Maxent Models, Conditional Estimation, and Optimization

Without Magic

That is, With Math!

Dan Klein and Chris Manning
Stanford University
http://nlp.stanford.edu/

HLT-NAACL 2003 and ACL 2003 Tutorial

Introduction

- In recent years there has been extensive use of *conditional* or *discriminative* probabilistic models in NLP, IR, and Speech
- 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

- Joint (generative) models place probabilities over both observed data and the hidden stuff (generate the observed data from hidden stuff):
  - All the best known StatNLP models:
    - *n*-gram models, Naive Bayes classifiers, hidden Markov models, probabilistic context-free grammars

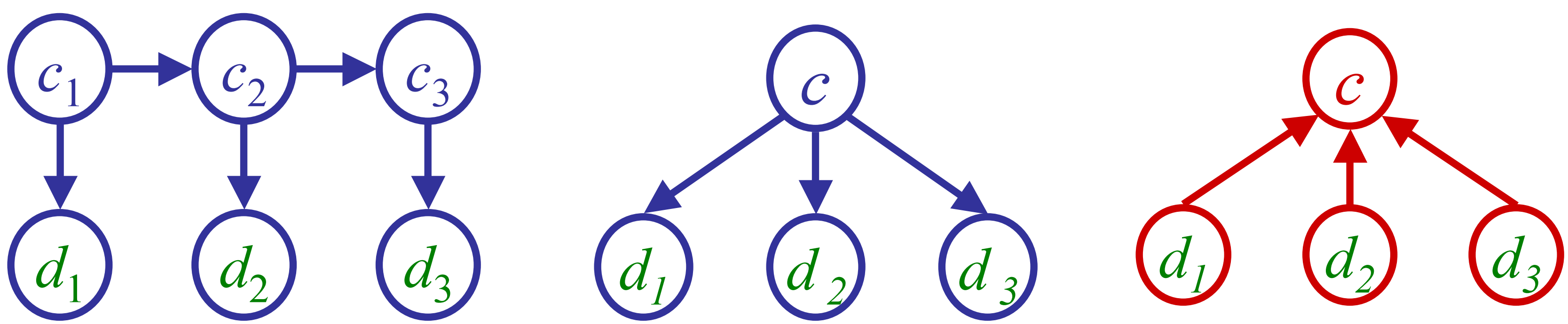
P(c,d)

- Discriminative (conditional) models take the data as given, and put a probability over hidden structure given the data:
    - Logistic regression, conditional loglinear models, maximum entropy markov models, (SVMs, perceptrons)

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



HMM Naive Bayes Logistic Regression

Generative Discriminative

Conditional models work well: Word Sense Disambiguation

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

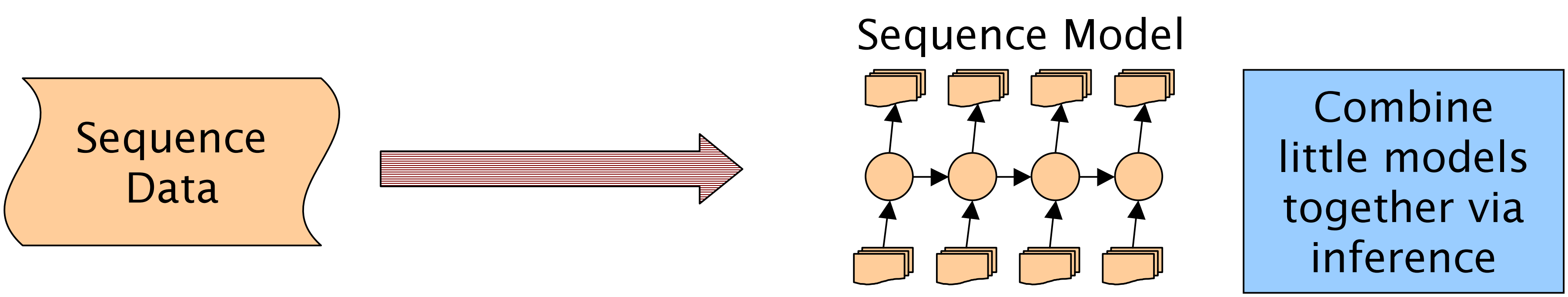
Test Set	
Objective	Accuracy
Joint Like.	73.6
Cond. Like.	76.1

- 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)

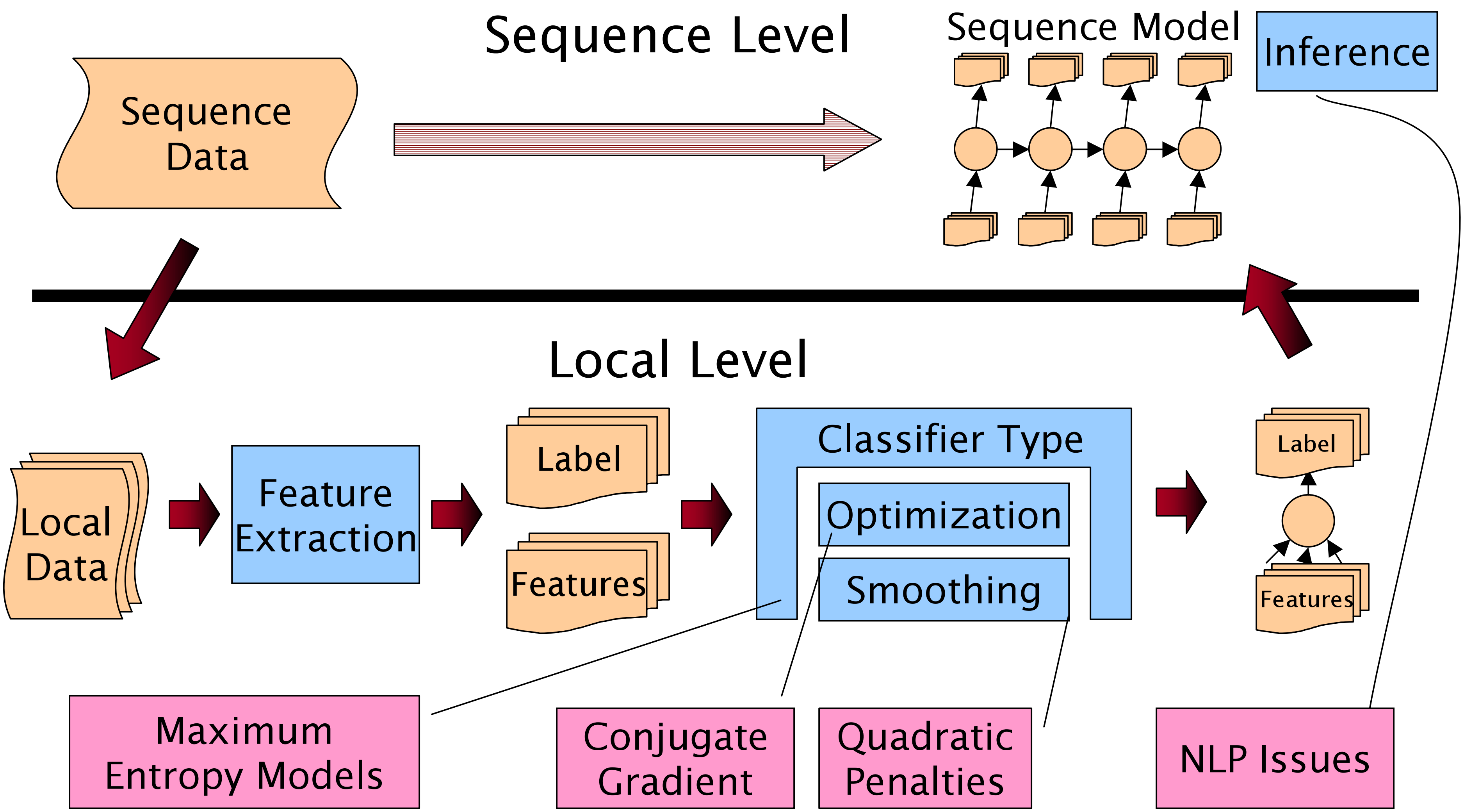
(Klein and Manning 2002, using Senseval-1 Data)

Overview: HLT Systems

- Typical Speech/NLP problems involve complex structures (sequences, pipelines, trees, feature structures, signals)
- Models are decomposed into individual local decision making locations
- Combining them together is the global inference problem



Overview: The Local Level



Tutorial Plan

1. Exponential/Maximum entropy models
2. Optimization methods
3. Linguistic issues in using these models

Part I: Maximum Entropy Models

a. Examples of Feature-Based Modeling

b. Exponential Models for Classification

c. Maximum Entropy Models

d. Smoothing

We will use the term "maxent" models, but will introduce them as loglinear or exponential models, deferring the interpretation as "maximum entropy models" until later.

Features

- In this tutorial and most maxent work: *features* are elementary pieces of evidence that link aspects of what we observe *d* with a category *c* that we want to predict.
- A feature has a real value: $f: C \times D \rightarrow \mathbf{R}$
- Usually features are indicator functions of properties of the input and a particular class (*every one we present is*). They pick out a subset.
  - $f_i(c, d) \equiv [\Phi(d) \wedge c = c_i]$ [Value is 0 or 1]
- We will freely say that $\Phi(d)$ is a feature of the data d, when, for each c_i, the conjunction $\Phi(d) \wedge c = c_i$ is a feature of the data-class pair (c, d).

Features

- For example:
  - $f_1(c, d) \equiv [c = \text{“NN”} \wedge \text{islower}(w_0) \wedge \text{ends}(w_0, \text{“d”})]$
  - $f_2(c, d) \equiv [c = \text{“NN”} \wedge w_{-1} = \text{“to”} \wedge t_{-1} = \text{“TO”}]$
  - $f_3(c, d) \equiv [c = \text{“VB”} \wedge \text{islower}(w_0)]$

IN NN	TO NN	TO VB	IN JJ
in bed	*to aid*	*to aid*	*in blue*

- Models will assign each feature a *weight*
- Empirical count (expectation) of a feature:

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

- Model expectation of a feature:

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

Feature-Based Models

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

Text Categorization	Word-Sense Disambiguation	POS Tagging
Data BUSINESS: Stocks hit a yearly low …	Data … to restructure bank:MONEY debt.	Data DT JJ NN … The previous fall …
Label BUSINESS Features {…, stocks, hit, a, yearly, low, …}	Label MONEY Features {…, P=restructure, N=debt, L=12, …}	Label NN Features {W=fall, PT=JJ PW=previous}

Example: Text Categorization

(Zhang and Oles 2001)

- Features are a word in document and class (they do feature selection to use reliable indicators)
- Tests on classic Reuters data set (and others)
  - Naïve Bayes: 77.0% F_1
  - Linear regression: 86.0%
  - Logistic regression: 86.4%
  - Support vector machine: 86.5%
- Emphasizes the importance of *regularization* (smoothing) for successful use of discriminative methods (not used in most early NLP/IR work)

Example: NER

(Klein et al. 2003; also, Borthwick 1999, etc.)

- Sequence model across words
- Each word classified by local model
- Features include the word, previous and next words, previous classes, previous, next, and current POS tag, character *n*-gram features and *shape* of word
  - Best model had > 800K features
- High (> 92% on English devtest set) performance comes from combining many informative features.
- With smoothing / regularization, more features never hurt!

Decision Point:

State for *Grace*

Local Context

	Prev	Cur	Next
Class	Other	???	???
Word	at	Grace	Road
Tag	IN	NNP	NNP
Sig	x	Xx	Xx

Example: NER

(Klein et al. 2003)

Decision Point:
State for *Grace*

Local Context

	Prev	Cur	Next
Class	Other	???	???
Word	at	Grace	Road
Tag	IN	NNP	NNP
Sig	x	Xx	Xx

Feature Weights

Feature Type	**Feature**	**PERS**	**LOC**
Previous word	*at*	-0.73	0.94
Current word	*Grace*	0.03	0.00
Beginning bigram	*<G*	0.45	-0.04
Current POS tag	NNP	0.47	0.45
Prev and cur tags	IN NNP	-0.10	0.14
Previous state	Other	-0.70	-0.92
Current signature	Xx	0.80	0.46
Prev state, cur sig	O-Xx	0.68	0.37
Prev-cur-next sig	x-Xx-Xx	-0.69	0.37
P. state - p-cur sig	O-x-Xx	-0.20	0.82
…			
Total:		**-0.58**	**2.68**

Example: Tagging

- Features can include:
  - Current, previous, next words in isolation or together.
  - Previous (or next) one, two, three tags.
  - Word-internal features: word types, suffixes, dashes, etc.

Decision Point

Local Context

-3	-2	-1	0	+1
DT	NNP	VBD	???	???
The	Dow	fell	22.6	%

(Ratnaparkhi 1996; Toutanova et al. 2003, etc.)

Features

W_0	22.6
W_{+1}	%
W_{-1}	fell
T_{-1}	VBD
T_{-1}-T_{-2}	NNP-VBD
hasDigit?	true
…	…

Other Maxent Examples

- Sentence boundary detection (Mikheev 2000)
  - Is period end of sentence or abbreviation?
- PP attachment (Ratnaparkhi 1998)
  - Features of head noun, preposition, etc.
- Language models (Rosenfeld 1996)
  - $P(w_0|w_{-n},\ldots,w_{-1})$. Features are word n-gram features, and trigger features which model repetitions of the same word.
- Parsing (Ratnaparkhi 1997; Johnson et al. 1999, etc.)
  - Either: Local classifications decide parser actions or feature counts choose a parse.

Conditional vs. Joint Likelihood

- We have some data $\{(d, c)\}$ and we want to place probability distributions over it.
- A *joint* model gives probabilities $P(d,c)$ and tries to maximize this 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.

Feature-Based Classifiers

- "Linear" classifiers:
  - Classify from features sets $\{f_i\}$ to classes $\{c\}$.
  - Assign a weight λ_i to each feature f_i.
  - 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 $\Sigma\lambda_i f_i(c,d) = \text{VB}$
  - There are many ways to chose weights
    - Perceptron: find a currently misclassified example, and nudge weights in the direction of a correct classification

Feature-Based Classifiers

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

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

Makes votes positive.

Normalizes votes.

    - P(NN|*to, aid,* TO) = $e^{1.2}e^{-1.8}/(e^{1.2}e^{-1.8} + e^{0.3}) = 0.29$
    - P(VB|*to, aid,* TO) = $e^{0.3}/(e^{1.2}e^{-1.8} + e^{0.3}) = 0.71$
  - The weights are the parameters of the probability model, combined via a "soft max" function
  - Given this model form, we will choose parameters $\{\lambda_i\}$ that *maximize the conditional likelihood* of the data according to this model.

Other Feature-Based Classifiers

- The exponential model approach is one way of deciding how to weight features, given data.
- It constructs not only classifications, but probability distributions over classifications.
- There are other (good!) ways of discriminating classes: SVMs, boosting, even perceptrons – though these methods are not as trivial to interpret as distributions over classes.
- We'll see later what maximizing the conditional likelihood according to the exponential model has to do with entropy.

Exponential Model Likelihood

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

- Exponential model form, for a data set (C,D):

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

Building a Maxent Model

- Define features (indicator functions) over data points.
  - Features represent sets of data points which are distinctive enough to deserve model parameters.
  - Usually features are added incrementally to "target" errors.
- For any given feature weights, we want to be able to calculate:
  - Data (conditional) likelihood
  - Derivative of the likelihood wrt each feature weight
    - Use expectations of each feature according to the model
- Find the optimum feature weights (next part).

