Probabilistic learning

Charles Elkan

November 8, 2012

Important: These lecture notes are based closely on notes written by Lawrence Saul. Text may be copied directly from his notes, or paraphrased. Also, these typeset notes lack illustrations. See the classroom lectures for figures and diagrams.

1 Learning in a Bayesian network

A Bayesian network is a directed graph with a CPT (conditional probability table) for each node. This section explains how to learn the CPTs from training data. As explained before, the training data are a matrix where each row is an instance and each column is a feature. Instances are also called examples, while features are also called nodes, random variables, and attributes. One entry in the matrix is one value of one feature, that is one outcome of one random variable.

We consider first the scenario where each instance is complete, that is the outcome of every node is observed for every instance. In this scenario, nothing is unknown, or in other words, there is no missing data. This scenario is also called fully visible, or no hidden nodes, or no latent variables.

We also assume that the graph of the Bayesian network is known, that its nodes X_1 to X_n constitute a finite set, and that each node is random variable with a discrete finite set of alternative values. In this scenario, what we need to learn is the CPT of each node. A single entry in one CPT is $p(X_i = x|\text{pa}(X_i) = \pi)$ where x is a specific outcome of X_i and π is a specific set of outcomes, also called a configuration, of the parent nodes of X_i.

The training data are T instances x_t, each of which is a complete configuration of X_1 to X_n. We write $x_t = (x_{t1}, \ldots, x_{tn})$. Remember the convention that the first subscript refers to rows while the second subscript refers to columns. To make learning feasible, we need a basic assumption about the training data.

Assumption. Each example is an independent and identically distributed (IID) random sample from the joint distribution defined by the Bayesian network.

This assumption has two parts. First, each x_t is identically distributed means that each sample is generated using the same CPTs. Second, being independent means that probabilities can be multiplied: $p(x_s, x_t) = p(x_s)p(x_t)$. With the IID assumption, we are ready to begin to derive a learning procedure. The probability of the training data is

$$P = \prod_{t=1}^{T} p(X_1 = x_{t1}, \ldots, X_n = x_{tn}).$$

The probability of example t is

$$\begin{aligned} p(X_1 = x_{t1}, \ldots, X_n = x_{tn}) &= \prod_{i=1}^{n} p(X_i = x_{ti} | X_1 = x_{t1}, \ldots, X_{i-1} = x_{t,i-1}) \\ &= \prod_{i=1}^{n} p(X_i = x_{ti} | \text{pa}(X_i) = \text{pa}_{ti}) \end{aligned}$$

The first equation above follows from the chain rule of probabilities, while the second follows from conditional independence in the Bayesian network.

Learning means choosing values, based on the available data, for the aspects of the model that are unknown. Here, the model is the probability distribution specified by the Bayesian network. Its graph is known but the parameters inside its CPTs are unknown. The principle of maximum likelihood says that we should choose values for unknown parameters in such a way that the overall probability of the training data is maximized. This principle is not a theorem that can be proved. It is simply a sensible guideline. One way to argue that the principle is sensible is to notice that, essentially, it says that we should assume that the training data are the most typical possible, that is, that the observed data are the mode of the distribution to be learned.

The principle of maximum likelihood says that we should choose values for the parameters of the CPTs that make P as large as possible. Let these parameters be called w. The principle says that we should choose $w^* = \text{argmax}_w P$. Because the logarithm function is monotone strictly increasing, this is equivalent to choosing $w^* = \text{argmax}_w \log P$. It is convenient to maximize the log because the log of

a product is a sum, and dealing with sums is easier. So, the goal is to maximize

$$\begin{aligned} L &= \log \prod_{t=1}^{T} \prod_{i=1}^{n} p(X_i = x_{ti} | \text{pa}(X_i) = \text{pa}_{ti}) \\ &= \sum_{t=1}^{T} \sum_{i=1}^{n} \log p(X_i = x_{ti} | \text{pa}(X_i) = \text{pa}_{ti}) \end{aligned}$$

Swapping the order of the summations gives

$$L = \sum_{i=1}^{n} \sum_{t=1}^{T} \log p(X_i = x_{ti} | \text{pa}(X_i) = \text{pa}_{ti}). \quad (1)$$

