Logistic Regression

COSI 114 – Computational Linguistics

James Pustejovsky

February 10, 2015

Brandeis University

Classification

- **Learn**: h: **X**->Y
  - **X** – features
  - Y – target classes
- Suppose you know P(Y|**X**) exactly, how should you classify?
  - Bayes classifier:

$$y^* = h_{bayes}(x) = \arg\max_y P(Y = y \mid X = x)$$

- Why?

Generative vs. Discriminative Classifiers - Intuition

- Generative classifier, e.g., Naïve Bayes:
  - Assume some functional form for **P(X|Y), P(Y)**
  - Estimate parameters of P(X|Y), P(Y) directly from training data
  - Use Bayes rule to calculate P(Y|X=x)
  - This is 'generative' model
    - Indirect computation of P(Y|X) through Bayes rule
    - But, can generate a sample of the data, $P(X) = \sum_y P(y)P(X \mid y)$
- Discriminative classifier, e.g., Logistic Regression:
  - Assume some functional form for **P(Y|X)**
  - Estimate parameters of P(Y|X) directly from training data
  - This is the 'discriminative' model
    - Directly learn P(Y|X)
    - But cannot sample data, because P(X) is not available

The Naïve Bayes Classifier

- Given:
  - Prior P(Y)
  - n conditionally independent features X given the class Y
  - For each Xi, we have likelihood $P(X_i|Y)$
- Decision rule:

$$y^* = h_{NB}(x) = \arg\max_y P(y)P(x_1,\ldots,x_n \mid y)$$

$$= \arg\max_y P(y)\prod_i P(x_i \mid y)$$

- If assumption holds, NB is optimal classifier!

History

- The concept of Maximum Entropy can be traced back along multiple threads to Jaynes 1957.
- Introduced to NLP area by Berger et. Al. (1996).
- Used in many NLP tasks: MT, Tagging, Parsing, PP attachment, LM, …

Outline

- **Modeling: Intuition, basic concepts, …**
- Parameter training
- Feature selection
- **Case study**

Reference papers

- (Jaynes, 1957a)
- (Jaynes, 1957b)
- (Ratnaparkhi, 1997)
- (Ratnaparkhi, 1996)
- (Berger et. al., 1996)
- (Klein and Manning, 2003)

Modeling

The basic idea

- Goal: estimate p

- Choose p with maximum entropy (or "uncertainty") subject to the constraints (or "evidence").

$$H(p) = -\sum_{x \in A \times B} p(x) \log p(x)$$

$$x = (a, b), \quad where \quad a \in A \wedge b \in B$$

Setting

- From training data, collect (a, b) pairs:
  - a: thing to be predicted (e.g., a class in a classification problem)
  - b: the context
  - Ex: POS tagging:
    - a=NN
    - b=the words in a window and previous two tags
- Learn the prob of each (a, b): p(a, b)

Features in POS tagging (Ratnaparkhi, 1996)

Condition	Features	
w_i is not rare	$w_i = X$	& $t_i = T$
w_i is rare	X is prefix of w_i, $\|X\| \leq 4$	& $t_i = T$
	X is suffix of w_i, $\|X\| \leq 4$	& $t_i = T$
	w_i contains number	& $t_i = T$
	w_i contains uppercase character	& $t_i = T$
	w_i contains hyphen	& $t_i = T$
$\forall\ w_i$	$t_{i-1} = X$	& $t_i = T$
	$t_{i-2}t_{i-1} = XY$	& $t_i = T$
	$w_{i-1} = X$	& $t_i = T$
	$w_{i-2} = X$	& $t_i = T$
	$w_{i+1} = X$	& $t_i = T$
	$w_{i+2} = X$	& $t_i = T$

context (a.k.a. history)

allowable classes

Maximum Entropy

- Why **maximum** entropy?
- Maximize entropy = Minimize commitment

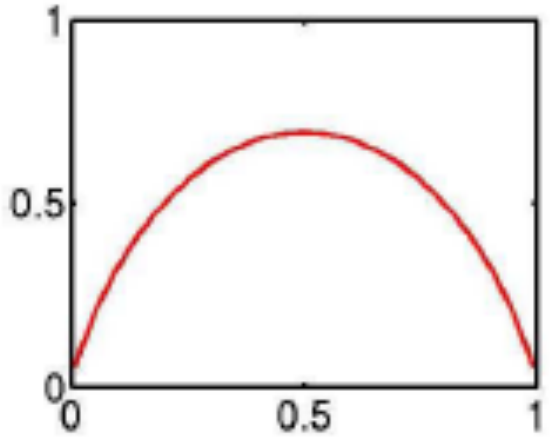
- Model all that is known and assume nothing about what is unknown.
  - Model all that is known: satisfy a set of constraints that must hold
  - Assume nothing about what is unknown: choose the most "uniform" distribution
    ➔ choose the one with maximum entropy

Ex1: Coin-flip example (Klein & Manning 2003)

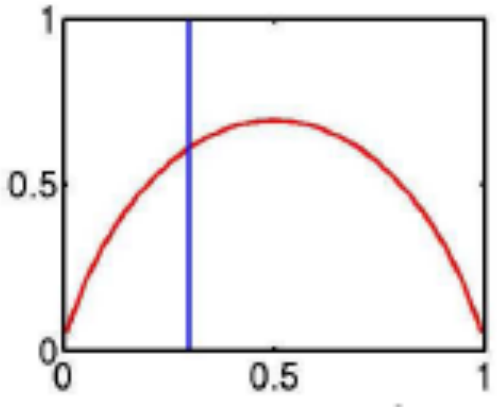
- Toss a coin: p(H)=p1, p(T)=p2.
- Constraint: p1 + p2 = 1
- Question: what' s your estimation of p=(p1, p2)?
- Answer: choose the p that maximizes H(p)

