CMP462: Natural Language Processing

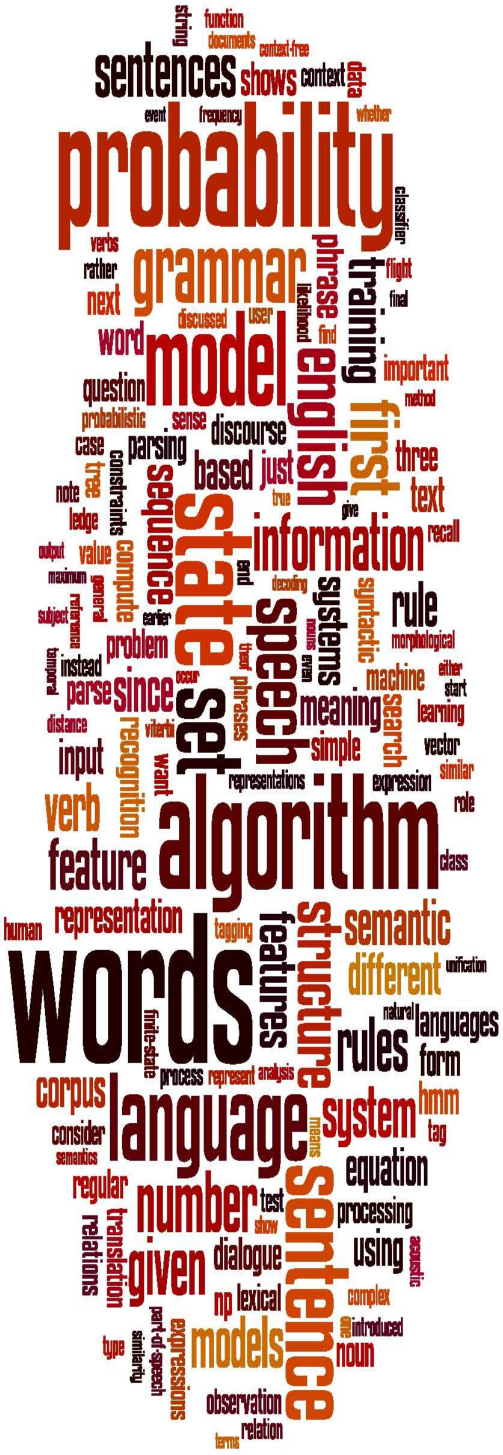
Lecture 06: Maximum Entropy Classifiers

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Spring 2013

Agenda

- Generative Vs Discriminative Models
- Features
- MaxEnt Models
- Training
- Smoothing

Acknowledgment:
Most slides adapted from Chris Manning and Dan Jurafsky's NLP class on Coursera.



Maxent Models and Discriminative Estimation

Generative vs. Discriminative models

Christopher Manning

Introduction

- So far we've looked at "generative models"
  - Language models, Naive Bayes
- But there is now much use of conditional or discriminative probabilistic 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 automatic 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 (generate the observed data from hidden stuff):
  - All the classic StatNLP models:
    - *n*-gram models, Naive Bayes classifiers, hidden Markov models, probabilistic context-free grammars, IBM machine translation alignment models

$P(c,d)$

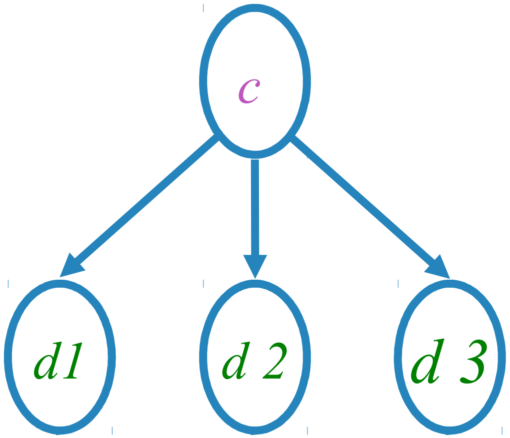
Joint vs. Conditional Models

- Discriminative (conditional) models take the data as given, and put a probability over hidden structure given the data:
  - Logistic regression, conditional loglinear or maximum entropy models, conditional random fields
  - Also, SVMs, (averaged) perceptron, etc. are discriminative classifiers (but not directly probabilistic)

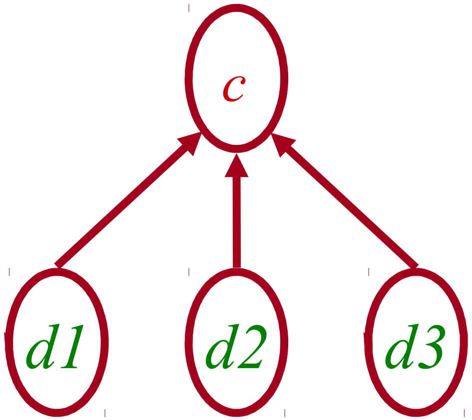
$P(c|d)$

Bayes Net/Graphical Models

- Bayes net diagrams draw circles for random variables, and lines for direct dependencies
- Some variables are observed; some are hidden
- Each node is a little classifier (conditional probability table) based on incoming arcs



Naive Bayes



Logistic Regression

Generative

Discriminative

Conditional vs. Joint Likelihood

- A *joint* model gives probabilities P(*d*,*c*) and tries to maximize this joint likelihood.
  - It turns out to be trivial to choose weights: just relative frequencies.
- A *conditional* model gives probabilities P(*c*|*d*). It takes the data as given and 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.

