Foundations of Statistical Natural Language Processing
Notes and Exercises

Chris Merck

July 12, 2011

2 Mathematical Foundations

Notes

- *cross entropy* – In their discussion of cross entropy, the authors do not say, at least in words, what they mean by cross entropy. We might take the first line of (2.46) to be a definition, but this makes Ex 2.13 seem odd because it asks the reader to prove that line! So, either they meant to say "motivate" or "explain" the first line of (2.46) rather than "prove", or they forgot to give a definition in words such as: "The cross entropy is the average number of bits per element needed to encode a message from X using an encoding optimized for q."

Exercises

2.1 Set Theoretic Probability Proofs

I'll just give the more interesting proofs:

a. The Addition Rule: easy.

b. $P(\emptyset) = 0$:
$1 = P(\Omega) = P(\Omega \cup \emptyset) = P(\Omega) + P(\emptyset) - 0$
$\therefore$ Q.E.D.

c. $P(\bar{A}) = 1 - P(A)$: easy.

d. $A \subset B \implies P(A) \leq P(B)$:
$P(B) = P(A \cup (B - A)) = P(A) + P(B - A)$
$P(B - A) \geq 0$ by the type of P
$\therefore$ Q.E.D.

e. $P(B - A) = P(B) - P(A \cap B)$: easy.

2.9 Entropy of Character Frequency Distribution

The entropy of a distribution p is given by

$$H(p) = \sum_x p(x) \log(p(x)),$$

which is computed for *Sense and Sensibility* by the `ex2.9-10.py` program and found to be 4.08 bits.

2.10 Kullback-Leibler Divergence

The KL divergence from a distribution p to a distribution q is given by

$$D(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)},$$

which is computed between two texts using first-, second-, and third-order Markov models. These computations are among those performed by the `ex2.9-10.py` program, the output of which is given below.

Note the order of magnitude increase in KL divergence with each increase in the order of the Markov models. This may be due more to the lower precision of the higher-order models due to the sparsity of training data than to an difference in the 'true' KL divergences.

```
[cmerck@caladan ch2]$ python2 ex2.9-10.py

Loading: 'Pride and Prejudice'...
 cleaned text length: 645822 chars
Loading: 'Sense and Sensibility'...
 cleaned text length: 632831 chars

Training n-Gram Markov Models using 'Pride and Prejudice'
. . . . . . . . . . . . . . . . . . DONE!

Computing Entropy of 'Sense and Sensibility' using those Models
model         entropy (bits/char)
zeroth-order  4.75
first-order   4.08
second-order  3.70
thrid-order   3.36

Computing KL Divergence between Texts
 Using First-Order Models:
  KLDiv(pap||sas) = 0.004186
  KLDiv(sas||pap) = 0.002258
 Using Second-Order Models:
  KLDiv(pap||sas) = 0.025596
  KLDiv(sas||pap) = 0.019338
 Using Third-Order Models:
  KLDiv(pap||sas) = 0.105399
  KLDiv(sas||pap) = 0.092103
```

2.11 A Noisy Channel Model of Word Sense Disambiguation

Using the noisy channel model from *Foundations*,

$$I \xrightarrow[p(o|i)]{\text{``noisy channel''}} O \xrightarrow[?]{\text{decoder}} \hat{I},$$

take the following identifications:

- $I =$ the true, unambiguious word senses (i.e., meanings),
- $O =$ the words, as they appear in the document,

- $\hat{I}$ = the disambiguated word senses,
- $p(o|i)$ = the author's diction.

2.12 Asymmetry of the KL Divergence

Take

$$X = \{a, b\}, p(a) = 1, p(b) = \epsilon, q(a) = 1/2, q(b) = 1/2.$$

Straight-forward computation shows that $D(p||q) = 1$ and $D(q||p) =$ positive infinity.

The interpretation of $D(p||q) = 1$ is that using an encoding optimized for q to encode data with distribution p wastes a whole bit per character. This makes sense because it should require no bits at all[1] to encode characters with distribution p, since we only have one character, a.

What $D(q||p) = +\infty$ tells us is that we cannot use an encoding optimized for p to encode characters from q. This makes sense because p does not contain the character b, and so an optimal encoding for p will make no provisions to encode b, which *is* found in q.