$$H(p) = -\sum_{x} p(x) \log p(x)$$

H



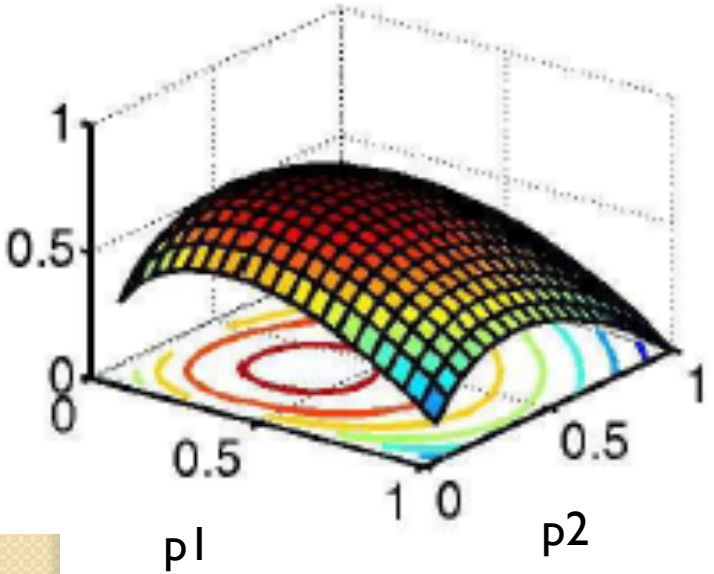
p1



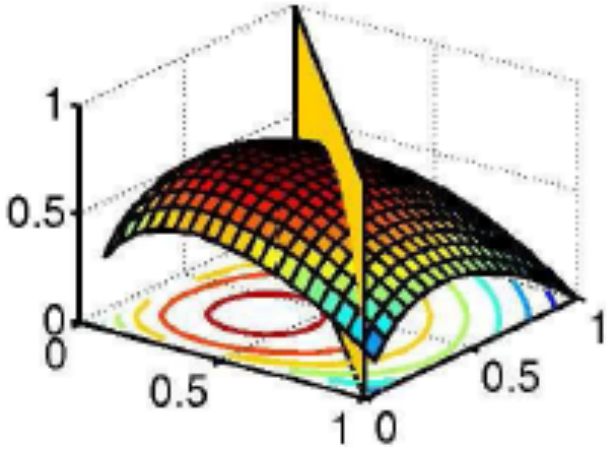
p1=0.3

Coin-flip example (cont)

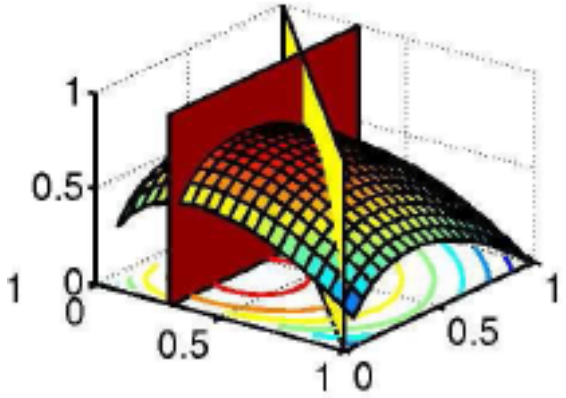
H



p1 p2



p1 + p2 = 1



p1+p2=1.0, p1=0.3

Ex2: An MT example (Berger et. al., 1996)

Possible translation for the word "in" is:

$$\{dans,\ en,\ \grave{a},\ au\ cours\ de,\ pendant\}$$

Constraint:

$$p(dans) + p(en) + p(\grave{a}) + p(au\ cours\ de) + p(pendant) = 1$$

Intuitive answer:

$$\begin{aligned} p(dans) &= 1/5 \\ p(en) &= 1/5 \\ p(\grave{a}) &= 1/5 \\ p(au\ cours\ de) &= 1/5 \\ p(pendant) &= 1/5 \end{aligned}$$

An MT example (cont)

Constraints:

$$p(dans) + p(en) = 3/10$$

$$p(dans) + p(en) + p(\grave{a}) + p(au\ cours\ de) + p(pendant) = 1$$

Intuitive answer:

$$p(dans) = 3/20$$

$$p(en) = 3/20$$

$$p(\grave{a}) = 7/30$$

$$p(au\ cours\ de) = 7/30$$

$$p(pendant) = 7/30$$

An MT example (cont)

Constraints:

$$p(dans) + p(en) = 3/10$$
$$p(dans) + p(en) + p(\grave{a}) + p(au\ cours\ de) + p(pendant) = 1$$
$$p(dans) + p(\grave{a}) = 1/2$$

Intuitive answer:

??

Ex3: POS tagging (Klein and Manning, 2003)

- 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

Ex3 (cont)

- 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

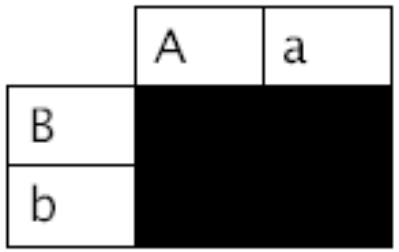
4/36	4/36	12/36	12/36	2/36	2/36

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

Ex4: overlapping features (Klein and Manning, 2003)

Empirical

	A	a
B	1	1
b	1	0

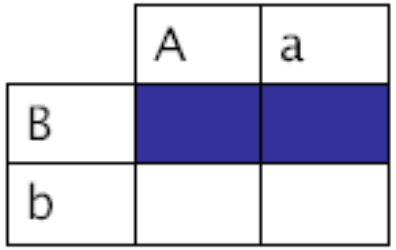


All = 1

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

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

Modeling the problem

- Objective function: H(p)

- Goal: Among all the distributions that satisfy the constraints, choose the one, p*, that maximizes H(p).

