Maxent Models and Discriminative Estimation

Christopher Manning
CS224N/Ling280

Introduction

- So far we've looked at "generative models"
  - Language models, Naive Bayes, IBM MT
- 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): P(c,d)
  - All the best known StatNLP models:
    - *n*-gram models, Naive Bayes classifiers, hidden Markov models, probabilistic context-free grammars
- Discriminative (conditional) models take the data as given, and put a probability over hidden structure given the data: P(c|d)
  - Logistic regression, conditional loglinear models, maximum entropy markov models, (SVMs, perceptrons)

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 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)

Features

- In these slides 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 (bounded) real value: $f: C \times D \rightarrow \mathbf{R}$
- Usually features specify an indicator function 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 *in bed* | TO NN *to aid* | TO VB *to aid* | IN JJ *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	Label: MONEY	Label: NN
Features: {…, stocks, hit, a, yearly, low, …}	Features: {…, P=restructure, N=debt, L=12, …}	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: POS 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	%

Features

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

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

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

1.2 −1.8 TO NN *to aid* TO VB *to aid* 0.3

  - Choose the class c which maximizes $\Sigma\lambda_i f_i(c,d) = \mathrm{VB}$
  - 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.

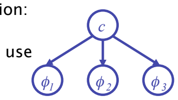
    - P(NN|*to, aid,* TO) = $e^{1.2}e^{-1.8}/(e^{1.2}e^{-1.8} + e^{0.3})$ = 0.29
    - P(VB|*to, aid,* TO) = $e^{0.3}/(e^{1.2}e^{-1.8} + e^{0.3})$ = 0.71
  - The weights are the parameters of the probability model, combined via a "soft max" function
  - Given this model form, we will choose parameters $\{\lambda_i\}$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

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:

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

Naïve-Bayes is just an exponential model.

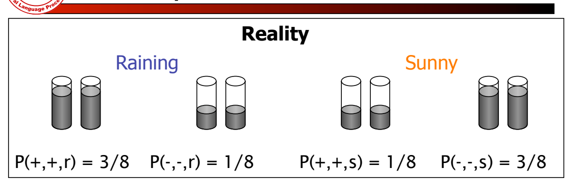
$$\Rightarrow \frac{\exp\left[\sum_i \lambda_{ic} f_{ic}(d,c)\right]}{\sum_{c'} \exp\left[\sum_i \lambda_{ic'} f_{ic'}(d,c')\right]}$$

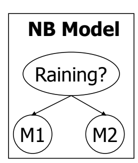
Comparison to Naïve-Bayes

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

Naïve-Bayes	Maxent
Trained to maximize joint likelihood of data and classes.	Trained to maximize the conditional likelihood of classes.
Features assumed to supply independent evidence.	Features weights take feature dependence into account.
Feature weights can be set independently.	Feature weights must be mutually estimated.
Features must be of the conjunctive $\Phi(d) \wedge c = c_i$ form.	Features need not be of the conjunctive form (but usually are).

Example: Sensors





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.

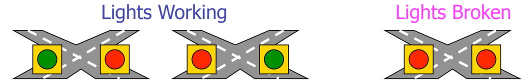
$$\frac{P(r\mid +\ldots+)}{P(s\mid +\ldots+)} = \frac{P(r)}{P(s)}\frac{P(+\mid r)}{P(+\mid s)}\cdots\frac{P(+\mid r)}{P(+\mid s)}$$

- 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

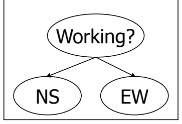
Reality

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!