2.13 Cross Entropy

Given a random variable X (over, say, utterances) and a pmf q (constructed, say, from training data), we wish to define a so-called *cross entropy* of X in q,[2] written $H(X, q)$. Although the authors' do not say it, this cross entropy value should be the number of bits per utterance needed to encode a message from X using an encoding optimized for q.

Intuitively, the cross entropy will be the inherent entropy of X plus the bits wasted due to using the not-quite-right encoding. We use the interpretation of KL divergence to write the second term, obtaining

$$H(X, q) = H(X) + D(p||q).$$

Using the definitions of $H()$ and $D(||)$, the $\log(p)$ terms cancel, and we are left with

$$\sum_x p(x) \log \frac{1}{q(x)}.$$

2.14 Nonergodic and Nonstationary Properties of Natural Languages

In the context of SNLP, we are generally concerned with corpa. I assume an infinitely long corpus, i.e. an algorithm for appending to a corpus new text as it is published.

We can see that the process which may be thought of as generating this corpus is nonstationary because of changes in vocabulary and grammar over time. It is also nonstationary in that it contains discontinuities at the end of each text.

Nonergodicity also follows from language change. However, we must more clearly define a *state of the process* first. We could trivially define states as the words (or characters) making up the text of the corpus. Under this definition the obsolescence of a word (or character) seals off that state from ever being visited again, making the process nonergodic.

Alternatively, we could take the more intuitive stance, thinking of the state of the process as the state of mind of the author at the time of composition. The interpretation of nonergodicity in this case is the possibility of falling into a blackhole of thought – no doubt an interesting and debatable topic, but outside the scope of this response!

[1] Technically, there is information carried by the length of the text. For the purposes of SNLP we can safely ignore this. However, a serious study of information theory cannot. See Chatin's *MetaMath!* [2]; look up *self-delimiting information* in the index.

[2] I dislike the authors' expression "cross entropy between X and q" because it suggests that the arguments X and q are interchangeable. So, I use "of ... in" instead.

2.15 A Reproduction of Shannon's Experiment

The reproduction of Shannon's experiment divides neatly into two tasks. The first is to write a computer program implementing the mechanics of loading a text snippet into memory, taking input from the test subject, and providing feedback to the subject as to whether their inputs were correct. The second portion of the exercise is the interpretation of the results – especially how to compute entropy.

Mechanically, the program was implemented in Python as `shannon.py` and operates in text-mode using the `curses` Python package. A typical screenshot is given below.

```
              *** Shannon's Experiment ***

         Try to guess the next letter of text.
           Please only use A-Z or SPACEBAR.

    ##############################################
    ###nly serve, after what had since passed, ###
    ##############################################

      Progress: 46%                  Score: 53
                                     Entropy: 0.88
```

The more difficult aspect of this project is the question of how to interpret the experimental data so as to arrive at an entropy measurement (or more correctly, a measured upper-bound on entropy).

Manning and Schuetze define the "average surprise" as the average of the pointwise surprises experienced by the model when it is run over a text. We use the following notation. A model m, given a history h of a text up to a certain point, is a pmf in the next element w, where $m(w|h)$ denotes the probability that the model m assigns to the element w given history h. The *pointwise entropy* (pointwise entropy) is

$$H(w|h) = -\log m(w|h).$$

Given this notation, the average surprise may be written

$$H_{\text{avg}} = \frac{1}{n}\sum_{j=1}^{n} \log m(w_j|w_1, w_2, ..., w_{j-1}).$$

This is what might be called a "white box" test of the model m, because computation of H requires complete knowledge of m, i.e. the actual probabilities, and therefore the actual level of surprise, experienced by the model as it is exposed to text. This is not the case in Shannon's experiment however. A single Shannon test subject contains the model within themselves, and the computer program receives only a keystroke's worth of information per character of text from the subject. Therefore, we cannot directly apply the formula for H_{avg} given above as we could to a computer model.

At this point we should decide how the model is used by the subject. If the subject were perfectly rational and had complete access to the model m, then given a history h, the test subject will choose what by their model is the most likely candidate next character

$$w_{\max} = \text{argmax}_w m(w|h).$$

So, under this interpretation of the subject-model system it is only the function $w_{\max}(h)$ which is transmitted from the subject to the computer program and not the entire model m.

