A gentle introduction to Maximum Entropy Models and their friends

Mark Johnson
Brown University

November 2007

Outline

Optimality theory analyses

- *Markedness constraints*
  - ONSET: Violated each time a syllable begins without an onset
  - PEAK: Violated each time a syllable doesn't have a peak V
  - NOCODA: Violated each time a syllable has a non-empty coda
  - *COMPLEX: Violated each time a syllable has a complex onset or coda
- *Faithfulness constraints*
  - FAITHV: Violated each time a V is inserted or deleted
  - FAITHC: Violated each time a C is inserted or deleted

	/ʔilk-hin/	PEAK	*COMPLEX	FAITHC	FAITHV	NOCODA
	ʔil.khin		*!			**
	ʔil.k.hin	*!				**
☞	ʔi.lik.hin				*	**
	ʔik.hin			*!		**

Optimal surface forms with strict domination

- OT constraints are *functions* f from (underlying form, surface form) pairs to *non-negative integers*
  - Example: FAITHC(/ʔilkhin/, [ʔik.hin]) = 1
- If $\boldsymbol{f} = (f_1, \ldots, f_m)$ is a *vector of constraints* and $x = (u, v)$ is a pair of an underlying form u and a surface form v, then $\boldsymbol{f}(x) = (f_1(x), \ldots, f_m(x))$
  - Ex: if $\boldsymbol{f} =$ (PEAK, *COMPLEX, FAITHC, FAITHV, NOCODA), then $\boldsymbol{f}$(/ʔilkhin/, [ʔik.hin]) $= (0, 0, 1, 0, 2)$
- If C is a (possibly infinite) set of (underlying form,candidate surface forms) pairs then:

$$x \in C \text{ is } \textit{optimal} \text{ in } C \quad \Leftrightarrow \quad \forall c \in C,\ \boldsymbol{f}(x) \leq \boldsymbol{f}(c)$$

  where $\leq$ is the standard (lexicographic) on vectors
- Generally all of the pairs in C have the same underlying form
- **Note:** the linguistic properties of a constraint f *don't matter* once we know $f(c)$ for each $c \in C$.

Optimality with linear constraint weights

- Each constraint f_k has a corresponding *weight* w_k
  - Ex: If $\boldsymbol{f} = (\textsc{Peak}, {}^{\star}\textsc{Complex}, \textsc{FaithC}, \textsc{FaithV}, \textsc{NoCoda})$, then $\boldsymbol{w} = (-2, -2, -2, -1, 0)$
- The *score* $s_{\boldsymbol{w}}(x)$ for an (underlying, surface form) pair x is:

$$s_{\boldsymbol{w}}(x) \;=\; \boldsymbol{w} \cdot \boldsymbol{f}(x) \;=\; \sum_{j=1}^{m} w_j \, f_j(x)$$

  - Ex: $\boldsymbol{f}(/\text{ʔilkhin}/, [\text{ʔik.hin}]) = (0, 0, 1, 0, 2)$, so $s_{\boldsymbol{w}}(/\text{ʔilkhin}/, [\text{ʔik.hin}]) = -2$
  - Called "linear" because the score is a linear function of the constraint values
- The *optimal candidate* is the one with the highest score

$$\text{Opt}(C) \;=\; \underset{x \in C}{\text{argmax}}\; s_{\boldsymbol{w}}(x)$$

- Again, all that matters are $\boldsymbol{w}$ and $\boldsymbol{f}(c)$ for $c \in C$

Constraint weight learning example

- All we need to know about the (underlying form,surface form) candidates x are their constraint vectors $\boldsymbol{f}(x)$

Winner x_i	**Losers** $C_i \setminus \{x_i\}$		
$(0,0,0,1,2)$	$(0,1,0,0,2)$	$(1,0,0,0,2)$	$(0,0,1,0,2)$
$(0,0,0,0,2)$	$(0,0,0,2,0)$	$(1,0,0,0,1)$	
$\ldots$	$\ldots$		

- The weight vector $\boldsymbol{w} = (-2,-2,-2,-1,0)$ correctly classifies this data
- Supervised learning problem: given data, find a weight vector $\boldsymbol{w}$ that correctly classifies every example in data

Supervised learning of constraint weights

- The training data is a vector D of pairs (C_i, x_i) where
  - C_i is a (possibly infinite) set of candidates
  - $x_i \in C_i$ is the correct realization from C_i