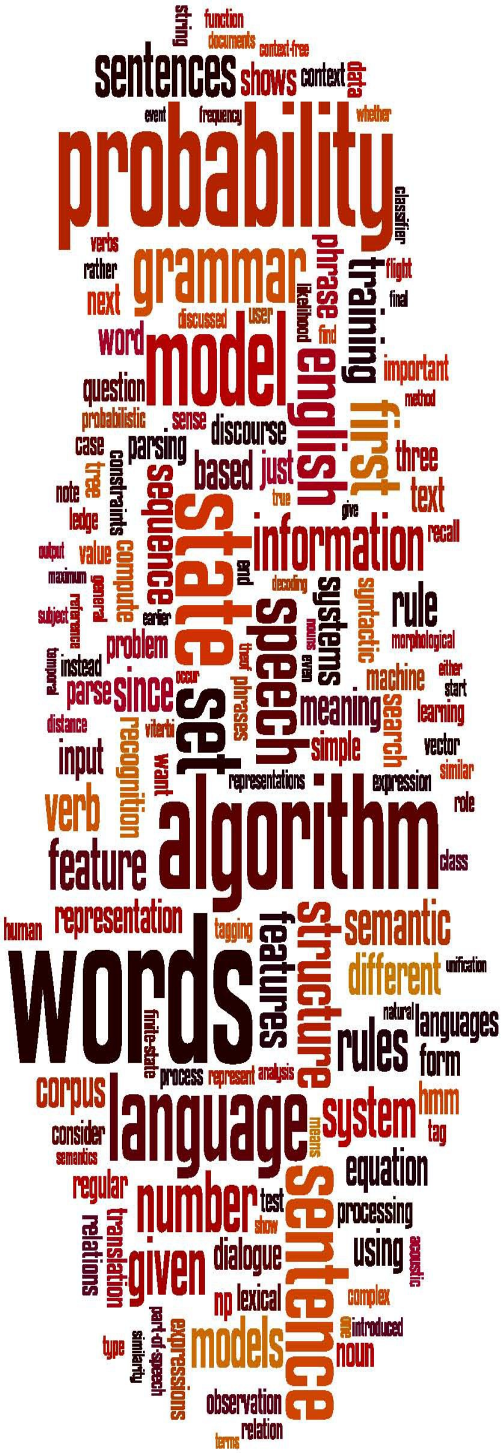
Conditional models work well: Word Sense Disambiguation

Training Set	
Objective	Accuracy
Joint Like.	86.8
Cond. Like.	98.5

Test Set	
Objective	Accuracy
Joint Like.	73.6
Cond. Like.	76.1

(Klein and Manning 2002, using Senseval-1 Data)

- Even with exactly the same features, changing from joint to conditional estimation increases performance
- That is, we use the same smoothing, and the same word-class features, we just change the numbers (parameters)



Discriminative Model Features

Making features from text for discriminative NLP models

Christopher Manning

Features

- In these slides and most maxent work: *features* f are elementary pieces of evidence that link aspects of what we observe *d* with a category *c* that we want to predict
- A feature is a function with a bounded real value: $f\colon C \times D \rightarrow \mathbb{R}$

Example features

- $f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$
- $f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$
- $f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$

- Models will assign to each feature a *weight:*
  - A positive weight votes that this configuration is likely correct
  - A negative weight votes that this configuration is likely incorrect

Feature Expectations

- We will crucially make use of two *expectations*
    - actual or predicted counts of a feature firing:
        - Empirical count (expectation) of a feature:

$$\text{empirical } E(f_i) = \sum_{(c,d)\in observed(C,D)} f_i(c,d)$$

        - Model expectation of a feature:

$$E(f_i) = \sum_{(c,d)\in observed(C,D)} P(c,d)\, f_i(c,d)$$

Features

- In NLP uses, usually a feature specifies
  1) an indicator function – a yes/no boolean matching function – of properties of the input and
  2) a particular class

$$f_i(c, d) \equiv [\Phi(d) \wedge c = c_j]$$

[Value is 0 or 1]

  - Each feature picks out a data subset and suggests a label for it

Feature-Based Models

- The decision about a data point is based only on the features active at that point.

Data

BUSINESS: Stocks hit a yearly low …

Label: BUSINESS

Features

{…, stocks, hit, a, yearly, low, …}

Text Categorization

e.g. f_i = [“stocks” occur and Label=”BUSINESS”]

Data

… to restructure bank:MONEY debt.

Label: MONEY

Features

{…, w-1=restructure, w+1=debt, L=12, …}

Word-Sense Disambiguation

Data

DT JJ NN …
The previous fall …

Label: NN

Features

{w=fall, t_{-1}=JJ w_{-1}=previous}

POS Tagging

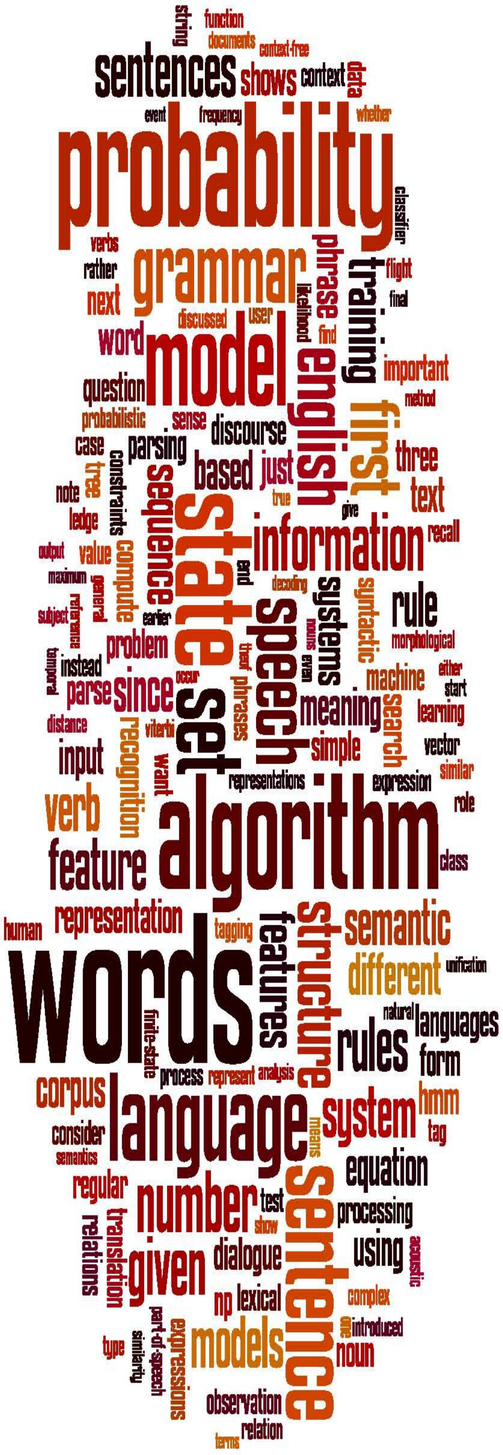
Example: Text Categorization

(Zhang and Oles 2001)