Now, notice that each inner sum over t involves a different CPT. These CPTs have parameters whose values can be chosen completely separately. Therefore, L can be maximized by maximizing each inner sum separately. We can decompose the task of maximizing L into n separate subtasks to maximize

$$M_i = \sum_{t=1}^{T} \log p(X_i = x_{ti} | \text{pa}(X_i) = \text{pa}_{ti})$$

for $i = 1$ to $i = n$. Consider one of these subtasks. The sum over t treats each training example separately. To make progress, we group the training examples into equivalence classes. Each class consists of all examples among the T that have the same outcome for X_i and the same outcome for the parents of X_i. Let x range over the outcomes of X_i and let π range over the outcomes of the parents of X_i. Let $\text{count}(x, \pi)$ be how many of the T examples have the value x and the configuration π. Note that

$$T = \sum_{x} \sum_{\pi} \text{count}(x, \pi).$$

We can write

$$M_i = \sum_{x} \sum_{\pi} \text{count}(x, \pi) \log p(X_i = x | \text{pa}(X_i) = \pi).$$

We want to choose parameter values for the CPT for node i to maximize this expression. These parameter values are the probabilities $p(X_i = x | \text{pa}(X_i) = \pi)$. These values are constrained by the fact that for each π

$$\sum_{x} \log p(X_i = x | \text{pa}(X_i) = \pi) = 1.$$

However, there is no constraint connecting the values for different π. Therefore, we can swap the order of the summations inside the expression for M_i and obtain a separate subtask for each π. Write $w_x = p(X_i = x|\text{pa}(X_i) = \pi)$ and $c_x = \text{count}(x, \pi)$. The problem to solve is

$$\text{maximize } \sum_x c_x \log w_x \text{ subject to } w_x \geq 0 \text{ and } \sum_x w_x = 1.$$

This problem can be solved using Lagrange multipliers. The solution is

$$w_x = \frac{c_x}{\sum_x c_x}.$$

In words, the maximum likelihood estimate of the probability that $X_i = x$, given that the parents of X_i are observed to be π, is

$$\begin{aligned} p(X_i = x|\text{pa}(X_i) = \pi) &= \frac{\text{count}(X_i = x, \text{pa}(X_i) = \pi)}{\sum_{x'} \text{count}(X_i = x', \text{pa}(X_i) = \pi)} \\ &= \frac{\text{count}(X_i = x, \text{pa}(X_i) = \pi)}{\text{count}(\text{pa}(X_i) = \pi)} \\ &= \frac{\sum_t I(x = x_{ti}, \pi = \text{pa}_{ti})}{\sum_t I(\pi = \text{pa}_{ti})} \end{aligned}$$

where the counts are with respect to the training data.

These estimates make sense intuitively. Each estimated probability is proportional to the corresponding frequency observed in the training data. If the value x is never observed for some combination π, then its conditional probability is estimated to be zero.

Although the estimates are intuitively sensible, only a formal derivation like the one above can show that they are correct (and unique). The derivation uses several mathematical manipulations that are common in similar arguments. These manipulations include changing products into sums, swapping the order of summations, and arguing that maximization subtasks are separate.

End of the lecture on Thursday October 25.

2 Markov models of language

Many applications involving natural language need a model that assigns probabilities to sentences. For example, the most successful translation systems nowadays

for natural language are based on probabilistic models. Let F be a random variable whose values are sentences written in French, and let E be a similar random variable ranging over English sentences. Given a specific French sentence f, the machine translation task is to find $e^* = \text{argmax}_e\ p(E = e|F = f)$. One way to decompose the task into subtasks is to use Bayes' rule and write

$$e^* = \text{argmax}_e \frac{p(F = f|E = e)p(E = e)}{p(F = f)} = \text{argmax}_e\ p(F = f|E = e)p(E = e).$$

The denominator $p(F = f)$ can be ignored because it is the same for all e. Although creating a model of $p(F = f|E = e)$ is presumably just as difficult as creating a model directly of $p(E = e|F = f)$, the model of $p(E = e)$ can overcome some errors in $p(F = f|E = e)$. For example, regardless of the original sentence in the foreign language, the English sentence "Colorless green ideas sleep furiously" should not be a high-probability translation. This section explains how to learn basic models of $p(E = e)$.

