CMPT 825: Natural Language Processing **Spring 2008**

Lecture 9 — Mar 17-26, 2008

Lecturer: Anoop Sarkar *Scribe: Chris Nell*

9.1 Introduction to Discriminative Models

For much of the course, we have been concerned with the problem of predicting the most likely associated "hidden" label sequence $\hat{\mathbf{y}}$ given an observed random variable sequence $\mathbf{x}$, where each $y_i \in \mathbf{y}$ is drawn from a set of labels Y, via the equation:

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\text{argmax}} P(\mathbf{y}|\mathbf{x}) \tag{9.1}$$

Up to this point, we have focused on so-called *generative models*, of which Hidden Markov Models (HMMs) are examples. In these models, we first use Bayes' rule to decompose $P(\mathbf{y}|\mathbf{x})$:

$$P(\mathbf{y}|\mathbf{x}) = \frac{P(\mathbf{x}|\mathbf{y})P(\mathbf{y})}{P(\mathbf{x})} \propto P(\mathbf{x}|\mathbf{y})P(\mathbf{y}) \tag{9.2}$$

In order to determine $\hat{\mathbf{y}}$ using Equation 9.1, we simply generate all possible candidate label sequences $\mathbf{y}$, and evaluate them using this decomposition. An advantage of this method is that for many model choices, the EM algorithm can be efficiently applied to learn appropriate values for $P(\mathbf{x}|\mathbf{y})$ and $P(\mathbf{y})$ from unlabeled training data[1], while it is not as easy to do so for $P(\mathbf{y}|\mathbf{x})$ directly. This makes generative models especially suited for situations in which unsupervised learning is desirable.

Of course, generative models are also capable of supervised learning using labelled training data. While this is very easy to implement ($P(\mathbf{x}|\mathbf{y})$ and $P(\mathbf{y})$ are simply set according to relative frequency values observed in the training data), it is not clear that the generative framework itself is the most natural choice for performing supervised learning. In particular, generating and evaluating every possible candidate sequence Y via $P(\mathbf{x}|\mathbf{y})P(\mathbf{y})$ seems an indirect way to compute our target quantity $P(\mathbf{y}|\mathbf{x})$. Instead, we would

[1]For example, the Baum-Welch (Forward-Backward) algorithm is a specialization of the EM algorithm for HMMs.

prefer a method which allows us to find $P(\mathbf{y}|\mathbf{x})$ directly, by minimizing some error function over the labelled training data.

In the remainder of this lecture, we introduce a class of *discriminative models* which are designed to enable direct evaluation of $P(\mathbf{y}|\mathbf{x})$. In particular, we will study *log-linear models*, one of the most fundamental examples of discriminative model.

9.1.1 Probabilities and Scoring

Consider the joint distribution $P(\mathbf{x}, \mathbf{y})$, where $\mathbf{x}$ is an input sequence, and $\mathbf{y}$ the corresponding label sequence. Our task, is to choose the distribution $P(\mathbf{x}, \mathbf{y})$ which best fits our labelled training data. In other words, we want to maximize the data likelihood (minimize entropy) via particular choice of model θ:

$$\max_{\theta} P(\theta|data) = \max_{\theta} P(\theta)P(data|\theta) \tag{9.3}$$

Readers familiar with search in general artificial intelligence contexts may wonder why we restrict ourselves to using probabilities in Equation 9.3, as opposed to heuristic scores. In fact, we can show that a probabilistic interpretation is not incompatible with heuristic scoring; if instead of "straight" probabilities, we work in terms of log-probabilities, we get:

$$\max_{\theta} \log P(\theta|data) = \max_{\theta} \left[\log P(\theta) + \log P(data|\theta)\right] \tag{9.4}$$

In Equation 9.4, our probabilistic terms act much like heuristic scores (they combine additively) while retaining benefits of probabilities (they can be automatically estimated from training data, whereas heuristic scores must be arbitrarily set). From this, we can see that using a probabilistic interpretation lets easily decompose a conceptually complex scoring for a model θ into a combination of more easily-calculated scores. Further, if we exponentiate these scores, we can treat them as probabilities (so long as they are appropriately normalized).