- Features are presence of each word in a document and the document class (they do feature selection to use reliable indicator words)
- Tests on classic Reuters data set (and others)
    - Naïve Bayes: 77.0% F1
    - Linear regression: 86.0%
    - Logistic regression: 86.4%
    - Support vector machine: 86.5%
- Paper emphasizes the importance of *regularization* (smoothing) for successful use of discriminative methods (not used in much early NLP/IR work)

Other Maxent Classifier Examples

- You can use a maxent classifier whenever you want to assign data points to one of a number of classes:
  - Sentence boundary detection (Mikheev 2000)
    - Is a period end of sentence or abbreviation?
  - Sentiment analysis (Pang and Lee 2002)
    - Word unigrams, bigrams, POS counts, …
  - PP attachment (Ratnaparkhi 1998)
    - Attach to verb or noun? Features of head noun, preposition, etc.
  - Parsing decisions in general (Ratnaparkhi 1997; Johnson et al. 1999, etc.)



Feature-based Linear Classifiers

How to put features into a classifier

Feature-Based Linear Classifiers

$f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$

$f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$

$f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$

- Linear classifiers at classification time:
  - Linear function from feature sets $\{f_i\}$ to classes $\{c\}$.
  - Assign a weight λ_i to each feature f_i.
  - We consider each class for an observed datum d
  - For a pair (c,d), features vote with their weights:
    - $\text{vote}(c) = \Sigma\lambda_i f_i(c,d)$

  - Choose the class c which maximizes $\Sigma\lambda_i f_i(c,d)$

PERSON: 0

LOCATION: 1.2

DRUG: 0.3

Feature-Based Linear Classifiers

There are many ways to chose weights for features

- Perceptron: find a currently misclassified example, and nudge weights in the direction of its correct classification
- Margin-based methods (Support Vector Machines)

Feature-Based Linear Classifiers

- Exponential (log-linear, maxent, logistic, Gibbs) models:
  - Make a probabilistic model from the linear combination $\Sigma\lambda_i f_i(c,d)$

$$P(c|d,\lambda)=\frac{\exp\left(\sum_i \lambda_i f_i(c,d)\right)}{\sum_{c'}\exp\left(\sum_i \lambda_i f_i(c',d)\right)}$$

Makes votes positive

Normalizes votes

- P(LOCATION|*in Québec*) = $e^{1.8}e^{-0.6}/(e^{1.8}e^{-0.6} + e^{0.3} + e^{0}) = 0.586$
- P(DRUG|*in Québec*) = $e^{0.3} /(e^{1.8}e^{-0.6} + e^{0.3} + e^{0}) = 0.238$
- P(PERSON|*in Québec*) = $e^{0} /(e^{1.8}e^{-0.6} + e^{0.3} + e^{0}) = 0.176$

  - The weights are the parameters of the probability model, combined via a “soft max” function

Feature-Based Linear Classifiers

- Exponential (log-linear, maxent, logistic, Gibbs) models:
  - Given this model form, we will choose parameters $\{\lambda_i\}$ that *maximize the conditional likelihood* of the data according to this model.
  - We construct not only classifications, but probability distributions over classifications.
    - There are other (good!) ways of discriminating classes – SVMs, boosting, even perceptrons – but these methods are not as trivial to interpret as distributions over classes.

Aside: logistic regression

- Maxent models in NLP are essentially the same as multiclass logistic regression models in statistics (or machine learning)
  - If you haven't seen these before, don't worry, this presentation is self-contained!
  - If you have seen these before you might think about:
    - The parameterization is slightly different in a way that is advantageous for NLP-style models with tons of sparse features (but statistically inelegant)
    - The key role of feature functions in NLP and in this presentation
      - The features are more general, with f also being a function of the class – when might this be useful?

Quizz Question

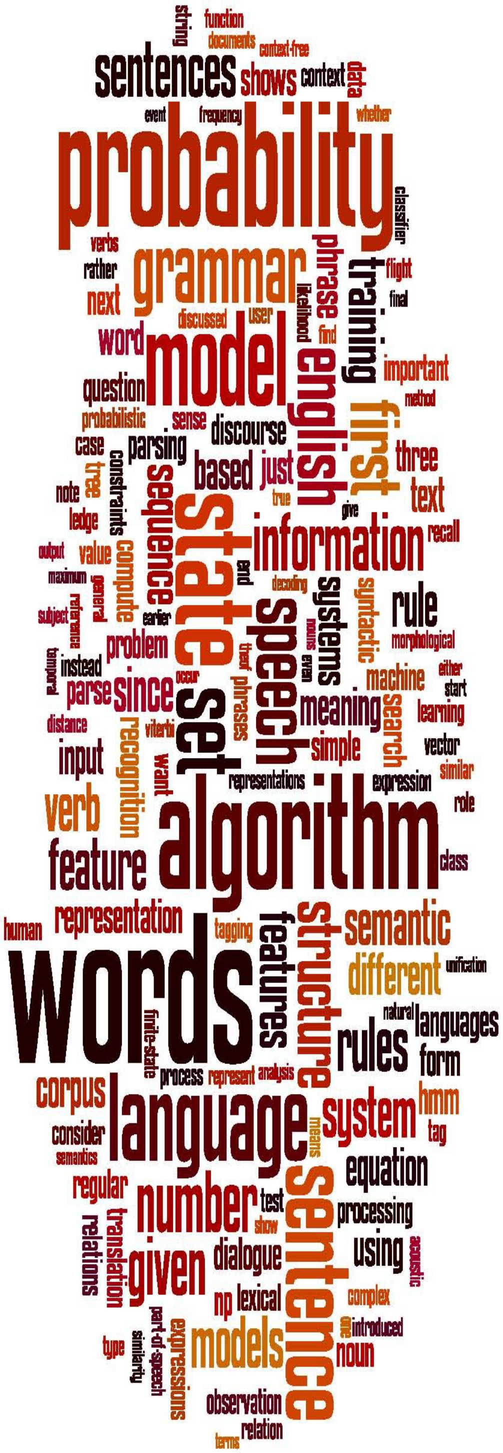
- Assuming exactly the same set up (3 class decision: LOCATION, PERSON, or DRUG; 3 features as before, maxent), what are:
  - P(PERSON | *by Goéric*) =
  - P(LOCATION | *by Goéric*) =
  - P(DRUG | *by Goéric*) =