Clearly the probability of a sentence depends on the words in it, and also on the order of the words. Consider a sentence that consists of the words w_1 to w_L in order. Let these words be the outcomes of random variables W_1 to W_L. The chain rule of probabilities says that

$$p(W_1, W_2, \ldots, W_L) = p(W_1)p(W_2|W_1) \cdots p(W_L|W_{L-1}, \ldots W_1).$$

Words that occur a long way before w_l in the sentence presumably influence the probability of w_l less, so to simplify this expression it is reasonable to fix a number n of previous words and write

$$\prod_{l=1}^{L} p(W_l|W_{l-n}, \ldots, W_{l-2}, W_{l-1})$$

with each word depending only on the previous n words. In the special case where $n = 0$, each word is independent of the previous words. A model of this type is called a Markov model of order n. A unigram model has order $n = 0$, a bigram model has order $n = 1$, and a trigram model has order $n = 2$.

A bigram model corresponds to a Bayesian network with nodes W_1 to W_L and an edge from each node W_l to W_{l+1}. Importantly, the same CPT $p(W_{l+1} = j|W_l = i)$ is used at each node W_{l+1}. Fixing the entries in different CPTs to be the same is called tying. Notice that technically we have a different Bayesian network for each different length L, but tying CPTs lets us treat all these networks as the same.

How can we learn the shared CPT? Each node W_l is a discrete random variable, but one with a very large set of values. The cardinality of this set is the size of the vocabulary, typically between 10^4 and 10^5 in applications. Since most words never follow each other, a document collection of size smaller than $(10^5)^2$ words can be adequate for training. Fortunately, nowadays it is easy to assemble and process collections of 10^8 and more words.

The maximum likelihood estimate of the CPT parameters is

$$p(W_l = j | W_{l-1} = i) = \frac{c_{ij}}{c_i}$$

where c_i is the number of times that word i occurs followed by any other word, and c_{ij} is the number of times that word i occurs followed by word j. A note on notation: it is convenient to assume that each word is an integer between 1 and the vocabulary size. Notation such as w_i instead of i for the ith word causes two difficulties: it leads to double subscripts, and it suggests that strings are mathematical objects.

Some issues occur with n-gram models. The first issue is that they do not handle novel words in an intelligent way. Typically we convert each word not in the predefined vocabulary into a special fixed token such as $\langle UNK \rangle$, and then treat this as an ordinary word. The second issue is that all sequences of words not seen in the training collection are assigned zero probability. For example, the bigram "pink flies" may be so uncommon that it occurs zero times in the training collection, but that does not mean it is impossible. Its probability should be small, but above zero. The higher the order of the n-gram model is, the more this second issue is important.

3 Linear regression

Linear regression is perhaps the most widely used method of modeling data in classical statistics. Here we see how it fits into the paradigm of learning the parameters of a Bayesian network via the principle of maximum likelihood.

We have independent nodes X_1 to X_d and a dependent node Y, with an edge $X_i \to Y$ for each i. Intuitively, the value of Y is a linear function of the values of X_1 to X_d, plus some random noise. Assuming that the noise has mean zero, we can write

$$E[Y] = \sum_{i=1}^{d} w_i x_i = \bar{w} \cdot \bar{x}$$

where w_1 to w_d are parameters describing the linear dependence. The standard choice to model the random noise is a Gaussian distribution with mean zero and variance σ^2. The probability density function of this distribution is

$$p(z) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{z^2}{2\sigma^2}.$$

Combining this with the expression for $E[y]$ gives

$$p(Y = y|\bar{X} = \bar{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{1}{2\sigma^2}(y - \bar{w} \cdot \bar{x})^2.$$

End of the lecture on Tuesday October 30.