9.2 Log-Linear Models

In motivating our first discriminative model, we start from a familiar generative model for a slightly simpler problem. Consider the Naive Bayes classifier, which makes strong conditional independence assumptions to simplify

calculation of $P(y|\mathbf{x})$, the likelihood of class label y given a data sequence $\mathbf{x} = \langle x_1, x_2, ... \rangle$:

$$\begin{aligned} P(y|\mathbf{x}) \propto P(y)P(\mathbf{x}|y) &= P(y)P(\mathbf{x}|y) \\ &= P(y)P(x_1|y)P(x_2|x_1, y)P(x_3|x_2, x_1, y)... \\ &\approx P(y)P(x_1|y)P(x_2|y)P(x_3|y)... \\ &= P(y)\prod_{i=1}^{|\mathbf{x}|} P(x_i|y) \end{aligned} \tag{9.5}$$

We would like to design an analogous model in which we can incorporate information other than just the words in $\mathbf{x}$. Note that in the Naive Bayes framework, if we include features which overlap (for example, if x_j was the j^{th} word and x_{2j} was the bigram $\langle x_j, x_{j+1} \rangle$), we end up underestimating the probability that these overlapping features co-occur; Equation 9.5 explicitly assumes all features are independent. Since we would like to maintain a probabilistic interpretation, we need to change our model. To this end, we introduce:

- m features, collectively defined by a feature function $\mathbf{f}(\mathbf{x}, y)$, where $f_k(\mathbf{x}, y)$ corresponds to the k^{th} feature. We assume for this discussion that $\mathbf{f} \in \{0, 1\}^m$; in general it is easy to relax assumption this to allow positive integers, but including negative-valued or continuous-valued features is more complex.

- A parameter vector $\mathbf{w} \in \mathbb{R}^m$ of *weights*, where weight w_k is associated with feature f_k.

- A score for tag y given an observation sequence $\mathbf{x}$, $s(\mathbf{x}, y)$:

$$s(\mathbf{x}, y) = \sum_{k=1}^{m} w_k f_k(\mathbf{x}, y) = \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y) \tag{9.6}$$

Note that $\mathbf{x}$ can now contain not only the observed word sequence, but also any other relevant local context (such as labels of previous words). For example, if we are considering the word "sing" in the POS-tagging task, knowing it follows a word tagged DT (determiner) might affect the chance that it should be tagged"VB" (verb). In a bigram HMM, this dependency is automatically modelled; in our discriminative framework we must include a

representative feature. In order to accomplish this, we would require that $\mathbf{x}$ contains information about the current word ω_0, and the previous tag t_{-1}.[2] Assume this is the 102^{nd} feature identified; in that case the feature would be represented as:

$$f_{102}(\mathbf{x}, \text{VB}) = \begin{cases} 1 \text{ if } t_{-1} = \text{DT and } \omega_0 = \text{“sing”} \\ 0 \text{ otherwise} \end{cases}$$

If this feature turns out to be very relevant in predicting the correct label, then we would expect to learn a large value for the corresponding weight w_{102}. In general, feature selection is not a trivial task; as such, it is discussed in more detail in Section 9.3.

We now re-express the conditional label probability in terms of these weighted features by exponentiating and renormalizing Equation 9.6:

$$P(y|\mathbf{x}, \mathbf{w}) = \frac{\exp(s(\mathbf{x}, y))}{\sum_{y' \in Y} \exp(s(\mathbf{x}, y'))} = \frac{\exp(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y))}{\sum_{y' \in Y} \exp(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y'))} \tag{9.7}$$

This provides the foundation for our *log-linear model*, so named because taking the logarithm of Equation 9.7 yields the linear equation:

$$\begin{aligned} \log P(y|\mathbf{x}, \mathbf{w}) &= s(\mathbf{x}, y) - \log \sum_{y' \in Y} \exp(s(\mathbf{x}, y')) \\ &= \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y) - \log \sum_{y' \in Y} \exp(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y')) \end{aligned} \tag{9.8}$$

In general, other discriminative methods (boosting, CRFs, SVMs, etc.) are variations on Equation 9.8, extending to label sequences, or adding other properties (frequently *loss terms*).

