Maximum Entropy Modelling

sgp@clg.ox.ac.uk

Some preliminaries: the 'expectation' for a (discrete random) variable X, written $E(X)$, is the sum of each possible value for X, weighted by the probability of that value occurring:

$E(X) = \sum_x x * P(X = x)$

The textbook example is the expectation for a dice: each of the possible values 1-6 for X is equally likely, i.e. $P(X = 2) = 1/6, P(X = 6) = 1/6$, etc. So the expectation for X is:

$$1 * P(X = 1) + 2 * P(X = 2) + ... + 6 * P(X = 6)$$
$$1/6 + 2/6 + 3/6 + 4/6 + 5/6 + 6/6 = 21/6 = 3.5$$

You can think of expectation as a kind of average expected value.

Entropy

Remember some things about logarithms:

$log_a\, x = y \;\leftrightarrow\; x = a^y$

$log_a\, xy = log_a\, x + log_a\, y; log_a\, \frac{x}{y} = log_a\, x - log_a\, y$

$log_a\, 1 = 0\ so\ log_a\, \frac{1}{x} = -log_a\, x$

If there are N events all equally probable, then the number of bits needed to decide which one you have is $log_2\, \frac{1}{N} = -log_2 N$.

The $\frac{1}{N}$ comes from the probability of each event.

The log is because if there are N events then you need a series of binary (yes/no) choices to recognise each member. E.g. if there are two objects you need one choice. If there are 3 or 4 you need at most two choices, 5-8 at most three choices, etc. The number of choices for N will be the number of times you need to multiply 2 by itself to get N, or alternatively, the number of times you have to divide N (in the worst case) into two sets of objects until you get a set containing just the one you want. So remembering the definition of a log the number of choices you need is $log_2 N$.

Now if we have a random variable X, which can take any number $x_1 ... x_n$ of values, possibly with differing probabilities, we can define the entropy of X, usually written H(X), as:

$$H(X) = -\sum_{i=1}^{n} p(x_i)\, log_2\, p(x_i)$$

The term $log_2\, p(x_i)$ is motivated by the same reasoning as above - the number of bits you need to select that event - except that since there are values of X with different probabilities we won't uniformly have $\frac{1}{|X|}$ but a different fraction for each value, its probability.

We also have to weight the number of binary choices by the same probability (because we want more highly probable values to need fewer choices) which gives us the first $p(x_i)$. The sum $\sum$ is there so we take into account each value x of X. The minus sign is there so that we end up with a positive number (because the log of a probability between 0 and 1 will be a negative number).

Conditional Entropy

You can think of the entropy of some probability distribution as a measure of how predictable it is. High entropy = low predictability.

Sometimes we need to deal with conditional entropy: i.e. the entropy of Y given that X is known:

$H(Y \mid X) = \sum_{x \in X} p(x) H(Y \mid X = x)$

$H(Y \mid X) = -\sum_{x \in X} p(x) \sum_{y \in Y} p(y \mid x) log\, p(y \mid x)$

$H(Y \mid X) = -\sum_{x \in X, y \in Y} p(x) p(y \mid x) log\, p(y \mid x)$

An example

We have some events: the, cat, sat, on, the, mat. Assume these words are all equally likely, although this will not be true in practice. We will have a variable X which can take two values, 1 if a word contains a 't' and 0 otherwise. Clearly P(X=1) = 5/6, and P(X=0)=1/6 Now the expectation of X, E(X), with respect to our event space is:

$$E(X) = \sum_x x * P(X = x)$$

= 0*1/6 + 1*5/6 = 5/6.

The entropy of this distribution is:

$$H(X) = -\sum_{i=1}^{n} p(x_i)\, log_2\, p(x_i)$$

$$H(X) = -((1/6*log_2 1/6)+(5/6*log_2 5/6)) = -(-0.43+-0.22) = 0.65$$

Now take X to have values 1 if a word contains an 'e', 2 if a word contains an 'a', and 3 otherwise (note that these are mutually exclusive events). Now the probabilities and expectation will be:

$$P(X = 1) = 2/6, P(X = 2) = 3/6, P(X = 3) = 1/6,$$

$$E(X) = 1 * 2/6 + 2 * 3/6 + 3 * 1/6 = 11/6 = 1.83,$$

and the entropy will be:

$$H(X) = -((2/6 * log_2 2/6) + (3/6 * log_2 3/6) + (1/6 * log_2 1/6)$$

= -(-0.52 + -0.5 + -0.43) = 1.45

Naive Bayes Classifiers

Now consider if we have some event space, and features like those above, and we wish to build a classifier which given new events will assign them to the appropriate class. In the so-called 'naive bayes' approach, to do this we choose a set of features that we suspect are good indicators of the presence of the category. We want to calculate, for each category, its conditional probability given the features present. By Bayes' Theorem:

$$P(Cat \mid Feat_1, ..., Feat_n) = \frac{P(Feat_1 ... Feat_n \mid Cat) * P(Cat)}{P(Feat_1 ... Feat_n)}$$