To learn the parameters w_1 to w_d we have training examples $(\bar{x}_t, y_t)$ for $t = 1$ to $t = T$. Assume that each $\bar{x}_t$ is a column vector. Given that these examples are IID, the log likelihood is

$$L = \sum_{t=1}^{T} \log p(y_t|\bar{x}_t) = -\sum_{t=1}^{T} \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2}(y - \bar{w} \cdot \bar{x}_t)^2.$$

We can maximize this expression in two stages: first find the optimal w_i values, and then find the optimal σ^2 value. The first subproblem is to minimize (not maximize)

$$S = \sum_{t=1}^{T} (y_t - \bar{w} \cdot \bar{x}_t)^2.$$

We can solve this by setting the partial derivatives of S to zero. We get the equations

$$\frac{\partial}{\partial w_i} S = \sum_{t=1}^{T} 2(y_t - \bar{w} \cdot \bar{x}_t) x_{it} = 0$$

for $i = 1$ to $i = d$, where we write x_{it} because $\bar{x}_t$ is a column vector. These yield the system of d linear equations

$$\sum_{t=1}^{T} y_t x_{it} = \sum_{t=1}^{T} (\bar{w} \cdot \bar{x}_t) x_{it}.$$

Note that each of the d equations involves all of the unknowns w_1 to w_d. In matrix notation, the system of equations is $\bar{b} = A\bar{w}$. Here, $\bar{b}$ is the column vector of

length d whose ith entry is $b_i = \sum_t y_t x_{it}$, that is $\bar{b} = \sum_t y_t \bar{x}_t$. The right side is $\sum_{t=1}^{T} x_{it}(\bar{x}_t^T \bar{w})$ where the superscript T means transpose and the dot product has been written as a matrix product. This yields

$$\bar{b} = \sum_{t=1}^{T} \bar{x}_t(\bar{x}_t^T \bar{w}) = (\sum_{t=1}^{T} \bar{x}_t \bar{x}_t^T)\bar{w} = A\bar{w}$$

where the $d \times d$ square matrix $A = \sum_t \bar{x}_t \bar{x}_t^T$. The row i column j entry of A is $A_{ij} = \sum_t x_{it} x_{jt}$.

Mathematically, the solution to the system $A\bar{w} = \bar{b}$ is $\bar{w} = A^{-1}\bar{b}$. Computationally, evaluating the inverse A^{-1} of A is more expensive than just solving the system of equations once for a specific vector $\bar{b}$. In practice, in Matlab one uses the backslash operator, and other programming environments have a similar feature.

The inverse of A is not well-defined when A does not have full rank. Since A is the sum of T matrices of rank one, this happens when $T < d$, and can happen when the input vectors $\bar{x}_t$ are not linearly independent. One way of overcoming this issue is to choose the solution $\bar{w}$ with minimum norm such that $A\bar{w} = \bar{b}$. Such a $\bar{w}$ always exists and is unique. Concretely, this solution is $\bar{w}^* = A^+\bar{b}$ where A^+ is the Moore-Penrose pseudo inverse of A, which always exists, and can be computed via the singular value decomposition (SVD) of A.

We said above that we can maximize the log likelihood in two stages, first finding the best w_i values, and then finding the best σ^2 value. The second stage is left as an exercise for the reader.

4 The general EM algorithm

Suppose that, in the data available for training, the outcomes of some random variables are unknown for some examples. These outcomes are called hidden or latent, and the examples are called incomplete or partial. Conceptually, it is not the case that the hidden outcomes do not exist. Rather, they do exist, but they have been concealed from the observer.

Let X be the set of all nodes of the Bayesian network. As before, suppose that there are T training examples, which are independent and identically distributed. For the tth training example, let V_t be the set of visible nodes and let H_t be the set of hidden nodes, so $X = V_t \cup H_t$. Note that different examples may have different hidden nodes.

As before, we want to maximize the log likelihood of the observed data:

$$\begin{aligned} L &= \sum_t \log p(V_t = v_t) \\ &= \sum_t \log \sum_{h_t} p(V_t = v_t, H_t = h_t) \\ &= \sum_t \log \sum_{h_t} \prod_{i=1}^{n} p(X_i = x_i | \text{pa}(X_i) = \text{pa}_i). \end{aligned}$$

In the last expression above, each X_i belongs to either V_t or H_t. Because of the sum over h_t, we cannot move the logarithm inside the product and we do not get a separate optimization subproblem for each node X_i. Expectation-maximization (EM) is the name for an approach to solving the combined optimization problem.

To simplify notation, assume initially that there is just one training example, with one observed random variable $X = x$ and one hidden random variable Z. Let θ be all the parameters of the joint model $p(X = x, Z = z; \theta)$. Following the principle of maximum likelihood, the goal is to choose θ to maximize the log likelihood function, which is $L(\theta; x) = \log p(x; \theta)$.