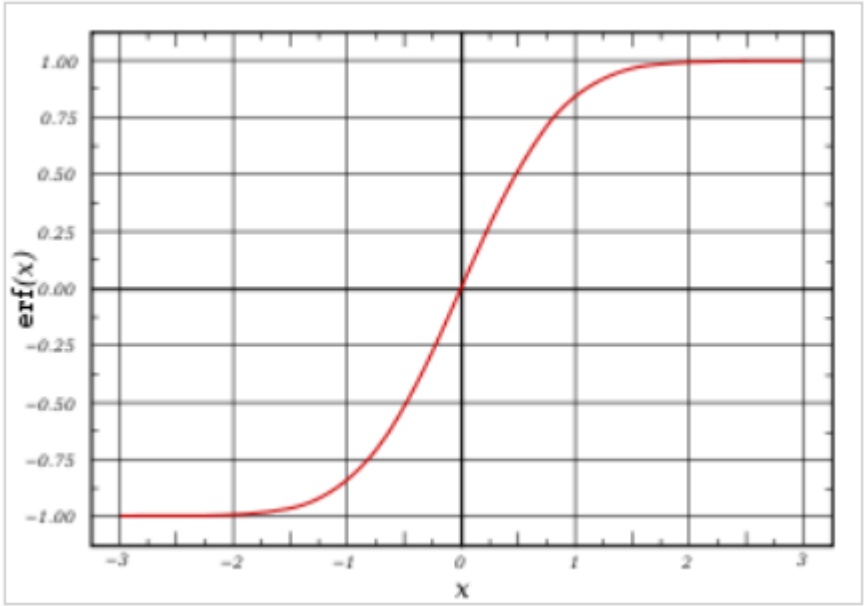
$$p^* = \arg\max_{p \in P} H(p)$$

- Question: How to represent constraints?

Logistic Regression

- Let X be the data instance, and Y be the class label: Learn P(Y|X) directly
  - Let W = (W1, W2, … Wn), X=(X1, X2, …, Xn), **WX** is the dot product
  - Sigmoid function:

$$P(Y=1|\mathbf{X}) = \frac{1}{1+e^{-\mathbf{wx}}}$$



Logistic Regression

- In logistic regression, we learn the conditional distribution P(y|x)
- Let $p_y(x;w)$ be our estimate of P(y|x), where w is a vector of adjustable parameters.
- Assume there are two classes, y = 0 and y = 1 and

$$p_1(\mathbf{x};\mathbf{w}) = \frac{1}{1+e^{-\mathbf{wx}}}$$

- This is equivalent to $p_0(\mathbf{x};\mathbf{w}) = 1 - \frac{1}{1+e^{-\mathbf{wx}}}$

- That is, the log odds of class 1 is a linear function of x
- Q: How to find **W**?

$$\log \frac{p_1(\mathbf{x};\mathbf{w})}{p_0(\mathbf{x};\mathbf{w})} = \mathbf{wx}$$

Constructing a Learning Algorithm

- The conditional data likelihood is the probability of the observed Y values in the training data, conditioned on their corresponding X values. We choose parameters w that satisfy

$$\mathbf{w} = \arg\max_{\mathbf{w}} \prod_{l} P(y^l \mid \mathbf{x}^l, \mathbf{w})$$

- where $\mathbf{w} = \langle w_0, w_1, \ldots, w_n \rangle$ is the vector of parameters to be estimated, y^l denotes the observed value of Y in the *l* th training example, and $\mathbf{x}^l$ denotes the observed value of **X** in the *l* th training example

Summary of Logistic Regression

- Learns the Conditional Probability Distribution P(y|x)
- Local Search.
  - Begins with initial weight vector.
  - Modifies it iteratively to maximize an objective function.
  - The objective function is the conditional log likelihood of the data – so the algorithm seeks the probability distribution P(y|x) that is most likely given the data.

What you should know about LR

- In general, NB and LR make different assumptions
  - NB: Features independent given class -> assumption on P(X|Y)
  - LR: Functional form of P(Y|X), no assumption on P(X|Y)
- LR is a linear classifier
  - decision rule is a hyperplane
- LR optimized by conditional likelihood
  - no closed-form solution
  - concave -> global optimum with gradient ascent

Relation Between Naïve Bayes and Logistic Regression

- Naïve Bayes with Gaussian distributions for features (GNB), can be shown to given the same functional form for the conditional distribution P(*Y*|*X*).
  - But converse is not true, so Logistic Regression makes a weaker assumption.
- Logistic regression is a **discriminative** rather than generative model, since it models the conditional distribution P(*Y*|*X*) and directly attempts to fit the training data for predicting *Y* from *X*. Does not specify a full joint distribution.
- When conditional independence is violated, logistic regression gives better generalization if it is given sufficient training data.
- GNB converges to accurate parameter estimates faster (O(log *n*) examples for *n* features) compared to Logistic Regression (O(*n*) examples).
  - Experimentally, GNB is better when training data is scarce, logistic regression is better when it is plentiful.

Maximum Entropy

- Why ***maximum*** entropy??
- Recall: so far, we always "liked"
  - minimum entropy...
    = minimum uncertainty
    = maximum predictive power
    .... distributions
  - always: relative to some "real world" data
  - always: clear relation between the data, model and parameters: e.g., n-gram language model
- This is still the case! But...

The Maximum Entropy Principle

- Given some set of constraints ("relations", "facts"), which ***must*** hold (i.e., we believe they correspond to the real world we model):

  What is the best distribution among those available?

- Answer: the one with ***maximum entropy***

  (of such distributions satisfying the constraints)

- Why? ...philosophical answer:
  - Occam's razor; Jaynes, ...:
    - make things as simple as possible, but not simpler;
    - do not pretend you know something you don't

Entropy

- Entropy(self-information)

$$H(p) = H(X) = -\sum_{x \in \chi} p(x) \log_2 p(x)$$

  - the amount of information in a random variable
  - average uncertainty of a random variable
  - the average length of the message needed to transmit an outcome of that variable
  - the size of the search space consisting of the possible values of a random variable and its associated probabilities

- Properties

$H(X) \geq 0$ $H(X) = 0$: providing no new information)

  - increases with message length

Entropy Example

- Simplified Polynesian
  - letter frequencies

i	p	t	k	a	i	u
P(i)	1/8	1/4	1/8	1/4	1/8	1/8

  - per-letter entropy

$$H(P) = -\sum_{i \in \{p,t,k,a,i,u\}} P(i) \log P(i) = 2.5 \text{ bits}$$

  - Coding

p	t	k	a	i	u
100	00	101	01	110	111

From Manning & Schutze, 1999

Perplexity and Entropy

- Both measure the (un)certainty of a model
  - How many choices are there at any given point
- Perplexity is $2^{(\text{entropy})}$ that is $2^{H(x1n,m)}$
  - Manning & Schutze hypothesize speech people want to show bigger gains when they reduce perplexity
  - Lowering perplexity from 940 to 540 is more impressive then reducing cross entropy from 9.9 to 9.1 bits

Example

- Throwing the "unknown" die
  - do not know anything → we should assume a fair die
    (uniform distribution ~ max. entropy distribution)
- Throwing unfair die
  - we know: p(4) = 0.4, p(6) = 0.2, nothing else
  - best distribution?
  - do not assume anything about the rest:

1	2	3	4	5	6
0.1	0.1	0.1	0.4	0.1	0.2

- What if we use instead:

1	2	3	4	5	6
0.25	0.05	0.05	0.4	0.05	0.2

?

