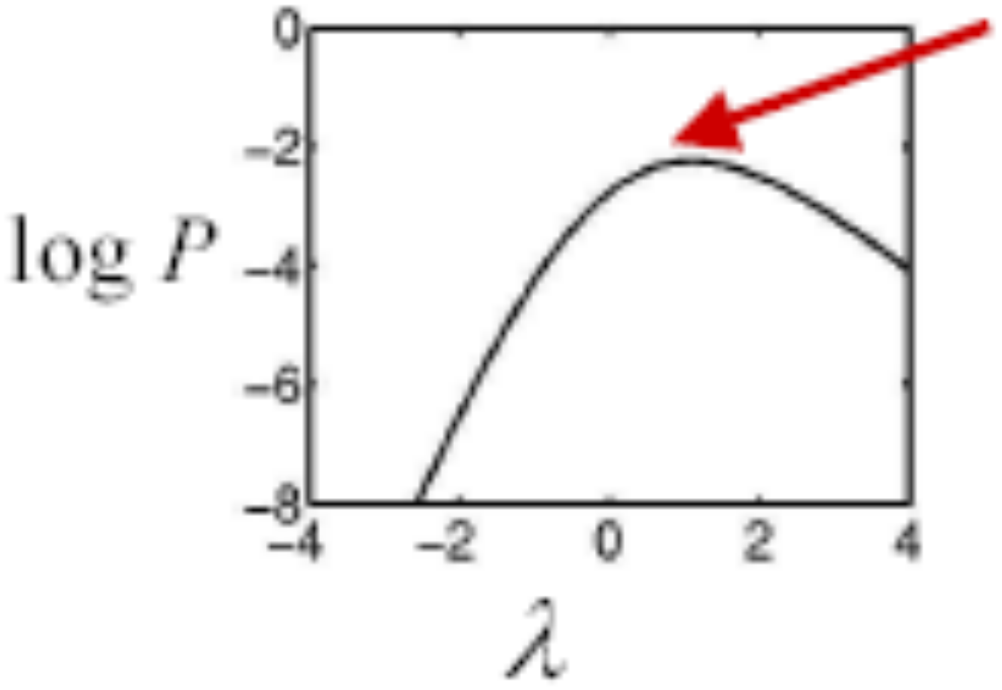
# An example

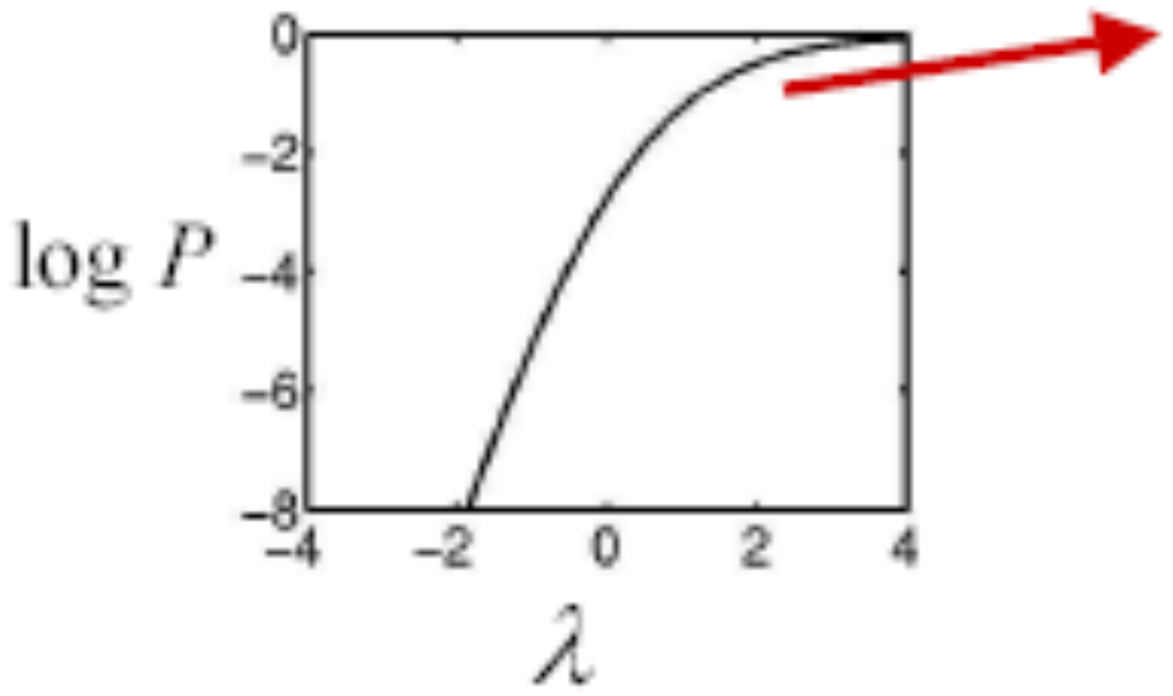
| Heads | Tails |
|---|---|
| 2 | 2 |

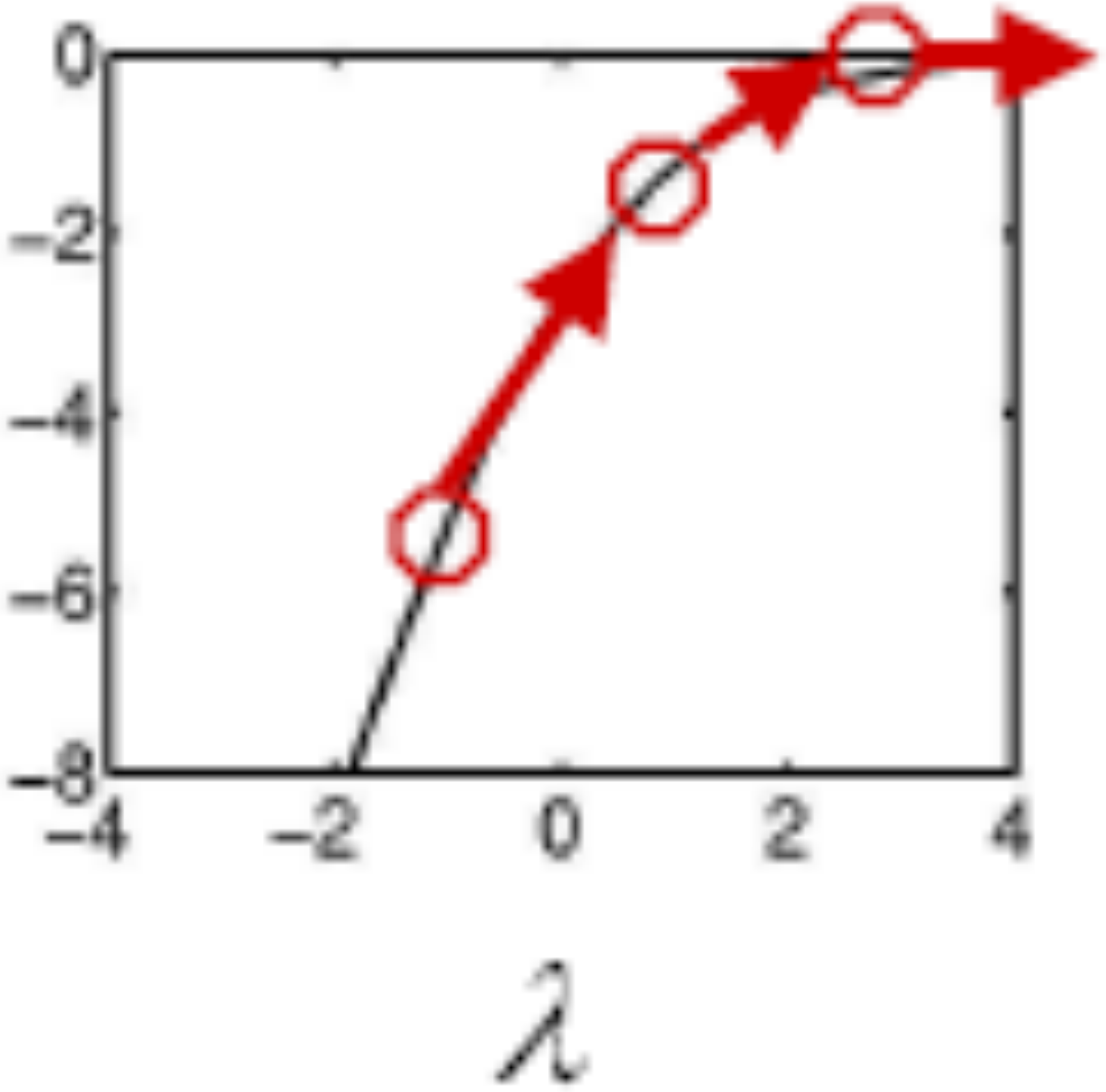
| Heads | Tails |
|---|---|
| 3 | 1 |

| 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

- Prior use of early stopping
  - Decision tree heuristics

- Similarly here
  - Stop training after a few iterations

  - The values of parameters will be finite.