As noted before, $p(x; \theta) = \sum_z p(x, z; \theta)$. Suppose we have a current estimate θ_t for the parameters. Multiplying inside this sum by $p(z|x; \theta_t)/p(z|x; \theta_t)$ gives that the log likelihood is

$$D = \log p(x; \theta) = \log \sum_z p(x, z; \theta) \frac{p(z|x; \theta_t)}{p(z|x; \theta_t)}.$$

Note that $\sum_z p(z|x; \theta_t) = 1$ and $p(z|x; \theta_t) \geq 0$ for all z. Therefore D is the logarithm of a weighted sum, so we can apply Jensen's inequality,[1] which says

[1] The mathematical fact on which the EM algorithm is based is known as Jensen's inequality. It is the following lemma.

Lemma: Suppose the weights w_j are nonnegative and sum to one, and let each x_j be any real number for $j = 1$ to $j = n$. Let $f : \mathbb{R} \to \mathbb{R}$ be any concave function. Then

$$f(\sum_{j=1}^{n} w_j x_j) \geq \sum_{j=1}^{n} w_j f(x_j).$$

Proof: The proof is by induction on n. For the base case $n = 2$, the definition of being concave says that $f(wa + (1 - w)b) \geq wf(a) + (1 - w)f(b)$. The logarithm function is concave, so Jensen's inequality applies to it.

$\log \sum_j w_j v_j \geq \sum_j w_j \log v_j$, given $\sum_j w_j = 1$ and each $w_j \geq 0$. Here, we let the sum range over the values z of Z, with the weight w_j being $p(z|x;\theta_t)$. We get

$$D \geq E = \sum_z p(z|x;\theta_t) \log \frac{p(x,z;\theta)}{p(z|x;\theta_t)}.$$

Separating the fraction inside the logarithm to obtain two sums gives

$$E = \big(\sum_z p(z|x;\theta_t) \log p(x,z;\theta)\big) - \big(\sum_z p(z|x;\theta_t) \log p(z|x;\theta_t)\big).$$

Since $E \leq D$ and we want to maximize D, consider maximizing E. The weights $p(z|x;\theta_t)$ do not depend on θ, so we only need to maximize the first sum, which is

$$\sum_z p(z|x;\theta_t) \log p(x,z;\theta).$$

In general, the E step of an EM algorithm is to compute $p(z|x;\theta_t)$ for all z. The M step is then to find θ to maximize $\sum_z p(z|x;\theta_t) \log p(x,z;\theta)$.

How do we know that maximizing E actually leads to an improvement in the likelihood? With $\theta = \theta_t$,

$$E = \sum_z p(z|x;\theta_t) \log \frac{p(x,z;\theta_t)}{p(z|x;\theta_t)} = \sum_z p(z|x;\theta_t) \log p(x;\theta_t) = \log p(x;\theta_t)$$

which is the log likelihood at θ_t. So any θ that maximizes E must lead to a likelihood that is better than the likelihood at θ_t.

5 EM with independent training examples

The EM algorithm derived above can be extended to the case where we have a training set $\{x_1, \ldots, x_n\}$ such that each x_i is independent, and they all share the same parameters θ. In this case the log likelihood is

$$D = \sum_i \log p(x_i;\theta).$$

Let the auxiliary random variables be a set $\{Z_1, \ldots, Z_n\}$ such that the distribution of each Z_i is a function only of the corresponding x_i and θ. Note that Z_i may be different for each i. By an argument similar to above,

$$D = \sum_i \log \sum_{z_i} p(x_i, z_i;\theta) \frac{p(z_i|x_i;\theta_t)}{p(z_i|x_i;\theta_t)}.$$

Using Jensen's inequality separately for each i gives

$$D \geq E = \sum_i \sum_{z_i} p(z_i|x_i;\theta_t) \log \frac{p(x_i, z_i;\theta)}{p(z_i|x_i;\theta_t)}.$$

As before, to maximize E we want to maximize the sum

$$\sum_i \sum_{z_i} p(z_i|x_i;\theta_t) \log p(x_i, z_i;\theta).$$

The E step is to compute $p(z_i|x_i;\theta_t)$ for all z_i for each i. The M step is then to find

$$\theta_{t+1} = \text{argmax}_\theta \sum_i \sum_{z_i} p(z_i|x_i;\theta_t) \log p(x_i, z_i;\theta).$$

End of the lecture on Thursday November 1.