  (can be generalized to permit multiple winners)
- Given data D and a constraint vector $\boldsymbol{f}$, *find a weight vector* $\boldsymbol{w}$ *that makes each* x_i *optimal for* C_i
- "Supervised" because underlying form is given in D
  - Unsupervised problem: underlying form is not given in D (*blind source separation*, *clustering*)
- The weight vector $\boldsymbol{w}$ may not exist.
  - If $\boldsymbol{w}$ exists then D is *linearly separable*
- We may want $\boldsymbol{w}$ to *correctly generalize* to examples not in D
- We may want $\boldsymbol{w}$ to be *robust to noise or errors* in D

$\Rightarrow$ *Probabilistic models of learning*

Aside: The OT supervised learning problem is often trivial

- There are typically tens of thousands of different underlying forms in a language
- But all the learner sees are the vectors $\boldsymbol{f}(c)$
- Many OT-inspired problems present very few different $\boldsymbol{f}(x)$ vectors ...
- so the correct surface forms can be identified by *memorizing* the $\boldsymbol{f}(x)$ vectors for all winners x

$\Rightarrow$ *generalization is often not necessary to identify optimal surface forms*

  - too many $\boldsymbol{f}(x)$ vectors to memorize if $\boldsymbol{f}$ contained all universally possible constraints?
  - maybe the supervised learning problem is unrealistically easy, and we should be working on unsupervised learning?

The probabilistic setting

- View training data D as a *random sample* from a (possibly much larger) "true" distribution $\mathrm{P}(x|C)$ over (C, x) triples
- Try to pick $\boldsymbol{w}$ so we do well on average over all (C, x)
- *Support Vector Machines* set $\boldsymbol{w}$ to maximize $\mathrm{P}(\mathrm{Opt}(C) = x)$, i.e., the probability that the optimal candidate is in fact correct
  - Although SVMs try to maximize the probability that the optimal candidate is correct, SVMs are not probabilistic models
- *Maximum Entropy models* set $\boldsymbol{w}$ to approximate $\mathrm{P}(x|C)$ as closely as possible with an exponential model, *or equivalently*
- find the probability distribution $\widehat{\mathrm{P}}(x|C)$ with *maximum entropy* such that $\mathrm{E}_{\widehat{\mathrm{P}}}[f_j|C] = \mathrm{E}_{\mathrm{P}}[f_j|C]$

Outline

Terminology, or Snow's "Two worlds"

Warning: Linguists and statisticians use same words to mean different things!

- *feature*
  - In linguistics, e.g., "voiced" is a function from phones to $+, -$
  - In statistics, what linguists call constraints (a function from candidates/outcomes to real numbers)
- *constraint*
  - In linguistics, what the statisticians call "features"
  - In statistics, a property that the estimated model $\widehat{\mathrm{P}}$ must have
- *outcome*
  - In statistics, the set of objects we're defining a probability distribution over (the set of all candidate surface forms)

Why are they *Maximum Entropy* models?

- Goal: learn a probability distribution $\widehat{\mathrm{P}}$ as close as possible to distribution P that generated training data D.
- But what does "as close as possible" mean?
  - Require $\widehat{\mathrm{P}}$ to have *same distribution of features* as D
  - As size of data $|D| \rightarrow \infty$, feature distribution in D will approach feature distribution in P
  - so distribution of features in $\widehat{\mathrm{P}}$ will approach distribution of features in P
- But there are many $\widehat{\mathrm{P}}$ that have same feature distributions as D. Which one should we choose?
  - The *entropy* measures the *amount of information* in a distribution
  - Higher entropy $\Rightarrow$ less information
  - Choose the $\widehat{\mathrm{P}}$ with *maximum entropy* that whose feature distributions agree with D
    $\Rightarrow \widehat{\mathrm{P}}$ has the least extraneous information possible

Maximum Entropy models

- A *conditional Maximum Entropy model* $\mathrm{P}_{\boldsymbol{w}}$ consists of a vector of features $\boldsymbol{f}$ and a vector of feature weights $\boldsymbol{w}$.
- The probability $\mathrm{P}_{\boldsymbol{w}}(x|C)$ of an outcome $x \in C$ is:

$$\begin{aligned}
\mathrm{P}_{\boldsymbol{w}}(x|C) &= \frac{1}{Z_{\boldsymbol{w}}(C)} \exp\left(s_{\boldsymbol{w}}(x)\right) \\
&= \frac{1}{Z_{\boldsymbol{w}}(C)} \exp\left(\sum_{j=1}^{m} w_j f_j(x)\right), \text{ where:} \\
Z_{\boldsymbol{w}}(C) &= \sum_{x' \in C} \exp\left(s_{\boldsymbol{w}}(x')\right)
\end{aligned}$$

- $Z_{\boldsymbol{w}}(C)$ is a normalization constant called the *partition function*