  - Commonly used in early MaxEnt work

# Feature selection

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

- Equivalent to setting the removed features' weights to be zero.

- Reduces model size, but 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:

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

Posterior Prior Likelihood

# MAP estimate**

- ML: Maximum likelihood

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

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

- MAP: Maximum A Posteriori

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

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

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

# The prior**

- Uniform distribution, Exponential prior, …
- Gaussian prior: $P(\lambda_i) = \frac{1}{\sigma_i\sqrt{2\pi}} \exp(-\frac{(\lambda_i - \mu_i)^2}{2\sigma^2})$

$$\log p(Y, \lambda \,|\, X) = \log P(\lambda) + \log P(Y|X, \lambda)$$

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

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

- Maximize P(Y|X, λ):

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

- Maximize P(Y, λ | 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 regularization**

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

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

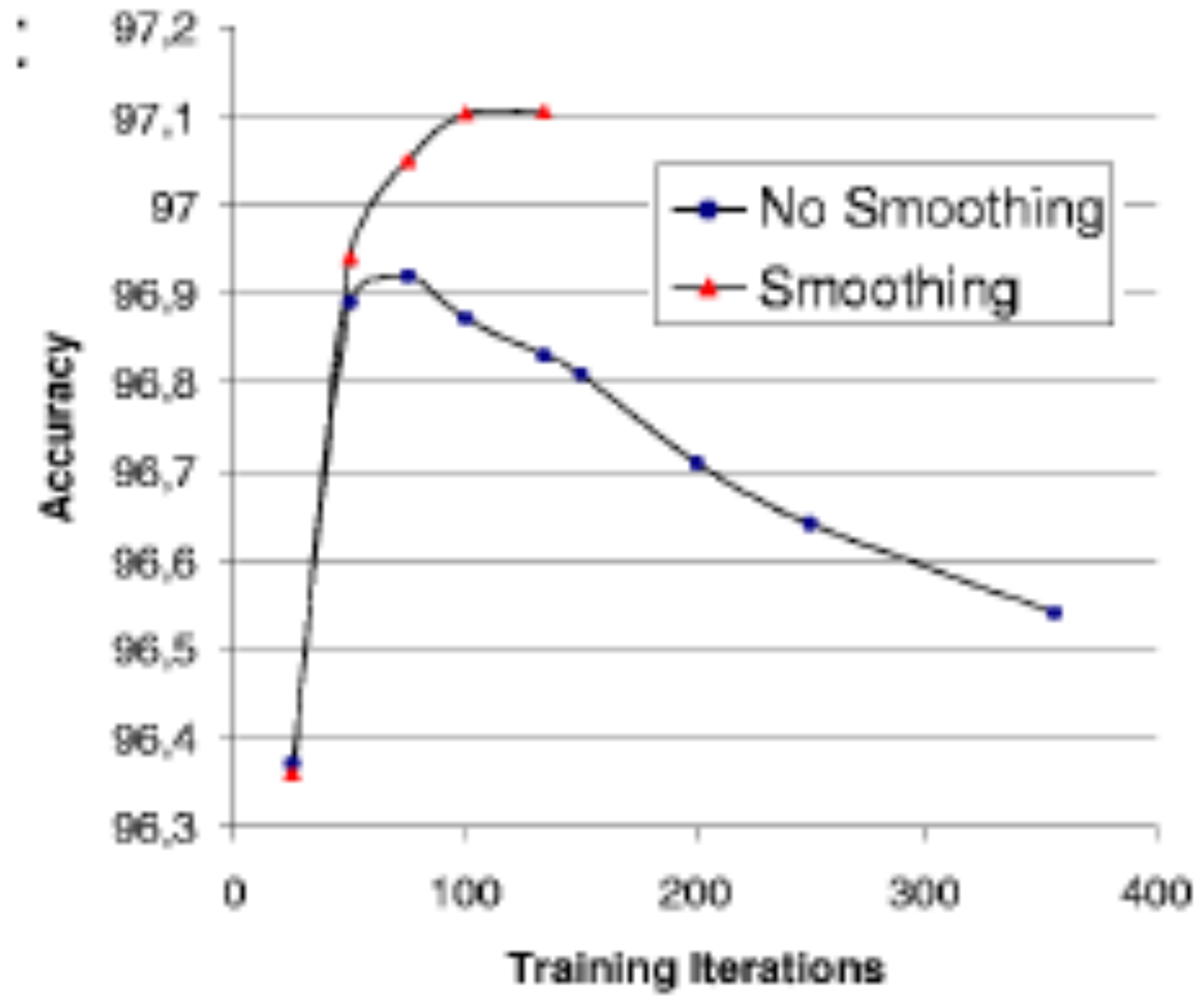
$$L_2 = \sum_i \log P(y_i, \lambda \mid x_i) - \frac{||\lambda||^2}{2\sigma^2}$$