6 EM for Bayesian networks

Let θ_0 be the current estimate of the parameters of a Bayesian network. For training example t, let v_t be the observed values of the visible nodes. The M step of EM is to choose new parameter values θ that maximize

$$F = \sum_t \sum_h p(h|v_t;\theta_0) \log p(h, v_t;\theta)$$

where the inner sum is over all possible combinations h of outcomes of the nodes that are hidden in the tth training example. We shall show that instead of summing explicitly over all possible combinations h, we can have a separate summation for each hidden node. The advantage of this is that separate summations are far more efficient computationally.

By the definition of a Bayesian network,

$$F = \sum_t \sum_h p(h|v_t;\theta_0) \log \prod_i p(X_i = x_i|\text{pa}(X_i) = \text{pa}_i;\theta)$$

where each x_i and each value in pa_i is part of either v_t or h. Converting the log product into a sum of logs, then moving this sum to the outside, gives

$$F = \sum_i \sum_t \sum_h p(h|v_t;\theta_0) \log p(x_i|\text{pa}_i;\theta).$$

For each i, the sum over h can be replaced by a sum over the alternative values x of X_i and π of the parents of X_i, yielding

$$F = \sum_i \sum_t \sum_{x,\pi} p(X_i = x, \text{pa}(X_i) = \pi | v_t; \theta_0) \log p(x|\pi; \theta).$$

Note that summing over alternative values for X_i and its parents makes sense even if some of these random variables are observed. If X_i happens to be observed for training example t, let its observed value be x'. In this case,

$$p(X_i = x, \text{pa}(X_i) = \pi | v_t; \theta_0) = 0 \text{ for all values } x \neq x'.$$

A similar observation is true for parents of X_i that are observed.

Changing the order of the sums again gives

$$F = \sum_i \sum_{x,\pi} [\sum_t p(x, \pi | v_t; \theta_0)] \log p(x|\pi; \theta).$$

For comparison, the log likelihood in Equation (1) on page 3 for the fully observed case can be rewritten as

$$\sum_i \sum_{x,\pi} [\sum_t I(x = x_{ti}, \pi = \text{pa}_{ti})] \log p(x|\pi; \theta).$$

The argument following Equation (1) says that the solution that maximizes this expression is

$$p(X_i = x | \text{pa}(X_i) = \pi) = \frac{\sum_t I(x = x_{ti}, \pi = \text{pa}_{ti})}{\sum_t I(\pi = \text{pa}_{ti})}.$$

A similar argument similar can be applied here to give that the solution for the new parameter values θ, in the partially observed case, is

$$p(x|\pi; \theta) = p(X_i = x | \text{pa}(X_i) = \pi) = \frac{\sum_t p(X_i = x, \text{pa}(X_i) = \pi | v_t; \theta_0)}{\sum_t p(\text{pa}(X_i) = \pi | v_t; \theta_0)}.$$

To appreciate the meaning of this result, remember that θ is shorthand for all the parameters of the Bayesian network, that is all the CPTs of the network. A single one of these parameters is one number in one CPT, written $p(x|\pi; \theta)$.

In the special case where X_i and its parents are fully observed, their values x_{ti} and pa_{ti} are part of v_t, and

$$p(X_i = x, \text{pa}(X_i) = \pi | v_t; \theta_0) = I(x = x_{ti}, \pi = \text{pa}_{ti}).$$

The maximum likelihood estimation method for θ explained at the end of Section 1 above is a special case of the expectation-maximization method described here.

7 Applying EM to modeling language

Section 2 above described n-gram models of language. A major issue with these models is that unigram models underfit the available data, while higher-order models tend to overfit. This section shows how to use expectation-maximization to fit a model with intermediate complexity, that can trade off between underfitting and overfitting.

The central idea is to introduce a hidden random variable called Z between the random variable W for a word and the variable W' for the following word. Specifically, the Bayesian network has edges $W \to Z$ and $Z \to W'$. The alternative values of the variable Z can be any discrete set. Intuitively, each of these values identifies a different possible linguistic context. Each context has a certain probability depending on the previous word, and each following word has a certain probability depending on the context. We can say that the previous word triggers each context with a word-specific probability, while each context suggests following words with word-specific probabilities.