Using Non-Maximum Entropy Distribution

- ME distribution: p:

1	2	3	4	5	6
0.1	0.1	0.1	0.4	0.1	0.2

- Using instead: q:

1	2	3	4	5	6
0.25	0.05	0.05	0.4	0.05	0.2

- Result depends on the real world:
  - real world ~ our constraints (p(4) = 0.4, p(6) = 0.2), everything else no specific constraints:
    - our average error: D(q||p) [recall: Kullback-Leibler distance]
  - real world ~ orig. constraints + p(1) = 0.25:
    - q is best (but hey, then we should have started with all 3 constraints!)

Things in Perspective: n-gram LM

- Is an n-gram model a ME model?
  - yes if we believe that trigrams are the all and only constraints
    - trigram model constraints: $p(z|x,y) = c(x,y,z)/c(x,y)$
  - no room for any "adjustments"
    - like if we say $p(2) = 0.7$, $p(6) = 0.3$ for a throwing die
- Accounting for the apparent inadequacy:
  - smoothing
  - ME solution: (sort of) smoothing "built in"
    - constraints from training, maximize entropy on training + heldout

Features and Constraints

- Introducing...
  - binary valued selector functions ("features"):
    - $f_i(y,x) \in \{0,1\}$, where
      - $y \in Y$ (sample space of the event being predicted, e.g. words, tags, ...),
      - $x \in X$ (space of contexts, e.g. word/tag bigrams, unigrams, weather conditions, of - in general - unspecified nature/length/size)

unusual!

  - constraints:
    - $E_p(f_i(y,x)) = E'(f_i(y,x))$ (= empirical expectation)
    - recall: expectation relative to distribution p: $E_p(f_i) = \sum_{y,x} p(x,y) f_i(y,x)$
    - empirical expectation: $E'(f_i) = \sum_{y,x} p'(x,y) f_i(y,x) = 1/|T| \sum_{t=1..T} f_i(y_t,x_t)$
    - notation: $E'(f_i(y,x)) = d_i$: constraints of the form $E_p(f_i(y,x)) = d_i$

Additional Constraint (Ensuring Probability Distribution)

- The model's p(y|x) should be probability distribution:
  - add an "omnipresent" feature $f_0(y,x) = 1$ for all y,x
  - constraint: $E_p(f_0(y,x)) = 1$
- Now, assume:
  - We know the set $S = \{f_i(y,x), i=0..N\}$ ($|S| = N+1$)
  - We know all the constraints
    - i.e. a vector d_i, one for each feature, i=0..N
- Where are the parameters?
  - ...we do not even know the form of the model yet

The Model

- Given the constraints, what is the form of the model which maximizes the entropy of p?
- Use Lagrangian Multipliers:
  - minimizing some function $\phi(z)$ in the presence of N constraints $g_i(z) = d_i$ means to minimize

    $$\phi(x) - \Sigma_{i=1..N}\lambda_i(g_i(x) - d_i) \qquad \text{(w.r.t. all } \lambda_i \text{ and } x)$$

  - our case, minimize

    $$A(p) = -H(p) - \Sigma_{i=1..N}\lambda_i(E_p(f_i(y,x)) - d_i) \quad \text{(w.r.t. all } \lambda_i \text{ and } p!)$$

  - i.e. $\phi(z) = -H(p)$, $g_i(z) = E_p(f_i(y,x))$ (variable z ~ distribution p)

Loglinear (Exponential) Model

- Minimize: for p, derive (partial derivation) and solve $A'(p) = 0$:

  $\delta[-H(p) - \Sigma_{i=0..N}\lambda_i(E_p(f_i(y,x)) - d_i)]/\delta p = 0$

  $\delta[\ \Sigma\ p\ \log(p) - \Sigma_{i=0..N}\lambda_i((\Sigma\ p\ f_i) - d_i)]/\delta p = 0$

  ...

  $1 + \log(p) - \Sigma_{i=0..N}\lambda_i\ f_i = 0$

  $1 + \log(p) = \Sigma_{i=1..N}\lambda_i\ f_i + \lambda_0$

  $p = e^{\Sigma_{i=1..N}\lambda_i\ f_i + \lambda_0 - 1}$

- $p(y,x) = (1/Z)\ e^{\Sigma_{i=1..N}\lambda_i f_i(y,x)}$ ($Z = e^{1-\lambda_0}$, the normalization factor)

Maximizing the Lambdas: Setup

- Model: $p(y,x) = (1/Z)\, e^{\Sigma_{i=1..N}\lambda_i f_i(y,x)}$
- Generalized Iterative Scaling (G.I.S.)
  - obeys form of model & constraints:
    - $E_p(f_i(y,x)) = d_i$
  - G.I.S. needs, in order to work, $\forall y,x\ \Sigma_{i=1..N} f_i(y,x) = C$
    - to fulfill, define additional constraint:
    - $f_{N+1}(y,x) = C_{max} - \Sigma_{i=1..N} f_i(y,x)$, where $C_{max} = \max_{x,y} \Sigma_{i=1..N} f_i(y,x)$
  - also, approximate (because $\Sigma_{x \in \text{All contexts}}$ is not (never) feasible)
    - $E_p(f_i) = \Sigma_{y,x} p(x,y) f_i(y,x) \cong 1/|T| \Sigma_{t=1..T} \Sigma_{y \in Y} p(y|x_t) f_i(y,x_t)$
      (use $p(y,x)=p(y|x)p'(x)$, where $p'(x)$ is empirical i.e. from data T)