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
- Can speed up convergence

# 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:

$$p(y \mid x) = \frac{e^{\sum_{j=1}^{k} \lambda_j f_j(x,y)}}{Z}$$

- Training**: compare empirical expectation and model expectation and modify the weights accordingly
- Smoothing**: change the objective function
- Case study: POS tagging

# Additional slides

# The "correction" feature function for GIS

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

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

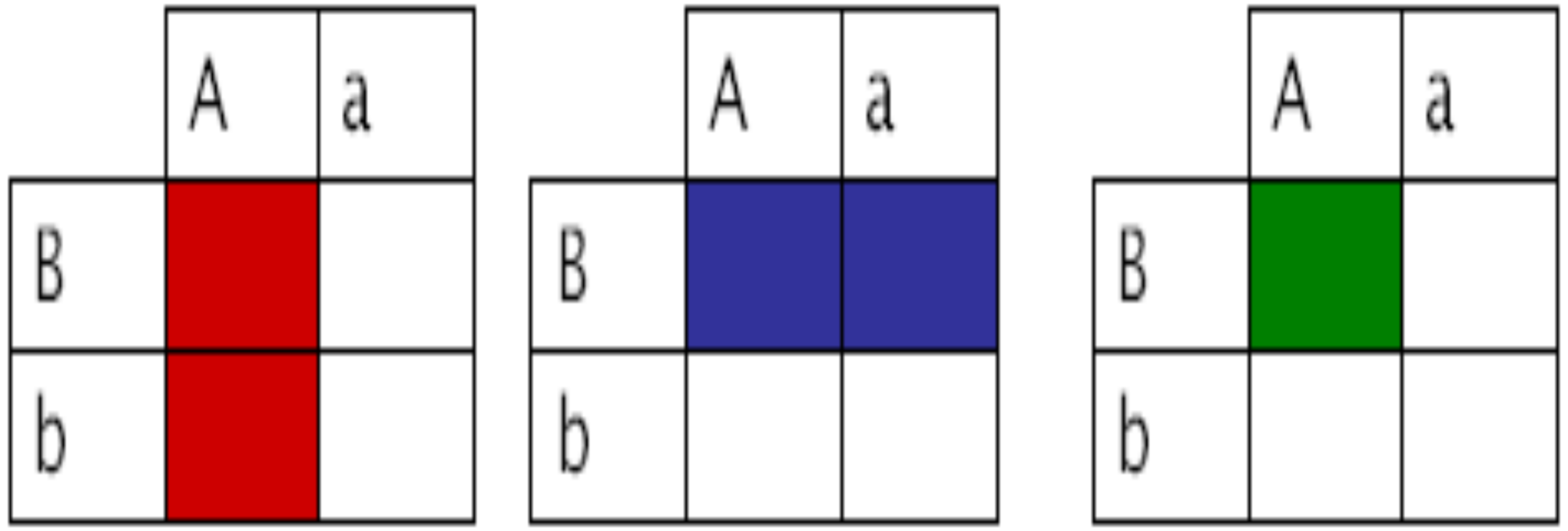
The weight of $f_{k+1}$ will not affect P(y | x).

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 calculated 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:

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

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

After adding a feature:

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

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

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.

$$p_{S \cup f}$$

- Repeat until the gain is small enough
  - For each candidate feature f
    - Computer the model 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.