Let the number of alternative contexts be c. Marginalizing out the variable Z gives

$$p(w'|w) = \sum_{z=1}^{c} p(w'|z)p(z|w).$$

This context model has $m(c-1) + c(m-1)$ parameters where m is the size of the vocabulary. If $c = 1$, the model reduces to the unigram model, while if $c = m$, the model has a quadratic number of parameters, like the bigram model.

End of the lecture on Tuesday November 6. The following M step derivation is the same as in the quiz.

The goal for training is to maximize the log likelihood of the training data, which is $\sum_t \log p(w_t, w'_t)$. (We ignore the complication that training examples are not independent, if they are taken from consecutive text.) Training the model means estimating $p(z|w)$ and $p(w'|z)$. Consider the former task first. The M step of EM is to perform the update

$$\begin{aligned} p(Z = z|W = w) &:= \frac{\sum_t p(Z = z, W = w|W = w_t, W' = w'_t)}{\sum_t p(W = w|W = w_t, W' = w'_t)} \\ &= \frac{\sum_t I(w_t = w)p(Z = z|W = w_t, W' = w'_t)}{\sum_t I(w_t = w)} \\ &= \frac{\sum_{t:w_t=w} p(Z = z|W = w_t, W' = w'_t)}{\text{count}(w_t = w)}. \end{aligned}$$

This M step is intuitively reasonable. First, the denominator says that the probability of context z given current word w depends only on training examples which have this word. Second, the numerator says that this probability should be high if z is compatible with the following word as well as with the current word.

The E step is to evaluate $p(z|w_t, w'_t)$ for all z, for each pair of consecutive words w_t and w'_t. By Bayes' rule this is

$$\begin{aligned} p(z|w, w') &= \frac{p(w'|z, w)p(z|w)}{\sum_{z'} p(w'|z', w)p(z'|w)} \\ &= \frac{p(w'|z)p(z|w)}{\sum_{z'} p(w'|z')p(z'|w)}. \end{aligned}$$

This result is also intuitively reasonable. It says that the weight of a context z is proportional to its probability given w and to the probability of w' given it.

Finally, consider estimating $p(w'|z)$. The M step for this is to perform the update

$$\begin{aligned} p(W' = w'|Z = z) &:= \frac{\sum_t p(W' = w', Z = z|W = w_t, W' = w'_t)}{\sum_t p(Z = z|W = w_t, W' = w'_t)} \\ &= \frac{\sum_t I(w' = w'_t)p(Z = z|W = w_t, W' = w'_t)}{\sum_t p(Z = z|W = w_t, W' = w'_t)} \\ &= \frac{\sum_{t:w'_t=w'} p(Z = z|W = w_t, W' = w'_t)}{\sum_t p(Z = z|W = w_t, W' = w'_t)}. \end{aligned}$$

The denominator here says that the update is based on all training examples, but each one is weighted according to the probability of the context z. The numerator selects, with the same weights, just those training examples for which the second word is w'. The E step is actually the same as above: to evaluate $p(z|w_t, w'_t)$ for all z, for each pair of consecutive words w_t and w'_t.

8 Mixture models

Suppose that we have alternative models $p(x; \theta_j)$ for $j = 1$ to $j = k$ that are applicable to the same data points x. The linear combination

$$p(x) = \sum_{j=1}^{k} \lambda_j p(x; \theta_j)$$

is a valid probability distribution if $\lambda_j \geq 0$ and $\sum_{j=1}^{k} \lambda_j = 1$. The combined model is interesting because it is more flexible than any individual model. It is often called a mixture model with k components, but it can also be called an interpolation model, or a cluster model with k clusters.

