Discriminative Estimation (Maxent models and perceptron)

Generative vs. Discriminative models

Many slides are adapted from slides by Christopher Manning and perceptron slides by Alan Ritter

Introduction

- So far we've looked at "generative 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

$P(c,d)$

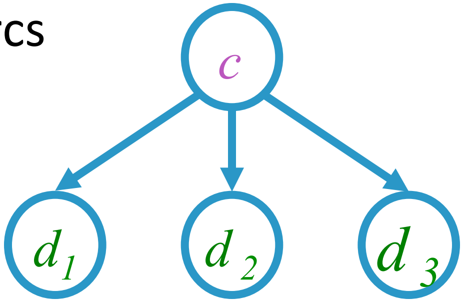
Joint vs. Conditional Models

- 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)

$P(c|d)$

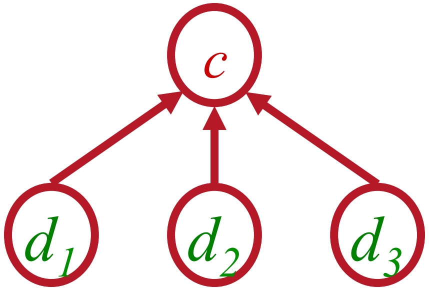
Bayes Net/Graphical Models

- 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

Generative



Logistic Regression

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)

Maxent Models and Discriminative Estimation

Generative vs. Discriminative models

Discriminative Model Features

Making features from text for discriminative NLP models

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: $f\colon C \times D \rightarrow \mathbb{R}$

A Belief: to create a data partition

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

Example features

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

LOCATION *in Arcadia* LOCATION *in Québec* DRUG *taking Zantac* PERSON *saw Sue*

- 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-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 {…, w_{-1}=restructure, w_{+1}=debt, …}	Label: NN Features {w=fall, t_{-1}=JJ w_{-1}=previous}

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% F_1
  - 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.)

Discriminative Model Features

Making features from text for discriminative NLP models

Feature-based Linear Classifiers

How to put features into a classifier

Feature-Based Linear Classifiers

- 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:
    - $\text{vote}(c) = \Sigma\lambda_i f_i(c,d)$

PERSON
in Québec

LOCATION
in Québec

DRUG
in Québec

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

Example features

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

LOCATION *in Arcadia* LOCATION *in Québec* DRUG *taking Zantac* PERSON *saw Sue*

- 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-Based Linear Classifiers

- 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:
    - vote(c) $= \Sigma\lambda_i f_i(c,d)$

PERSON *in Québec*

1.8 LOCATION *in Québec* –0.6

0.3 DRUG *in Québec*

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

Feature-Based Linear Classifiers

There are many ways to chose weights for features

With different loss functions as the optimization goal

- 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 $\Sigma\lambda_i f_i(c,d)$

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

Makes votes positive

Normalizes votes

- P(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)
    - 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

Quiz 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 $f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$
- -0.6 $f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$
- 0.3 $f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$

PERSON *by Goéric*

LOCATION *by Goéric*

DRUG *by Goéric*

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

Feature-based Linear Classifiers

How to put features into a classifier

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 (index)
  - 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
  - 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).