9.3 Feature Selection

In Section 9.2, we gave a simple example of a feature corresponding to a kind of dependency modelled automatically in standard bigram HMMs. In general, a standard n-gram HMM considers only the current word's lexical form, and the labels of the $n-1$ preceding words. A discriminative framework

[2] $\mathbf{x}$ may also contain additional information irrelevant to this particular feature.

relaxes this restriction – we are free to use any combination of features from the local context $\mathbf{x}$. In principle, this should allow our discriminative model to perform better than the corresponding generative model.

When selecting the features to use in a discriminative model, we consider a *window* of historical context around word ω_0. The size of this window is the number of previous (and/or subsequent) words included; a window of $(-2, 0)$ would consider the previous two words as well as ω_0, analogous to a trigam model. A choice of window size induces a *template* containing all of the local information available in that window.

Given such a template, we generally consider any combination of the constituent information as a possible feature in the discriminative model. It is important to note that we must include features not seen in the training data in the model, in order to allow new data to be recognized and classified at test time.

9.3.1 Example: POS Tagging

To further illustrate the feature vector representation, we work through an example from the POS tagging task. Consider the partially-tagged sentence, where we assume left-to-right decoding:

> Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/?? from which Spain expanded its empire into the rest of the Western Hemisphere.

In this POS-tagging task, where we aim to predict t_0, a (-2,0) window might induce the template:

$$\text{(-2,0) template: } \begin{bmatrix} \omega_{-2} & \omega_{-1} & \omega_0 \\ t_{-2} & t_{-1} & \end{bmatrix}$$

Corresponding to this template, we would choose for inclusion in the local context $\mathbf{x}$ 5-tuples of the form:

$$x_i = \langle \omega_{-2}, \omega_{-1}, \omega_0, t_{-2}, t_{-1} \rangle$$

We can then identify 3 classes of features for inclusion in the feature space:

1. word/tag features; for example:
$$f(x_i, t) = \begin{cases} 1 \text{ if } \omega_0 = \text{“base” and } t = \text{NN} \\ 0 \text{ otherwise} \end{cases}$$

2. spelling features (prefixes, suffixes, capitalization, etc.)[3]; for example:
$$f(x_i, t) = \begin{cases} 1 \text{ if } \omega_0 \text{ ends in "ing" and } t = \text{VBG} \\ 0 \text{ otherwise} \end{cases}$$

3. contextual features; for example:
$$f(x_i, t) = \begin{cases} 1 \text{ if } t_{-2} = t_{-1} = \text{DT and } t = \text{VBD} \\ 0 \text{ otherwise} \end{cases}$$

Thus, we see that through the selection of a template/history, we prevent ourselves from having to consider truly arbitrary features.

Regarding unseen training data, if we were to include word/tag features involving ω_{-1} and t_{-1}, and assuming a vocabulary of size $|V|$ and $|T|$ possible POS tags, we should include all $|V||T|$ corresponding features in the model, not just the particular word/tag pairs observed in training.

As an interesting aside, imagine we had a feature which associated the tag VBG to words ending in "ing". For example, this feature would encourage "googling" to be tagged as VBG, even if we had never seen the word in our test data. Further, imagine we had a second feature which associated the presence of capitalization in a word with the tag NNP; this feature would encourage "Google" to be tagged NNP. Finally, assume that both of these features were assigned equal weight. What should we expect the tag of a third unseen word "Boeing" to be? There is no way to say; instead, we need another feature to break the tie.

9.4 Weight Estimation

In order to perform supervised training on the log-linear model of Equation 9.8, our problem has become one of finding the weights $\hat{\mathbf{w}}$ which maximize the log likelihood given the labelled training data $\mathbf{D} = \langle(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ...\rangle$:

$$\begin{aligned} L(\mathbf{w}|\mathbf{D}) &= \sum_{i=1}^{|\mathbf{D}|} \log P(y_i|\mathbf{x}_i, \mathbf{w}) \\ &= \sum_{i=1}^{|\mathbf{D}|} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y_i) - \sum_{i=1}^{|\mathbf{D}|} \log \sum_{y' \in Y} \exp(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y')) \end{aligned} \tag{9.9}$$

[3]Note that spelling features can frequently overlap.

This is not an easy problem in general, and no efficient exact algorithms are known. Instead, we rely on common search techniques employed successfully in other areas of artificial intelligence.