Generalized Iterative Scaling

- 1. Initialize $\lambda_i^{(1)}$ (any values, e.g. 0), compute d_i, i=1..N+1
- 2. Set iteration number n to 1.
- 3. Compute current model distribution expected values of all the constraint expectations

  $E_{p^{(n)}}(f_i)$ (based on $p^{(n)}(y|x_t)$)
  - [pass through data, see previous slide; at each data position t, compute $p^{(n)}(y,x_t)$, normalize]
- 4. Update $\lambda_i^{(n+1)} = \lambda_i^{(n)} + (1/C) \log(d_i/E_{p^{(n)}}(f_i))$
- 5. Repeat 3.,4. until convergence.

Comments on Features

- Advantage of "variable" (~ not fixed) context in f(y,x):
  - any feature o.k. (examples mostly for tagging):
    - previous word's part of speech is VBZ or VB or VBP, y is DT
    - next word: capitalized, current: ".", and y is a sentence break (SB detect)
    - y is MD, and the current sentence is a question (last word: question mark)
    - tag assigned by a different tagger is VBP, and y is VB
    - it is before Thanksgiving and y is "turkey" (Language modeling)
    - even (God forbid!) manually written rules, e.g. y is VBZ and there is ...
  - remember, the predicted event plays a role in a feature:
    - also, a set of events: f(y,x) is true if y is NNS or NN, and x is ...
    - x can be ignored as well ("unigram" features)

Feature Selection

- Advantage:
  - throw in many features
    - typical case: specify templates manually (pool of features P), fill in from data, possibly add some specific manually written features
    - let the machine select
    - Maximum Likelihood ~ Minimum Entropy on training data
    - after, of course, computing the λ_i's using the MaxEnt algorithm
- Naive (greedy of course) algorithm:
  - start with empty S, add feature at a time (MLE after ME)
  - too costly for full computation (|S| x |P| x |ME-time|)
  - Solution: see Berger & DellaPietras

Logistic Regression

- Assumes a parametric form for directly estimating P(*Y* | *X*). For binary concepts, this is:

$$P(Y=1 \mid X) = \frac{1}{1+\exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$

$$P(Y=0 \mid X) = 1 - P(Y=1 \mid X)$$

$$= \frac{\exp(w_0 + \sum_{i=1}^{n} w_i X_i)}{1+\exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$

- Equivalent to a one-layer backpropagation neural net.
  - Logistic regression is the source of the sigmoid function used in backpropagation.
  - Objective function for training is somewhat different.

Logistic Regression as a Log-Linear Model

- Logistic regression is basically a linear model, which is demonstrated by taking logs.

$$\text{Assign label } Y = 0 \text{ iff } 1 < \frac{P(Y=0 \mid X)}{P(Y=1 \mid X)}$$

$$1 < \exp(w_0 + \sum_{i=1}^{n} w_i X_i)$$

$$0 < w_0 + \sum_{i=1}^{n} w_i X_i$$

$$\text{or equivalently} \quad w_0 > \sum_{i=1}^{n} -w_i X_i$$

- Also called a **maximum entropy model** (**MaxEnt**) because it can be shown that standard training for logistic regression gives the distribution with maximum entropy that is consistent with the training data.

Logistic Regression Training

- Weights are set during training to maximize the **conditional data likelihood** :

$$W \leftarrow \underset{W}{\operatorname{argmax}} \prod_{d \in D} P(Y^d \mid X^d, W)$$

where D is the set of training examples and Y^d and X^d denote, respectively, the values of Y and X for example d.

- Equivalently viewed as maximizing the **conditional log likelihood** (CLL)

$$W \leftarrow \underset{W}{\operatorname{argmax}} \sum_{d \in D} \ln P(Y^d \mid X^d, W)$$

Logistic Regression Training

- Like neural-nets, can use standard gradient descent to find the parameters (weights) that optimize the CLL objective function.
- Many other more advanced training methods are possible to speed convergence.
  - Conjugate gradient
  - Generalized Iterative Scaling (GIS)
  - Improved Iterative Scaling (IIS)
  - Limited-memory quasi-Newton (L-BFGS)

Preventing Overfitting in Logistic Regression

- To prevent overfitting, one can use **regularization** (a.k.a. smoothing) by penalizing large weights by changing the training objective:

$$W \leftarrow \underset{W}{\operatorname{argmax}} \sum_{d \in D} \ln P(Y^d \mid X^d, W) - \frac{\lambda}{2}\|W\|^2$$

Where λ is a constant that determines the amount of smoothing

- This can be shown to be equivalent to assuming a Guassian prior for *W* with zero mean and a variance related to 1/λ.

Parameter estimation

Algorithms

- Generalized Iterative Scaling (GIS): (Darroch and Ratcliff, 1972)

- Improved Iterative Scaling (IIS): (Della Pietra et al., 1995)

GIS: setup

Requirements for running GIS:

- Obey form of model and constraints:

$$p(x) = \frac{e^{\sum_{j=1}^{k} \lambda_j f_j(x)}}{Z} \qquad E_p f_j = d_j$$

- An additional constraint:

$$\forall x \in \varepsilon \quad \sum_{j=1}^{k} f_j(x) = C$$

Let $$C = \max_{x \in \varepsilon} \sum_{j=1}^{k} f_j(x)$$

Add a new feature f_{k+1}:

$$\forall x \in \varepsilon \quad f_{k+1}(x) = C - \sum_{j=1}^{k} f_j(x)$$

GIS algorithm

- Compute d_j, j=1, …, k+1
- Initialize $\lambda_j^{(1)}$ (any values, e.g., 0)
- Repeat until converge
  - For each j
    - Compute $$E_{p^{(n)}} f_j = \sum_{x \in \varepsilon} p^{(n)}(x) f_j(x)$$

      where $$p^{(n)}(x) = \frac{e^{\sum_{j=1}^{k+1} \lambda_j{}^{(n)} f_j(x)}}{Z}$$

    - Update $$\lambda_j^{(n+1)} = \lambda_j^{(n)} + \frac{1}{C}(\log \frac{d_i}{E_{p^{(n)}} f_j})$$

