Maximum Entropy

Lecture #20

Introduction to Natural Language Processing

CMPSCI 585, Spring 2004

University of Massachusetts Amherst

Andrew McCallum

(Slides from Jason Eisner)

summary of half of the course (statistics)

Probability is Useful

- We love probability distributions!
  - We've learned how to define & **use** p(…) functions.
- Pick best output text T from a set of candidates
  - speech recognition; machine translation; OCR; spell correction...
  - maximize $p_1(T)$ for some appropriate distribution p_1
- Pick best annotation T for a fixed input I
  - text categorization; parsing; part-of-speech tagging …
  - maximize p(T | I); equivalently maximize joint probability p(I,T)
    - often define p(I,T) by noisy channel: p(I,T) = p(T) * p(I | T)
  - speech recognition & other tasks above are cases of this too:
    - we're maximizing an appropriate $p_1(T)$ defined by p(T | I)
- Pick best probability distribution (a meta-problem!)
  - really, pick best parameters θ: train HMM, PCFG, n-grams, clusters …
  - maximum likelihood; smoothing; EM if unsupervised (incomplete data)
  - Bayesian smoothing: max $p(\theta|\text{data})$ = max $p(\theta, \text{data}) = p(\theta)p(\text{data}|\theta)$

summary of other half of the course (linguistics)

Probability is Flexible

- We love probability distributions!
  - We've learned how to **define** & use p(…) functions.
- We want p(…) to define probability of *linguistic* objects
  - Trees of (non)terminals (PCFGs; CKY, Earley, pruning, inside-outside)
  - Sequences of words, tags, morphemes, phonemes (n-grams, FSAs, FSTs; regex, Viterbi, forward-backward, collocations)
  - Vectors (naïve Bayes; clustering word senses)
- We've also seen some not-so-probabilistic stuff
  - Syntactic features, morphology. Could be stochasticized?
  - Methods can be quantitative & data-driven but not fully probabilistic: clustering, collocations,…
- But probabilities have wormed their way into most things
- **p(…) has to capture our intuitions about the ling. data**

really so alternative?

An ~~Alternative~~ Tradition

- Old AI hacking technique:
  - Possible parses (or whatever) have scores.
  - Pick the one with the best score.
  - How do you define the score?
    - Completely ad hoc!
    - Throw anything you want into the stew
    - Add a bonus for this, a penalty for that, etc.
- "Learns" over time – as you adjust bonuses and penalties by hand to improve performance.
- Total kludge, but totally flexible too …
  - Can throw in **any** intuitions you might have

really so alternative?

An ~~Alternative~~ Tradition

- Old A
  - Po .
  - Pi
  - Ho
    -
    -
    -
- "Lea uses and pena nce. ϑ
- Total
  - Ca

Probabilistic Revolution Not Really a Revolution, Critics Say

Log-probabilities no more than scores in disguise

"We're just adding stuff up like the old corrupt regime did," admits spokesperson

Nuthin' but adding weights

- n-grams: … + log p(w7 | w5,w6) + log(w8 | w6, w7) + …
- PCFG: log p(NP VP | S) + log p(Papa | NP) + log p(VP PP | VP) …
- HMM tagging: … + log p(t7 | t5, t6) + log p(w7 | t7) + …
- Noisy channel: [log p(source)] + [log p(data | source)]
- Naïve Bayes:
  log p(Class) + log p(feature1 | Class) + log p(feature2 | Class) …
- *Note:* Today we'll use +logprob not –logprob: i.e., bigger weights are **better**.

Nuthin' but adding weights

- n-grams: … + log p(w7 | w5,w6) + log(w8 | w6, w7) + …
- PCFG: log p(NP VP | S) + log p(Papa | NP) + log p(VP PP | VP) …
  - Can regard any linguistic object as a collection of features (here, tree = a collection of context-free rules)
  - Weight of the object = total weight of features
  - Our weights have always been conditional log-probs (≤ 0)
    - but that is going to change in a few minutes!
- HMM tagging: … + log p(t7 | t5, t6) + log p(w7 | t7) + …
- Noisy channel: [log p(source)] + [log p(data | source)]
- Naïve Bayes:

Probabilists Rally Behind their Paradigm

".2, .4, .6, .8! We're not gonna take your bait!"