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

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

- 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:

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

Fitting the Model

- To find the parameters $\lambda_1, \lambda_2, \lambda_3$ write out the conditional log-likelihood of the training data and maximize it

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

- The log-likelihood is concave and has a single maximum; use your favorite numerical optimization package
- Good large scale techniques: conjugate gradient or limited memory quasi-Newton

Fitting the Model Generalized Iterative Scaling

- A simple optimization algorithm which works when the features are non-negative
- We need to define a slack feature to make the features sum to a constant over all considered pairs from $D \times C$
- Define $M = \max_{i,c} \sum_{j=1}^{m} f_j(d_i, c)$
- Add new feature

$$f_{m+1}(d,c) = M - \sum_{j=1}^{m} f_j(d,c)$$

Generalized Iterative Scaling

- Compute empirical expectation for all features
  $E_{\tilde{p}}(f_j) = \frac{1}{N} \sum_{i=1}^{n} f_j(d_i, c_i)$
- Initialize $\lambda_j = 0, j = 1 \ldots m+1$
- Repeat
  - Compute feature expectations according to current model
    $E_{p^t}(f_j) = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} P(c_k | d_i) f_j(d_i, c_k)$
  - Update parameters
    $$\lambda_j^{(t+1)} = \lambda_j^{(t)} + \frac{1}{M} \log\left(\frac{E_{\tilde{p}}(f_j)}{E_{p^t}(f_j)}\right)$$
- Until converged

Maximum Entropy Models

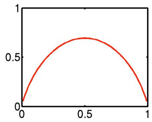
- An equivalent approach:
  - Lots of distributions out there, most of them very spiked, specific, overfit.
  - We want a distribution which is uniform except in specific ways we require.
  - Uniformity means high entropy – we can search for distributions which have properties we desire, but also have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

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

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



p_{HEADS}

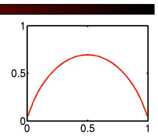
A coin-flip is most uncertain for a fair coin.

Maxent Examples I

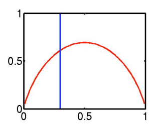
- What do we want from a distribution?
  - Minimize commitment = maximize entropy.
  - Resemble some reference distribution (data).
- Solution: maximize entropy H, subject to feature-based constraints:

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

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

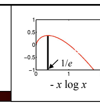


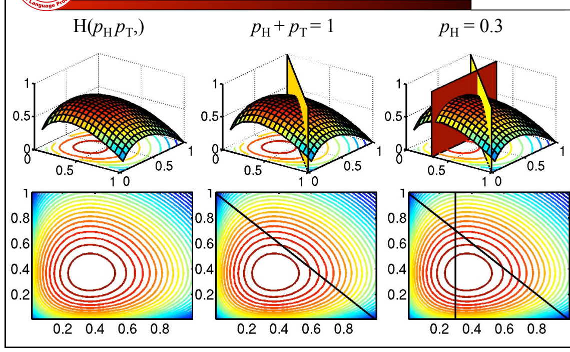
Unconstrained, max at 0.5



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

Maxent Examples II





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

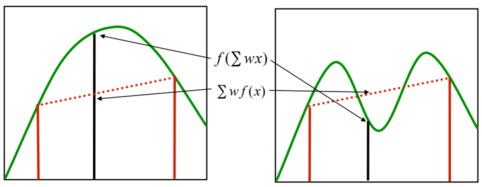
- … and proper nouns are more frequent than common nouns, so we add f_{P} = {NNP, NNPS}, with E[f_{P}] =24/36

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.

Convexity

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

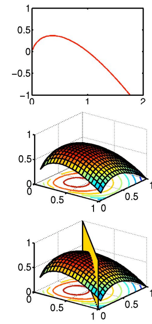


Convex Non-Convex

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.



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		
b		

A = 2/3

	A	a
B		
b		

A = 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	1/3	1/6
b	1/3	1/6

	A	a
B		
b		

	A	a
B	λ_A	
b	λ_A	

	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

	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		
b		

B = 2/3

	A	a
B		
b		

AB = 1/3

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

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

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

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

	A	a
B		
b		

	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

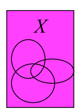
Feature Type	Feature	PERS	LOC
Previous word	*at*	-0.73	0.94
Current word	*Grace*	0.03	0.00
Beginning bigram	*<G*	0.45	-0.04
Current POS tag	NNP	0.47	0.45
Prev and cur tags	IN NNP	-0.10	0.14
Previous state	Other	-0.70	-0.92
Current signature	Xx	0.80	0.46
Prev state, cur sig	O-Xx	0.68	0.37
Prev-cur-next 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**

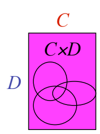
Classification

- What do these joint models of P(X) have to do with conditional models P(C|D)?
- Think of the space C×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 P(C,D).
- This will involve calculating expectations of features (over C×D):

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

- Generally impractical: can't enumerate d efficiently.