Since the features are the same for each different category given a particular event, the denominator is constant and we can ignore it. Now the numerator is equivalent to the joint probability of the category and the features:

$$P(F_1, \ldots, F_n, C)$$

By the chain rule, this is equivalent to:

$$P(F_1 \mid F_2, \ldots, F_n, C) * P(F_2 \mid F_3, \ldots, F_n, C) \ldots P(F_n \mid C) * P(C)$$

Now if we assume that every feature is independent of every other feature (the 'naive' part), this will be equivalent to:

$$P(F_1 \mid C) * P(F_2 \mid C) \ldots P(F_n \mid C) * P(C)$$

We can now estimate P(C) for each C by counting the relative frequency of C in the training corpus, and estimate $P(F \mid C)$ as $\frac{|F,C|}{|C|}$, making sure that we smooth appropriately to avoid having zero probabilities.

Given a new event with a set of N features, we want to find the most likely category with respect to those features, i.e.

$$Max_{Cat}[P(Cat) \prod_{i=1}^{n} P(Feat_i \mid Cat)]$$

Non-independent features

Usually the assumption that the features are independent of each other are safe. But sometimes it isn't: one case that arises frequently in practice is where features are automatically generated.

An example might be a classifier for guessing the part of speech of unknown words. Features like 'begins with uppercase' etc. usually combined with information about prefixes and suffixes: pre-, -ing, etc. Typically these (for English) would be of length 1-4:

1 -s, a-, 2 un-, -ed, 3 pre-, -ing, 4 anti-, -ment, etc.

These are generated automatically and so in training we will get features like -ing, -ng, and -g. But these are not completely independent: any word that has suffix -ing will also have -ng and -g. This can lead to incorrect classifications

Maximum Entropy

* need to combine information from different sources, possibly with complex overlapping dependencies.

* produce an overall probability distribution which reflects training data, but which makes no further assumptions about interdependencies.

* therefore we must find the distribution which has maximum entropy with respect to the training sample.

If A is the set of possible classes/labels, and B the set of possible features, then we want to find the conditional distribution *p* that gives the largest value for:

$$H(A \mid B) = - \sum_{a \in A, b \in B} p(b) p(a \mid b) \, log \, p(a \mid b)$$

Note that this requires us to look at every combination of features and classes: for some values of b we can make estimates of p(b) from the training data, but for others we cannot. We want the final distribution to be consistent with the observed estimates, and to be 'evenly spread' where we have no evidence.

Feature representation

In the ME framework we usually represent features as 'indicator functions': functions from events to $\{0,1\}$.

f_1 = if word ends in t and is N then 1, else 0.
f_2 = if word ends in t and is V then 1, else 0.
...
f_2 = if word is preceded by 'the' and is V then 1, else 0.
etc.

Now the requirement that the final distribution is consistent with the observed evidence can be stated in terms of the expectation of a feature: $\boxed{E_p f_j = E_{\tilde{p}} f_j}$, where $\boxed{E_p f_j}$ is the expectation of the feature in the distribution or model we are computing, and $\boxed{E_{\tilde{p}} f_j}$ is the expectation we observed in the training set. We require our conditional distribution to be consistent with these constraints, and to maximise:

$$\boxed{H(A \mid B) = -\sum_{a \in A, b \in B} \tilde{p}(b) p(a \mid b) \, log \, p(a \mid b)}$$

- where $\tilde{p}(b)$ is the empirical probability from the training set (we can't estimate from things we haven't seen).

Max Ent distribution

So of all the distributions that satisfy $E_p f_j = E_{\tilde{p}} f_j$ for every feature j, we want the one that maximises the entropy. It can be shown (I won't: see Berger ref) that there is a unique distribution, call it p*, that satisfies these constraints, and it is of the form:

$p^*(a \mid b) = \frac{1}{Z(b)} \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}$

k is the number of features, and Z(b) is a normalising factor to make sure that we get a genuine probability distribution, i.e.