1. Can estimate <u>our</u> parameters *automatically*
   - e.g., log p(t7 | t5, t6) (trigram tag probability)
   - from supervised or unsupervised data
2. <u>Our</u> results are more meaningful
   - Can use probabilities to place bets, quantify risk
   - e.g., how sure are we that this is the correct parse?
3. <u>Our</u> results can be meaningfully combined ⇒ modularity!
   - Multiply indep. conditional probs – normalized, unlike scores
   - p(English text) * p(English phonemes | English text) * p(Jap. phonemes | English phonemes) * p(Jap. text | Jap. phonemes)
   - p(semantics) * p(syntax | semantics) * p(morphology | syntax) * p(phonology | morphology) * p(sounds | phonology)

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage
- Consider e.g. Naïve Bayes for text categorization:
  - `Buy this supercalifragilistic Ginsu knife set for only $39 today …`
- Some useful features:
  - Contains `Buy`
  - Contains `supercalifragilistic`
  - Contains a dollar amount under `$100` — spam .5 | ling .02
  - Contains an imperative sentence
  - Reading level = 8^{th} grade
  - Mentions money (use word classes and/or regexp to detect this) — spam .9 | ling .1
- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …
- What assumption does Naïve Bayes make? True here?

Probabilists Regret Being Bound by Principle

- Ad-hoc approach does have one advantage
- Consider e.g. Naïve Bayes for text categorization:
  - `Buy this supercalifragilistic Ginsu knife set for only $39 today …`
- Some useful features:

50% of spam has this – 25x more likely than in ling

  - Contains a dollar amount under `$100` — spam .5 | ling .02

*but **here** are the emails with both features – only 25x!*

90% of spam has this – 9x more likely than in ling

  - Mentions money — spam .9 | ling .1

Naïve Bayes claims .5*.9=45% of spam has **both** features – 25*9=225x more likely than in ling.

- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …
- What assumption does Naïve Bayes make? True here?

Probabilists Regret Being Bound by Principle

- But ad-hoc approach does have one advantage
  - **Can adjust scores to compensate for feature overlap …**
- Some useful features of this message:

spam	ling	Feature	log prob spam	log prob ling	adjusted spam	adjusted ling
.5	.02	Contains a dollar amount under `$100`	-1	-5.6	-.85	-2.3
.9	.1	Mentions money	-.15	-3.3	-.15	-3.3

- Naïve Bayes: pick C maximizing p(C) * p(feat 1 | C) * …
- What assumption does Naïve Bayes make? True here?

Revolution Corrupted by Bourgeois Values

- Naïve Bayes needs overlapping but independent features
- But not clear how to restructure these features like that:
  - Contains `Buy`
  - Contains `supercalifragilistic`
  - Contains a dollar amount under `$100`
  - Contains an imperative sentence
  - Reading level = 7th grade
  - Mentions money (use word classes and/or regexp to detect this)
  - …
- Boy, we'd like to be able to throw all that useful stuff in without worrying about feature overlap/independence.
- Well, maybe we can add up scores and pretend like we got a log probability:

Revolution Corrupted by Bourgeois Values

- Naïve Bayes needs overlapping but independent features
- But not clear how to restructure these features like that:
  - +4 Contains `Buy`
  - +0.2 Contains `supercalifragilistic`
  - +1 Contains a dollar amount under `$100`
  - +2 Contains an imperative sentence
  - -3 Reading level = 7th grade
  - +5 Mentions money (use word classes and/or regexp to detect this)
  - … …

  total: 5.77

- Boy, we'd like to be able to throw all that useful stuff in without worrying about feature overlap/independence.
- Well, maybe we can add up scores and pretend like we got a log probability: **log p(feats | spam) = 5.77**

♣ Oops, then p(feats | spam) = exp 5.77 = 320.5

Renormalize by 1/Z to get a Log-Linear Model

- p(feats | spam) = exp 5.77 = 320.5

  scale down so everything < 1 and sums to 1!