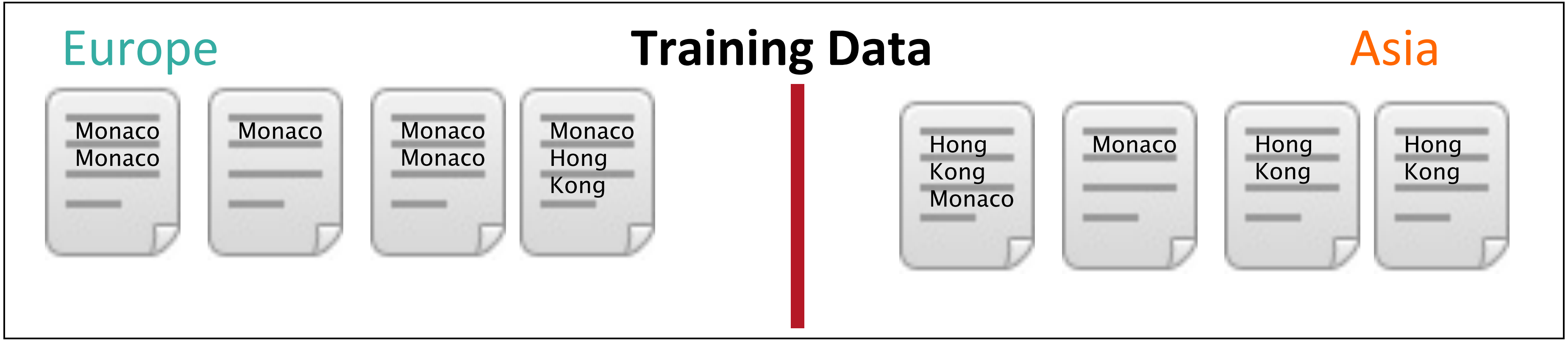
Building a Maxent Model

The nuts and bolts

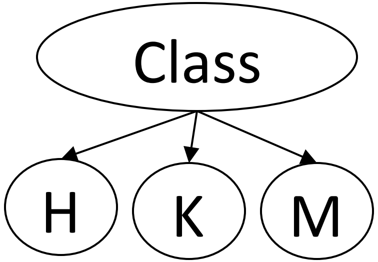
Naive Bayes vs. Maxent models

Generative vs. Discriminative models: The problem of overcounting evidence

Text classification: Asia or Europe



NB Model



NB FACTORS:

- P(A) = P(E) =
- P(M|A) =
- P(M|E) =
- P(H|A) = P(K|A) =
- P(H|E) = PK|E) =

PREDICTIONS:

- P(A,H,K,M) =
- P(E,H,K,M) =
- P(A|H,K,M) =
- P(E|H,K,M) =

Naive Bayes vs. Maxent Models

- Naive Bayes models multi-count correlated evidence
  - Each feature is multiplied in, even when you have multiple features telling you the same thing
- Maximum Entropy models (pretty much) solve this problem
  - As we will see, this is done by weighting features so that model expectations match the observed (empirical) expectations

Naive Bayes vs. Maxent models

Generative vs. Discriminative models: The problem of overcounting evidence

Maxent Models and Discriminative Estimation

Maximizing the likelihood

Feature Expectations

- We will crucially make use of two *expectations*
  - actual or predicted counts of a feature firing:

- Empirical count (expectation) of a feature:

Goal: well fit the data

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

- Model expectation of a feature:

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

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:

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

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

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

The Likelihood Value

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

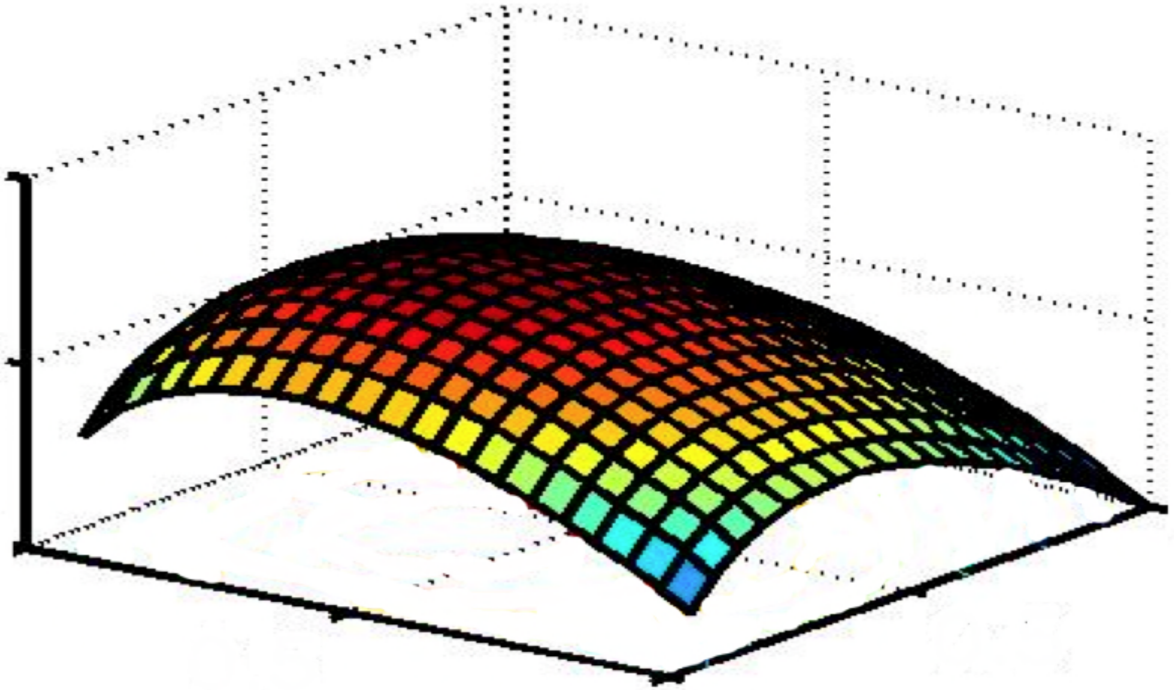
Finding the optimal parameters

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

$$CLogLik(D) = \sum_{i=1}^{n} \log P(c_i \mid d_i)$$

- 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, 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

Maxent Models and Discriminative Estimation

Maximizing the likelihood

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.

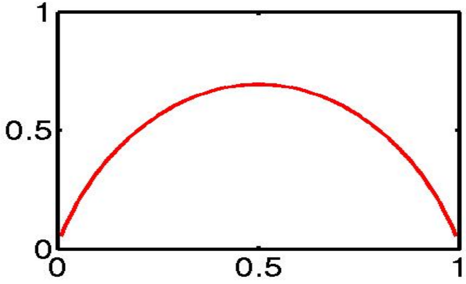
Ignorance is preferable to error and he is less remote from the truth who believes nothing than he who believes what is wrong – Thomas Jefferson (1781)

Maxent Examples I

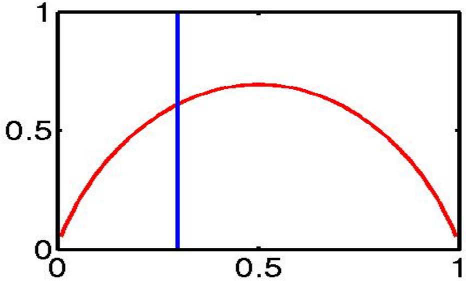
- What do we want from a distribution?
  - Minimize commitment = maximize entropy.
  - Resemble some reference distribution (data).
- Solution: maximize entropy *H*, subject to feature-based constraints:

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

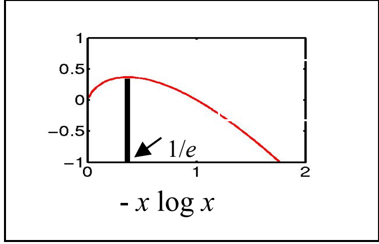
- Adding constraints (features):
  - Lowers maximum entropy
  - Raises maximum likelihood of data
  - Brings the distribution further from uniform
  - Brings the distribution closer to data



