6.6 Maximum Entropy Models: Background

We turn now to a second probabilistic machine learning framework called **maximum entropy** modeling, **MaxEnt** for short. MaxEnt is more widely known as **multinomial logistic regression**.

Our goal in this chapter is to introduce the use of MaxEnt for sequence classification. Recall that the task of sequence classification or sequence labeling is to assign a label to each element in some sequence, such as assigning a part-of-speech tag to a word. The most common MaxEnt sequence classifier is the **maximum entropy Markov model** or **MEMM**, introduced in Section 6.8. But before we describe the use of MaxEnt as a sequence classifier, we need to introduce non-sequential classification.

The task of classification is to take a single observation, extract some useful features describing the observation, and then, based on these features, to **classify** the observation into one of a set of discrete classes. A **probabilistic** classifier does slightly more than this; in addition to assigning a label or class, it gives the **probability** of the observation being in that class; indeed, for a given observation, a probabilistic classifier gives a probability distribution over all classes.

Such non-sequential classification tasks occur throughout speech and language processing. For example, in **text classification** we might need to decide whether a particular email should be classified as spam. In **sentiment analysis** we have to determine whether a particular sentence or document expresses a positive or negative **opinion**. In many tasks, we'll need to know where the sentence boundaries are, and so we'll need to classify a period character ('.') as either a sentence boundary or not. We show more examples of the need for classification throughout this book.

Log-linear classifier

MaxEnt belongs to the family of classifiers known as the **exponential** or **log-linear** classifiers. MaxEnt works by extracting some set of features from the input, combining them **linearly** (meaning that each feature is multiplied by a weight and then added up), and then, for reasons discussed below, using this sum as an exponent.

Let's flesh out this intuition just a bit more. Assume that we have some input x (perhaps it is a word that needs to be tagged or a document that needs to be classified) from which we extract some features. A feature for tagging might be *this word ends in -ing* or *the previous word was 'the'*. For each such feature f_i, we have some weight w_i.

Given the features and weights, our goal is to choose a class (e.g., a part-of-speech tag) for the word. MaxEnt does this by choosing the most probable tag; the probability of a particular class c given the observation x is

$$p(c\,x) = \frac{1}{Z}\exp(\sum_i w_i f_i) \tag{6.44}$$

Here Z is a normalizing factor, used to make the probabilities correctly sum to 1; and as usual $\exp(x) = e^x$. As we show later, this is a simplified equation in various ways; for example, in the actual MaxEnt model, the features f and weights w both depend on the class c (i.e., we'll have different features and weights for different classes).

To explain the details of the MaxEnt classifier, including the definition of the normalizing term Z and the intuition of the exponential function, we must first understand

both **linear regression**, which lays the groundwork for prediction by using features, and **logistic regression**, which is our introduction to exponential models. We cover these areas in the next two sections; readers with a background in regression may want to skip ahead. Then, in Section 6.7 we introduce the details of the MaxEnt classifier. Finally, in Section 6.8 we show how the MaxEnt classifier is used for sequence classification in the **maximum entropy Markov model (MEMM)**.

6.6.1 Linear Regression

In statistics we use two different names for tasks that map some input features into some output value: we use the word **regression** when the output is real-valued and **classification** when the output is one of a discrete set of classes.

You may already be familiar with linear regression from a statistics class. The idea is that we are given a set of observations, each observation associated with some features, and we want to predict some real-valued outcome for each observation. Let's see an example from the domain of predicting housing prices. Levitt and Dubner (2005) showed that the words used in a real estate ad can be a good predictor of whether a house will sell for more or less than its asking price. They showed, for example, that houses whose real estate ads had words like *fantastic*, *cute*, or *charming*, tended to sell for lower prices, while houses whose ads had words like *maple* and *granite* tended to sell for higher prices. Their hypothesis was that real estate agents used vague positive words like *fantastic* to mask the lack of any specific positive qualities in the house. Just for pedagogical purposes, we created the fake data in Fig. 6.17.

# of Vague Adjectives	Amount House Sold Over Asking Price
4	0
3	$1000
2	$1500
2	$6000
1	$14000
0	$18000

Figure 6.17 Some made-up data on the number of vague adjectives (*fantastic*, *cute*, *charming*) in a real estate ad and the amount the house sold for over the asking price.