  - 1.8 $f1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$
  - -0.6 $f2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$
  - 0.3 $f3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$

PERSON
by Goéric

LOCATION
by Goéric

DRUG
by Goéric



Building a Maxent Model

The nuts and bolts

Building a Maxent Model

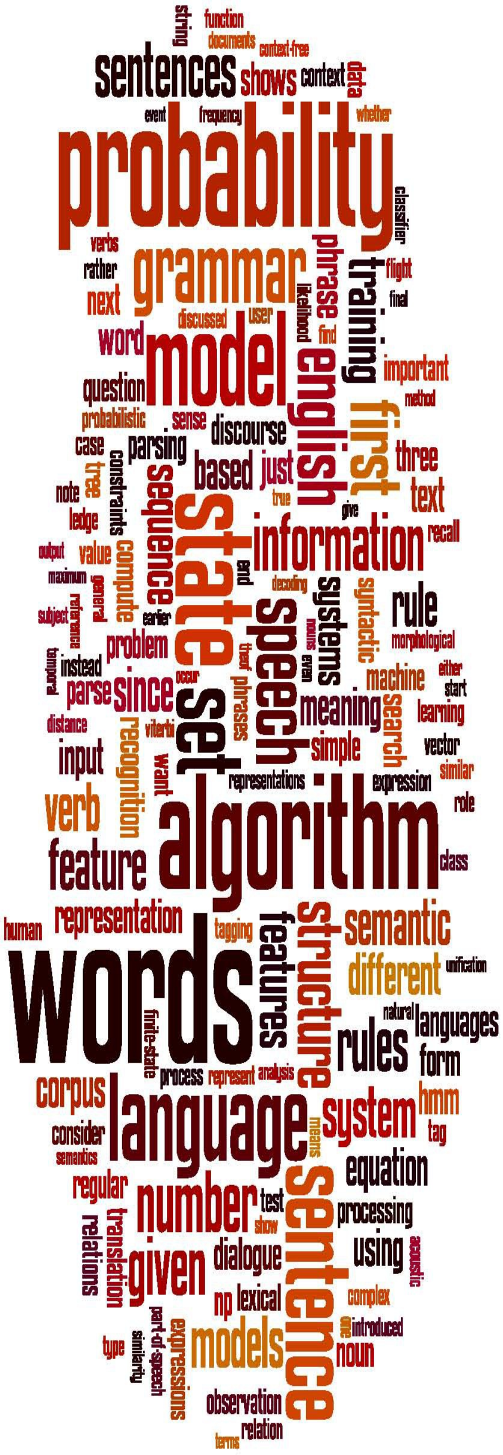
- We define features (indicator functions) over data points
  - Features represent sets of data points which are distinctive enough to deserve model parameters.
    - Words, but also "word contains number", "word ends with *ing*", etc.

- We will simply encode each Φ feature as a unique String
  - A datum will give rise to a set of Strings: the active Φ features
  - Each feature $f_i(c, d) \equiv [\Phi(d) \wedge c = c_j]$ gets a real number weight

- We concentrate on Φ features but the math uses i indices of f_i

Building a Maxent Model

- Features are often added during model development to target errors i.e. to get rid of training errors
  - Often, the easiest thing to think of are features that mark bad combinations
- Then, for any given feature weights, we want to be able to calculate:
  - Data conditional likelihood
  - Derivative of the likelihood wrt each feature weight
    - Uses expectations of each feature according to the model
- We can then find the optimum feature weights (discussed later).



Maxent Models and Discriminative Estimation

Maximizing the likelihood

Exponential Model Likelihood

- Maximum (Conditional) Likelihood Models :
  - Given a model form, choose values of parameters to maximize the (conditional) likelihood of the data.

$$\log P(C \mid D, \lambda) = \sum_{(c,d) \in (C,D)} \log P(c \mid d, \lambda) = \sum_{(c,d) \in (C,D)} \log \frac{\exp \sum_i \lambda_i f_i(c,d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c',d)}$$

The Likelihood Value

- The (log) conditional likelihood of iid data (*C*,*D*) according to maxent model is a function of the data and the parameters λ:

$$\log P(C|D,\lambda)=\log \prod_{(c,d)\in(C,D)} P(c|d,\lambda)=\sum_{(c,d)\in(C,D)} \log P(c|d,\lambda)$$

- If there aren't many values of *c*, it's easy to calculate:

$$\log P(c|d,\lambda)=\sum_{(c,d)\in(C,D)} \log \frac{\exp\left(\sum_i \lambda_i f_i(c,d)\right)}{\sum_{c'} \exp\left(\sum_i f_i(c',d)\right)}$$