Feature dependence ⇒ MaxEnt models

- Many probabilistic models *assume that features are independently distributed* (e.g., Hidden Markov Models, Probabilistic Context-Free Grammars)

⇒ Estimating feature weights is simple (relative frequency)

- But features in most linguistic theories interact in complex ways
  - Long-distance and local dependencies in syntax
  - Many markedness and faithfulness constraints interact to determine a single syllable's shape

⇒ *These features are not independently distributed*

- MaxEnt models can handle these feature interactions
- Estimating feature weights of MaxEnt models is more complicated
  - generally requires numerical optimization

A rose by any other name …

- Like most other good ideas, Maximum Entropy models have been invented many times …
  - In statistical mechanics (physics) as the *Gibbs* and *Boltzmann distributions*
  - In probability theory, as *Maximum Entropy models*, *log-linear models*, *Markov Random Fields* and *exponential families*
  - In statistics, as *logistic regression*
  - In neural networks, as *Boltzmann machines*

A brief history of MaxEnt models in Linguistics

- Logistic regression used in socio-linguistics to model "variable rules" (Sedergren and Sankoff 1974)
- Hinton and Sejnowski (1986) and Smolensky (1986) introduce the Boltzmann machine for neural networks
- Berger, Dell Pietra and Della Pietra (1996) propose Maximum Entropy Models for *language models with non-independent features*
- Abney (1997) proposes MaxEnt models for *probabilistic syntactic grammars with non-independent features*
- (Johnson, Geman, Canon, Chi and Riezler (1999) propose conditional estimation of regularized MaxEnt models)

Outline

Finding the MaxEnt model by maximizing likelihood

- Can prove that the MaxEnt model $\mathrm{P}_{\widehat{w}}$ for features $\boldsymbol{f}$ and data $D = ((C_1, x_1), \ldots, (C_n, x_n))$ is:

$$\mathrm{P}_{\widehat{w}}(x \mid C) \;=\; \frac{1}{Z_{\widehat{w}}(C)} \exp(\, s_{\widehat{w}}(x)\,) \;=\; \frac{1}{Z_{\widehat{w}}(C)} \exp \sum_{j=1}^{m} \widehat{w}_j \, f_j(x)$$

  where $\widehat{\boldsymbol{w}}$ *maximizes the likelihood* $\mathrm{L}_D(\boldsymbol{w})$ *of the data* D

$$\widehat{\boldsymbol{w}} \;=\; \operatorname*{argmax}_{\boldsymbol{w}} \mathrm{L}_D(\boldsymbol{w}) \;=\; \operatorname*{argmax}_{\boldsymbol{w}} \prod_{i=1}^{n} \mathrm{P}_{\boldsymbol{w}}(x_i \mid C_i)$$

  I.e., choose $\widehat{\boldsymbol{w}}$ to make the winners x_i *as likely as possible* compared to losers $C_i \setminus \{x_i\}$

Finding the feature weights $\hat{w}$

- Standard method: use a *gradient-based* numerical optimizer to *minimize the negative log likelihood* $-\log \mathrm{L}_D(\boldsymbol{w})$
  (Limited memory variable metric optimizers seem to be best)

$$
\begin{aligned}
-\log \mathrm{L}_D(\boldsymbol{w}) &= \sum_{i=1}^{n} -\log \mathrm{P}_{\boldsymbol{w}}(x_i \mid C_i) \\
&= \sum_{i=1}^{n} \left(\log Z_{\boldsymbol{w}}(C_i) - \sum_{j=1}^{m} w_j f_j(x_i) \right) \\
\frac{\partial \ -\log \mathrm{L}_D(\boldsymbol{w})}{\partial w_j} &= \sum_{j=1}^{n} \left(\mathrm{E}_{\boldsymbol{w}}[f_j | C_i] - f_j(x_i) \right), \text{ where:} \\
\mathrm{E}_{\boldsymbol{w}}[f_j | C_i] &= \sum_{x' \in C_i} f_j(x') \, \mathrm{P}_{\boldsymbol{w}}(x')
\end{aligned}
$$

- I.e., find feature weights $\hat{w}$ that make the *model's distribution of features over C_i equal distribution of features in winners x_i*