The Likelihood Value

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

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

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

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

- We can separate this into two components:

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

$$\log P(C \mid D, \lambda) = N(\lambda) - M(\lambda)$$

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

The Derivative I: Numerator

$$\frac{\partial N(\lambda)}{\partial \lambda_i} = \frac{\partial \sum_{(c,d)\in(C,D)} \log \exp \sum_i \lambda_{ci} f_i(c,d)}{\partial \lambda_i} = \frac{\partial \sum_{(c,d)\in(C,D)} \sum_i \lambda_i f_i(c,d)}{\partial \lambda_i}$$

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

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

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

The Derivative II: Denominator

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

$$= \sum_{(c,d)\in(C,D)} \frac{1}{\sum_{c''} \exp \sum_i \lambda_i f_i(c'',d)} \frac{\partial \sum_{c'} \exp \sum_i \lambda_i f_i(c',d)}{\partial \lambda_i}$$

$$= \sum_{(c,d)\in(C,D)} \frac{1}{\sum_{c''} \exp \sum_i \lambda_i f_i(c'',d)} \sum_{c'} \frac{\exp \sum_i \lambda_i f_i(c',d)}{1} \frac{\partial \sum_i \lambda_i f_i(c',d)}{\partial \lambda_i}$$

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

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

The Derivative III

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

- 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)
  - Always exists (if features counts are from actual data).
- Features can have high model expectations (predicted counts) either because they have large weights or because they occur with other features which have large weights.

Summary so far

- We have a function to optimize:

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

- We know the function's derivatives:

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

- Perfect situation for general optimization (Part II)
- But first … what has all this got to do with maximum entropy models?

Maximum Entropy Models

- An equivalent approach:
  - Lots of distributions out there, most of them very spiked, specific, overfit.
  - We want a distribution which is uniform except in specific ways we require.
  - Uniformity means high entropy – we can search for distributions which have properties we desire, but also have high entropy.

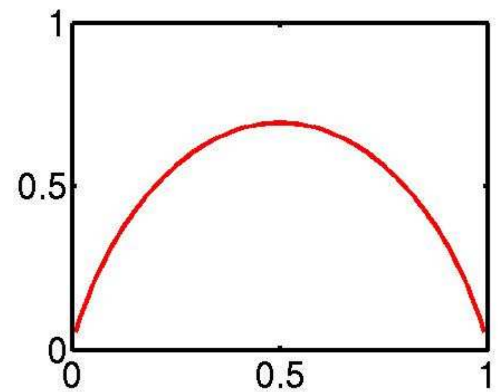
(Maximum) Entropy

- Entropy: the uncertainty of a distribution.
- Quantifying uncertainty ("surprise"):
  - Event x
  - Probability p_x
  - "Surprise" $\log(1/p_x)$
- Entropy: expected surprise (over p):

$$\mathrm{H}(p) = E_p\left[\log \frac{1}{p_x}\right]$$

$$\mathrm{H}(p) = -\sum_x p_x \log p_x$$

H



p_{HEADS}

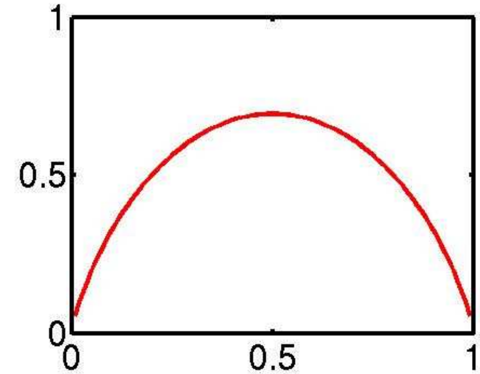
A coin-flip is most uncertain for a fair coin.

Maxent Examples I

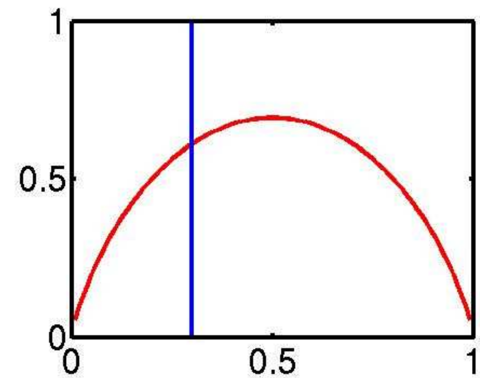
- What do we want from a distribution?
  - Minimize commitment = maximize entropy.
  - Resemble some reference distribution (data).
- Solution: maximize entropy H, subject to feature-based constraints:

$$E_p[f_i] = E_{\hat{p}}[f_i] \Leftrightarrow \sum_{x \in f_i} p_x = C_i$$

- Adding constraints (features):
  - Lowers maximum entropy
  - Raises maximum likelihood of data
  - Brings the distribution further from uniform
  - Brings the distribution closer to data

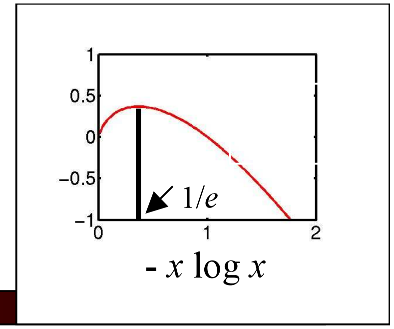


Unconstrained, max at 0.5

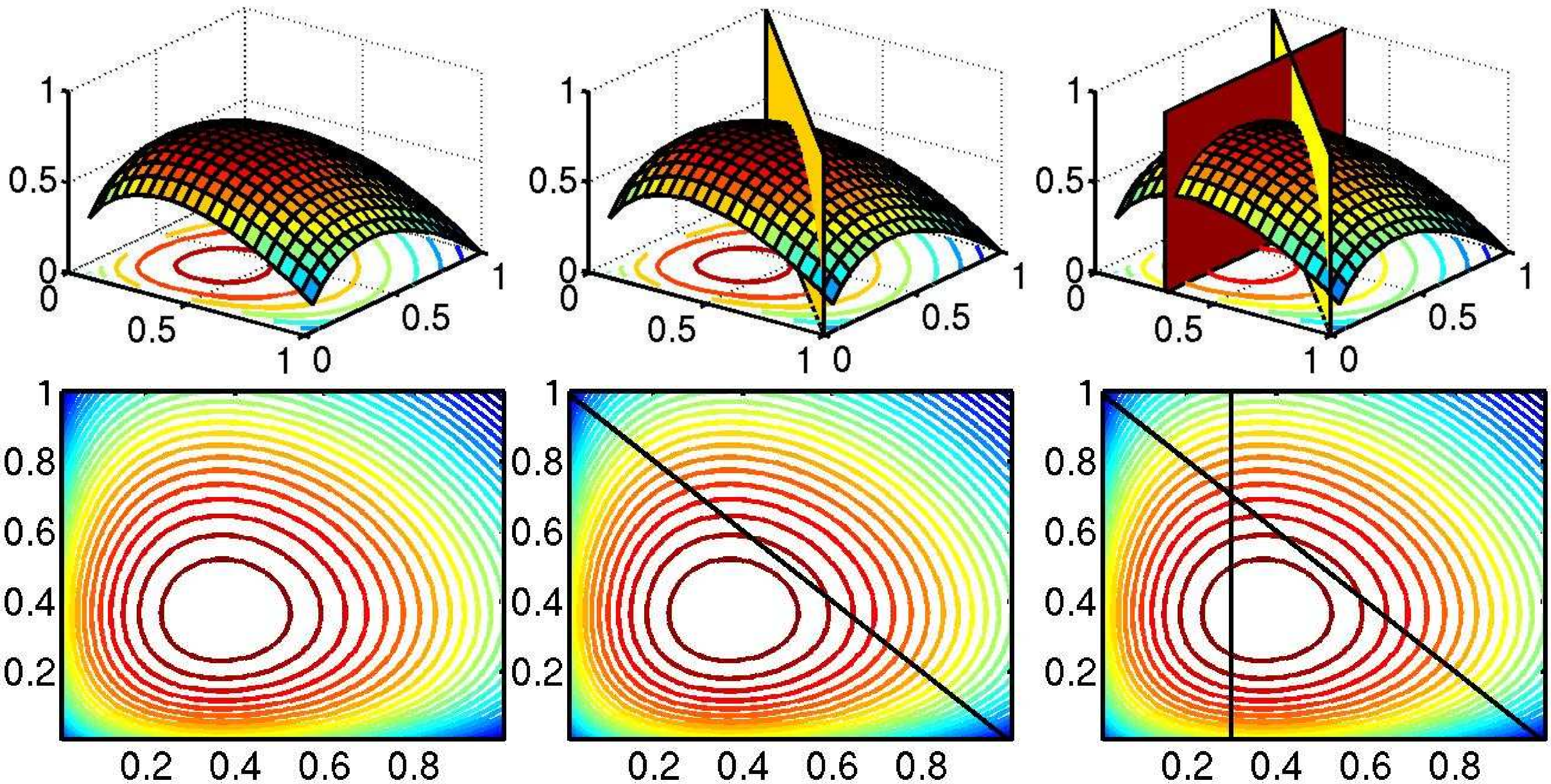


Constraint that $p_{\text{HEADS}} = 0.3$

Maxent Examples II



$H(p_H p_T,)$ $\quad$ $p_H + p_T = 1$ $\quad$ $p_H = 0.3$



Maxent Examples III

- Lets say we have the following event space:

NN	NNS	NNP	NNPS	VBZ	VBD

- … and the following empirical data:

3	5	11	13	3	1

- Maximize H:

$1/e$	$1/e$	$1/e$	$1/e$	$1/e$	$1/e$

- … want probabilities: E[NN,NNS,NNP,NNPS,VBZ,VBD] = 1

1/6	1/6	1/6	1/6	1/6	1/6

Maxent Examples IV

- Too uniform!
- N* are more common than V*, so we add the feature f_N = {NN, NNS, NNP, NNPS}, with E[f_N] =32/36

NN	NNS	NNP	NNPS	VBZ	VBD
8/36	8/36	8/36	8/36	2/36	2/36

- … and proper nouns are more frequent than common nouns, so we add f_P = {NNP, NNPS}, with E[f_P] =24/36

4/36	4/36	12/36	12/36	2/36	2/36

- … we could keep refining the models, e.g. by adding a feature to distinguish singular vs. plural nouns, or verb types.

Feature Overlap

- Maxent models handle overlapping features well.
- Unlike a NB model, there is no double counting!

Empirical

	A	a
B	2	1
b	2	1

	A	a
B		
b		

All = 1

	A	a
B	1/4	1/4
b	1/4	1/4

	A	a
B		
b		

	A	a
B		
b		

A = 2/3

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

	A	a
B	λ_A	
b	λ_A	

	A	a
B		
b		

A = 2/3

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

	A	a
B	$\lambda'_A+\lambda''_A$	
b	$\lambda'_A+\lambda''_A$	

Example: NER Overlap

Grace is correlated with PERSON, but does not add much evidence on top of already knowing prefix features.

Local Context

	Prev	Cur	Next
State	Other	???	???
Word	at	Grace	Road
Tag	IN	NNP	NNP
Sig	x	Xx	Xx

Feature Weights

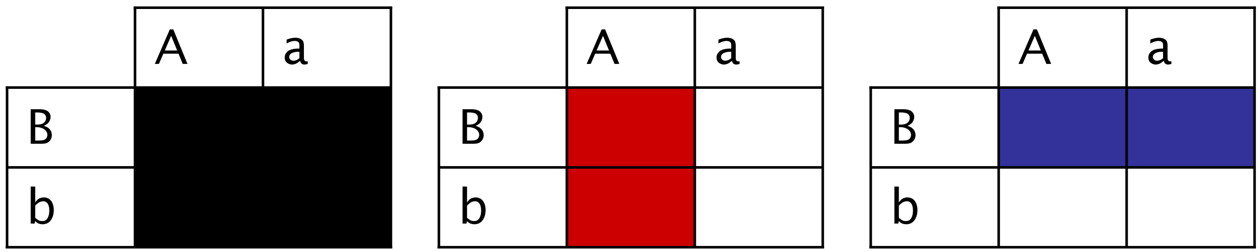
Feature Type	**Feature**	**PERS**	**LOC**
Previous word	*at*	-0.73	0.94
Current word	*Grace*	0.03	0.00
Beginning bigram	*<G*	0.45	-0.04
Current POS tag	NNP	0.47	0.45
Prev and cur tags	IN NNP	-0.10	0.14
Previous state	Other	-0.70	-0.92
Current signature	Xx	0.80	0.46
Prev state, cur sig	O-Xx	0.68	0.37
Prev-cur-next sig	x-Xx-Xx	-0.69	0.37
P. state - p-cur sig	O-x-Xx	-0.20	0.82
…			
Total:		**-0.58**	**2.68**

Feature Interaction

- Maxent models handle overlapping features well, but do not automatically model feature interactions.

Empirical

	A	a
B	1	1
b	1	0



All = 1 | A = 2/3 | B = 2/3

	A	a
B	1/4	1/4
b	1/4	1/4

	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	0	0
b	0	0

	A	a
B	λ_A	
b	λ_A	

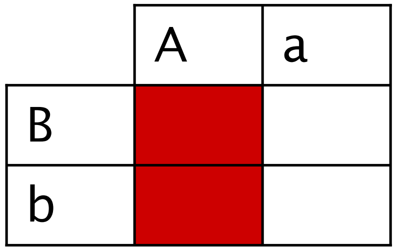
	A	a
B	$\lambda_A+\lambda_B$	λ_B
b	λ_A	

Feature Interaction

- If you want interaction terms, you have to add them:

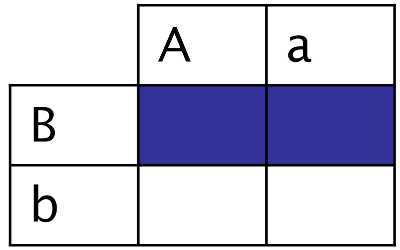
Empirical

	A	a
B	1	1
b	1	0



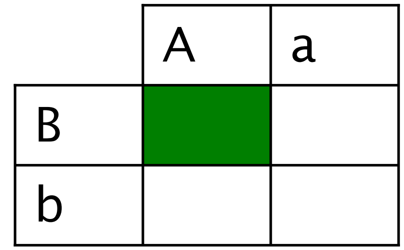
A = 2/3

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



B = 2/3

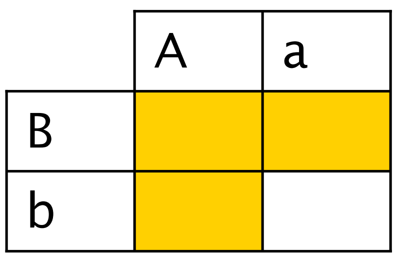
	A	a
B	4/9	2/9
b	2/9	1/9



AB = 1/3

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

- A disjunctive feature would also have done it (alone):



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

Feature Interaction

- For loglinear/logistic regression models in statistics, it is standard to do a greedy stepwise search over the space of all possible interaction terms.
- This combinatorial space is exponential in size, but that's okay as most statistics models only have 4–8 features.
- In NLP, our models commonly use hundreds of thousands of features, so that's not okay.
- Commonly, interaction terms are added by hand based on linguistic intuitions.

Example: NER Interaction

Previous-state and current-signature have interactions, e.g. P=PERS-C=Xx indicates C=PERS much more strongly than C=Xx and P=PERS independently.

This feature type allows the model to capture this interaction.