Regression line

Figure 6.18 shows a graph of these points, with the feature (# of adjectives) on the x-axis, and the price on the y-axis. We have also plotted a **regression line**, which is the line that best fits the observed data. The equation of any line is $y = mx + b$; as we show on the graph, the slope of this line is $m = -4900$, and the intercept is 16550. We can think of these two parameters of this line (slope m and intercept b) as a set of weights that we use to map from our features (in this case x, number of adjectives) to our output value y (in this case, price). We can represent this linear function by using w to refer to weights as follows:

$$\text{price} = w_0 + w_1 * \text{Num\_Adjectives} \tag{6.45}$$

Thus, Eq. 6.45 gives us a linear function that lets us estimate the sales price for any number of these adjectives. For example, how much would we expect a house whose

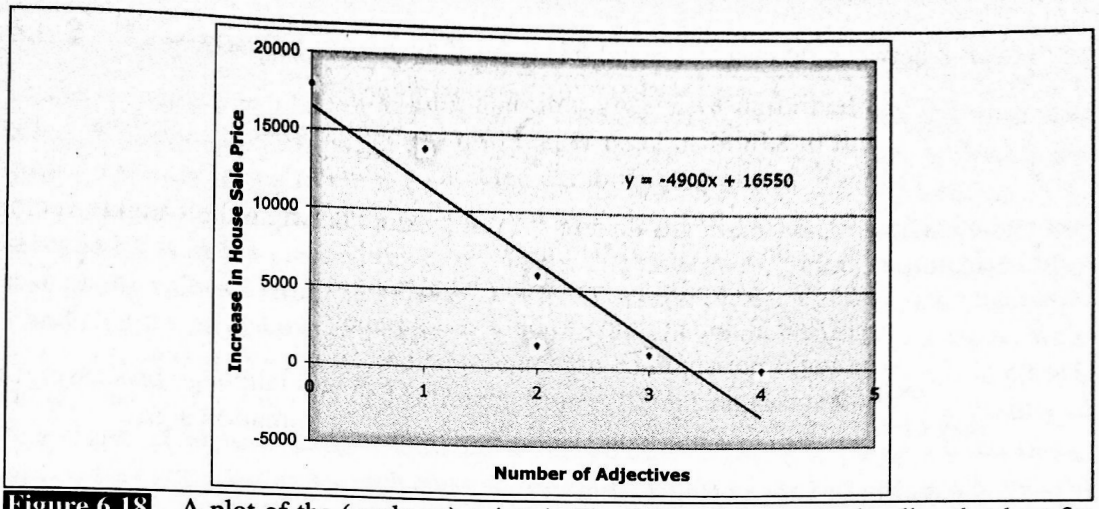

Figure 6.18 A plot of the (made-up) points in Fig. 6.17 and the regression line that best fits them, with the equation $y = -4900x + 16550$.

ad has five adjectives to sell for?

The true power of linear models comes when we use more than one feature (technically, we call this **multiple linear regression**). For example, the final house price probably depends on many factors, such as the current mortgage rate, the number of unsold houses on the market, etc. We could encode each of these as a variable, and the importance of each factor would be the weight on that variable, as follows:

$$\text{price} = w_0 + w_1 * \text{Num\_Adjectives} + w_2 * \text{Mortgage\_Rate} + w_3 * \text{Num\_Unsold\_Houses}$$

Feature

In speech and language processing, we often call each of these predictive factors, like the number of adjectives or the mortgage rate, a **feature**. We represent each observation (each house for sale) by a vector of these features. Suppose a house has one adjective in its ad and the mortgage rate was 6.5 and there were 10,000 unsold houses in the city. The feature vector for the house would be $\vec{f} = (1, 6.5, 10000)$. Suppose the weight vector that we had previously learned for this task was $\vec{w} = (w_0, w_1, w_2, w_3) = (18000, -5000, -3000, -1.8)$. Then the predicted value for this house would be computed by multiplying each feature by its weight:

$$\text{price} = w_0 + \sum_{i=1}^{N} w_i \quad f_i \tag{6.46}$$

In general, we will pretend that there is an extra feature, f_0, that has the value 1, an **intercept feature**, which make the equations simpler with regard to that pesky w_0, and so in general we can represent a linear regression for estimating the value of y as

$$\textbf{linear regression:} \qquad y = \sum_{i=0}^{N} w_i \quad f_i \tag{6.47}$$