Finding the optimal feature weights $\hat{w}$

- Numerically optimizing likelihood involves calculating $-\log \mathrm{L}_D(\boldsymbol{w})$ and its derivatives
- Need to calculate $Z_{\boldsymbol{w}}(C_i)$ and $\mathrm{E}_{\boldsymbol{w}}[f_j|C_i]$, which are sums over C_i, the set of candidates for example i
- If C_i can be infinite:
  - depending on $\boldsymbol{f}$ and C, might be possible to *explicitly calculate* $Z_{\boldsymbol{w}}(C_i)$ and $\mathrm{E}_{\boldsymbol{w}}[f_j|C_i]$, or
  - may be able to *approximate* $Z_{\boldsymbol{w}}(C_i)$ and $\mathrm{E}_{\boldsymbol{w}}[f_j|C_i]$, especially if $\mathrm{P}_{\boldsymbol{w}}(x|C)$ is concentrated on few x.
- Aside: using MaxEnt for *unsupervised learning* requires $Z_{\boldsymbol{w}}$ and $\mathrm{E}_{\boldsymbol{w}}[f_j]$, but these are typically *hard to compute*
- If feature weights w_j should be negative (e.g., OT constraint violations can only "hurt" a candidate), then replace optimizer with a *numerical optimizer/constraint solver* (e.g., TAO package from Argonne labs)

Outline

Why regularize?

- MaxEnt selects $\widehat{w}$ so that winners are as likely as possible
- Might not want to do this with *noisy training data*
- *Pseudo-maximal or minimal* features cause numerical problems
  - A feature f_j is *pseudo-minimal* iff for all $i = 1, \ldots, n$ and $x' \in C_i$, $f_j(x_i) \leq f(x')$ (i.e., $f_j(x_i)$ is the minimum value f_j has in C_i)
  - If f_j is *pseudo-minimal*, then $\widehat{w}_j = -\infty$
- Example: Features 1, 2 and 3 are pseudo-minimal below:

Winner x_i	**Losers** $C_i \setminus \{x_i\}$		
$(0,0,0,1,2)$	$(0,1,0,0,2)$	$(1,0,0,0,2)$	$(0,0,1,0,2)$
$(0,0,0,0,2)$	$(0,0,0,2,0)$	$(1,0,0,0,1)$	
$\ldots$	$\ldots$		

so we can make (some of) the losers have arbitrarily low probability by setting the corresponding feature weights as negative as possible

Regularization, or "keep it simple"

- Slavishly optimizing likelihood leads to over-fitting or numerical problems

$\Rightarrow$ Regularize or smooth, i.e., try to find a "good" $\widehat{\boldsymbol{w}}$ that is "not too complex"

- Minimize the *penalized negative log likelihood*

$$\widehat{\boldsymbol{w}} \quad = \quad \underset{\boldsymbol{w}}{\operatorname{argmin}} \quad -\log \mathrm{L}_D(\boldsymbol{w}) + \alpha \sum_{j=1}^{m} |w_j|^k$$

where $\alpha \geq 0$ is a parameter (often set by *cross-validation on held-out training data*) controlling amount of regularization

Aside: Regularizers as Bayesian priors

- Bayes inversion formula

$$\underbrace{\mathrm{P}(\boldsymbol{w} \mid D)}_{\text{posterior}} \propto \underbrace{\mathrm{P}(D \mid \boldsymbol{w})}_{\text{likelihood}} \underbrace{\mathrm{P}(\boldsymbol{w})}_{\text{prior}}$$

  or in terms of log probabilities:

$$-\log \mathrm{P}(\boldsymbol{w} \mid D) = \underbrace{-\log \mathrm{P}(D \mid \boldsymbol{w})}_{\text{– log likelihood}} \underbrace{-\log \mathrm{P}(\boldsymbol{w})}_{\text{– log prior}} + c$$

$\Rightarrow$ The regularized estimate $\widehat{\boldsymbol{w}}$ is also the Bayesian *maximum a posteriori* (MAP) estimate with prior

$$\mathrm{P}(\boldsymbol{w}) \propto \exp\left(-\alpha \sum_{j=1}^{m} |w_j|^k\right)$$

- When $k = 2$ this is a *Gaussian prior*

Understanding the effects of the priors

- The log penalty term for a *Gaussian prior* ($k = 2$) is $\alpha \sum_j w_j^2$ so its derivative $2\alpha w_j \to 0$ as $w_j \to 0$
- Effect of Gaussian prior decreases as w_j is small

$\Rightarrow$ Gaussian prior prefers all w_j to be small but not necessarily zero

- The log penalty term for a *1-norm prior* ($k = 1$) is $\alpha \sum_j |w_j|$ so its derivative $\alpha \text{sign}(w_j)$ is α or $-\alpha$ unless $w_j = 0$
- Effect of 1-norm prior is constant no matter how small w_j is

$\Rightarrow$ 1-norm prior prefers most w_j to be zero (*sparse solutions*)

- My personal view: If most features in your problem are irrelevant, prefer a sparse feature vector.
  But if most features are noisy and weakly correlated with the solution, prefer a dense feature vector (averaging is the solution to noise).