Local Context

	Prev	Cur	Next
State	Other	???	???
Word	at	Grace	Road
Tag	IN	NNP	NNP
Sig	x	Xx	Xx

Feature Weights

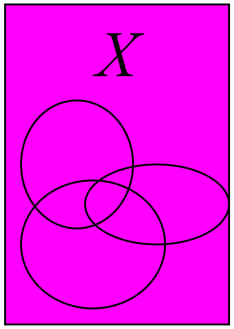
Feature Type	Feature	PERS	LOC
Previous word	*at*	-0.73	0.94
Current word	*Grace*	0.03	0.00
Beginning bigram	*<G*	0.45	-0.04
Current POS tag	NNP	0.47	0.45
Prev and cur tags	IN NNP	-0.10	0.14
Previous state	Other	-0.70	-0.92
Current signature	Xx	0.80	0.46
Prev state, cur sig	O-Xx	0.68	0.37
Prev-cur-next sig	x-Xx-Xx	-0.69	0.37
P. state - p-cur sig	O-x-Xx	-0.20	0.82
…			
Total:		**-0.58**	**2.68**

Classification

- What do these joint models of $\mathrm{P}(X)$ have to do with conditional models $\mathrm{P}(C|D)$?
- Think of the space $C \times D$ as a complex X.
  - C is generally small (e.g., 2-100 topic classes)
  - D is generally huge (e.g., number of documents)
- We can, in principle, build models over $\mathrm{P}(C,D)$.
- This will involve calculating expectations of features (over $C \times D$):

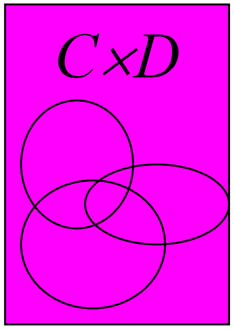
$$E(f_i) = \sum_{(c,d)\in(C,D)} P(c,d) f_i(c,d)$$

- Generally impractical: can't enumerate d efficiently.



C

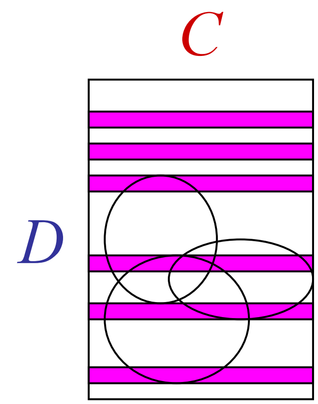
D



Classification II

- D may be huge or infinite, but only a few d occur in our data.
- What if we add one feature for each d and constrain its expectation to match our empirical data?

$$\forall (d) \in D \quad P(d) = \hat{P}(d)$$



- Now, most entries of $P(c,d)$ will be zero.
- We can therefore use the much easier sum:

$$E(f_i) = \sum_{(c,d)\in(C,D)} P(c,d) f_i(c,d)$$
$$= \sum_{(c,d)\in(C,D)\wedge \hat{P}(d)>0} P(c,d) f_i(c,d)$$

Classification III

- But if we've constrained the D marginals

$$\forall(d) \in D \quad P(d) = \hat{P}(d)$$

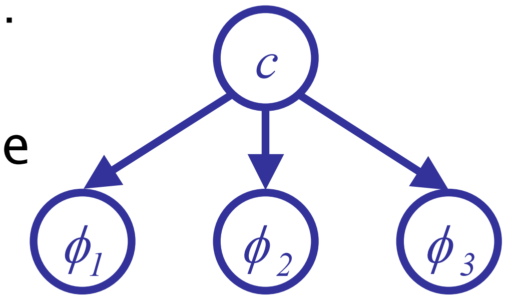
then the only thing that can vary is the conditional distributions:

$$P(c,d) = P(c \mid d)P(d)$$
$$= P(c \mid d)\hat{P}(d)$$

- This is the connection between joint and conditional maxent / exponential models:
  - Conditional models can be thought of as joint models with marginal constraints.
- Maximizing joint likelihood and conditional likelihood of the data in this model are equivalent!

Comparison to Naïve-Bayes

- Naïve-Bayes is another tool for classification:
  - We have a bunch of random variables (data features) which we would like to use to predict another variable (the class):



  - The Naïve-Bayes likelihood over classes is:

$$P(c \mid d, \lambda) = \frac{P(c)\prod_i P(\phi_i \mid c)}{\sum_{c'} P(c')\prod_i P(\phi_i \mid c')} \Rightarrow \frac{\exp\left[\log P(c) + \sum_i \log P(\phi_i \mid c)\right]}{\sum_{c'} \exp\left[\log P(c') + \sum_i \log P(\phi_i \mid c')\right]}$$

$$\Rightarrow \frac{\exp\left[\sum_i \lambda_{ic} f_{ic}(d, c)\right]}{\sum_{c'} \exp\left[\sum_i \lambda_{ic'} f_{ic'}(d, c')\right]}$$

Naïve-Bayes is just an exponential model.

Comparison to Naïve-Bayes

- The primary differences between Naïve-Bayes and maxent models are:

Naïve-Bayes

Trained to maximize joint likelihood of data and classes.
Features assumed to supply independent evidence.
Feature weights can be set independently.

Features must be of the conjunctive $\Phi(d) \wedge c = c_i$ form.

Maxent

Trained to maximize the conditional likelihood of classes.
Features weights take feature dependence into account.
Feature weights must be mutually estimated.

Features need not be of the conjunctive form (but usually are).

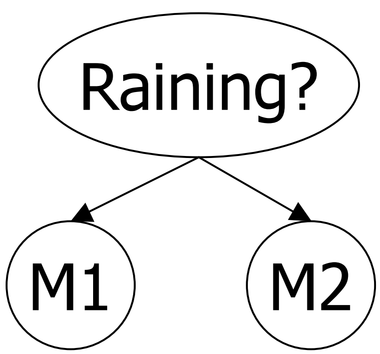
Example: Sensors

Reality

Raining — Sunny

P(+,+,r) = 3/8 P(-,-,r) = 1/8 P(+,+,s) = 1/8 P(-,-,s) = 3/8

NB Model



NB FACTORS:

- P(s) = 1/2
- P(+|s) = 1/4
- P(+|r) = 3/4

PREDICTIONS:

- P(r,+,+) = (½)(¾)(¾)
- P(s,+,+) = (½)(¼)(¼)
- P(r|+,+) = 9/10
- P(s|+,+) = 1/10

Example: Sensors

- Problem: NB multi-counts the evidence.

$$\frac{P(r \mid +\ldots+)}{P(s \mid +\ldots+)} = \frac{P(r)}{P(s)} \frac{P(+ \mid r)}{P(+ \mid s)} \cdots \frac{P(+ \mid r)}{P(+ \mid s)}$$

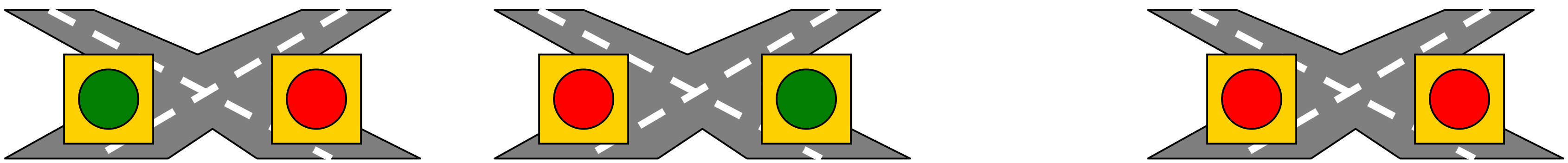
- Maxent behavior:
  - Take a model over $(M_1, \ldots M_n, R)$ with features:
    - f_{ri}: **M_i=+, R=r** weight: λ_{ri}
    - f_{si}: **M_i=+, R=s** weight: λ_{si}
  - $\exp(\lambda_{ri}-\lambda_{si})$ is the factor analogous to $P(+|r)/P(+|s)$
  - … but instead of being 3, it will be $3^{1/n}$
  - … because if it were 3, $E[f_{ri}]$ would be far higher than the target of 3/8!

Example: Stoplights

Reality

Lights Working

Lights Broken

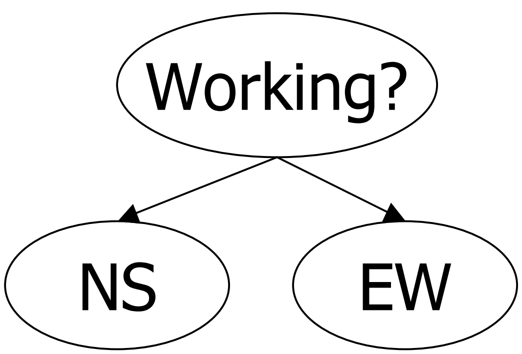


$P(g,r,w) = 3/7$

$P(r,g,w) = 3/7$

$P(r,r,b) = 1/7$

NB Model



NB FACTORS:

- $P(w) = 6/7$
- $P(r|w) = 1/2$
- $P(g|w) = 1/2$
- $P(b) = 1/7$
- $P(r|b) = 1$
- $P(g|b) = 0$

Example: Stoplights

- What does the model say when both lights are red?
  - P(b,r,r) = (1/7)(1)(1) = 1/7 = 4/28
  - P(w,r,r) = (6/7)(1/2)(1/2) = 6/28 = 6/28
  - P(w|r,r) = 6/10!
- We'll guess that (r,r) indicates lights are working!

- Imagine if P(b) were boosted higher, to 1/2:
  - P(b,r,r) = (1/2)(1)(1) = 1/2 = 4/8
  - P(w,r,r) = (1/2)(1/2)(1/2) = 1/8 = 1/8
  - P(w|r,r) = 1/5!
- Changing the parameters, bought conditional accuracy at the expense of data likelihood!

Smoothing: Issues of Scale

- Lots of features:
  - NLP maxent models can have over 1M features.
  - Even storing a single array of parameter values can have a substantial memory cost.
- Lots of sparsity:
  - Overfitting very easy – 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:

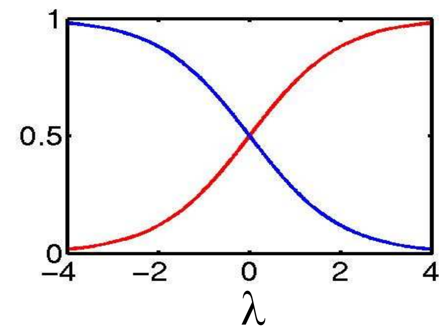
Heads	Tails
h	t

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

$$p_{\text{HEADS}} = \frac{e^{\lambda_H}}{e^{\lambda_H} + e^{\lambda_T}} \qquad p_{\text{TAILS}} = \frac{e^{\lambda_T}}{e^{\lambda_H} + e^{\lambda_T}}$$

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

$$p_{\text{HEADS}} = \frac{e^{\lambda_H} e^{-\lambda_T}}{e^{\lambda_H} e^{-\lambda_T} + e^{\lambda_T} e^{-\lambda_T}} = \frac{e^{\lambda}}{e^{\lambda} + e^{0}} \quad p_{\text{TAILS}} = \frac{e^{0}}{e^{\lambda} + e^{0}}$$

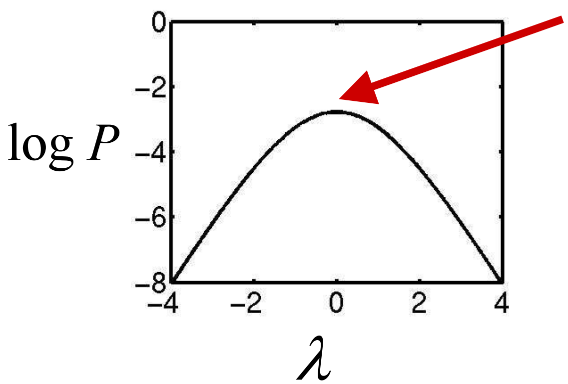


Smoothing: Issues

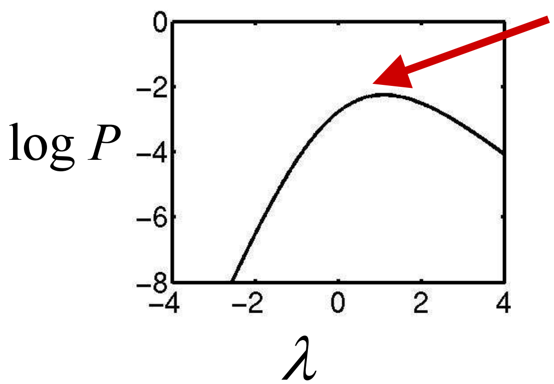
- The data likelihood in this model is:

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

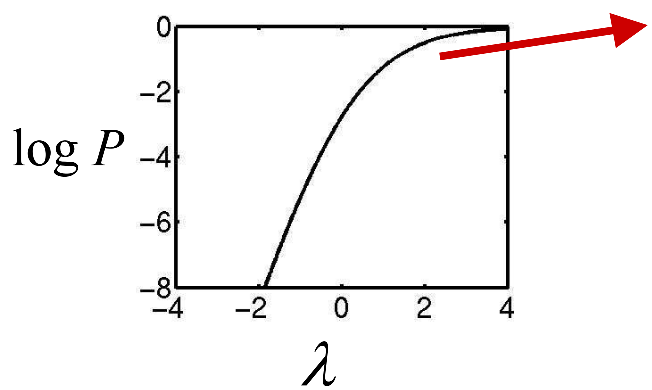
$$\log P(h,t \mid \lambda) = h\lambda - (t+h)\log(1+e^{\lambda})$$



Heads	Tails
2	2



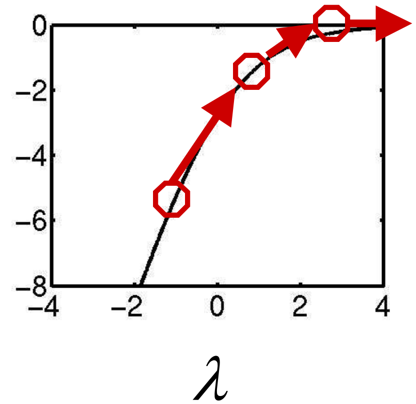
Heads	Tails
3	1



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:

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

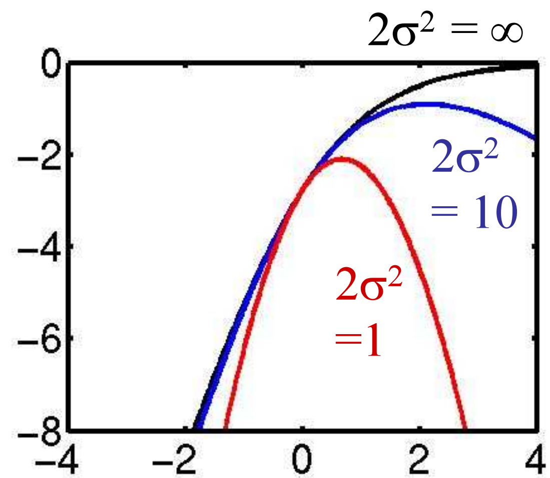
Posterior Prior Evidence

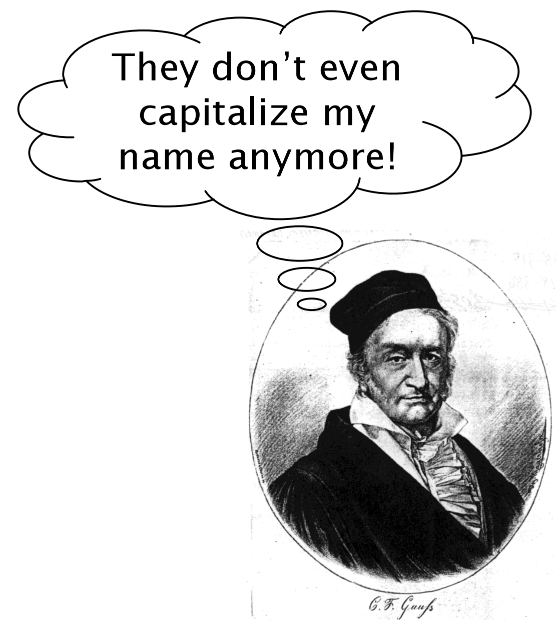
Smoothing: Priors

- Gaussian, or quadratic, 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 σ^2.