The Likelihood Value

- We can separate this into two components:

$$\log P(c|d,\lambda) = \sum_{(c,d)\in(C,D)} \log\exp\left(\sum_i \lambda_i f_i(c,d)\right) - \sum_{(c,d)\in(C,D)} \log\sum_{c'} \exp\left(\sum_i f_i(c',d)\right)$$

$$\log P(c|d,\lambda) = N(\lambda) - M(\lambda)$$

- The derivative is the difference between the derivatives of each component

The Derivative I: Numerator

$$\frac{\partial N(\lambda)}{\partial \lambda_i} = \frac{\partial}{\partial \lambda_i}\left[\sum_{(c,d)\in(C,D)} \log \exp\left(\sum_i \lambda_i f_i(c,d)\right)\right]$$

$$= \sum_{(c,d)\in(C,D)} \frac{\partial}{\partial \lambda_i}\left[\sum_i \lambda_i f_i(c,d)\right]$$

$$= \sum_{(c,d)\in(C,D)} f_i(c,d)$$

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

The Derivative II: Denominator

$$\frac{\partial M(\lambda)}{\partial \lambda_i} = \frac{\partial}{\partial \lambda_i}\left[\sum_{(c,d)\in(C,D)} \log \sum_{c'} \exp\left(\sum_i \lambda_i f_i(c',d)\right)\right]$$

$$= \sum_{(c,d)} \frac{\partial}{\partial \lambda_i}\left[\log \sum_{c'} \exp\left(\sum_i \lambda_i f_i(c',d)\right)\right]$$

$$= \sum_{(c,d)} \frac{\frac{\partial}{\partial \lambda_i}\left[\sum_{c'} \exp\left(\sum_i \lambda_i f_i(c',d)\right)\right]}{\sum_{c''} \exp\left(\sum_i \lambda_i f_i(c'',d)\right)}$$

$$= \sum_{(c,d)} \frac{\sum_{c'} \exp\left(\sum_i \lambda_i f_i(c',d)\right)\left(\frac{\partial}{\partial \lambda_i}\left[\sum_i \lambda_i f_i(c',d)\right]\right)}{\sum_{c''} \exp\left(\sum_i \lambda_i f_i(c'',d)\right)}$$

The Derivative II: Denominator

$$\frac{\partial M(\lambda)}{\partial \lambda_i} = \sum_{(c,d)} \frac{\sum_{c'} \exp\left(\sum_i \lambda_i f_i(c',d)\right)\left(\frac{\partial}{\partial \lambda_i}\left[\sum_i \lambda_i f_i(c',d)\right)\right]}{\sum_{c''} \exp\left(\sum_i \lambda_i f_i(c'',d)\right)}$$

$$= \sum_{(c,d)} \frac{\sum_{c'} \exp\left(\sum_i \lambda_i f_i(c',d)\right)\left(f_i(c',d)\right)}{\sum_{c''} \exp\left(\sum_i \lambda_i f_i(c'',d)\right)}$$

$$= \sum_{(c,d)} \sum_{c'} \frac{\exp\left(\sum_i \lambda_i f_i(c',d)\right)}{\sum_{c''} \exp\left(\sum_i \lambda_i f_i(c'',d)\right)} f_i(c',d)$$

$$= \sum_{(c,d)} \sum_{c'} P(c'|d,\lambda) f_i(c',d)$$ predicted count(f_i, λ)

The Derivative III

$$\frac{\partial}{\partial \lambda_i} \log P(C|D,\lambda) = \text{actual count}(f_i, C) - \text{predicted count}(f_i, \lambda)$$

- The optimum parameters are the ones for which each feature's predicted expectation equals its empirical expectation. The optimum distribution is:
  - Always unique (but parameters may not be unique) as it's convex
  - Always exists (if feature counts are from actual data).
- These models are also called maximum entropy models because we find the model having maximum entropy and satisfying the constraints:

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

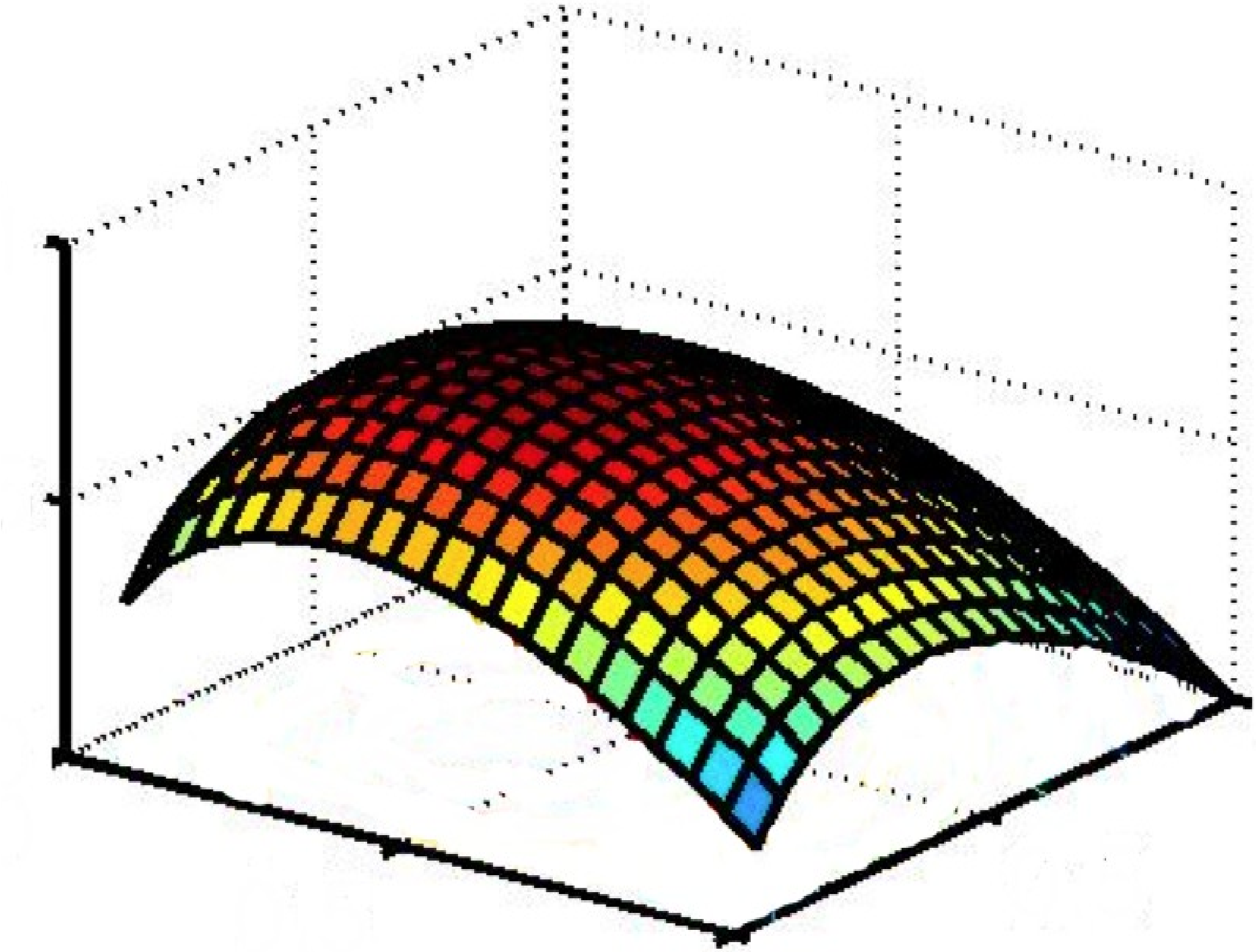
Finding the optimal parameters

- We want to choose parameters $\lambda_1, \lambda_2, \lambda_3, \ldots$ that maximize the conditional log-likelihood of the training data

$$CLogLike(D) = \sum_i \log P(c_i | d_i)$$

- To be able to do that, we've worked out how to calculate the function value and its partial derivatives (its gradient)

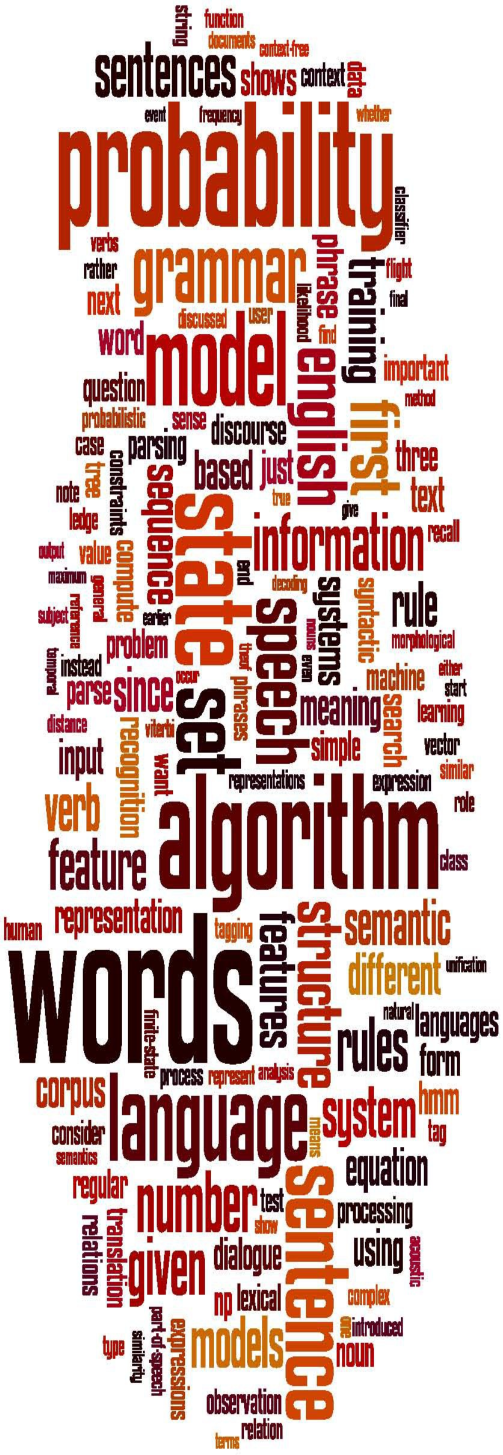
A likelihood surface



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 (used in the homework)



Smoothing/Priors/ Regularization for Maxent Models

Smoothing: Issues of Scale

- Lots of features:
  - NLP maxent models can have well over a 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 will never occur again at test time.
- Optimization problems:
  - Feature weights can be infinite, and iterative solvers can take a long time to get to those infinities.

Smoothing: Issues

- Assume the following empirical distribution:

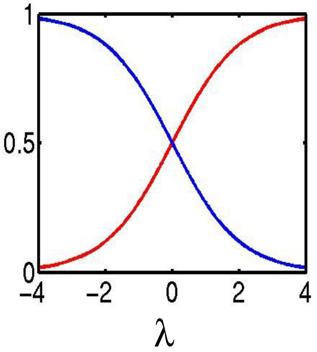
Heads	Tails
h	*t*

- Features: {Heads}, {Tails}
- We'll have the following model distribution:

$$p_{\text{HEADS}} = \frac{e^{\lambda_{\text{H}}}}{e^{\lambda_{\text{H}}} + e^{\lambda_{\text{T}}}} \qquad p_{\text{TAILS}} = \frac{e^{\lambda_{\text{T}}}}{e^{\lambda_{\text{H}}} + e^{\lambda_{\text{T}}}}$$

- Really, only one degree of freedom ($\lambda = \lambda_H - \lambda_T$)

$$p_{\text{HEADS}} = \frac{e^{\lambda_{\text{H}}} e^{-\lambda_{\text{T}}}}{e^{\lambda_{\text{H}}} e^{-\lambda_{\text{T}}} + e^{\lambda_{\text{T}}} e^{-\lambda_{\text{T}}}} = \frac{e^{\lambda}}{e^{\lambda} + e^{0}} \qquad p_{\text{TAILS}} = \frac{e^{0}}{e^{\lambda} + e^{0}}$$

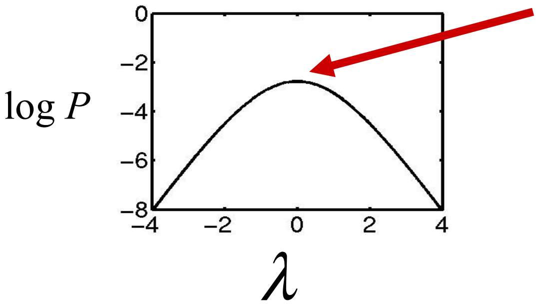


Smoothing: Issues

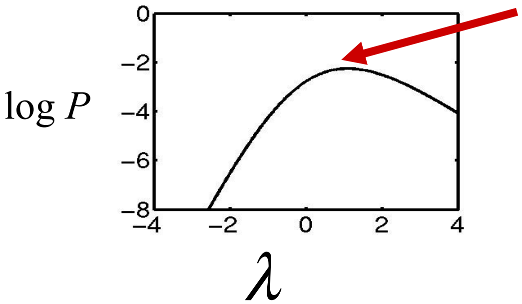
- The data likelihood in this model is:

$$\log P(h,t\,|\,\lambda) = h \log p_{\mathrm{HEADS}} + t \log p_{\mathrm{TAILS}}$$