Dot product

Taking two vectors and creating a scalar by multiplying each element in a pairwise fashion and summing the results is called the **dot product**. Recall that the dot product

$a \ b$ between two vectors a and b is defined as

$$\textbf{dot product:} \qquad a \ b = \sum_{i=1}^{N} a_i b_i = a_1 b_1 + a_2 b_2 + \quad + a_n b_n \tag{6.48}$$

Thus, Eq. 6.47 is equivalent to the dot product between the weights vector and the feature vector:

$$y = w \ f \tag{6.49}$$

Vector dot products frequently occur in speech and language processing; we often rely on the dot product notation to avoid the messy summation signs.

Learning in Linear Regression

How do we learn the weights for linear regression? Intuitively, we'd like to choose weights that make the estimated values y as close as possible to the actual values that we saw in a training set.

Consider a particular instance $x^{(j)}$ from the training set that has an observed label in the training set $y^{(j)}_{obs}$ (we'll use superscripts in parentheses to represent training instances). Our linear regression model predicts a value for $y^{(j)}$ as follows:

$$y^{(j)}_{pred} = \sum_{i=0}^{N} w_i \quad f^{(j)}_i \tag{6.50}$$

We'd like to choose the whole set of weights W so as to minimize the difference between the predicted value $y^{(j)}_{pred}$ and the observed value $y^{(j)}_{obs}$, and we want this difference minimized over all the M examples in our training set. Actually, we want to minimize the absolute value of the difference since we don't want a negative distance in one example to cancel out a positive difference in another example, so for simplicity (and differentiability), we minimize the square of the difference. Thus, the total value we want to minimize, which we call the **sum-squared error**, is this cost function of the current set of weights W:

Sum-squared error

$$\text{cost}(W) = \sum_{j=0}^{M} \left(y^{(j)}_{pred} - y^{(j)}_{obs} \right)^2 \tag{6.51}$$

We don't give here the details of choosing the optimal set of weights to minimize the sum-squared error. But, briefly, it turns out that if we put the entire training set into a single matrix X with each row in the matrix consisting of the vector of features associated with each observation $x^{(i)}$ and put all the observed y values in a vector $\vec{y}$, there is a closed-form formula for the optimal weight values W that will minimize cost(W):

$$W = (X^T X)^{-1} X^T \vec{y} \tag{6.52}$$

Implementations of this equation are widely available in statistical packages like SPSS or R.

6.6.2 Logistic Regression

Linear regression is what we want when we are predicting a real-valued outcome. But often in speech and language processing we are doing **classification**, in which the output y we are trying to predict takes on one from a small set of discrete values.

Consider the simplest case of binary classification: classifying whether some observation x is in the class (true) or not in the class (false). In other words, y can only take on the values 1 (true) or 0 (false), and we'd like a classifier that can take features of x and return true or false. Furthermore, instead of just returning the 0 or 1 value, we'd like a model that can give us the **probability** that a particular observation is in class 0 or 1. This is important because in most real-world tasks we're passing the results of this classifier to some further classifier to accomplish some task. Since we are rarely completely certain about which class an observation falls in, we'd prefer not to make a hard decision at this stage, ruling out all other classes. Instead, we'd like to pass on to the later classifier as much information as possible: the entire set of classes, with the probability value that we assign to each class.

Could we modify our linear regression model to use it for this kind of probabilistic classification? Suppose we just tried to train a linear model to predict a probability as follows:

$$P(y = \text{true}\ x) = \sum_{i=0}^{N} w_i \quad f_i \tag{6.53}$$

$$= w \ f \tag{6.54}$$

We could train such a model by assigning each training observation the target value $y = 1$ if it was in the class (true) and the target value $y = 0$ if it was not (false). Each observation x would have a feature vector f, and we would train the weight vector w to minimize the predictive error from 1 (for observations in the class) or 0 (for observations not in the class). After training, we would compute the probability of a class given an observation by just taking the dot product of the weight vector with the features for that observation.