$Z(b) = \sum_a \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}$

The α_j are weighting factors, and there is one for each feature. In probspeak, they are the 'parameters' for the 'model' $p(a \mid b)$.

What does this formula mean?

Our features all have 0 or 1 as their value. So this means that if some feature *j* does not apply to x, its value will be 0. In this case the value of $\alpha_j^{f_j(a,b)}$ will be 1, since anything to the power of 0 is 1. If the value of the feature is 1, then the value of $\alpha_j^{f_j(a,b)}$ will just be the value of α_j, again because any quantity *q* to the power of 1 is just *q*. So the expression $\prod_{j=1}^{k} \alpha_j^{f_j(a,b)}$ will just be the product of the weights of the features that apply to (a,b).

Since the weights themselves are only indirectly related to our initial probability estimates (they derive from the 'iterative scaling' method described later) we must make sure that the value we get can be interpreted as a probability distribution, and so we scale it with the normalising factor Z(b). Z(b) is just the sum of the values of the expression $\prod_{j=1}^{k} \alpha_j^{f_j(a,b)}$ for **all** values of a:

$$Z = \sum_a \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}$$

Log linear models

ME models are often called 'log linear' models. Looking back again at the definition of logarithms, we can see that this is because when we take the logarithm of both sides of the equation defining p* we get:

$log\, p^*(x) = -log\, Z + \sum_{i=1}^{k} f_i(x) log\, \alpha_i$

Again the fact that the values of f_i will be 0 or 1 means that only those features that apply will get counted, and $log\, p^*(x)$ will be a linear combination* of the logs of the weights (and the scaling factor). As usual with anything to do with probabilities, implementations use logarithms rather than the original numbers to avoid computational floating point limitations.

*Linear combination? A linear combination is a sum of the elements from some set with constant coefficients placed in front of each. E.g. $2x + 3y + z$ is a linear combination of x, y, z.

Exponential form

There is an alternative formulation of a max ent distribution which is often used:

$$p(a \mid b) = \frac{e^{\sum_i \lambda_i f_i(a,b)}}{Z(b)}$$

Original: $p(a \mid b) = \frac{\prod_i \alpha_i^{f_i(a,b)}}{Z(b)}$

$p(a \mid b) = \frac{e^{log(\prod_i \alpha_i^{f_i(x)})}}{Z(b)}$ because $e^{logf} = f$

$$p(a \mid b) = \frac{e^{\sum_i log(\alpha_i^{f_i(a,b)})}}{Z(b)}$$

$p(a \mid b) = \frac{e^{\sum_i f_i(a,b) log(\alpha_i)}}{Z(b)}$ since $log\, x^k = k.logx$

$$p(a \mid b) = \frac{e^{\sum_i \lambda_i f_i(a,b)}}{Z(b)}$$

where $\lambda_i = log(\alpha_i)$

Generalised Iterative Scaling

How do we determine the right values for the weights, α_j? The original GIS algorithm requires that the sum of the feature values for each possible event x (=(a,b)) should be equal to some constant C:

$\forall x.\ \sum_{i=1}^{k} f_i(x) = C$

Define C to be the biggest possible value for $\sum_{i=1}^{k} f_i(x)$. Then define a new 'correction' feature, f_{k+1} such that:

$\forall x.\ f_{k+1}(x) = C - \sum_{i=1}^{k} f_i(x)$

Although we need C, it turns out that we do not need the correction feature (which would be the only feature with a value other than 0 or 1).

1. There is a version of the algorithm called Improved Iterative Scaling which does not need the correction feature (Della Pietra ref)

2. Curran and Clark (see ref) gave a proof that even Generalised Iterative Scaling does not need the feature: the scaling procedure will still converge, and the resulting model is not significantly different to one computed with the feature.

Iterative scaling: Initialisation

The training set will be a set of samples of the form:

$$\{\langle \text{ClassLabel}_1, \text{Context}_1\rangle \ldots \langle \text{ClassLabel}_N, \text{Context}_N\rangle\}.$$

Several different features might apply to the Context: the ClassLabel will be something like 'N', or 'V', depending on the task.

First compute the empirical expectation of each feature from the training set. Remembering the definition of expectation this will be: $\boxed{E_{\tilde{P}} f_i = \sum_x \tilde{P}(x) f_i(x)}$ where $\tilde{P}$ is the empirical probability. In this case, the expression will be equivalent to: $\boxed{E_{\tilde{P}} f_i = \frac{1}{N} \sum_{j=1}^{N} f_i(x_j)}$ where N is the size of the training sample, because of the fact that the ME features return either 0 or 1. So this is the average number of times a feature fires in the training data.