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

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



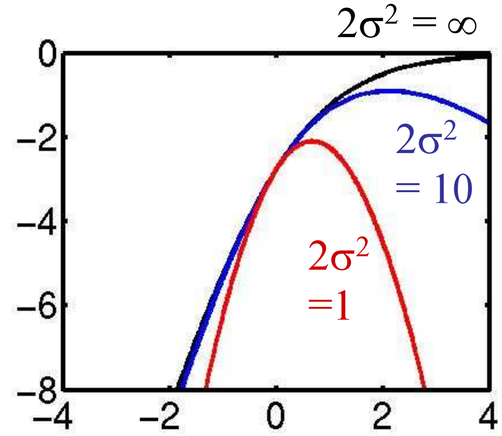


Smoothing: Priors

- 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:

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

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

	Prev	Cur	Next
State	Other	???	???
Word	at	Grace	Road
Tag	IN	NNP	NNP
Sig	x	Xx	Xx

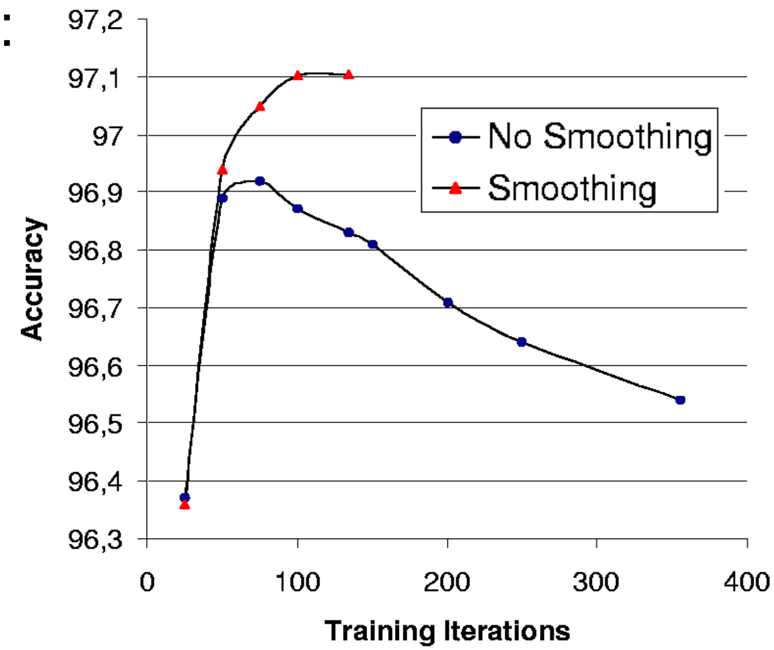
Feature Weights

Feature Type	**Feature**	**PERS**	**LOC**
Previous word	*at*	-0.73	0.94
Current word	*Grace*	0.03	0.00
Beginning bigram	*<G*	0.45	-0.04
Current POS tag	NNP	0.47	0.45
Prev and cur tags	IN NNP	-0.10	0.14
Previous state	Other	-0.70	-0.92
Current signature	Xx	0.80	0.46
Prev state, cur sig	O-Xx	0.68	0.37
Prev-cur-next sig	x-Xx-Xx	-0.69	0.37
P. state - p-cur sig	O-x-Xx	-0.20	0.82
…			
Total:		**-0.58**	**2.68**

Example: POS Tagging

- From (Toutanova et al., 2003):

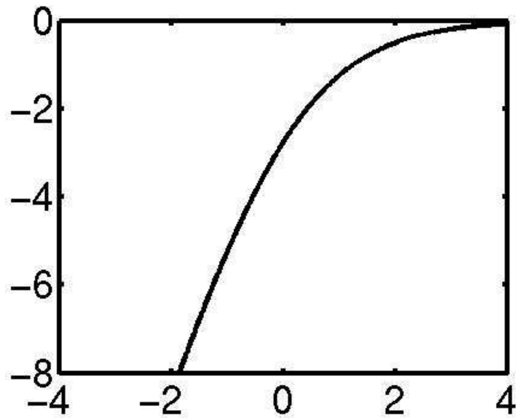
	Overall Accuracy	Unknown Word Acc
Without Smoothing	96.54	85.20
With Smoothing	97.10	88.20



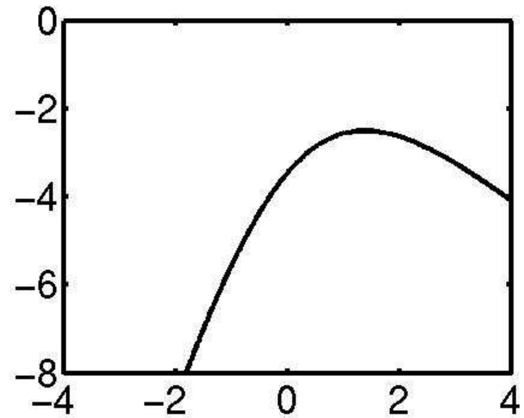
- Smoothing helps:
  - Softens distributions.
  - Pushes weight onto more explanatory features.
  - Allows many features to be dumped safely into the mix.
  - Speeds up convergence (if both are allowed to converge)!

Smoothing: 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 were usually 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.

Part II: Optimization

a. Unconstrained optimization methods

b. Constrained optimization methods

c. Duality of maximum entropy and exponential models

Function Optimization

- To estimate the parameters of a maximum likelihood model, we must find the λ which maximizes:

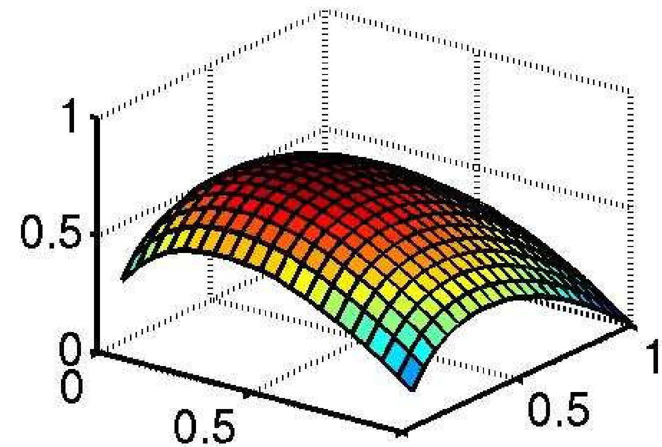
$$\log P(C \mid D, \lambda) = \sum_{(c,d)\in(C,D)} \log \frac{\exp \sum_i \lambda_i f_i(c,d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c',d)}$$

- We'll approach this as a general function optimization problem, though special-purpose methods exist.
- An advantage of the general-purpose approach is that no modification needs to be made to the algorithm to support smoothing by priors.

Notation

- Assume we have a function $f(x)$ from R^n to R.

f



- The gradient $\nabla f(x)$ is the $n\times 1$ vector of partial derivatives $\partial f/\partial x_i$.

$$\nabla f = \begin{bmatrix} \partial f / \partial x_1 \\ \vdots \\ \partial f / \partial x_n \end{bmatrix}$$

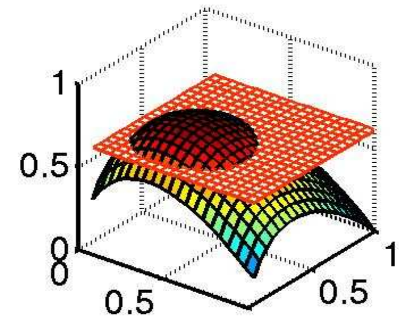
- The Hessian $\nabla^2 f$ is the $n\times n$ matrix of second derivatives $\partial^2 f/\partial x_i \partial x_j$.

$$\nabla^2 f = \begin{bmatrix} \partial^2 f / \partial x_1 \partial x_1 & \cdots & \partial^2 f / \partial x_1 \partial x_n \\ \vdots & \ddots & \vdots \\ \partial^2 f / \partial x_n \partial x_1 & \cdots & \partial^2 f / \partial x_n \partial x_n \end{bmatrix}$$

Taylor Approximations

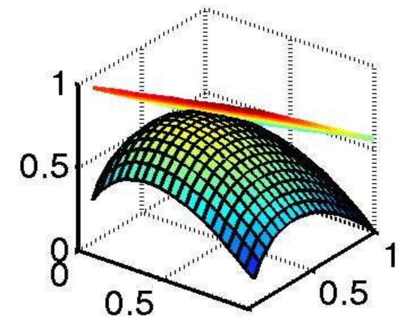
- Constant (zeroth-order):

$$f_{x_0}^{0}(x) = f(x_0)$$



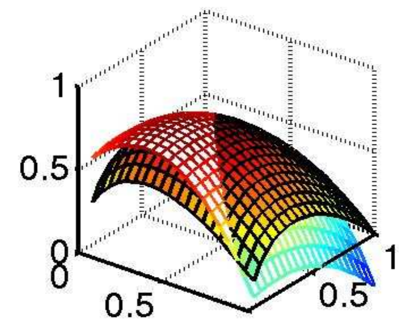
- Linear (first-order):

$$f_{x_0}^{1}(x) = f(x_0) + \nabla f(x_0)^{\mathrm{T}}(x - x_0)$$



- Quadratic (second-order):

$$f_{x_0}^{2}(x) = f(x_0) + \nabla f(x_0)^{\mathrm{T}}(x - x_0) + \frac{1}{2}(x - x_0)^{\mathrm{T}} \nabla^2 f(x_0)(x - x_0)$$



Unconstrained Optimization

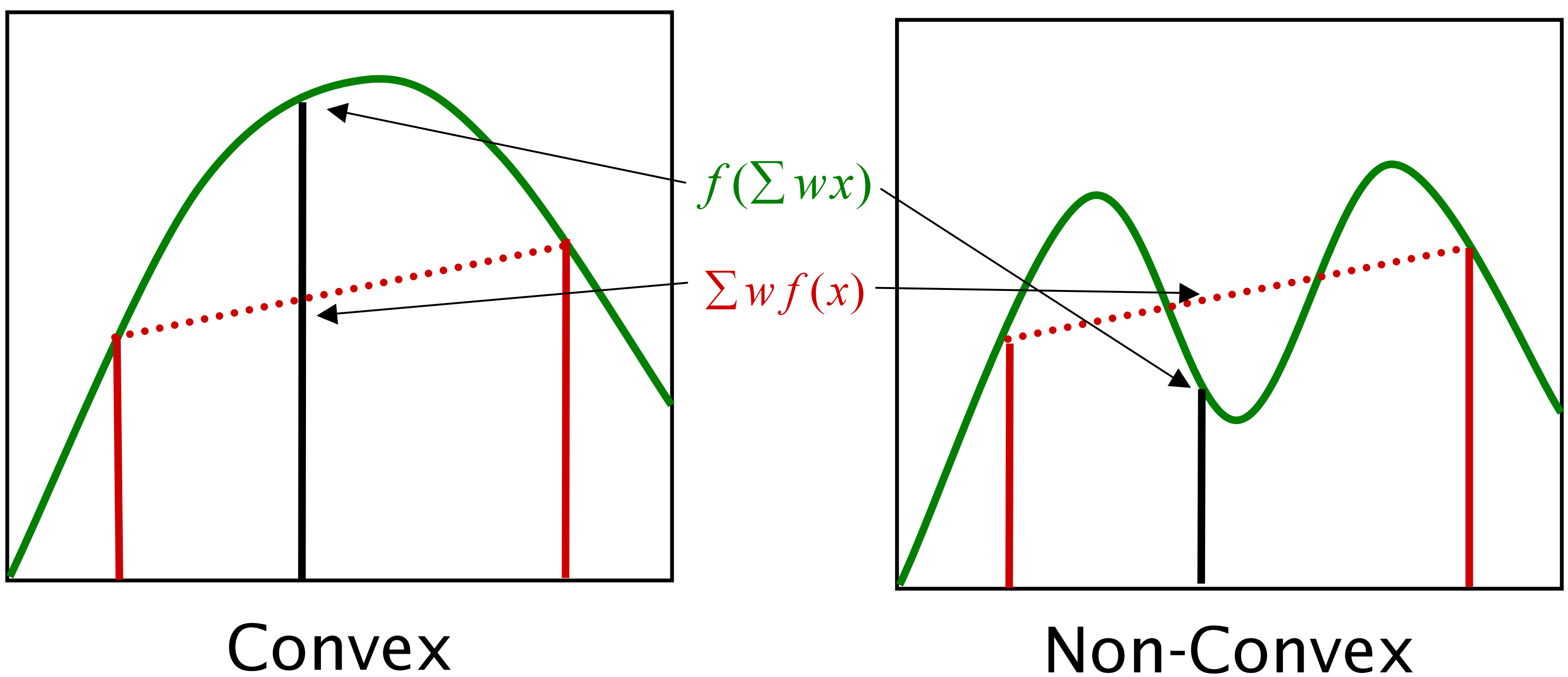
- Problem: $x^* = \arg\max_x f(x)$

- Questions:
  - Is there a unique maximum?
  - How do we find it efficiently?
  - Does f have a special form?

- Our situation:
  - f is convex.
  - f's first derivative vector ∇f is known.
  - f's second derivative matrix $\nabla^2 f$ is not available.

Convexity

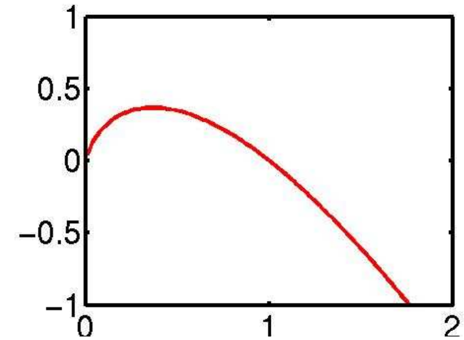
$$f\left(\sum_i w_i x_i\right) \geq \sum_i w_i f(x_i) \qquad \sum_i w_i = 1$$

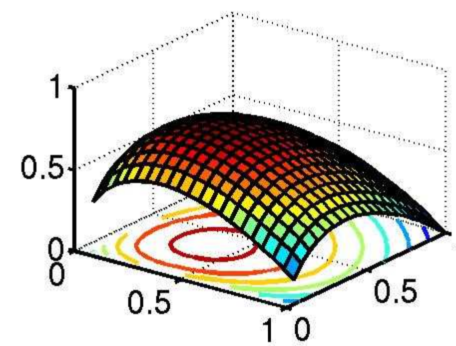


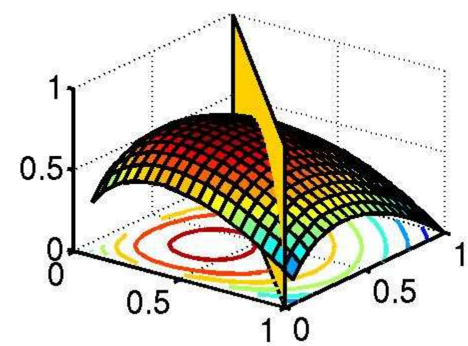
Convexity guarantees a single, global maximum because any higher points are greedily reachable.