- p(m | spam) = (1/Z(λ)) exp $\sum_i \lambda_i f_i(m)$ where

  m is the email message

  λ_i is weight of feature i

  $f_i(m) \in \{0,1\}$ according to whether m has feature i

  More generally, allow $f_i(m)$ = count or strength of feature.

  1/Z(λ) is a normalizing factor making $\sum_m$ p(m | spam)=1

  (summed over all possible messages m! hard to find!)

- The weights we add up are basically arbitrary.
- They don't have to mean anything, so long as they give us a good probability.
- Why is it called "log-linear"?

Why Bother?

- Gives us probs, not just scores.
  - Can use them to bet, or combine w/ other probs.
- We can now learn weights from data!
  - Choose weights λ_i that maximize logprob of labeled training data = $\log \prod_j p(c_j)\, p(m_j \mid c_j)$
    - where $c_j \in$ {ling,spam} is classification of message m_j
    - and $p(m_j \mid c_j)$ is log-linear model from previous slide
  - Convex function – easy to maximize! (why?)
- **But:** $p(m_j \mid c_j)$ for a given λ requires $Z(\lambda)$: hard!

Attempt to Cancel out Z

- Set weights to maximize $\prod_j p(c_j)\, p(m_j \mid c_j)$
  - where $p(m \mid \text{spam}) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m)$
  - **But** normalizer $Z(\lambda)$ is awful sum over all possible emails
- So instead: Maximize $\prod_j p(c_j \mid m_j)$
  - Doesn't model the emails m_j, only their classifications c_j
  - Makes more sense anyway given our feature set

- p(spam | m) = p(spam)p(m|spam) / (p(spam)p(m|spam)+p(ling)p(m|ling))
- Z appears in both numerator and denominator
- Alas, doesn't cancel out because Z differs for the spam and ling models
- But we can fix this …

So: Modify Setup a Bit

- Instead of having separate models

  p(m|spam)*p(spam) vs. p(m|ling)*p(ling)
- Have just one joint model p(m,c)

  gives us both p(m,spam) and p(m,ling)
- Equivalent to changing feature set to:
  - spam ⇓ weight of this feature is log p(spam) + a constant
  - spam and Contains `Buy` ⇓old spam model's weight for "contains Buy"
  - spam and Contains `supercalifragilistic`
  - …
  - ling ⇓ weight of this feature is log p(ling) + a constant
  - ling and Contains `Buy` ⇓old ling model's weight for "contains Buy"
  - ling and Contains `supercalifragilistic`
- No real change, but 2 categories now share single feature set and single value of $Z(\lambda)$

Now we can cancel out Z

Now $p(m,c) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m,c)$ where $c \in \{ling, spam\}$

- Old: choose weights λ_i that maximize prob of labeled training data = $\prod_j p(m_j, c_j)$
- New: choose weights λ_i that maximize prob of labels given messages = $\prod_j p(c_j \mid m_j)$

- Now Z cancels out of conditional probability!
  - p(spam | m) = p(m,spam) / (p(m,spam) + p(m,ling))

    $= \exp \sum_i \lambda_i f_i(m,spam) / (\exp \sum_i \lambda_i f_i(m,spam) + \exp \sum_i \lambda_i f_i(m,ling))$
  - Easy to compute now …
  - $\prod_j p(c_j \mid m_j)$ is still convex, so easy to maximize too

Maximum Entropy

- Suppose there are 10 classes, A through J.
- I don't give you any other information.
- Question: Given message m: what is your guess for p(C I m)?

- Suppose I tell you that 55% of all messages are in class A.
- Question: Now what is your guess for p(C I m)?

- Suppose I also tell you that 10% of all messages contain `Buy` and 80% of these are in class A or C.
- Question: Now what is your guess for p(C I m), if m contains `Buy`?
- OUCH!