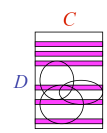
Classification II

- 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!

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:

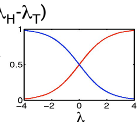
Heads	Tails
h	t

- 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 ($\lambda = \lambda_H - \lambda_T$)

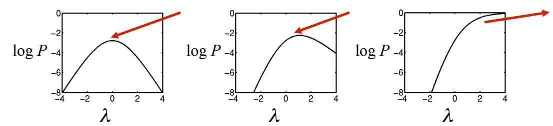
$$p_{\text{HEADS}} = \frac{e^{\lambda_H}e^{-\lambda_T}}{e^{\lambda_H}e^{-\lambda_T}+e^{\lambda_T}e^{-\lambda_T}} = \frac{e^{\lambda}}{e^{\lambda}+e^{0}} \quad p_{\text{TAILS}} = \frac{e^{0}}{e^{\lambda}+e^{0}}$$



Smoothing: Issues

- The data likelihood in this model is:

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



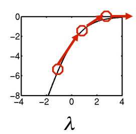
Heads	Tails
2	2

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)

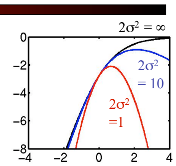
- What if we had a prior expectation that parameter values wouldn't be very large?
- We could then balance evidence suggesting large parameters (or infinite) against our prior.
- The evidence would never totally defeat the prior, and parameters would be smoothed (and kept finite!).
- We can do this explicitly by changing the optimization objective to maximum posterior likelihood:

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

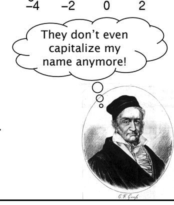
Posterior Prior Evidence

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.

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

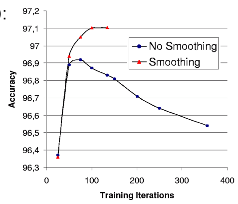
Feature Weights

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):

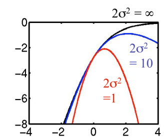
	Overall Accuracy	Unknown Word Acc
Without Smoothing	96.54	85.20
With Smoothing	97.10	88.20



- 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: Priors

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



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

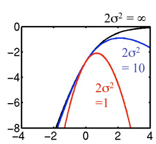
$$\log P(C,\lambda \mid D) = \sum_{(c,d)\in(C,D)} P(c \mid d,\lambda) - \sum_i \frac{(\lambda_i-\mu_i)^2}{2\sigma_i^2} + k$$

- Change the derivative:

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

Smoothing: Priors

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



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

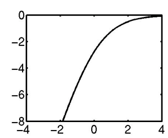
$$\log P(C,\lambda \mid D) = \sum_{(c,d)\in(C,D)} P(c \mid d,\lambda) - \sum_i \frac{\lambda_i^2}{2\sigma_i^2} + k$$

- Change the derivative:

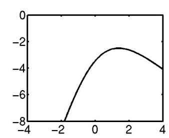
$$\partial \log P(C,\lambda \mid D)/\partial\lambda_i = \text{actual}(f_i,C) - \text{predicted}(f_i,\lambda) \quad -\lambda_i/\sigma^2$$

Smoothing: Virtual Data

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



Heads	Tails
4	0



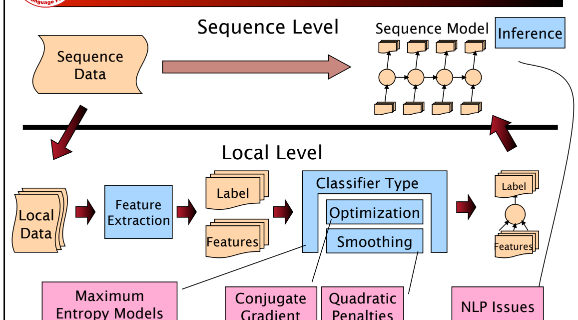
Heads	Tails
5	1

  - 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.