Convexity II

- Constrained $H(p) = -\sum x \log x$ is convex:
  - $-x \log x$ is convex
  - $-\sum x \log x$ is convex (sum of convex functions is convex).
  - The feasible region of constrained H is a linear subspace (which is convex)
  - The constrained entropy surface is therefore convex.
- The maximum likelihood exponential model (dual) formulation is also convex.







Optimization Methods

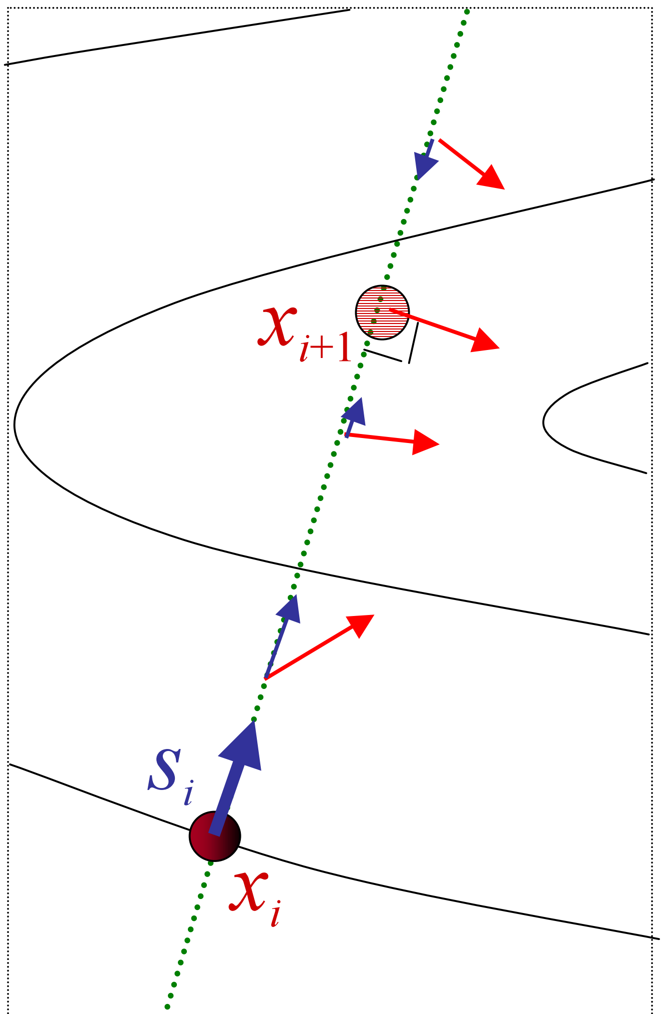
- Iterative Methods:
  - Start at some x_i.
  - Repeatedly find a new x_{i+1} such that $f(x_{i+1}) \geq f(x_i)$.

- Iterative Line Search Methods:
  - Improve x_i by choosing a search direction s_i and setting

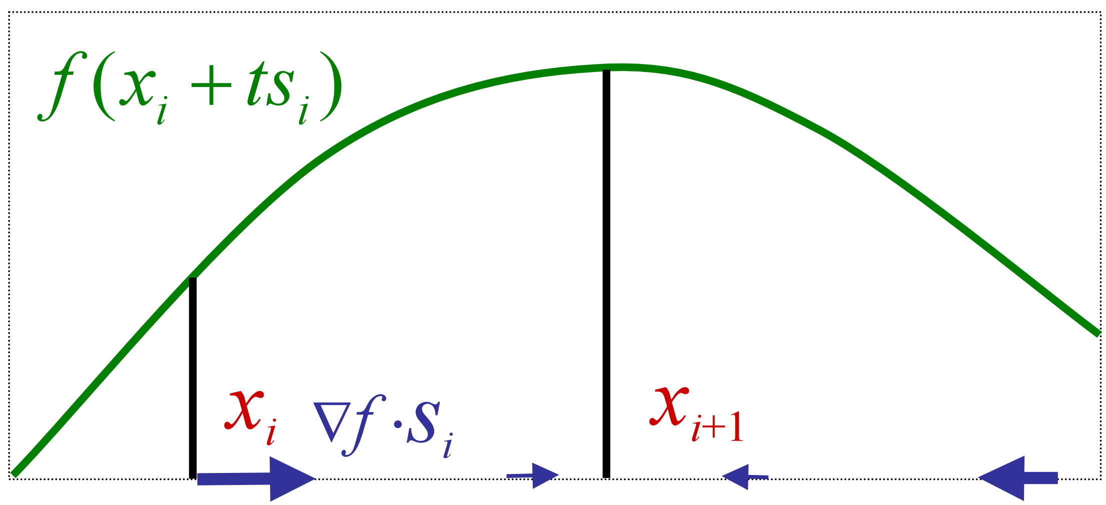
$$x_{i+1} = \arg\max_{x_i + ts_i} f(x_i + ts_i)$$

- Gradient Methods:
  - s_i is a function of the gradient ∇f at x_i.

Line Search I

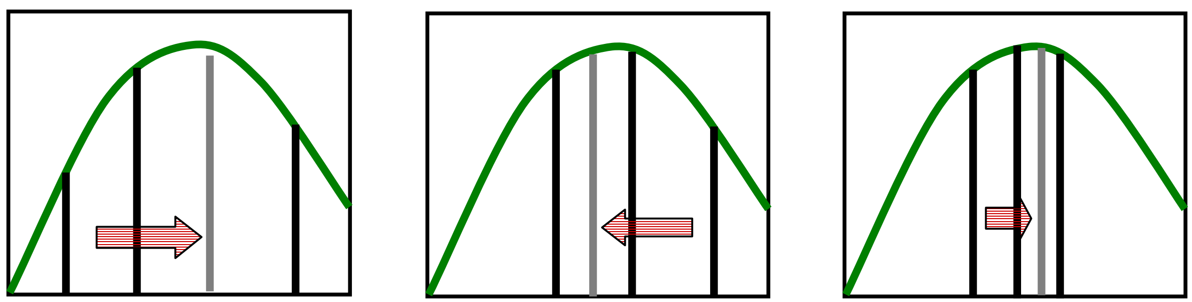


- Choose a start point x_i and a search direction s_i.
- Search along s_i to find the line maximizer: $x_{i+1} = \arg\max_{x_i + ts_i} f(x_i + ts_i)$
- When are we done?

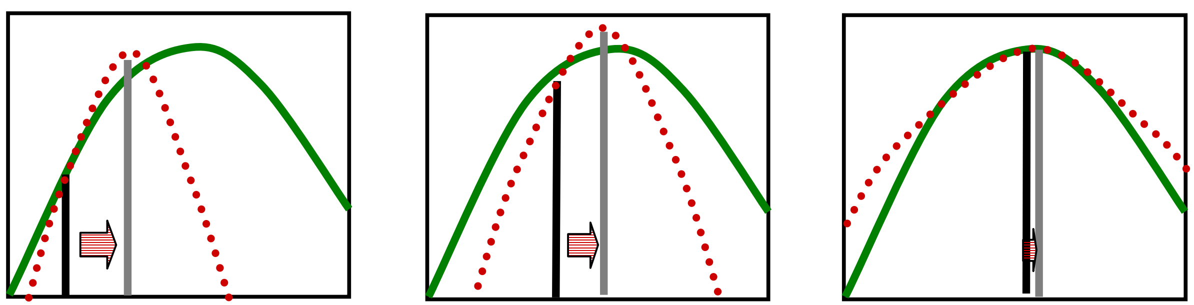


Line Search II

- One dimensional line search is much simpler than multidimensional search.
- Several ways to find the line maximizer:
  - Divisive search: narrowing a window containing the max.



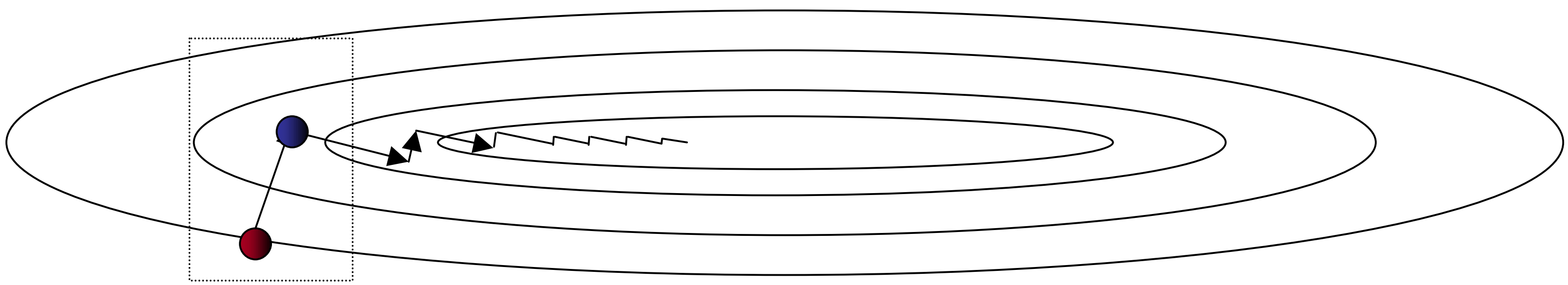
  - Repeated approximation:

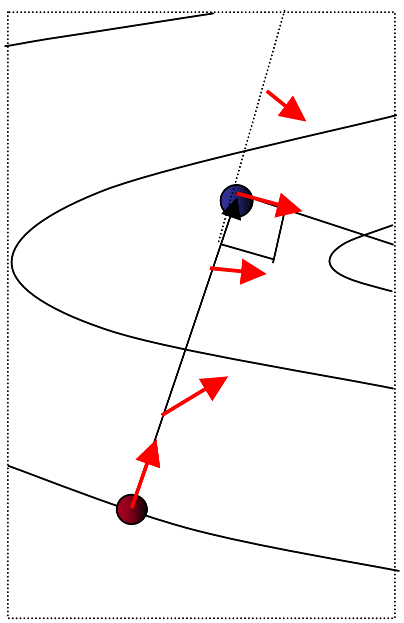


Gradient Ascent I

- Gradient Ascent:
  - Until convergence:
    1. Find the derivative $\nabla f(x)$.
    2. Line search along $\nabla f(x)$.
- Each iteration improves the value of $f(x)$.
- Guaranteed to find a local optimum (in theory could find a saddle point).
- Why would you ever want anything else?
  - Other methods chose better search directions.
  - E.g., $\nabla f(x)$ may be maximally "uphill", but you'd rather be pointed straight at the solution!

Gradient Ascent II





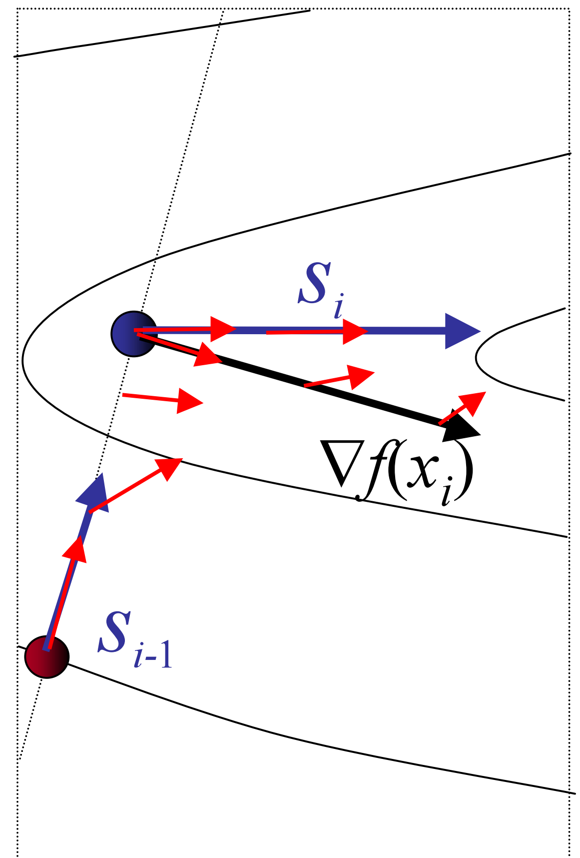
- The gradient is always perpendicular to the level curves.
- Along a line, the maximum occurs when the gradient has no component in the line.
- At that point, the gradient is orthogonal to the search line, so the next direction will be orthogonal to the last.

What Goes Wrong?

- Graphically:
  - Each new gradient is orthogonal to the previous line search, so we'll keep making right-angle turns. It's like being on a city street grid, trying to go along a diagonal – you'll make a lot of turns.
- Mathematically:
  - We've just searched along the old gradient direction $s_{i-1} = \nabla f(x_{i-1})$.
  - The new gradient is $\nabla f(x_i)$ and we know $s_{i-1}{}^{T} \cdot \nabla f(x_i) = \nabla f(x_{i-1})^{T} \cdot \nabla f(x_i) = 0$.
  - As we move along $s_i = \nabla f(x_i)$, the gradient becomes $\nabla f(x_i + ts_i) \approx \nabla f(x_i) + t\nabla^2 f(x_i)\, s_i = \nabla f(x_i) + t\nabla^2 f(x_i) \nabla f(x_i)$.
  - What about that old direction s_{i-1}?
    - $s_{i-1}{}^{T} \cdot (\nabla f(x_{i-1}) + t\nabla^2 f(x_i) \nabla f(x_i)) =$
    - $\nabla f(x_{i-1})^{T} \nabla f(x_i) + t\nabla f(x_{i-1})^{T} \nabla^2 f(x_i) \nabla f(x_i) =$
    - $0 + t\nabla f(x_{i-1})^{T} \nabla^2 f(x_i) \nabla f(x_i)$
    - … so the gradient is regrowing a component in the last direction!

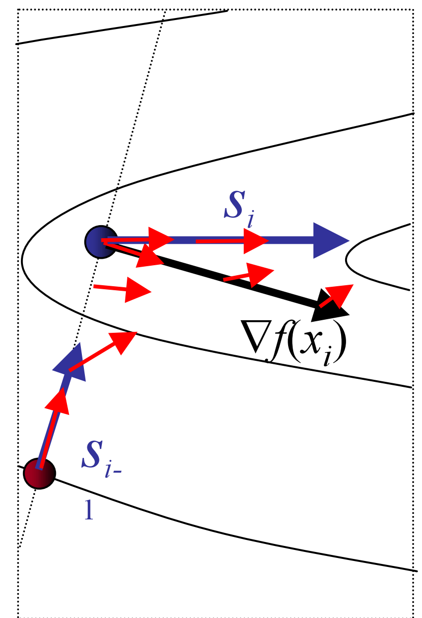
Conjugacy I

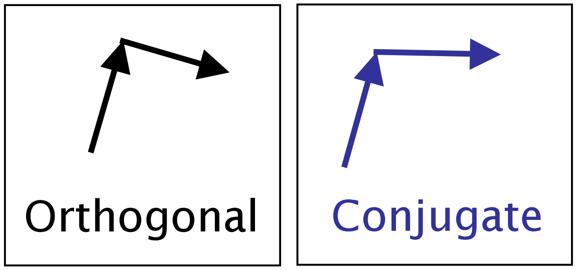
- Problem: with gradient ascent, search along s_i ruined optimization in previous directions.
- Idea: choose s_i to keep the gradient in the previous direction(s) zero.
- If we choose a direction s_i, we want:
  - $\nabla f(x_i + t s_i)$ to stay orthogonal to previous s
  - $s_{i-1}{}^{T} \cdot [\nabla f(x_i + t s_i)] = 0$
  - $s_{i-1}{}^{T} \cdot [\nabla f(x_i) + t \nabla^2 f(x_i) s_i] = 0$
  - $s_{i-1}{}^{T} \cdot \nabla f(x_i) + s_{i-1}{}^{T} \cdot t \nabla^2 f(x_i)\, s_i = 0$
  - $0 + t\, [s_{i-1}{}^{T} \cdot \nabla^2 f(x_i)\, s_i] = 0$
- If $\nabla^2 f(x)$ is constant, then we want: $s_{i-1}{}^{T} \nabla^2 f(x) s_i = 0$



Conjugacy II

- The condition $s_{i-1}{}^{T}\nabla^2 f(x_i) s_i = 0$ almost says that the new direction and the last should be orthogonal – it says that they must be $\nabla^2 f(x_i)$-orthogonal, or conjugate.
- Various ways to operationalize this condition.
- Basic problems:
  - We generally don't know $\nabla^2 f(x_i)$.
  - It wouldn't fit in memory anyway.