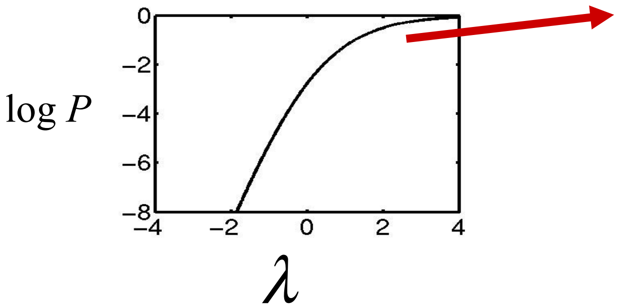
$$\log P(h,t\,|\,\lambda) = h\lambda - (t+h)\log(1+e^{\lambda})$$



Heads	Tails
2	2



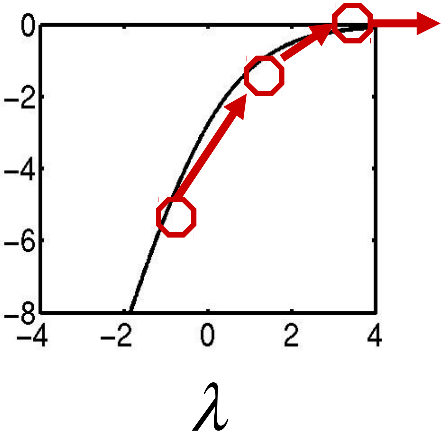
Heads	Tails
3	1



Heads	Tails
4	0

Smoothing: Early Stopping

- In the 4/0 case, there were two problems:
  - The optimal value of λ was ∞, which is a long 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 λ will be finite (but presumably big).
  - The optimization won't take forever (clearly).
  - Commonly used in early maxent work.

Heads	Tails
4	0

Input

Heads	Tails
1	0

Output

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:

$$\log P(C,\lambda \mid D) = \log P(\lambda) + \log P(C \mid D,\lambda)$$

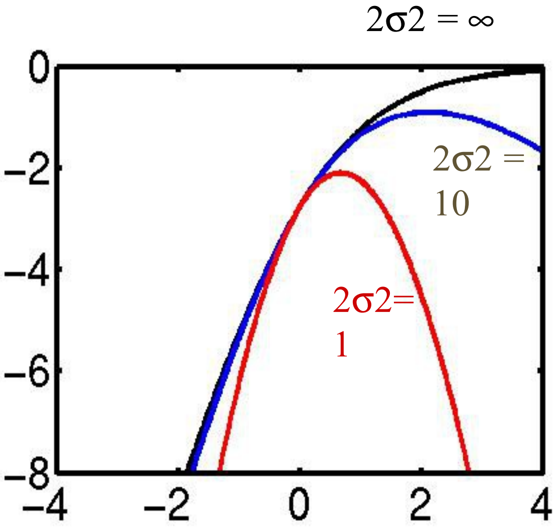
Posterior Prior Evidence

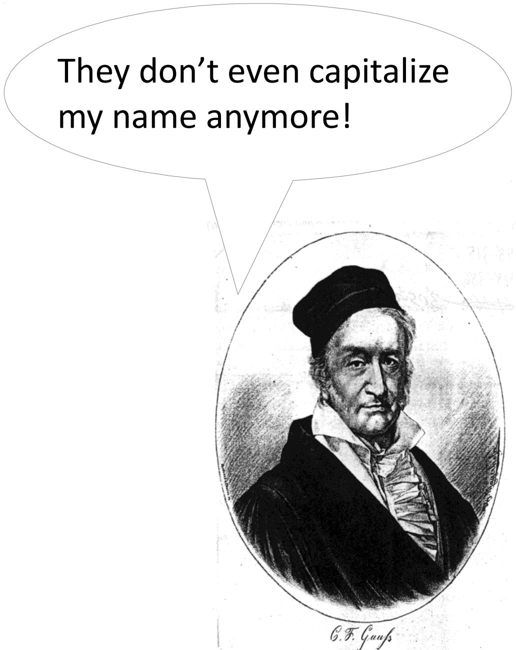
Smoothing: Priors

- Gaussian, or quadratic, or L2 priors:
  - Intuition: parameters shouldn't be large.
  - Formalization: prior expectation that each parameter will be distributed according to a gaussian with mean μ and variance σ2.

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

  - Penalizes parameters for drifting to far from their mean prior value (usually μ=0).
  - $2\sigma^2$=1 works surprisingly well.



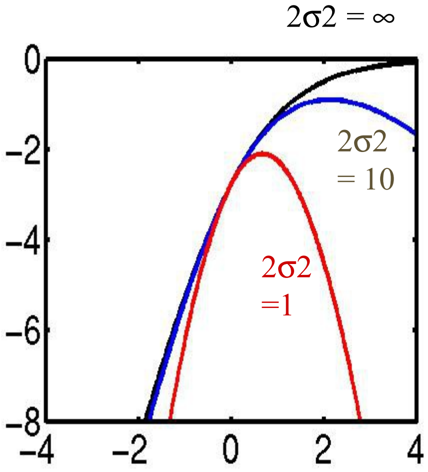


Smoothing: Priors

- If we use gaussian priors:
  - Trade off some expectation-matching for smaller parameters.
  - When multiple features can be recruited to explain a data point, the more common ones generally receive more weight.
  - Accuracy generally goes up!
- Change the objective:

$$\log P(C,\lambda \mid D) = \log P(C \mid D,\lambda) + \log P(\lambda)$$

$$\log P(C,\lambda \mid D) = \sum_{(c,d)\in(C,D)} P(c \mid d,\lambda) - \sum_i \frac{(\lambda_i - \mu_i)^2}{2\sigma_i^2} + k$$