Approximation for calculating feature expectation

$$E_p f_j = \sum_{x \in \varepsilon} p(x) f_j(x) = \sum_{a \in A, b \in B} p(a,b) f_j(a,b)$$

$$= \sum_{a \in A, b \in B} p(b) p(a \mid b) f_j(a,b)$$

$$\approx \sum_{a \in A, b \in B} \tilde{p}(b) p(a \mid b) f_j(a,b)$$

$$= \sum_{b \in B} \tilde{p}(b) \sum_{a \in A} p(a \mid b) f_j(a,b)$$

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{a \in A} p(a \mid b_i) f_j(a, b_i)$$

Properties of GIS

- $L(p^{(n+1)}) >= L(p^{(n)})$
- The sequence is guaranteed to converge to p^*.
- The converge can be very slow.

- The running time of each iteration is O(NPA):
  - N: the training set size
  - P: the number of classes
  - A: the average number of features that are active for a given event (a, b).

IIS algorithm

- Compute d_j, j=1, …, k+1 and $f^{\#}(x) = \sum_{j=1}^{k} f_j(x)$
- Initialize $\lambda_j^{(1)}$ (any values, e.g., 0)
- Repeat until converge
  - For each j
    - Let $\Delta\lambda_j$ be the solution to

$$\sum_{x \in \varepsilon} p^{(n)}(x) f_j(x) e^{\Delta\lambda_j f^{\#}(x)} = d_j$$

    - Update $\lambda_j^{(n+1)} = \lambda_j^{(n)} + \Delta\lambda_j$

Calculating $\Delta\lambda_j$

If $\forall x \in \varepsilon \quad \sum_{j=1}^{k} f_j(x) = C$

Then $\Delta\lambda_j = \frac{1}{C}(\log \frac{d_i}{E_{p^{(n)}} f_j})$

GIS is the same as IIS

Else $\Delta\lambda_j$ must be calcuated numerically.

Feature selection

Feature selection

- Throw in many features and let the machine select the weights
  - Manually specify feature templates
- Problem: too many features
- An alternative: greedy algorithm
  - Start with an empty set S
  - Add a feature at each iteration

Notation

With the feature set S:

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

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

After adding a feature:

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

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

The gain in the log-likelihood of the training data:

$$\Delta L(S, \hat{f}) \equiv L(p_{S \cup \hat{f}}) - L(p_S)$$

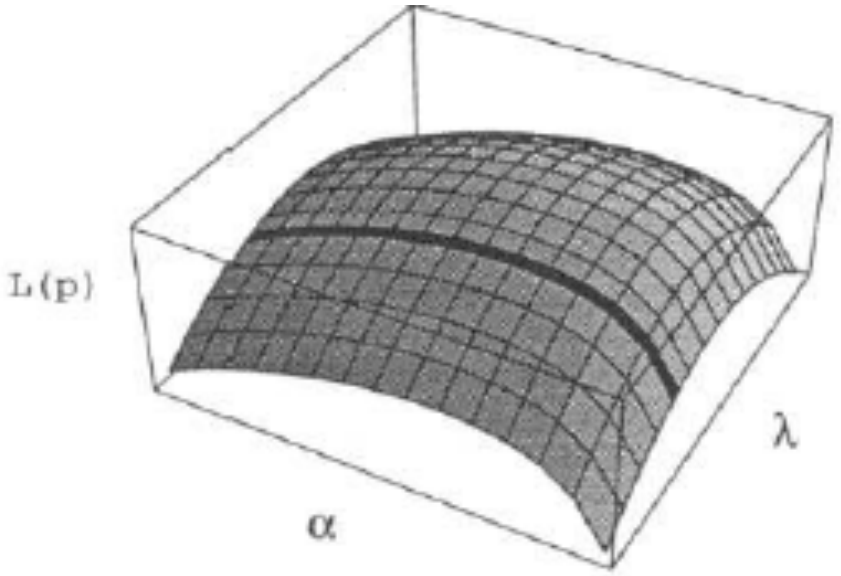
Feature selection algorithm (Berger et al., 1996)

- Start with S being empty; thus p_s is uniform.
- Repeat until the gain is small enough
  - For each candidate feature f
    - Computer the model $p_{S \cup f}$ using IIS
    - Calculate the log-likelihood gain
  - Choose the feature with maximal gain, and add it to S

➔ Problem: too expensive

Approximating gains (Berger et. al., 1996)

- Instead of recalculating all the weights, calculate only the weight of the new feature.



Training a MaxEnt Model

Scenario #1:

- Define features templates
- Create the feature set
- Determine the optimum feature weights via GIS or IIS

Scenario #2:

- Define feature templates
- Create candidate feature set S
- At every iteration, choose the feature from S (with max gain) and determine its weight (or choose top-n features and their weights).

Case study

POS tagging (Ratnaparkhi, 1996)

- Notation variation:
  - $f_j(a, b)$: a: class, b: context
  - $f_j(h_i, t_i)$: h: history for i^{th} word, t: tag for i^{th} word
- History:

$$h_i = \{w_i, w_{i-1}, w_{i-2}, w_{i+1}, w_{i+2}, t_{i-1}, t_{i-2}\}$$

- Training data:
  - Treat it as a list of (h_i, t_i) pairs.
  - How many pairs are there?