Case study: MaxEnt in syntactic parsing

- MaxEnt model used to *pick correct parse from 50 parses* produced by Charniak parser
  - C_i is set of 50 parses from Charniak parser, x_i is best parse in C_i
  - Charniak parser's accuracy ≈ 0.898 (picking tree it likes best)
  - Oracle accuracy is ≈ 0.968
  - EM-like method for dealing with ties (training data C_i contains several equally good "best parses" for a sentence i)
- MaxEnt model uses 1,219,273 features, encoding a wide variety of syntactic information
  - including the Charniak model's *log probability* of the tree
  - trained on parse trees for 36,000 sentences
  - prior weight α set by *cross-validation* (don't need to be accurate)
- Gaussian prior results in all feature weights non-zero
- L1 prior results in $\approx 25,000$ non-zero feature weights
- Accuracy with both Gaussian and L1 priors ≈ 0.916 (Andrew and Gao, *ICML 2007*)

Outline

Stochastic gradient ascent

- MaxEnt: choose $\widehat{w}$ to maximize log likelihood
- If $w \neq \widehat{w}$ and δ is sufficiently small, then

$$\log \mathrm{L}_D \left(w + \delta \frac{\partial \log \mathrm{L}_D(w)}{\partial w} \right) > \log \mathrm{L}_D(w)$$

  i.e., small steps in direction of derivative increase likelihood

$$\frac{\partial \log \mathrm{L}_D(w)}{\partial w_j} = \sum_{j=1}^{n} \left(f_j(x_i) - \mathrm{E}_w[f_j \mid C_i] \right), \text{ where:}$$

$$\mathrm{E}_w[f_j|C_i] = \sum_{x' \in C_i} f_j(x') \, \mathrm{P}_w(x')$$

- *Gradient ascent* optimizes the log likelihood in this manner.
  - It is usually *not* an efficient optimization method
- *Stochastic gradient ascent* updates immediately in direction of contribution of training example i to derivative
  - It is a simple and sometimes very efficient method

Perceptron updates as a MaxEnt approx

- Perceptron learning rule: Let $x_i^\star$ be the model's current prediction of the optimal candidate in C_i

$$x_i^\star \quad = \quad \underset{x' \in C_i}{\operatorname{argmax}}\ s_{\boldsymbol{w}}(x')$$

  If $x_i^\star \neq x_i$, where x_i is the correct candidate in C_i, then increment the current weights $\boldsymbol{w}$ with:

$$\delta \left(\boldsymbol{f}(x_i) - \boldsymbol{f}(x_i^\star) \right)$$

- MaxEnt stochastic gradient ascent update:

$$\delta \, \frac{\partial \, \log \mathrm{L}_D(\boldsymbol{w})}{\partial \, \boldsymbol{w}} \quad = \quad \delta \left(\boldsymbol{f}(x_i) - \mathrm{E}_{\boldsymbol{w}}[\boldsymbol{f} \mid C_i] \right)$$

  If $\mathrm{P}_{\boldsymbol{w}}(x \mid C_i)$ is peaked around $x_i^\star$, then $\mathrm{E}_{\boldsymbol{w}}[\boldsymbol{f} \mid C_i] \approx \boldsymbol{f}(x_i^\star)$

$\Rightarrow$ The Perceptron rule approximates the MaxEnt stochastic gradient ascent update

Regularization as weight decay

- When we approximate regularized MaxEnt as either Stochastic Gradient Ascent or the Perceptron update, *regularization corresponds to weight decay* (a popular smoothing method for neural networks)

- Contribution of *Gaussian prior* to log likelihood is $-\alpha \sum_j w_j^2$ so derivative of regularizer is $-2\alpha w_j$

$\Rightarrow$ weights *decay proportionally to their current value* each iteration

- Contribution of *1-norm prior* to log likelihood is $-\alpha \sum_j |w_j|$ so derivative of regularizer is $-\alpha \ \text{sign}(w_j)$

$\Rightarrow$ non-zero weights *decay by a constant amount* each iteration

Outline

Summary

- Phonological problems, once expressed in Optimality Theory, can often also be viewed as statistical problems
- Because the OT features (OT constraints) aren't independent, MaxEnt (and SVMs?) are natural ways of modeling these problems
- MaxEnt (and SVM) models are particularly suited to *supervised learning* problems (which may not be realistic in phonology)
- Regularization controls over-learning, and by choosing an appropriate prior we can prefer sparse solutions (a.k.a. *feature selection*)
- MaxEnt is closely related to other popular learning algorithms such as Stochastic Gradient Ascent and the Perceptron