Conjugate Gradient Methods

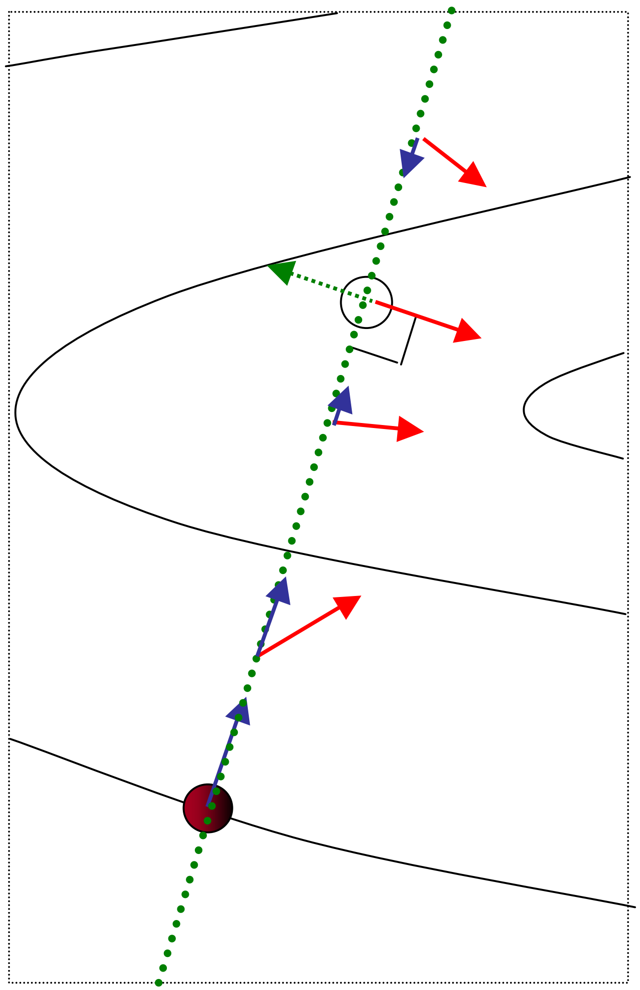
- The general CG method:
  - Until convergence:
    1. Find the derivative $\nabla f(x_i)$.
    2. Remove components of $\nabla f(x_i)$ not conjugate to previous directions.
    3. Line search along the remaining, conjugate projection of $\nabla f(x_i)$.
- The variations are in step 2.
  - If we know $\nabla^2 f(x_i)$ and track all previous search directions, we can implement this directly.
  - If we do not know $\nabla^2 f(x_i)$ – we don't for maxent modeling – and it isn't constant (it's not), there are other (better) ways.
    - Sufficient to ensure conjugacy to the single previous direction.
    - Can do this with the following recurrences [Fletcher-Reeves]:

$$s_i = \nabla f(x_i) + \beta_i s_{i-1} \qquad \beta_i = \frac{\nabla f(x_i)^{\mathrm{T}} \nabla f(x_i)}{\nabla f(x_{i-1})^{\mathrm{T}} \nabla f(x_{i-1})}$$

Constrained Optimization

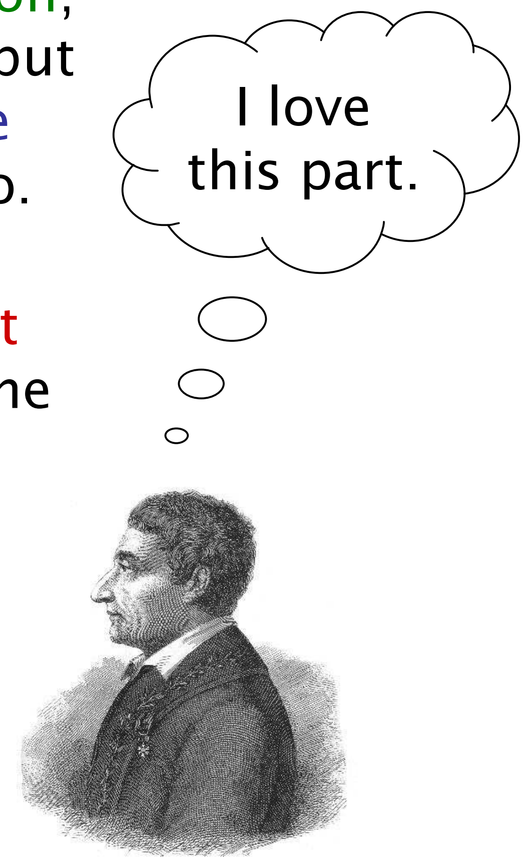
- Goal: $$x^* = \arg\max_x f(x)$$
  subject to the constraints:
  $$\forall i : g_i(x) = 0$$
- Problems:
  - Have to ensure we satisfy the constraints.
  - No guarantee that $\nabla f(x^*) = 0$, so how to recognize the max?
- Solution: the method of Lagrange Multipliers

Lagrange Multipliers I



- At a global max, $\nabla f(x^*) = 0$.
- Inside a constraint region, $\nabla f(x^*)$ can be non-zero, but its projection inside the constraint must be zero.
- In two dimensions, this means that the gradient must be a multiple of the constraint normal:

$$\nabla f(x) = \lambda \nabla g(x)$$



Lagrange Multipliers II

- In multiple dimensions, with multiple constraints, the gradient must be in the span of the surface normals:

$$\nabla f(x) = \sum_i \lambda_i \nabla g_i(x)$$

- Also, we still have constraints on :

$$\forall i : g_i(x) = 0$$

- We can capture both requirements by looking for critical points of the Lagrangian:

$$\Lambda(x, \lambda) = f(x) - \sum_i \lambda_i g_i(x)$$

$\partial\Lambda/\partial x = 0$ recovers the gradient-in-span property.

$\partial\Lambda/\partial\lambda_i = 0$ recovers constraint i.

The Lagrangian as an Encoding

- The Lagrangian:

$$\Lambda(x,\lambda) = f(x) - \sum_i \lambda_i g_i(x)$$

- Zeroing the x_j derivative recovers the jth component of the gradient span condition:

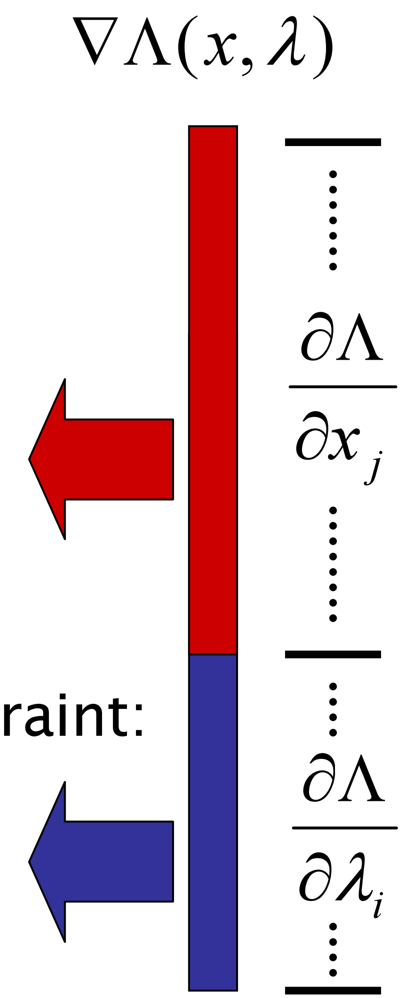
$$\frac{\partial \Lambda(x,\lambda)}{\partial x_j} = \frac{\partial f(x)}{\partial x_j} - \sum_i \lambda_i \frac{\partial g_i(x)}{\partial x_j}$$

$$0 = \nabla f(x) - \sum_i \lambda_i \nabla g_i(x)$$

- Zeroing the λ_i derivative recovers the ith constraint:

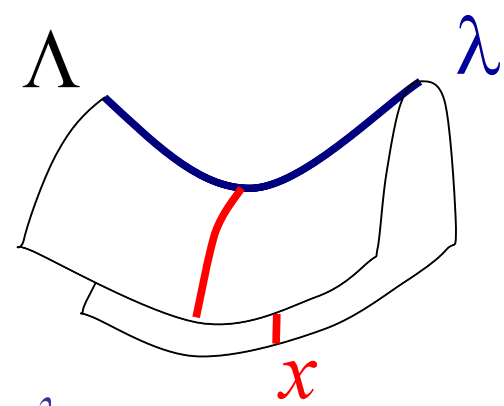
$$\frac{\partial \Lambda(x,\lambda)}{\partial \lambda_i} = 0 - g_i(x)$$

$$0 = g_i(x)$$



A Duality Theorem

- Constrained maxima x^* occur at critical points (x^*, λ^*) of Λ where:
  1. x^* is a local maximum of $\Lambda(x, \lambda^*)$
  2. λ^* is a local minimum of $\Lambda(x^*, \lambda)$



- Proof bits:
  - At a constrained maximum x^*:
    - All constraints i must be satisfied at x^*.
    - The gradient span condition holds at x^* for some λ.
  - [Local max in x] If we change x^*, slightly, while staying in the constraint region, $f(x)$ must drop. However, each $g_i(x)$ will stay zero, so $\Lambda(x, \lambda)$ will drop.
  - [Local min in λ] If we change λ^*, slightly, then find the x which maximizes Λ, the max Λ can only be greater than the old one, because at x^* Λ's value is independent of λ, so we can still get it.

Direct Constrained Optimization

- Many methods for constrained optimization are outgrowths of Lagrange multiplier ideas.

- Iterative Penalty Methods
  - Can add an increasing penalty to the objective for violating constraints:

$$f_{PENALIZED}(x,k) = f(x) - \sum_i k\, g_i(x)^2 / 2$$

  - This works by itself (though not well) as you increase k.
    - For any k, an unconstrained optimization will balances the penalty against gains in function value
    - k may have to be huge to get constraint violations small.

Direct Constrained Optimization

- Better method: shift the force exerted by the penalty onto Lagrange multipliers:

$$\Lambda_{PENALIZED}(x, \lambda, k) = f(x) - \sum_i \lambda_i g_i(x) - \sum_i k\, g_i(x)^2 / 2$$

- Fix $\lambda=0$ and $k=k_0$.
- Each round:
  - $x = \arg\max \Lambda(x, \lambda^*, k)$
  - $k = \alpha k$
  - $\lambda_i = \lambda_i + k\, g_i(x)$

Max over the penalized surface.

Penalty cost grows each round.

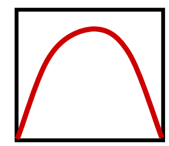
Lagrange multipliers take over the force that the penalty function exerted in the current round.

- This finds both the optimum x^* and λ^* at the same time!

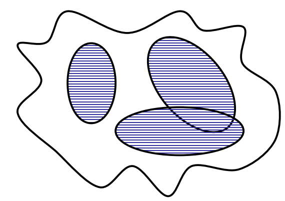
Maximum Entropy

- Recall our example of constrained optimization:

maximize $$H(p) = -\sum_x p_x \log p_x$$



subject to $$\forall i : \sum_{x \in f_i} p_x = C_{f_i}$$



- We can build its Lagrangian:

$$\Lambda(p, \lambda) = -\sum_x p_x \log p_x - \sum_i \lambda_i \left[C_{f_i} - \sum_x p_x f_i(x) \right]$$

- We *could* optimize this directly to get our maxent model.

Lagrangian: Max-and-Min

- Can think of constrained optimization as:

$$\max_x \min_\lambda \Lambda(x,\lambda) = f(x) - \sum_i \lambda_i g_i(x)$$

- Penalty methods work somewhat in this way:
  - Stay in the constrained region, or your function value gets clobbered by penalties.
- Duality lets you reverse the ordering:

$$\min_\lambda \max_x \Lambda(x,\lambda) = f(x) - \sum_i \lambda_i g_i(x)$$

- Dual methods work in this way:
  - Solve the maximization for a given set of λs.
  - Of these solutions, minimize over the space of λs.

The Dual Problem

- For fixed λ, we know that Λ has a maximum where:

$$\frac{\partial \Lambda(p,\lambda)}{\partial p_x} = \frac{-\partial \sum_x p_x \log p_x}{\partial p_x} + \frac{-\partial \sum_i \lambda_i \left[C_i - \sum_x p_x f_i(x) \right]}{\partial p_x} = 0$$

- … and:

$$\frac{\partial \sum_x p_x \log p_x}{\partial p_x} = 1 + \log p_x \qquad \frac{\partial \sum_i \lambda_i \left[C_i - \sum_x p_x f_i(x) \right]}{\partial p_x} = -\sum_i \lambda_i f_i(x)$$

- … so we know:

$$1 + \log p_x = \sum_i \lambda_i f_i(x)$$

$$p_x \propto \exp \sum_i \lambda_i f_i(x)$$

The Dual Problem

- We know the maximum entropy distribution has the exponential form:

$$p_x(\lambda) \propto \exp \sum_i \lambda_i f_i(x)$$

- By the duality theorem, we want to find the multipliers λ that *minimize* the Lagrangian:

$$\Lambda(p,\lambda) = -\sum_x p_x \log p_x - \sum_i \lambda_i \left[C_{f_i} - \sum_x p_x f_i(x) \right]$$

- The Lagrangian is the negative data log-likelihood (next slides), so this is the same as finding the λ which maximize the data likelihood – our original problem in part I.