Using a MaxEnt Model

- Modeling:
- Training:
  - Define features templates
  - Create the feature set
  - Determine the optimum feature weights via GIS or IIS
- Decoding:

Modeling

$$P(t_1,\ldots,t_n \mid w_1,\ldots,w_n)$$

$$= \prod_{i=1}^{n} p(t_i \mid w_1^n, t_1^{i-1})$$

$$\approx \prod_{i=1}^{n} p(t_i \mid h_i)$$

$$p(t \mid h) = \frac{p(h,t)}{\sum_{t' \in T} p(h,t')}$$

Training step 1: define feature templates

Condition	Features	
w_i is not rare	$w_i = X$	& $t_i = T$
w_i is rare	X is prefix of w_i, $\|X\| \leq 4$	& $t_i = T$
	X is suffix of w_i, $\|X\| \leq 4$	& $t_i = T$
	w_i contains number	& $t_i = T$
	w_i contains uppercase character	& $t_i = T$
	w_i contains hyphen	& $t_i = T$
$\forall\ w_i$	$t_{i-1} = X$	& $t_i = T$
	$t_{i-2}t_{i-1} = XY$	& $t_i = T$
	$w_{i-1} = X$	& $t_i = T$
	$w_{i-2} = X$	& $t_i = T$
	$w_{i+1} = X$	& $t_i = T$
	$w_{i+2} = X$	& $t_i = T$

History h_i Tag t_i

Step 2: Create feature set

Word:	the	stories	about	well-heeled	communities	and	developers
Tag:	DT	NNS	IN	JJ	NNS	CC	NNS
Position:	1	2	3	4	5	6	7

$$
\begin{aligned}
w_i &= \texttt{about} && \& \; t_i = \texttt{IN} \\
w_{i-1} &= \texttt{stories} && \& \; t_i = \texttt{IN} \\
w_{i-2} &= \texttt{the} && \& \; t_i = \texttt{IN} \\
w_{i+1} &= \texttt{well-heeled} && \& \; t_i = \texttt{IN} \\
w_{i+2} &= \texttt{communities} && \& \; t_i = \texttt{IN} \\
t_{i-1} &= \texttt{NNS} && \& \; t_i = \texttt{IN} \\
t_{i-2}t_{i-1} &= \texttt{DT NNS} && \& \; t_i = \texttt{IN}
\end{aligned}
$$

➔Collect all the features from the training data
➔Throw away features that appear less than 10 times

Step 3: determine the feature weights

- GIS
- Training time:
  - Each iteration: O(NTA):
    - N: the training set size
    - T: the number of allowable tags
    - A: average number of features that are active for a (h, t).
  - About 24 hours on an IBM RS/6000 Model 380.
- How many features?

Decoding: Beam search

- Generate tags for w_1, find top N, set s_{1j} accordingly, j=1, 2, …, N
- For i=2 to n (n is the sentence length)
  - For j=1 to N
    - Generate tags for wi, given $s_{(i-1)j}$ as previous tag context
    - Append each tag to $s_{(i-1)j}$ to make a new sequence.
  - Find N highest prob sequences generated above, and set s_{ij} accordingly, j=1, …, N
- Return highest prob sequence s_{n1}.

Beam search

- 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 search

- 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).

Decoding (cont)

- Tags for words:
  - Known words: use tag dictionary
  - Unknown words: try all possible tags

- Ex: "time flies like an arrow"

- Running time: O(NTAB)
  - N: sentence length
  - B: beam size
  - T: tagset size
  - A: average number of features that are active for a given event

Experiment results

MF tag	O	7.66	
Markov 1-gram	B	6.74	
Markov 3-gram	W	3.7	
Markov 3-gram	B	3.64	
Decision tree	M	3.5	
Transformation	B	3.39	
Maxent	R	3.37	
Maxent	O	3.11	±.07
Multi-tagger Voting	B	2.84	±.03

Comparison with other learners

- HMM: MaxEnt uses more context
- SDT: MaxEnt does not split data
- TBL: MaxEnt is statistical and it provides probability distributions.

MaxEnt Summary

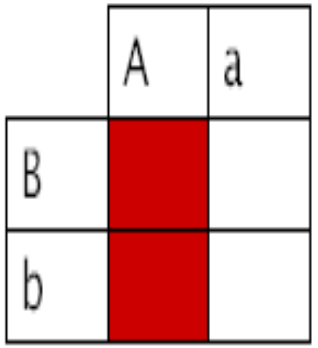
- Concept: choose the p* that maximizes entropy while satisfying all the constraints.
- Max likelihood: p* is also the model within a model family that maximizes the log-likelihood of the training data.
- Training: GIS or IIS, which can be slow.
- MaxEnt handles overlapping features well.
- In general, MaxEnt achieves good performances on many NLP tasks.

Additional slides

Ex4 (cont)

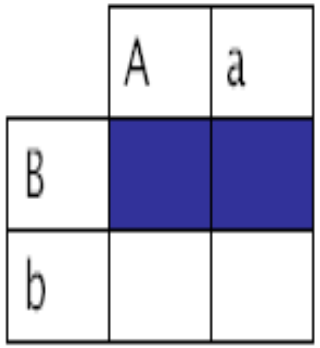
Empirical

	A	a
B	1	1
b	1	0



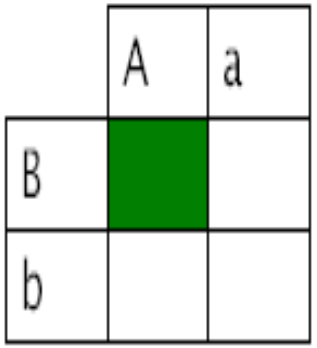
A = 2/3

	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

??

Multinomial Logistic Regression

- Logistic regression can be generalized to multi-class problems (where *Y* has a multinomial distribution).
- Effectively constructs a linear classifier for each category.

The Task, Again

- Recall:
  - tagging ~ morphological disambiguation
  - tagset $V_T \subset (C_1, C_2, \ldots C_n)$
    - C_i - morphological categories, such as POS, NUMBER, CASE, PERSON, TENSE, GENDER, ...
  - mapping $w \rightarrow \{t \in V_T\}$ exists
    - restriction of Morphological Analysis: $A^+ \rightarrow 2^{(L,C1,C2,\ldots,Cn)}$

      where A is the language alphabet, L is the set of lemmas
  - extension to punctuation, sentence boundaries (treated as words)

Maximum Entropy Tagging Model

- General

  $p(y,x) = (1/Z)\ e^{\Sigma_{i=1..N}\lambda_i f_i(y,x)}$

  Task: find λ_i satisfying the model and constraints

  - $E_p(f_i(y,x)) = d_i$

  where

  - $d_i = E'(f_i(y,x))$ (empirical expectation i.e. feature frequency)

- Tagging

  $p(t,x) = (1/Z)\ e^{\Sigma_{i=1..N}\lambda_i f_i(t,x)}$ (λ_0 might be extra: cf. μ in AR(?)