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	.051	.0025	.029	.0025	.0025	.0025	.0025	.0025	.0025	.0025
Other	.499	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446

- Column A sums to 0.55 ("55% of all messages are in class A")

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	.051	.0025	.029	.0025	.0025	.0025	.0025	.0025	.0025	.0025
Other	.499	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446

- Column A sums to 0.55
- Row `Buy` sums to 0.1 (“10% of all messages contain `Buy`”)

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	.051	.0025	.029	.0025	.0025	.0025	.0025	.0025	.0025	.0025
Other	.499	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446

- Column A sums to 0.55
- Row `Buy` sums to 0.1
- (`Buy`, A) and (`Buy`, C) cells sum to 0.08 (“80% of the 10%”)

♣ Given these constraints, fill in cells “as equally as possible”: maximize the entropy (related to cross-entropy, perplexity)

Entropy = -.051 log .051 - .0025 log .0025 - .029 log .029 - …

Largest if probabilities are evenly distributed

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	.051	.0025	.029	.0025	.0025	.0025	.0025	.0025	.0025	.0025
Other	.499	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446

- Column A sums to 0.55
- Row `Buy` sums to 0.1
- (`Buy`, A) and (`Buy`, C) cells sum to 0.08 (“80% of the 10%”)

- Given these constraints, fill in cells “as equally as possible”: maximize the entropy
- Now p(`Buy`, C) = .029 and p(C | `Buy`) = .29
- We got a compromise: p(C | `Buy`) < p(A | `Buy`) < .55

Generalizing to More Features

<$100

Other

	A	B	C	D	E	F	G	H	…
Buy	.051	.0025	.029	.0025	.0025	.0025	.0025	.0025	
Other	.499	.0446	.0446	.0446	.0446	.0446	.0446	.0446	

What we just did

- For each feature (“contains Buy”), see what fraction of training data has it
- Many distributions p(c,m) would predict these fractions (including the unsmoothed one where all mass goes to feature combos we’ve actually seen)
- Of these, pick distribution that has max entropy
- Amazing Theorem: This distribution has the form $p(m,c) = (1/Z(\lambda)) \exp \sum_i \lambda_i f_i(m,c)$
  - So it is log-linear. In fact it is the same log-linear distribution that maximizes $\prod_j p(m_j, c_j)$ as before!
- Gives another motivation for our log-linear approach.

Log-linear form derivation

- Say we are given some ***constraints*** in the form of feature expectations:

$$\sum_x p(x) f_i(x) = \alpha_i$$

- In general, there may be many distributions p(x) that satisfy the constraints. Which one to pick?
- The one with maximum entropy (making fewest possible additional assumptions---Occum’s Razor)
- This yields an optimization problem

$$\max H(p(x)) = -\sum_x p(x) \log p(x)$$

$$\text{Subject to } \sum_x p(x) f_i(x) = \alpha_i, \forall i \text{ and } \sum_x p(x) = 1$$

Log-linear form derivation

- To solve the maxent problem, we use Lagrange multipliers:

$$L = -\sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x}) - \sum_i \theta_i \left(\sum_{\mathbf{x}} p(\mathbf{x}) f_i(\mathbf{x}) - \alpha_i \right) - \mu \left(\sum_{\mathbf{x}} p(\mathbf{x}) - 1 \right)$$

$$\frac{\partial L}{\partial p(\mathbf{x})} = 1 + \log p(\mathbf{x}) - \sum_i \theta_i f_i(\mathbf{x}) - \mu$$

$$p^*(\mathbf{x}) = e^{\mu - 1} \exp \left\{ \sum_i \theta_i f_i(\mathbf{x}) \right\}$$

$$Z(\theta) = e^{1-\mu} = \sum_{\mathbf{x}} \exp \left\{ \sum_i \theta_i f_i(\mathbf{x}) \right\}$$

$$p(\mathbf{x}|\theta) = \frac{1}{Z(\theta)} \exp \left\{ \sum_i \theta_i f_i(\mathbf{x}) \right\}$$

- So feature constraints + maxent implies exponential family.
- Problem is convex, so solution is unique.