The Dual Problem

$$\Lambda(p,\lambda) = -\sum_x p_x \log p_x - \sum_i \lambda_i \left[C_{f_i} - \sum_x p_x f_i(x) \right]$$

$$= -\sum_x p_x \log \frac{\exp \sum_i \lambda_i f_i(x)}{\sum_{x'} \exp \sum_i \lambda_i f_i(x')} - \sum_i \lambda_i \left[C_{f_i} - \sum_x p_x f_i(x) \right]$$

$$= \left[-\sum_x p_x \sum_i \lambda_i f_i(x) \right] + \left[\log \sum_{x'} \exp \sum_i \lambda_i f_i(x') \right]$$

$$- \sum_i \lambda_i C_{f_i} + \left[\sum_x p_x \sum_i \lambda_i f_i(x) \right]$$

The Dual Problem

$$\Lambda(p,\lambda) = \left[\log \sum_x \exp \sum_i \lambda_i f_i(x)\right] - \sum_i \lambda_i C_{f_i}$$

$$C_{f_i} = \sum_x \hat{p}_x f_i(x)$$

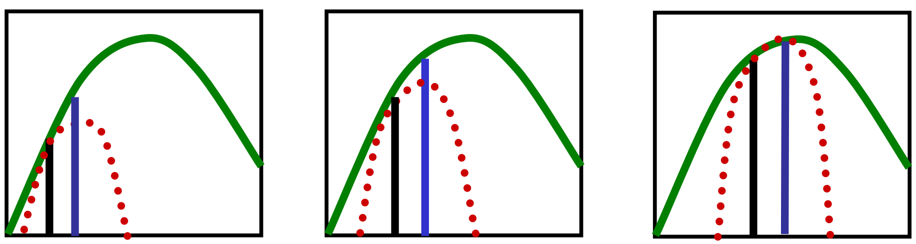
$$= \left[\log \sum_x \exp \sum_i \lambda_i f_i(x)\right] - \sum_x \sum_i \hat{p}_x \lambda_i f_i(x)$$

$$= \left[\log \sum_x \exp \sum_i \lambda_i f_i(x)\right] - \sum_x \hat{p}_x \log \exp \sum_i \lambda_i f_i(x)$$

$$= -\sum_x \hat{p}_x \log \left[\frac{\exp \sum_i \lambda_i f_i(x)}{\sum_x \exp \sum_i \lambda_i f_i(x)}\right] = -\sum_x \hat{p}_x \log p_x$$

Iterative Scaling Methods

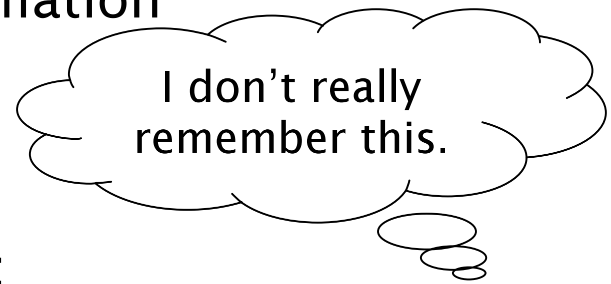
- Iterative Scaling methods are an alternative optimization method. (Darroch and Ratcliff, 72)
- Specialized to the problem of finding maxent models.
- They are iterative lower bounding methods [so is EM]:
  - Construct a lower bound to the function.
  - Optimize the bound.

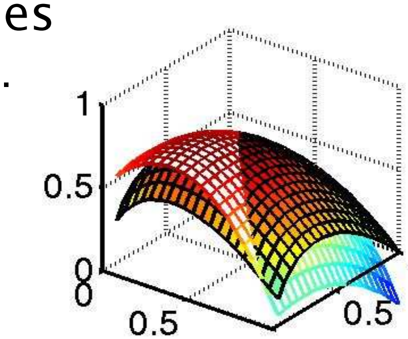


  - Problem: lower bound can be loose!
- People have worked on many variants, but these algorithms are neither simpler to understand, nor empirically more efficient.

Newton Methods

- Newton Methods are also iterative approximation algorithms.
  - Construct a quadratic approximation.
  - Maximize the approximation.
- Various ways of doing each approximation:
  - The pure Newton method constructs the tangent quadratic surface at x, using $\nabla f(x)$ and $\nabla^2 f(x)$.
  - This involves inverting the $\nabla^2 f(x)$, (slow). Quasi-Newton methods use simpler approximations to $\nabla^2 f(x)$.
  - If the number of dimensions (number of features) is large, $\nabla^2 f(x)$ is too large to store; limited-memory quasi-Newton methods use the last few gradient values to implicitly approximate $\nabla^2 f(x)$ (CG is a special case).
- Limited-memory quasi-Newton methods like in (Nocedal 1997) are possibly the most efficient way to train maxent models (Malouf 2002).

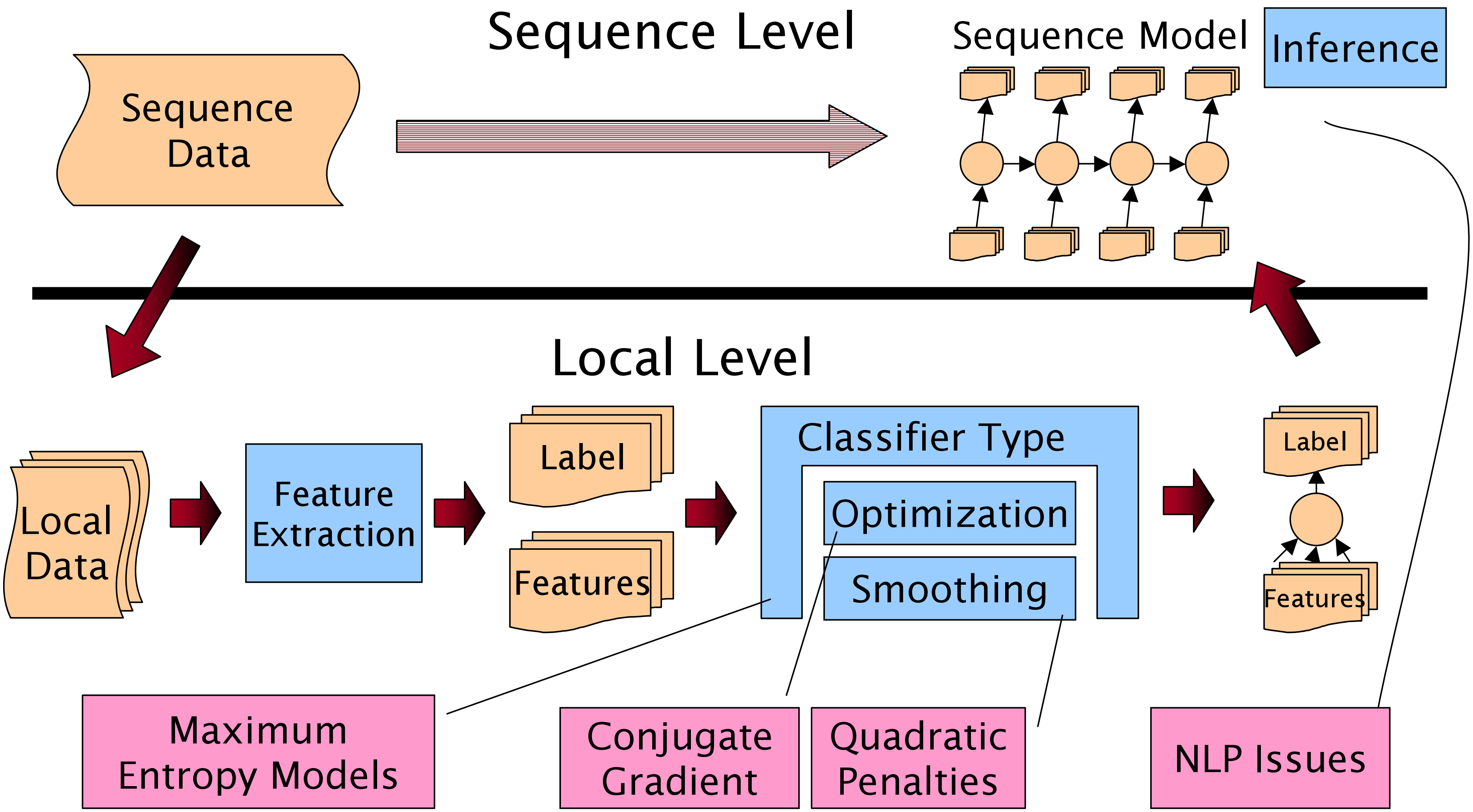




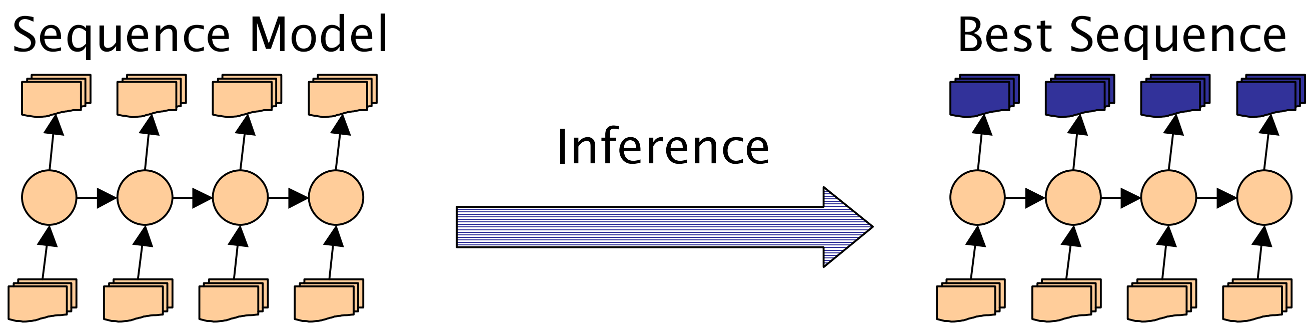
Part III: NLP Issues

- Sequence Inference
- Model Structure and Independence Assumptions
- Biases of Conditional Models

Inference in Systems

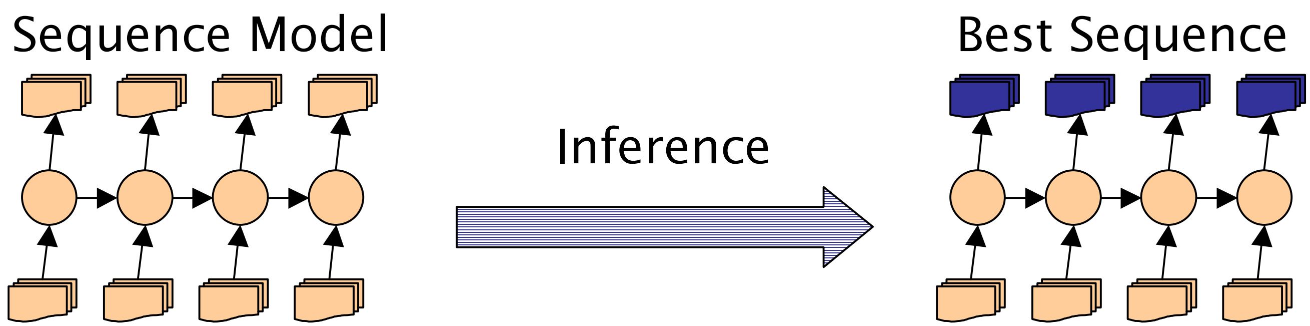


Beam Inference



- 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; 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.

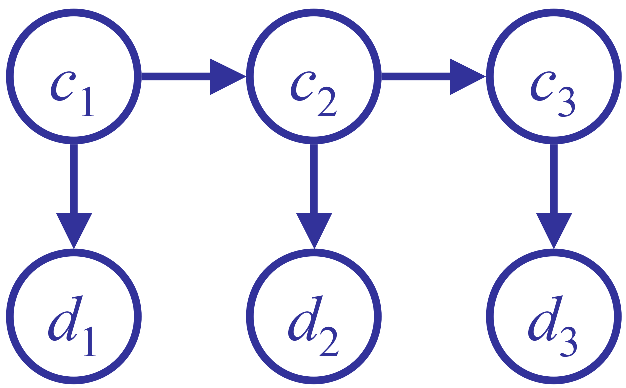
Viterbi Inference



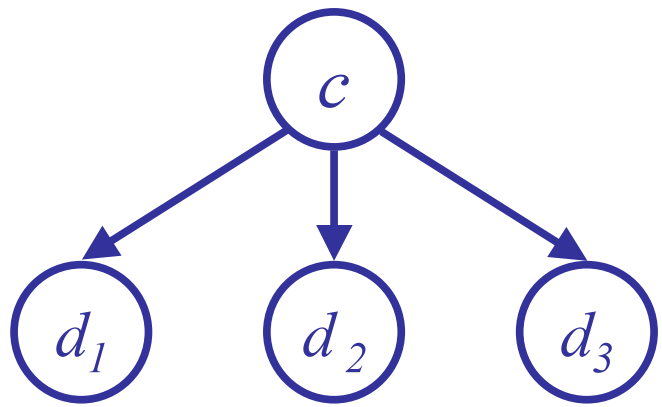
- Viterbi inference:
  - Dynamic programming or memoization.
  - Requires small window of state influence (e.g., past two states are relevant).
- Advantage:
  - Exact: the global best sequence is returned.
- Disadvantage:
  - Harder to implement long-distance state-state interactions (but beam inference tends not to allow long-distance resurrection of sequences anyway).

Independence Assumptions

- Graphical models describe the conditional independence assumptions implicit in models.



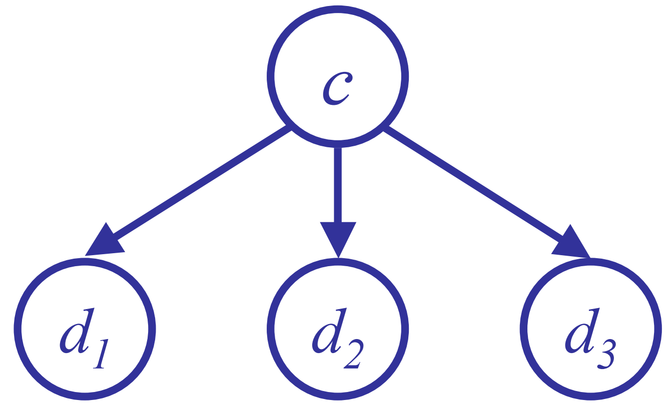
HMM



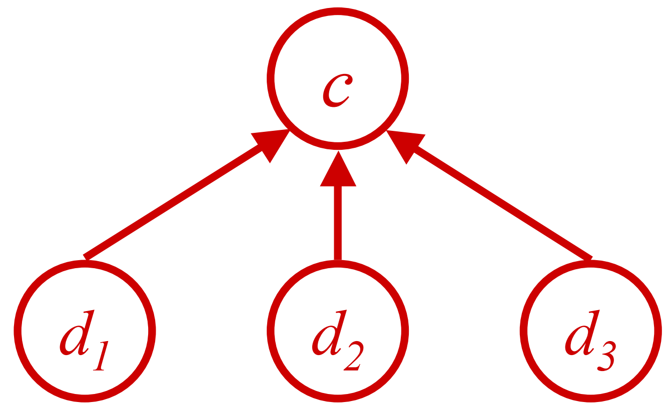
Naïve-Bayes

Causes and Effects

- Effects
  - Children (the w_i here) are effects in the model.
  - When two arrows exit a node, the children are (independent) effects.

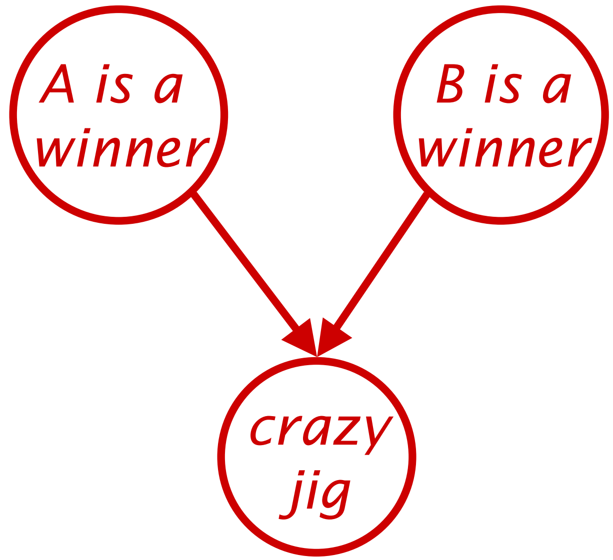


- Causes
  - Parents (the w_i here) are causes in the model.
  - When two arrows enter a node (a v-structure), the parents are in causal competition.