We can formulate the task of learning the coefficients from training examples using a Bayesian network that has an observed node X, an unobserved node Z, and one edge $Z \to X$. The CPT for Z is simply $p(Z = j) = \lambda_j$, while the CPT for X is $p(x|z) = p(x; \theta_z)$. The goal is to maximize the log likelihood of training examples x_1 to x_T. Marginalizing over Z, then using the product rule, shows that

$$p(x) = \sum_z p(x, z) = \sum_z p(z)p(x|z) = \sum_{j=1}^{k} \lambda_j p(x; \theta_j)$$

which is the same mixture model. The CPT of the node Z can be learned using EM. The E step is to compute $p(Z = j|x_t)$ for all j, for each training example x_t. Using Bayes' rule, this is

$$p(Z = j|x_t) = \frac{p(x_t|Z = j)p(Z = j)}{p(x_t)} = \frac{p(x; \theta_j)\lambda_j}{\sum_{i=1}^{k} \lambda_i p(x; \theta_i)}$$

The general M step for Bayesian networks is

$$p(X_i = x|\text{pa}_i = \pi) := \frac{\sum_t p(X_i = x, \text{pa}_i = \pi|v_t)}{\sum_{x'} \sum_t p(X_i = x', \text{pa}_i = \pi|v_t)}.$$

For the application here, X_i is Z and the parents of X_i are the empty set. We get the update

$$p(Z = j) = \lambda_j := \frac{\sum_t p(Z = j|x_t)}{\sum_{i=1}^{k} \sum_t p(Z = i|x_t)} = \frac{\sum_t p(Z = j|x_t)}{T}$$

where T is the number of training examples.

End of the lecture on Thursday November 8.

9 Interpolating language models

As a special case of training a mixture model, consider a linear combination of language models of different orders:

$$p(w_l|w_{l-1}, w_{l-2}) = \lambda_1 p_1(w_l) + \lambda_1 p_1(w_l|w_{l-1}) + \lambda_3 p_3(w_l|w_{l-1}, w_{l-2})$$

where all three component models are trained on the same corpus A. What is a principled way to estimate the interpolation weights λ_i? The first important point is that the weights should be trained using a different corpus, say C. Specifically, we can choose the weights to optimize the log likelihood of C. If the weights are estimated on A, the result will always be $\lambda_n = 1$ and $\lambda_i = 0$ for $i < n$, where n indicates the highest order model, because this model fits the A corpus the most closely. When testing the final combined model, we must use a third corpus B, since the weights will overfit C, at least slightly.

We can formulate the task of learning the λ_i weights using a Bayesian network. The network has nodes W_{l-2}, W_{l-1}, W_l, and Z, with edges $W_{l-2} \to W_l$, $W_{l-1} \to W_l$, and $Z \to W_l$. The CPT for Z is simply $p(Z = i) = \lambda_i$, while the CPT for W_l is

$$p(w_l|w_{l-1}, w_{l-2}, z) = \begin{cases} p_1(w_l) & \text{if } z = 1 \\ p_2(w_l|w_{l-1}) & \text{if } z = 2 \\ p_3(w_l|w_{l-1}, w_{l-2}) & \text{if } z = 3 \end{cases}$$

The goal is to maximize the log likelihood of the tuning corpus C. Marginalizing over Z, then using the product rule and conditional independence, shows that

$$p(w_l|w_{l-1}, w_{l-2}) = \lambda_1 p_1(w_l) + \lambda_1 p_1(w_l|w_{l-1}) + \lambda_3 p_3(w_l|w_{l-1}, w_{l-2})$$

as above. To learn values for the parameters $\lambda_i = p(Z = i)$, the E step is to compute the posterior probability $p(Z = i|w_l, w_{l-1}, w_{l-2})$. Using Bayes' rule, this is

$$p(Z = i|w_l, w_{l-1}, w_{l-2}) = \ldots$$

The M step is to update λ_i values. The general M step for Bayesian networks is

$$p(X_i = x|\text{pa}_i = \pi) := \frac{\sum_t p(X_i = x, \text{pa}_i = \pi|v_t)}{\sum_{x'} \sum_t p(X_i = x', \text{pa}_i = \pi|v_t)}.$$

For the application here, training example number t is the word triplet ending in w_l, X_i is Z, and the parents of X_i are the empty set. We get the update

$$p(Z = i) := \frac{\sum_l p(Z = i|w_l, w_{l-1}, w_{l-2})}{\sum_{j=1}^3 \sum_l p(Z = j|w_l, w_{l-1}, w_{l-2})} = \frac{\sum_l p(Z = i|w_l, w_{l-1}, w_{l-2})}{L}$$

where L is the number of words in the corpus.