Perhaps the simplest method of finding $\hat{\mathbf{w}} = \underset{\mathbf{w}}{\text{argmax}} L(\mathbf{w}|\mathbf{D})$ is via gradient ascent. Here, we maximize $L(\mathbf{w}|\mathbf{D})$ by differentiating with respect to $\mathbf{w}$, and finding a root. We can employ a trick to do this: by using the definition of $P(y|\mathbf{x}, \mathbf{w})$ in Equation 9.7, we can see that:

$$\frac{\partial L(\mathbf{w}|\mathbf{D})}{\partial \mathbf{w}} = \text{Observed counts - Expected counts} \tag{9.10}$$

Since $\mathbf{w} \cdot \mathbf{f}$ is a convex linear combination, Equation 9.10 has a unique root corresponding to a global maximum, so a simple gradient ascent search will eventually find $\hat{\mathbf{w}}$ exactly (although it may take a long time). We can optimize this search procedure by using one of many possible local search techniques; *line search* and iterative scaling have commonly been used for this purpose.

9.4.1 Line Search

In a problem with m features and weight vector $\mathbf{w}$ initialized to some value in $\mathbf{w}_0 \in \mathbb{R}^m$, line search first finds the direction of maximum positive gradient from $\mathbf{w}_0$ in the m-dimensional search space of $L(\mathbf{w}|\mathbf{D})$. It then slowly ascends on a line in this direction to reach a new point $\mathbf{w}_1$. This procedure is repeated until the gradient is negative in all directions, at which point the desired $\hat{\mathbf{w}}$ corresponding to a maximum of $L(\mathbf{w}|\mathbf{D})$ has been found.

9.4.2 Iterative Scaling

Although conceptually simple and guaranteed to find $\hat{\mathbf{w}}$, the limitations of line search become clear when we realize that our feature space may become very large (easily millions of distinct features). As line search considers the weights associated with each of these features together at every step, it must traverse a very high-dimensional search space. Despite this, we know that only a few features actually occur with any particular example. Thus, we'd prefer a method which treats features independently, as we expect to achieve much faster convergence this way.

To this end, we introduce Algorithm 1, the *iterative scaling* algorithm for finding $\hat{\mathbf{w}}$.[4] This algorithm is both easier to implement and faster to converge than gradient ascent, but it is still quite slow. We can improve performance by using a hash table when implementing the feature function, where $f_k(\mathbf{x}, y)$ is keyed by $(k, \mathbf{x}, y)$. Even so, training a POS-tagger on 40,000 sentences can take on the order of 24 hours using iterative scaling.

Algorithm 1 Iterative Scaling

Require: labelled training data $\langle(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ...\rangle$
Require: feature function satisfying $f_k(\mathbf{x}_i, y_i) \geq 0 \ \forall \ i$, k over m features
Require:
Initialize $\mathbf{w} = \mathbf{0}$
Calculate empirical counts: $\mathbf{H} = \sum_{i=1}^{|\mathbf{D}|} \mathbf{f}(\mathbf{x}_i, y_i)$
Find most common feature: $C = \max_{y' \in Y, i \in \{1:|\mathbf{D}|\}} \sum_{k=1}^{m} \mathbf{f}(\mathbf{x}_i, y')$
while $\mathbf{w}$ changed in the last iteration **do**
  Calculate expected counts: $\mathbf{E}(\mathbf{w}) = \sum_{i=1}^{|\mathbf{D}|} \sum_{y' \in Y} \mathbf{f}(\mathbf{x}_i, y') P(y'|\mathbf{x}_i, \mathbf{w})$
  for each feature $k \in \{1 : m\}$ **do**
    Scale the associated weight: $w_k = w_k + \frac{1}{C} \log \frac{H_k}{E_k(\mathbf{w})}$
  end for
end while
return $\mathbf{w}$

9.4.3 Maximum Entropy

While we have seen a method for estimating weights of features observed in training data, it is not clear how we should handle unseen features. The statistical principle of maximum entropy states that the least biased distribution that encodes certain given information is that which maximizes the information entropy. We can use this principle to unambiguously assign weights to unseen features. One can think of this process as a kind of smoothing.

[4] The lengthy derivation and justification of the iterative scaling algorithm is omitted here.