Explaining-Away

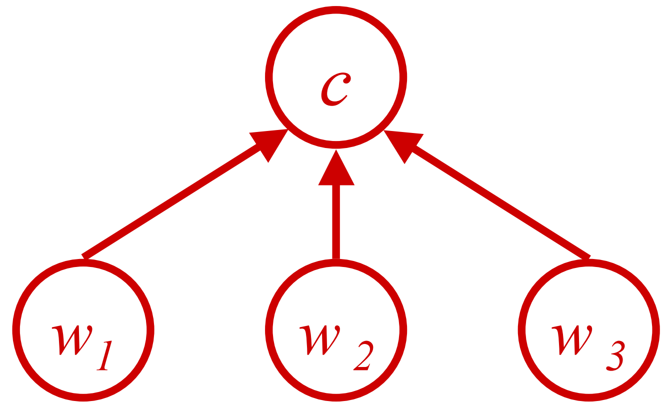
- When nodes are in causal competition, a common interaction is explaining-away.
- In explaining-away, discovering one cause leads to a lowered belief in other causes.



Example: I buy lottery tickets A and B. You assume neither is a winner. I then do a crazy jig. You then believe one of my two lottery tickets must be a winner, 50%-50%. If you then find that ticket A did indeed win, you go back to believing that B is probably not a winner.

Data and Causal Competition

- Problem in NLP in general:
  - Some singleton words are noise.
  - Others are your only only glimpse of a good feature.



- Maxent models have an interesting, *potentially* NLP-friendly behavior.
  - Optimization goal: assign the correct class.
  - Process: assigns more weight ("blame") to features which are needed to get classifications right.
  - Maxent models effectively have the structure shown, putting features into causal competition.

Example WSD Behavior I

- line$_2$ (a phone line)

  A) "thanks anyway, the transatlantic line$_2$ died."

  B) "… phones with more than one line$_2$, plush robes, exotic flowers, and complimentary wine."

- In A, "died" occurs with line$_2$ 2/3 times.
- In B, "phone(s)" occurs with line$_2$ 191/193 times.
- "transatlantic" and "flowers" are both singletons in data

- We'd like "transatlantic" to indicate line$_2$ more than "flowers" does...

Example WSD Behavior II

- Both models use "add one" pseudocount smoothing
- With Naïve-Bayes:

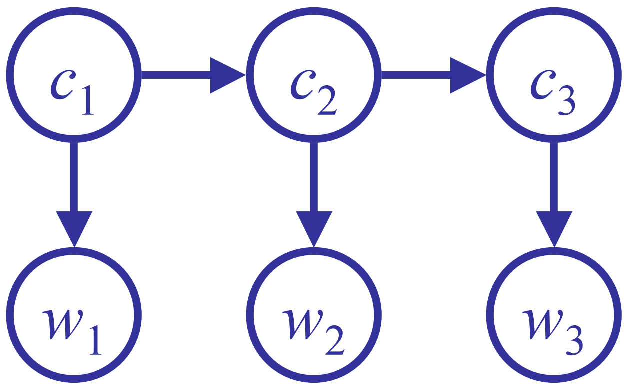
$$\frac{P_{NB}(flowers \mid 2)}{P_{NB}(flowers \mid 1)} = 2 \qquad \frac{P_{NB}(transatlantic \mid 2)}{P_{NB}(transatlantic \mid 1)} = 2$$

- With a word-featured maxent model:

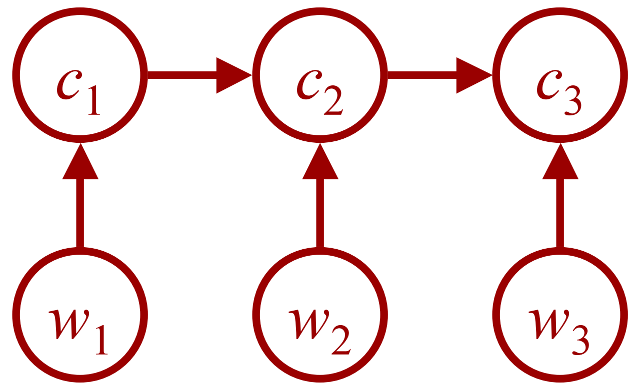
$$\frac{P_{ME}(flowers \mid 2)}{P_{ME}(flowers \mid 1)} = 2.05 \quad \frac{P_{ME}(transatlantic \mid 2)}{P_{ME}(transatlantic \mid 1)} = 3.74$$

- Of course, "thanks" is just like "transatlantic"!

Markov Models for POS Tagging



Joint HMM



Conditional CMM

- Need $\mathrm{P}(c|c_{-1})$, $\mathrm{P}(w|c)$
- Advantage: easy to train.
- Could be used for language modeling.

- Need $\mathrm{P}(c|w,c_{-1})$, $\mathrm{P}(w)$
- Advantage: easy to include features.
- Typically split $\mathrm{P}(c|w,c_{-1})$

WSJ Results

- Tagging WSJ sentences, using only previous-tag and current-word features.

Penn Treebank WSJ, Test Set	
HMM	CMM
91.2	89.2

- Very similar experiment to (Lafferty et al. 2001)

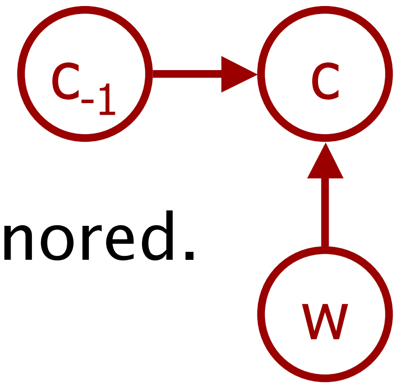
- Details:
  - Words occurring less than 5 times marked UNK
  - No other smoothing.

Label Bias

- Why does the conditional CMM underperform the joint model, given the same features?
- Idea: label bias (Bottou 1991)
  - Classes with low exit entropy will be preferred.
  - "Mass preservation" – if a class has only one exit, that exit is taken with conditional probability 1, regardless of the next observation.
- Example:
  - If we tag a word as a pre-determiner (PDT), then the next word will almost surely be a determiner (DT).
  - Previous class determines current class regardless of word

States and Causal Competition

- In the conditional model shown, C_{-1} and W are competing causes for C.



- Label bias is explaining-away.
  - The C_{-1} explains C so well that W is ignored.
- The reverse explaining-away effect:
  - "Observation bias"
  - The W explains C so well that C_{-1} is ignored.
- We can check experimentally for these effects.

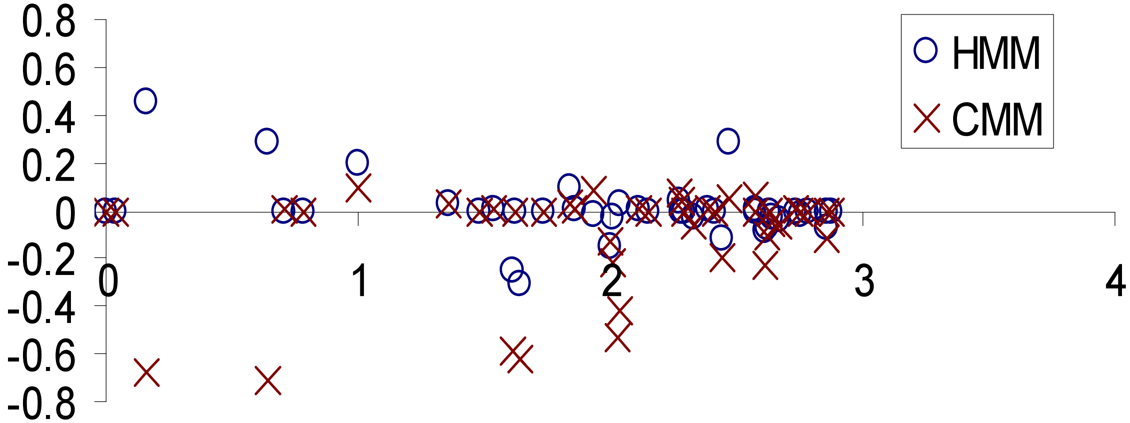
Example: Observation Bias

						Log Probability	
						HMM	CMM
Correct Tags	PDT	DT	NNS	VBD	.	-0.0	-1.3
Incorrect Tags	DT	DT	NNS	VBD	.	-5.4	-0.3
Words	All	the	indexes	dove	.		

- “All” is usually a DT, not a PDT.
- “the” is virtually always a DT.
- The CMM is happy with the (rare) DT-DT sequence, because having “the” explains the second DT.

Label Bias?

- Label exit entropy vs. overproposal rate:



  … if anything, low-entropy states are dispreferred by the CMM.

- Label bias might well arise in models with more features, or observation bias might not.
  - Top-performing maxent taggers have next-word features that can mitigate observation bias.

CRFs

- Another sequence model: Conditional Random Fields (CRFs) of (Lafferty et al. 2001).
- A whole-sequence conditional model rather than a chaining of local models.

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

- The space of c's is now the space of sequences, and hence must be summed over using dynamic programming.
- Training is slow, but CRFs avoid causal-competition biases.

Model Biases

- Causal competition between hidden variables seems to generally be harmful for NLP.
  - Classes vs. observations in tagging.
  - Empty input forcing reductions in shift-reduce parsing.

- Maxent models can and do have these issues, but…
  - The model with the better features usually wins.
  - Maxent models are easy to stuff huge numbers of non-independent features into.
  - These effects seem to be less troublesome when you include lots of conditioning context
  - Can avoid these biases with global models, but the efficiency cost can be huge.

Part IV: Resources

- Our Software
- Other Software Resources
- References

Classifier Package

- Our Java software package:
  - Classifier interface
  - General linear classifiers
    - Maxent classifier factory
    - Naïve-Bayes classifier factory
  - Optimization
    - Unconstrained CG Minimizer
    - Constrained Penalty Minimizer
- Available at:
  - http://nlp.stanford.edu/downloads/classifier.shtml

↑ **NB!**

Other software sources

- http://maxent.sourceforge.net/
  - Jason Baldridge et al. Java maxent model library. GIS.
- http://www-rohan.sdsu.edu/~malouf/pubs.html
  - Rob Malouf. Frontend maxent package that uses PETSc library for optimization. GIS, IIS, gradient ascent, CG, limited memory variable metric quasi-Newton technique.
- http://search.cpan.org/author/TERDOEST/
  - Hugo WL ter Doest. Perl 5. GIS, IIS.

Other software non-sources

- http://www.cis.upenn.edu/~adwait/statnlp.html
  - Adwait Ratnaparkhi. Java bytecode for maxent POS tagger and sentence boundary finder. GIS.
- http://www.cs.princeton.edu/~ristad/
  - Eric Ristad once upon a time distributed a maxent toolkit to accompany his ACL/EACL 1997 tutorial, but that was many moons ago. GIS.
- http://www.cs.umass.edu/~mccallum/mallet/
  - Andrew McCallum announced a package at NIPS 2002 that includes a maxent classifier also using a limited memory quasi-Newton optimization technique. But delivery seems to have been "delayed".

References: Optimization/Maxent

Adam Berger, Stephen Della Pietra, and Vincent Della Pietra. 1996. “A maximum entropy approach to natural language processing.” Computational Linguistics. 22.

J. Darroch and D. Ratcliff. 1972. “Generalized iterative scaling for log-linear models.” Ann. Math. Statistics, 43:1470-1480.

John Lafferty, Fernando Pereira, and Andrew McCallum. 2001. “Conditional random fields: Probabilistic models for segmenting and labeling sequence data.” In Proceedings of the International Conference on Machine Learning (ICML-2001).

Robert Malouf. 2002. "A comparison of algorithms for maximum entropy parameter estimation." In Proceedings of the Sixth Conference on Natural Language Learning (CoNLL-2002). Pages 49-55.

Thomas P. Minka. 2001. Algorithms for maximum-likelihood logistic regression. Statistics Tech Report 758, CMU.

Jorge Nocedal. 1997. “Large-scale unconstrained optimization.” In A. Watson and I. Duff, eds., The State of the Art in Numerical Analysis, pp 311-338. Oxford University Press.

References: Regularization

Stanley Chen and Ronald Rosenfeld. A Survey of Smoothing Techniques for ME Models. *IEEE Transactions on Speech and Audio Processing*, 8(1), pp. 37--50. January 2000.

M. Johnson, S. Geman, S. Canon, Z. Chi and S. Riezler. 1999. Estimators for Stochastic "Unification-based" Grammars. *Proceedings of ACL 1999*.

References: Named Entity Recognition

Andrew Borthwick. 1999. A Maximum Entropy Approach to Named Entity Recognition. Ph.D. Thesis. New York University.

Dan Klein, Joseph Smarr, Huy Nguyen, and Christopher D. Manning. 2003. Named Entity Recognition with Character-Level Models. *Proceedings the Seventh Conference on Natural Language Learning (CoNLL 2003).*

References: POS Tagging

James R. Curran and Stephen Clark (2003). Investigating GIS and Smoothing for Maximum Entropy Taggers. *Proceedings of the 11th Annual Meeting of the European Chapter of the Association for Computational Linguistics (EACL'03),* pp.91-98, Budapest, Hungary

Adwait Ratnaparkhi. A Maximum Entropy Part-Of-Speech Tagger. In *Proceedings of the Empirical Methods in Natural Language Processing Conference*, May 17-18, 1996. University of Pennsylvania

Kristina Toutanova and Christopher D. Manning. 2000. Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000),* pp. 63-70. Hong Kong.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. HLT-NAACL 2003.

References: Other Applications

Tong Zhang and Frank J. Oles. 2001. Text Categorization Based on Regularized Linear Classification Methods. *Information Retrieval* 4: 5–31.

Ronald Rosenfeld. A Maximum Entropy Approach to Adaptive Statistical Language Modeling. *Computer, Speech and Language* 10, 187--228, 1996.

Adwait Ratnaparkhi. A Linear Observed Time Statistical Parser Based on Maximum Entropy Models. In Proceedings of the Second Conference on Empirical Methods in Natural Language Processing. Aug. 1-2, 1997. Brown University, Providence, Rhode Island.

Adwait Ratnaparkhi. Unsupervised Statistical Models for Prepositional Phrase Attachment. In Proceedings of the Seventeenth International Conference on Computational Linguistics, Aug. 10-14, 1998. Montreal.

Andrei Mikheev. 2000. Tagging Sentence Boundaries. *NAACL 2000,* pp. 264-271.

References: Linguistic Issues

Léon Bottou. 1991. Une approche theorique de l'apprentissage connexioniste; applications a la reconnaissance de la parole. Ph.D. thesis, Université de Paris XI.

Mark Johnson. 2001. Joint and conditional estimation of tagging and parsing models. In ACL 39, pages 314–321.

Dan Klein and Christopher D. Manning. 2002. Conditional Structure versus Conditional Estimation in NLP Models. 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002), pp. 9-16.

Andrew McCallum, Dayne Freitag and Fernando Pereira. 2000. Maximum Entropy Markov Models for Information Extraction and Segmentation. *ICML.*

Riezler, S., T. King, R. Kaplan, R. Crouch, J. Maxwell and M. Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and Discriminative Estimation Techniques. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics.*