The problem with this model is that nothing forces the output to be a legal probability, that is, to lie between 0 and 1. The expression $\sum_{i=0}^{N} w_i \quad f_i$ produces values from $-\infty$ to ∞. How can we fix this problem? Suppose that we keep our linear predictor $w \ f$, but instead of having it predict a probability, we have it predict a *ratio* of two probabilities. Specifically, suppose we predict the ratio of the probability of being in the class to the probability of not being in the class. This ratio is called the **odds**. If an event has probability .75 of occurring and probability .25 of not occurring, we say the **odds** of occurring is $.75/.25 = 3$. We could use the linear model to predict the odds of y being true:

Odds

$$\frac{p(y = \text{true})\ x}{1 - p(y = \text{true}\ x)} = w \ f \tag{6.55}$$

This last model is close: a ratio of probabilities can lie between 0 and ∞. But we need the left-hand side of the equation to lie between $-\infty$ and ∞. We can achieve this by taking the natural log of this probability:

$$\ln\left(\frac{p(y=\text{true}\ x)}{1-p(y=\text{true}\ x)}\right) = w\ \ f \tag{6.56}$$

Now both the left and right hand lie between $-\infty$ and ∞. This function on the left (the log of the odds) is known as the **logit function**:

Logit function

$$\text{logit}\,(p(x)) = \ln\left(\frac{p(x)}{1-p(x)}\right) \tag{6.57}$$

Logistic regression

The model of regression in which we use a linear function to estimate the logit of the probability rather than the probability is known as **logistic regression**. If the linear function is estimating the logit, what is the actual formula in logistic regression for the probability $P(y = \text{true})$? You should stop here and with Eq. 6.56 apply some simple algebra to solve for the probability $P(y = \text{true})$.

Hopefully, when you solved for $P(y = \text{true})$, you came up with a derivation something like the following:

$$\begin{aligned}
&\ln\left(\frac{p(y=\text{true}\ x)}{1-p(y=\text{true}\ x)}\right) = w\ \ f \\
&\frac{p(y=\text{true}\ x)}{1-p(y=\text{true}\ x)} = e^{w\ f} \qquad (6.58)\\
&p(y=\text{true}\ x) = (1-p(y=\text{true}\ x))e^{w\ f} \\
&p(y=\text{true}\ x) = e^{w\ f} - p(y=\text{true}\ x)e^{w\ f} \\
&p(y=\text{true}\ x) + p(y=\text{true}\ x)e^{w\ f} = e^{w\ f} \\
&p(y=\text{true}\ x)(1+e^{w\ f}) = e^{w\ f} \\
&p(y=\text{true}\ x) = \frac{e^{w\ f}}{1+e^{w\ f}} \qquad (6.59)
\end{aligned}$$

Once we have this probability, it is easy to state the probability of the observation not belonging to the class, $p(y = \text{false}\ x)$, as the two must sum to 1:

$$p(y=\text{false}\ x) = \frac{1}{1+e^{w\ f}} \tag{6.60}$$

Here are the equations again in explicit summation notation:

$$p(y=\text{true}\ x) = \frac{\exp(\sum_{i=0}^{N} w_i f_i)}{1+\exp(\sum_{i=0}^{N} w_i f_i)} \tag{6.61}$$

$$p(y=\text{false}\ x) = \frac{1}{1+\exp(\sum_{i=0}^{N} w_i f_i)} \tag{6.62}$$

We can express the probability $P(y = \text{true}\ x)$ in a slightly different way, by dividing the numerator and denominator in (6.59) by $e^{-w\ f}$:

$$p(y=\text{true}\ x) = \frac{e^{w\ f}}{1+e^{w\ f}} \tag{6.63}$$

$$= \frac{1}{1+e^{-wf}} \tag{6.64}$$

Logistic function

The last equation is now in the form of what is called the **logistic function** (the function that gives logistic regression its name). The general form of the logistic function is

$$\frac{1}{1+e^{-x}} \tag{6.65}$$

The logistic function maps values from $-\infty$ and ∞ to lie between 0 and 1. Again, we can express $P(y = \text{false } x)$ so as to make the probabilities sum to 1:

$$p(y = \text{false } x) = \frac{e^{-wf}}{1+e^{-wf}} \tag{6.66}$$

6.6.3 Logistic Regression: Classification

Classification
Inference

Given a particular observation, how do we decide which of the two classes ('true' or 'false') it belongs to? This is the task of **classification**, also called **inference**. Clearly, the correct class is the one with the higher probability. Thus, we can safely say that our observation should be labeled 'true' if

$$\begin{aligned} p(y = \text{true } x) &> p(y = \text{false } x) \\ \frac{p(y = \text{true } x)}{p(y = \text{false } x)} &> 1 \\ \frac{p(y = \text{true } x)}{1 - p(y = \text{true } x)} &> 1 \end{aligned}$$

and substituting from Eq. 6.58 for the odds ratio:

$$\begin{aligned} e^{wf} &> 1 \\ w\ f &> 0 \end{aligned} \tag{6.67}$$

or with the explicit sum notation:

$$\sum_{i=0}^{N} w_i f_i > 0 \tag{6.68}$$

Thus, to decide if an observation is a member of the class, we just need to compute the linear function and see if its value is positive; if so, the observation is in the class.

A more advanced point: the equation $\sum_{i=0}^{N} w_i f_i = 0$ is the equation of a **hyperplane** (a generalization of a line to N dimensions). The equation $\sum_{i=0}^{N} w_i f_i > 0$ is thus the part of N-dimensional space above this hyperplane. Thus, we can see the logistic regression function as learning a hyperplane which separates points in space that are in the class ('true') from points that are not in the class.

6.6.4 Advanced: Learning in Logistic Regression

Conditional maximum likelihood estimation

In linear regression, learning consisted of choosing the weights w that minimized the sum-squared error on the training set. In logistic regression we use **conditional maximum likelihood estimation**. What this means is that we choose the parameters w that make the probability of the observed y values in the training data to be the highest, given the observations x. In other words, for an individual training observation x, we want to choose the weights as follows:

$$\hat{w} = \operatorname*{argmax}_w P(y^{(i)} \; x^{(i)}) \tag{6.69}$$

And we'd like to choose the optimal weights for the entire training set:

$$\hat{w} = \operatorname*{argmax}_w \prod_i P(y^{(i)} \; x^{(i)}) \tag{6.70}$$

We generally work with the log likelihood:

$$\hat{w} = \operatorname*{argmax}_w \sum_i \log P(y^{(i)} \; x^{(i)}) \tag{6.71}$$

So, more explicitly:

$$\hat{w} = \operatorname*{argmax}_w \sum_i \log \begin{cases} P(y^{(i)} = 1 \; x^{(i)})) & \text{for } y^{(i)} = 1 \\ P(y^{(i)} = 0 \; x^{(i)})) & \text{for } y^{(i)} = 0 \end{cases} \tag{6.72}$$

This equation is unwieldy, so we usually apply a convenient representational trick. Note that if $y = 0$, the first term goes away, while if $y = 1$ the second term goes away:

$$\hat{w} = \operatorname*{argmax}_w \sum_i y^{(i)} \log P(y^{(i)} = 1 \; x^{(i)}) + (1 - y^{(i)}) \log P(y^{(i)} = 0 \; x^{(i)}) \tag{6.73}$$

Now if we substitute in Eq. 6.64 and Eq. 6.66, we get

$$\hat{w} = \operatorname*{argmax}_w \sum_i y^{(i)} \log \frac{1}{1 + e^{-w \, f}} + (1 - y^{(i)}) \log \frac{e^{-w \, f}}{1 + e^{-w \, f}} \tag{6.74}$$

Convex optimization

Finding the weights that result in the maximum log-likelihood according to Eq. 6.74 is a problem in the field known as **convex optimization**. Among the most commonly used algorithms are **quasi-Newton** methods like L-BFGS (Nocedal, 1980; Byrd et al., 1995), as well as gradient ascent, conjugate gradient, and various iterative scaling algorithms (Darroch and Ratcliff, 1972; Della Pietra et al., 1997; Malouf, 2002). These learning algorithms are available in MaxEnt modeling toolkits but are too complex to define here; interested readers should see the Historical Notes at the end of the chapter.