We might try to approximate m by recording responses from a number of test subjects using the same text. Realistically, different subjects will give different responses to the same text. Keeping the $w_{\max}$ interpretation, this evidence leads us to believe that the models of the various subjects are not precisely the true (or mean) model m but rather some deviation therefrom $m + \Delta m$. Unfortunately, we are still no better off. Although $m + \Delta m$ may yield different w_{max}s in some cases, the argmax function prevents us from reproducing m from these data. It would certainly be convenient if the subjects instead stochastically evaluated their models, making a response w with probability $m(w|h)$. If that were the case, then the data from a large enough test group, combined with smoothing could yield a measured m which could be passed to the equation for H_{avg}, solving our conundrum.

Many-subject possibilities notwithstanding, due to limited resources (e.g. lack of test subjects), I will take a different route to measuring the entropy upper-bound which requires but a single subject. I model the subject as guessing, on the jth occasion of being asked for input, between n_j equally expected[3] candidates, one of which is correct. The probability of guessing correctly is $p_j = 1/n_j$ or, in terms of the number b_j of bits of uncertainty, $p_j = 2^{-b_j}$. In a particular run, let r_j be 1 if the subject correctly guessed the jth character and 0 otherwise. If the run is long (say, length $N >> 1$), then $\bar{r}$ approximates $\bar{p}$:

$$\bar{r} = \frac{1}{N}\sum_j r_j \approx \bar{p} = \frac{1}{N}\sum_j 2^{-b}.$$

It is tempting to assume now that all characters are equientropic (i.e. b is constant), so that we can apply a logarithm over the summation and simplify to

$$b \approx -\log \bar{r}. \;\; (*)$$

However, it seems reasonable that some characters have higher true pointwise entropy than others, and, moreover, from experience with the experiment it is clear that the pointwise surprise varies greatly – some characters being easily guessed and others near impossible to guess. So, we must at least consider the effect of the unequal distribution of entropy on the experiment.

It is impossible for us to know the actual b_j values; however, we might assume that they are distributed log-normally. Even if we do not know the standard deviation σ_b of the distribution, it is easy to show that given fixed $\bar{b}$, increasing σ_b will increase $\bar{r}$. That is, estimates of entropy made by assuming $\sigma_b = 0$ will be too low. This is disastrous for us, because we want an upper-bound on entropy.

Abandoning that approach, we propose third formula for H_{avg}. When the subject correctly guesses a character, assume that the surprise is zero. When the subject incorrectly guesses however, assume that the subject guessed the character at random among the 27 possible characters and so the surprise is the same which would be experienced by a zeroth-order Markov chain: 4.76 bits (i.e. $\log 27$). In symbols, we have

$$H_{\text{avg}} = (1 - \bar{r}) \log 27.$$

This formula does not provide a very tight upper-bound, but it is robust. This is the formula used by the `shannon.py` program.

A tighter upper-bound could be obtained by augmenting the experiment with a good computer predictive model (say, a Markov chain) and providing the machine's prediction to the subject. However, this "assisted Shannon experiment" is here out of scope.

2.16 'Easy' and 'Hard' Texts

On a 180-character excerpt of *Pride and Prejudice*, which I consider an easy text because of the regularity of the Austin's English and my familiarity with period style, I correctly guessed the next character 60% of the time, which corresponds to an entropy of no more than 1.9 bits per character in my reckoning (as justified above). That this number is somewhat higher than Shannon's result of 1.3 is most likely my choice of a conservative formula for H_{avg}.

On a 200-character excerpt of the introduction to a neuropsychopharmacological paper I fared even better, with 62% correct. Difficulty with the few acronyms and technical terms in the text was offset by the

[3] I take 'expectation' as the antonym of 'surprise'.

relative predictability of the scientific writing style – or perhaps by my passing familiarity with the pharma industry.

So, it seems that for prose we have a 60% predictability. I did not put myself through the pain of running the experiment over poetry, but I imagine that the subjects would not fare so well in that case!

References

[1] Manning and Schuetze, *Foundations of Statistical Natural Language Processing*, The MIT Press, Cambridge (1999).

[2] Chatin, G., *MetaMath! The Quest for Omega*, `http://www.umcs.maine.edu/~chaitin/omega.html`, Random House (2005).