  - $t \in$ Tagset,
  - x ~ context (words and tags alike; say, up to three positions R/L)

Features for Tagging

- Context definition
  - two words back and ahead, two tags back, current word:
    - $x_i = (w_{i-2}, t_{i-2}, w_{i-1}, t_{i-1}, w_i, w_{i+1}, w_{i+2})$
  - features may ask any information from this window
    - e.g.:
      - previous tag is DT
      - previous two tags are PRP$ and MD, and the following word is “be”
      - current word is “an”
      - suffix of current word is “ing”
    - do not forget: feature also contains t_i, the current tag:
      - feature #45: suffix of current word is “ing” & the tag is VBG $\Leftrightarrow f_{45} = 1$

Feature Selection

- The PC[1] way (see also yesterday's class):
  - (try to) test all possible feature combinations
    - features may overlap, or be redundant; also, general or specific - impossible to select manually
  - greedy selection:
    - add one feature at a time, test if (good) improvement:
      - keep if yes, return to the pool of features if not
  - even this is costly, unless some shortcuts are made
    - see Berger & DPs for details
- The other way:
  - use some heuristic to limit the number of features

[1] Politically (or, Probabilistically-stochastically) Correct

Limiting the Number of Features

- Always do (regardless whether you're PC or not):
  - use contexts which appear in the training data (lossless selection)
- More or less PC, but entails huge savings (in the number of features to estimate λ_i weights for):
  - use features appearing only L-times in the data (L ~ 10)
  - use w_i-derived features which appear with rare words only
  - do not use all combinations of context (this is even "LC[1]")
  - but then, use all of them, and compute the λ_i only once using the Generalized Iterative Scaling algorithm

[1] Linguistically Correct

Feature Examples (Context)

- From A. Ratnaparkhi (EMNLP, 1996, UPenn)
  - $t_i = T, w_i = X$ (frequency c > 4):
    - $t_i = VBG, w_i$ = selling
  - $t_i = T, w_i$ contains uppercase char (rare):
    - t_i = NNP, tolower(w_i) $\neq w_i$
  - $t_i = T, t_{i-1} = Y, t_{i-2} = X$:
    - $t_i = VBP, t_{i-2}$ = PRP, t_{i-1} = RB
- Other examples of possible features:
  - $t_i = T, t_j$ is X, where j is the closest left position where Y
    - $t_i = VBZ, t_j$ = NN, $Y \Leftrightarrow t_j \in$ {NNP, NNS, NN}

Feature Examples (Lexical/Unknown)

- From A. Ratnaparkhi :
  - t_i = T, suffix(w_i)= X (length X < 5):
    - t_i = JJ, suffix(w_i) = eled (traveled, leveled, ...)
  - t_i = T, prefix(w_i)= X (length X < 5):
    - t_i = JJ, prefix(w_i) = well (well-done, well-received,...)
  - t_i = T, w_i contains hyphen:
    - t_i = JJ, '-' in w_i (open-minded, short-sighted,...)
- Other possibility, for example:
  - t_i = T, w_i contains X:
    - t_i = NounPl, w_i contains umlaut (ä,ö,ü) (Wörter, Länge,...)

"Specialized" Word-based Features

- List of words with most errors (WSJ, Penn Treebank):
  - about, that, more, up, ...
- Add "specialized", detailed features:
  - t_i = T, w_i = X, t_{i-1} = Y, t_{i-2} = Z:
    - t_i = IN, w_i = about, t_{i-1} = NNS, t_{i-2} = DT
  - possible only for relatively high-frequency words
- Slightly better results (also, inconsistent [test] data)

Maximum Entropy Tagging: Results

- For details, see A Ratnaparkhi
- Base experiment (133k words, < 3% unknown):
  - 96.31% word accuracy
- Specialized features added:
  - 96.49% word accuracy
- Consistent subset (training + test)
  - 97.04% word accuracy (97.13% w/specialized features)
- ~~This is the best result on WSJ so far.~~

Discriminative models

Shift-reduce parser Ratnaparkhi (98)

- Learns a distribution P(T|S) of parse trees given sentences using the sequence of actions of a shift-reduce parser

$$P(T \mid S) = \prod_{i=1}^{n} P(a_i \mid a_1 \ldots a_{i-1} S)$$

- Uses a maximum entropy model to learn conditional distribution of parse action given history
- Suffers from independence assumptions that actions are independent of future observations as CMM
- Higher parameter estimation cost to learn local maximum entropy models
- **Lower** but still good accuracy 86% - 87% labeled precision/ recall

Discriminative Models – Distribution Free Re-ranking

- Represent sentence-parse tree pairs by a feature vector F(X,Y)
- Learn a linear ranking model with parameters $\overline{\alpha}$ using the boosting loss

Model	LP	LR
Collins 99 (Generative)	88.3%	88.1%
Collins 00 (BoostLoss)	89.9%	89.6%

13% error reduction

Still very close in accuracy to generative model (Charniak 00)

Comparison of Generative-Discriminative Pairs

Johnson (2001) have compared simple PCFG trained to maximize L(T,S) and L(T|S)

A Simple PCFG has parameters

$$\Theta \equiv \{\theta_{ij} = P(A_i -> \alpha_j \mid A_i), \sum_{j=1..m_i} \theta_{ij} = 1, \forall\ i\}$$

Models:

$$MLE = \arg\max_{\theta} \sum_{i=1..n} Log(P(T_i, S_i)), \theta \in \Theta$$

$$MCLE = \arg\max_{\theta} \sum_{i=1..n} Log(P(T_i \mid S_i)), \theta \in \Theta$$

Results:

Model	LPrecision	LRecall
MLE	0.815	0.789
MCLE	**0.817**	**0.794**