- Change the derivative:

$$\partial \log P(C,\lambda \mid D)/\partial \lambda_i = \text{actual}(f_i,C) - \text{predicted}(f_i,\lambda) - (\lambda_i - \mu_i)/\sigma^2$$

Smoothing: Priors

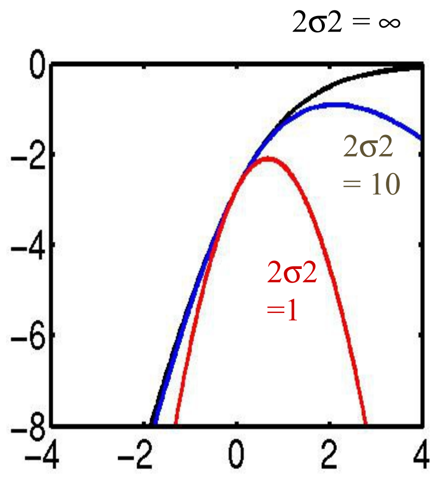
- If we use gaussian priors:
  - Trade off some expectation-matching for smaller parameters.
  - When multiple features can be recruited to explain a data point, the more common ones generally receive more weight.
  - Accuracy generally goes up!
- Change the objective:

$$\log P(C,\lambda \mid D) = \log P(C \mid D,\lambda) + \log P(\lambda)$$

$$\log P(C,\lambda \mid D) = \sum_{(c,d)\in(C,D)} P(c \mid d,\lambda) - \sum_i \frac{\lambda_i^2}{2\sigma_i^2} + k$$

- Change the derivative:

$$\partial \log P(C,\lambda \mid D)/\partial \lambda_i = \text{actual}(f_i,C) - \text{predicted}(f_i,\lambda) - \lambda_i/\sigma^2$$

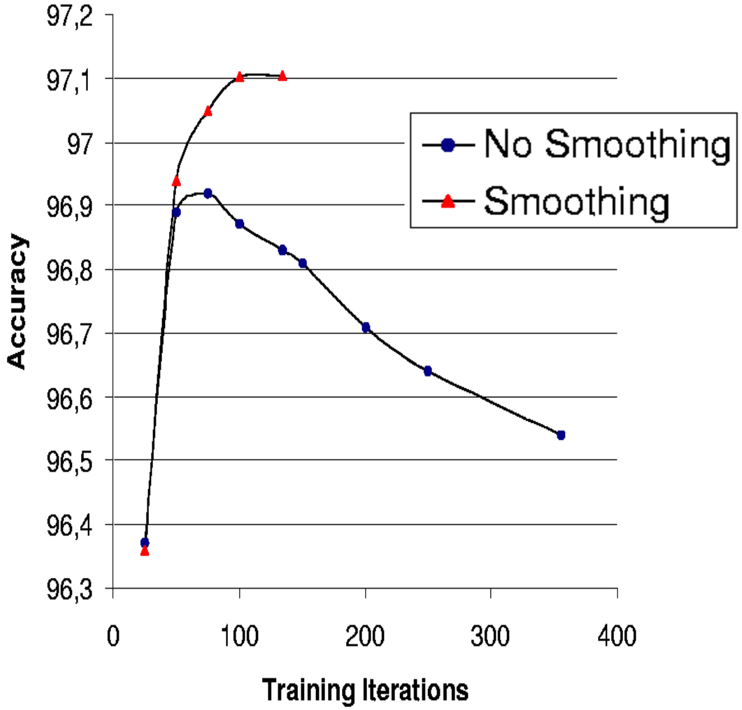


Taking prior mean as 0

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



- 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)!

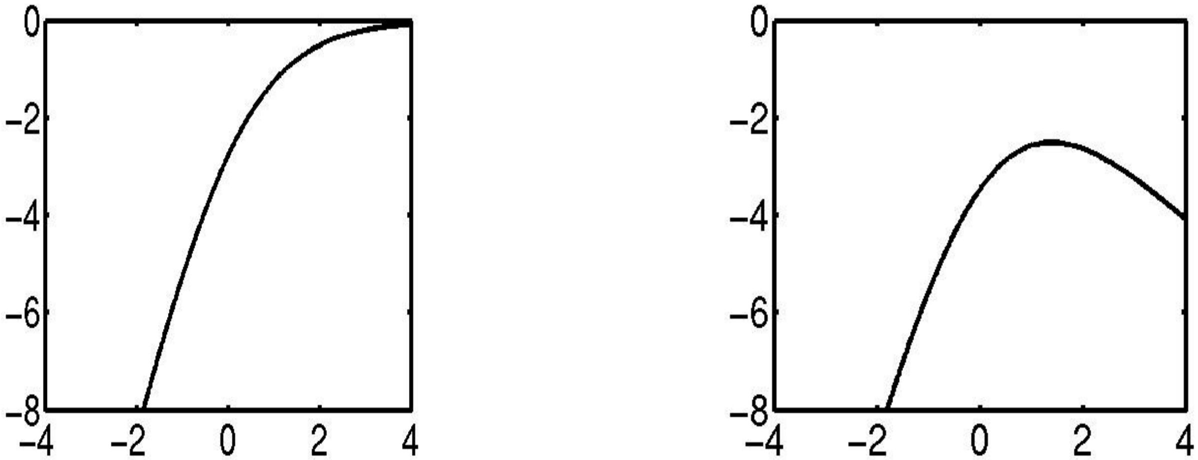
Smoothing: Regularization

- 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

Smoothing: Virtual Data

- Another option: smooth the data, not the parameters.
- Example:



Heads	Tails
4	0

→

Heads	Tails
5	1

  - Equivalent to adding two extra data points.
  - Similar to add-one smoothing for generative 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.
- We recommend: don’t use count cutoffs unless absolutely necessary for memory usage reasons.

Recap

- Generative Vs Discriminative Models
- Features
- MaxEnt Models
- Training
- Smoothing