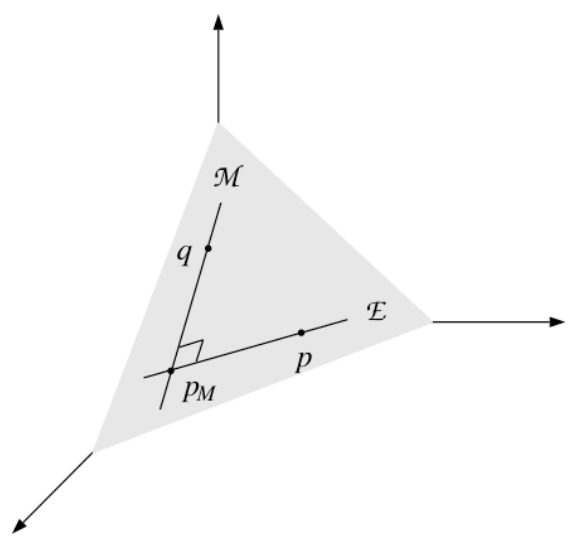
MaxEnt = Max Likelihood

Define two submanifolds on the probability simplex $p(\mathbf{x})$.

The first is $\mathcal{E}$, the set of all exponential family distributions based on a particular set of features $f_i(\mathbf{x})$.

The second is $\mathcal{M}$, the set of all distributions that satisfy the feature expectation constraints.

They intersect at a single distribution p_M, the maxent, maximum likelihood



$$
\begin{aligned}
\ell(\theta; \mathcal{D}) &= \sum_{\mathbf{x}} n(\mathbf{x}) \log p(\mathbf{x}|\theta) \\
&= \sum_{\mathbf{x}} n(\mathbf{x}) \left(\sum_i \theta_i f_i(\mathbf{x}) - \log Z(\theta) \right) \\
&= \sum_{\mathbf{x}} n(\mathbf{x}) \sum_i \theta_i f_i(\mathbf{x}) - N \log Z(\theta) \\
\frac{\partial \ell}{\partial \theta_i} &= \sum_{\mathbf{x}} n(\mathbf{x}) f_i(\mathbf{x}) - N \frac{\partial}{\partial \theta_i} \log Z(\theta) \\
&= \sum_{\mathbf{x}} n(\mathbf{x}) f_i(\mathbf{x}) - N \sum_{\mathbf{x}} p(\mathbf{x}|\theta) f_i(\mathbf{x}) \\
\Rightarrow \quad \sum_{\mathbf{x}} p(\mathbf{x}|\theta) f_i(\mathbf{x}) &= \sum_{\mathbf{x}} \frac{n(\mathbf{x})}{N} f_i(\mathbf{x}) = \sum_{\mathbf{x}} \bar{p}(\mathbf{x}) f_i(\mathbf{x})
\end{aligned}
$$

Derivative of log partition function is the expectation of the feature.

At ML estimate, model expectations match empirical feature counts.

Recipe for a Conditional MaxEnt Classifier

1. Gather *constraints* from training data:
   $$\alpha_{iy} = \tilde{E}[f_{iy}] = \sum_{x_j, y_j \in D} f_{iy}(x_j, y_j)$$
2. Initialize all parameters to zero.
3. Classify training data with current parameters. Calculate *expectations*. $$E_\Theta[f_{iy}] = \sum_{x_j \in D} \sum_{y'} p_\Theta(y'|x_j) f_{iy}(x_j, y')$$
4. Gradient is $\tilde{E}[f_{iy}] - E_\Theta[f_{iy}]$
5. Take a step in the direction of the gradient
6. Until convergence, return to step 3.

Overfitting

- If we have too many features, we can choose weights to model the training data perfectly.

- If we have a feature that only appears in spam training, not ling training, it will get weight ∞ to maximize p(spam | feature) at 1.

- These behaviors overfit the training data.
- Will probably do poorly on test data.

Solutions to Overfitting

1. Throw out rare features.
   - Require every feature to occur > 4 times, and > 0 times with ling, and > 0 times with spam.
2. Only keep 1000 features.
   - Add one at a time, always greedily picking the one that most improves performance on held-out data.
3. Smooth the observed feature counts.
4. Smooth the weights by using a prior.
   - max $p(\lambda|data)$ = max $p(\lambda, data) = p(\lambda)p(data|\lambda)$
   - decree $p(\lambda)$ to be high when most weights close to 0