Now calculate $\boxed{C = max_{x_j} \sum_{i=1}^{k} f_i(x_j)}$, i.e. C is the largest number of features that fire for some member of the training set.

Then set initial values for each $\boxed{\alpha_i^0}$ as some number, say 1. (0 if you are doing the log version.)

Main Loop

Repeat until convergence:

$$\alpha_i^{(n+1)} = \alpha_i^n (\frac{E_{\tilde{P}} f_i}{E_{P^n} f_i})^{\frac{1}{C}}$$

or the log version:

$$\alpha_i^{(n+1)} = \alpha_i^n + \frac{1}{C} log \frac{E_{\tilde{P}} f_i}{E_{P^n} f_i}$$

'Convergence' here can be interpreted to mean either 'after some fixed number of iterations' or 'when the change in successive values falls below some threshold'.

We saw the formula for the empirical expectation on the previous slide. The tricky thing is the computation of the model expectation: $\boxed{E_P^n f_i = \sum_x P^n(x) f_i(x)}$, where $\boxed{P^n(x) = \frac{1}{Z} \prod_{j=1}^k (\alpha^n)^{f_j(x)}}$.

Approximating the model expectation

Unlike the empirical expectation, the model expectation requires a summation over the probabilities for all possible feature-class pairs, but we only have estimates for those we have seen. We have to approximate, and for this we look at the empirical probabilities of the features (in the usual sense) and the conditional probability of the class label given the other features from the current iteration.

$$E^n f_i \approx \sum_{j=1}^{N} \tilde{P}(b_j) \sum_a P^n(a \mid b_j) f_i(a, b_j)$$

What does this formula say? $\boxed{\sum_{j=1}^{N} \tilde{P}(b_j)}$ sums over the empirical probability of each feature, i.e. the relative frequency of that feature in the training data, multiplied by the sum, for each class label a, of the value of the term $\boxed{P^n(a \mid b_j) f_i(a, b_j)}$. $\boxed{P^n(a \mid b_j)}$ is the conditional probability of the class label a given feature b and is computed in the same way as P^* given earlier.

So each time round the loop we change the weights a little.

Note that only the items in the training sample for which the feature is 'active', i.e. gives value 1, will contribute to the summation. The complexity of each iteration is therefore going to be no worse than:

|training data| * |class-labels| * |avge-active-features|

but of course, for a problem with a large training set, a large number of labels (e.g. POS tagging) and a large number of features this may still be quite slow. In general some 100s of iterations may be needed.

Maximum Entropy modelling is one of the most widely used techniques in Computational Linguistics. It has been used to build POS taggers, chunkers, and parsers, and is also used as a classifier for disambiguation (distinguishing good from bad parses), document classification, and many other tasks. An extremely good introduction to the technique and some applications of it can be found in Ratnaparkhi's PhD thesis, to be found at:

`http://www.inf.ed.ac.uk/resources/nlp/local_doc/mxpost_thesis.pdf`

References

A maximum entropy approach to natural language processing Adam Berger, Stephen Della Pietra, and Vincent Della Pietra Computational Linguistics, (22-1), March 1996;

S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. IEEE Transactions on pattern analysis and machine intelligence, 19(4), 380-393, April, 1997

James R. Curran and Stephen Clark, 2003, Investigating GIS and Smoothing for Maximum Entropy Taggers, EACL, pp91-98, `http://acl.ldc.upenn.edu/E/E03/E03-1071.pdf`.

A. Ratnaparkhi 1996 A maximum entropy model for POS tagging, ACL proceedings,
`http://acl.ldc.upenn.edu/W/W96/W96-0213.pdf`

C. Manning and F Schütze: Foundations of Statistical NLP, Section 16.2, NB the version in the book is buggy. The corrected version is at:
`http://nlp.stanford.edu/fsnlp/class/fsnlp-new-maxent.pdf`

Raymond Lau's thesis motivates the model expectation approximation: `http://www.raylau.com/SMThesis.pdf`

Class Exercise
to be handed in to Comlab reception
by Friday of week 5

Read the section (16.2) on Maximum Entropy Modelling in Manning and Schütze's Foundations of Statistical Natural Language Processing. Note that there is an amended version of this available at:

`http://nlp.stanford.edu/fsnlp/class/fsnlp-new-maxent.pdf`

Look at their small document classification example and try to do as many of the associated exercises 16.7 - 16.10 as you have time for.