The entropy of a distribution our data set $\mathbf{D}$ is defined as:

$$H(P|\mathbf{D}) = -\left(\frac{1}{|\mathbf{D}|}\sum_{i=1}^{|\mathbf{D}|}\sum_{y'\in Y} P(y'|\mathbf{x}_i)\log P(y'|\mathbf{x}_i)\right) \tag{9.11}$$

In general, entropy is maximized if $P(y'|\mathbf{x}_i) = 1/|Y| \;\forall\; y'$,$\mathbf{x}_i$ (the uniform distribution). However, we should only consider distributions which are consistent with our training data; that is, distributions for which the expected label counts equal the empirically observed label counts in data set $\mathbf{D}$. We define the set of all such consistent distributions as:

$$\mathcal{P} = \left\{ P \left| \sum_{i=1}^{|\mathbf{D}|}\mathbf{f}(\mathbf{x}_i, y_i) = \sum_{i=1}^{|\mathbf{D}|}\sum_{y'\in Y} P(y'|\mathbf{x}_i)\mathbf{f}(\mathbf{x}_i, y') \right.\right\} \tag{9.12}$$

Clearly $\mathcal{P}$ is not empty, because there is a trivial distribution which memorizes $\mathbf{D}$:

$$P(y|\mathbf{x}_i) = \left\{ \begin{array}{l} 1 \text{ if } y = y_i \\ 0 \text{ otherwise} \end{array} \right. \in \mathcal{P}$$

Let us consider this set of consistent distributions in log-linear form:

$$\mathcal{Q} = \left\{ P \left| P(y|\mathbf{x}_i) = \frac{\exp(\mathbf{w}\cdot\mathbf{f}(\mathbf{x}_i, y))}{\sum_{y'\in Y}\exp(\mathbf{w}\cdot\mathbf{f}(\mathbf{x}_i, y'))}, \mathbf{w}\in\mathbb{R}^m \right.\right\} \tag{9.13}$$

We can define the log likelihood of a distribution Q given the data:

$$L(Q|\mathbf{D}) = -\sum_{i=1}^{|\mathbf{D}|}\log Q(y_i|\mathbf{x}_i) \tag{9.14}$$

Thus, the maximum likelihood distributions are given by:

$$\hat{Q} = \underset{Q\in\mathcal{Q}}{\text{argmin}} L(Q|\mathbf{D}) \tag{9.15}$$

$$\hat{P} = \underset{P\in\mathcal{P}}{\text{argmax}} H(P|\mathbf{D}) \tag{9.16}$$

While we will not prove it here, one can show that there is a unique distribution $\hat{R} \in \mathcal{P}\cap\mathcal{Q}$ such that $\hat{P} = \hat{Q} = \hat{R}$. Thus, if we can find this distribution, we will have found the distribution which is consistent with our training data, and which imposes the weakest possible assumptions on the weights for unseen features. This is exactly what we wanted.

9.4.4 Example of Maximum Entropy

Consider a toy example where we have two labels $Y = 0, 1$, and a two-word vocabulary such that $x_i \in= a, b$. Consider further a single feature function given by:

$$f_7(x_i, y) = \begin{cases} 1 \text{ if } y = 0 \\ 0 \text{ otherwise} \end{cases}$$

and assume that this feature was observed in 60% of the training cases; thus, we set $w_7 = 0.6$.

In our feature space, we should also include all other features induced by our chosen history/template. In other words, we should have a feature and a weight for each of the possible word/tag combinations. We can represent these features and their weights in tabular form:

	0	1	
a	w_1	w_2	w_3
b	w_4	w_5	w_6
	0.6	w_8	w_9

We immediately conclude that $w_8 = 0.4$, $w_9 = 1.0$ because $P(y = 0) + P(y = 1) = 1$. These provide constraints we can use with the maximum entropy principle to find the remaining values.

First consider that we have no information about the distribution of as and bs, so we must assume a uniform distribution: $P(x = a) = P(x = b) = 0.5$. Thus, $w_3 = w_6 = 0.5$.

Similarly, we assume a uniform distribution for $P(x|y = 0)$. Combined with the normalization constraint $P(x = a|y = 0) + P(x = b|y = 0) = P(y = 0)$, we find $w_1 = w_4 = 0.3$.

Finally, by the same argument for $P(x|y = 1)$, we find $w_2 = w_5 = 0.2$. This yields our unique maximum-entropy distribution:

	0	1	
a	0.3	0.2	0.5
b	0.3	0.2	0.5
	0.6	0.4	1