Unconstrained, max at 0.5

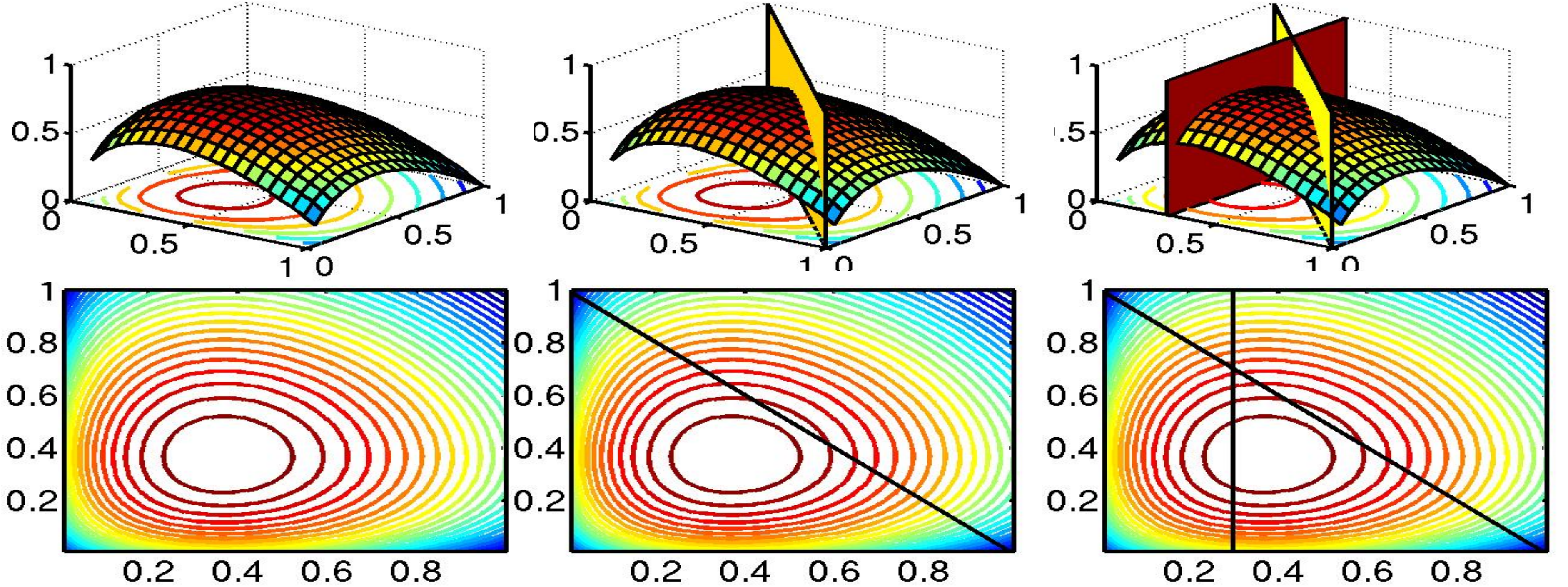


Constraint that $p_{\mathrm{HEADS}} = 0.3$



Maxent Examples II

$\mathrm{H}(p_{\mathrm{H}}\,p_{\mathrm{T}},)$ $\qquad$ $p_{\mathrm{H}} + p_{\mathrm{T}} = 1$ $\qquad$ $p_{\mathrm{H}} = 0.3$



Maxent Examples III

- Let's 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/ Feature Interaction

How overlapping features work in maxent models

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: Named Entity Feature 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

	A	a
B		
b		

All = 1

	A	a
B		
b		

A = 2/3

	A	a
B		
b		

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	a
B		
b		

A = 2/3

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

	A	a
B		
b		

B = 2/3

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

	A	a
B		
b		

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		
b		

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

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/Priors/ Regularization

- Combating over fitting

- Intuition: don't let the weights get very large

$$w_{\mathrm{MLE}} = \mathrm{argmax}_w \log P(y_1, \ldots, y_d | x_1, \ldots, x_d; w)$$

$$\mathrm{argmax}_w \log P(y_1, \ldots, y_d | x_1, \ldots, x_d; w) - \delta \sum_{i=1}^{V} w_i^2$$

Perceptron Algorithm

- Algorithm is Very similar to logistic regression
- Not exactly computing gradients

```
Initalize weight vector w = 0
Loop for K iterations
    Loop For all training examples x_i
        if sign(w * x_i) != y_i
            w += (y_i - sign(w * x_i)) * x_i
```

MaxEnt v.s Perceptron

- Batch v.s Online learning
- Perceptron doesn’t always make updates
- Probabilities v.s scores

Multi-class Perceptron Algorithm

```
Initalize weight vector w = 0
Loop for K iterations
    Loop For all training examples x_i
        y_pred = argmax_y w_y * x_i
        if y_pred != y_i
            w_y_gold += x_i
            w_y_pred -= x_i
```

Regularization in the Perceptron Algorithm

- No gradient computed, so can’t directly include a regularizer in an object function.
- Instead run different numbers of iterations
- Use parameter averaging, for instance, average of all parameters after seeing each data point