Inference in Systems



MEMM inference in systems

- For a Conditional Markov Model (CMM) a.k.a. a Maximum Entropy Markov Model (MEMM), the classifier makes a single decision at a time, conditioned on evidence from observations and previous decisions.
- A larger space of sequences is explored via search

Local Context

Decision Point

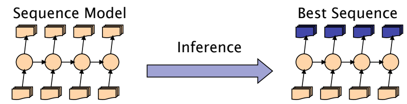
-3	-2	-1	0	+1
DT	NNP	VBD	???	???
The	Dow	fell	22.6	%

Features

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

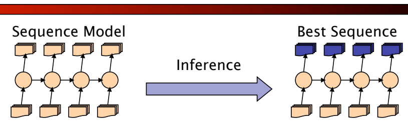
(Ratnaparkhi 1996; Toutanova et al. 2003, etc.)

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).

CRFs [Lafferty, Pereira, and McCallum 2001]

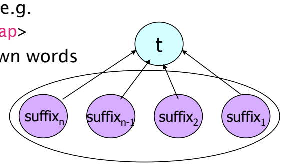
- Another sequence model: Conditional Random Fields (CRFs)
- A whole-sequence conditional model rather than a chaining of local models.

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

- The space of c's is now the space of sequences
  - But if the features f_i remain local, the conditional sequence likelihood can be calculated exactly using dynamic programming
- Training is slow, but CRFs avoid causal-competition biases
- These (or a variant using a max margin criterion) are seen as the state-of-the-art these days

HMM Tagging Models - Brants 2000

- Highly competitive with other state-of-the art models
- Trigram HMM with smoothed transition probabilities
- Capitalization feature becomes part of the state – each tag state is split into two e.g.
  NN → <NN,cap>,<NN,not cap>
- Suffix features for unknown words



$$P(w \mid tag) = P(suffix \mid tag)(w \mid suffix)$$
$$\approx \hat{P}(suffix)\tilde{P}(tag \mid suffix) / \hat{P}(tag)$$

$$\tilde{P}(tag \mid suffix_n) = \lambda_1 \hat{P}(tag \mid suffix_n) + \lambda_2 \hat{P}(tag \mid suffix_{n-1}) + \ldots + \lambda_n \hat{P}(tag)$$

MEMM Tagging Models -II

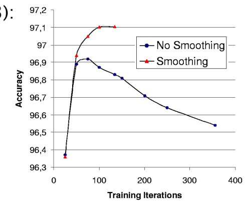
- Ratnaparkhi (1996): local distributions are estimated using maximum entropy models
  - Previous two tags, current word, previous two words, next two words, suffix, prefix, hyphenation, and capitalization features for unknown words
- Toutanova et al. (2003)
  - Richer features, bidirectional inference, better smoothing, better unknown word handling

Model	Overall Accuracy	Unknown Words
HMM (Brants 2000)	96.7	85.5
MEMM (Ratn. 1996)	96.63	85.56
MEMM (T. et al 2003)	97.24	89.04

Smoothing: 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)!

Summary of Tagging

For tagging, the change from generative to discriminative model **does not by itself** result in great improvement

One profits from discriminative models for specifying dependence on **overlapping features of the observation** such as spelling, suffix analysis,etc

A CMM allows integration of rich features of the observations, but can suffer strongly from assuming independence from following observations; this effect can be relieved by adding dependence on following words

This additional power (of the CMM ,CRF, Perceptron models) has been shown to result in improvements in accuracy

The **higher accuracy** of discriminative models comes at the price of **much slower training**