CS224N/Ling 237 Homework #2

Due: Wed, Apr 30, 2003

This assignment begins with written questions, but also includes practical questions involving working with corpora and Unix tools and part-of-speech taggers. The total is 30pts.

1. **Zipf's Law (3pts).** Zipf's law states that if we rank words according to decreasing frequency, then the product of a word's rank r and its frequency f will be constant. Since frequency is proportional to probability, this can be stated as $p(w)r(w) = k$ for some constant k. Zipf claimed that this law holds for text generated according to natural language processes.

   Let S be an alphabet of N symbols. Let X be a uniform random process that produces each symbol of S, plus a space symbol, with uniform probability $1/(N+1)$. X will thus generate text where words w over S have a certain probability $p(w)$ of occurring. If we rank words according to p, however, there will be huge stretches where the ranking r is arbitrary, since all words of equal length have equal probability. Let $r'(w)$ be the average rank that a word of length $|w|$ would be assigned according to r.

   (a) Show that, in the limit (as word length ℓ increases), "Zipf's law" holds for text randomly generated according to X. In particular, show that in the limit there is a linear relationship between $\log p(w)$ and $\log r'(w)$.

   (b) What is the slope of the line in the limit?

2. **I know there was some reason why I learned calculus (3pts).** The simplest estimate of a probability distribution is the relative frequency estimate, where we give events a probability equal to their relative frequency in the training data. This estimate is also called the *maximum likelihood estimate* since it is the estimate that makes the data most likely. Let X be a random variable representing a possibly unfair coin flip. In this case, the model has only one parameter: given p_H, the probability of a flip coming up heads, we also know the probability of tails $(1 - p_H)$. Let's say we flip the coin N times, and get the data $D = \{d_1, d_2, ...d_n\}$ of N_H heads and $N_T = N - N_H$ tails, then the MLE will be that $p_H = \frac{N_H}{N}$.

   (a) Prove that this is the value of p_H which makes D most likely.

   (b) Considering such an example, briefly explain why it is not surprising that MLE leads to estimates which overfit the training data, and why we might want to smooth, or make use of a prior distribution.

3. **Smoothing probability models (3pts).**

   In the absolute discounting model of smoothing, all non-zero MLE frequencies are discounted by a constant amount δ:

Absolute discounting: If $C(w_n|w_1 \cdots w_{n-1}) = r$,

$$P_{\text{abs}}(w_n|w_1 \cdots w_{n-1}) = \begin{cases} (r-\delta)/N & \text{if } r > 0 \\ \frac{(V-N_0)\delta}{N_0 N} & \text{otherwise} \end{cases}$$

(Recall that V is the size of the vocabulary, N is the total number of times $w_1 \cdots w_{n-1}$ has been seen, and N_0 is the number of word types that were unseen after this context.)

Whereas under linear discounting the estimates count of seen words is discounted by a certain percentage, for example 5%:

Linear discounting: If $C(w_1 \cdots w_n) = r$,

$$P(w_1 \cdots w_n) = \begin{cases} (1-\alpha)r/N & \text{if } r > 0 \\ \alpha/N_0 & \text{otherwise} \end{cases} \tag{1}$$

(a) Show that absolute discounting yields a probability model. In particular show that $\sum_{x \in V} P_{\text{abs}}(x) = 1$, and discuss what conditions are needed on δ.

(b) Show that linear discounting also yields a probability model.

(c) Which of these is a better estimator for natural language data (such as for estimaing $P(w_3|w_1, w_2)$ in the *comes across* __ example)? Explain your answer with a concrete example.

4. **More linguistic issues (3pts).**

(a) Briefly discuss the validity of the "one sense per discourse" constraint for different types of ambiguity (types of usages, homonyms, etc.). Discuss cases where the constraint is expected to do well and cases where it is expected to do poorly.

(b) The n-gram models we've shown make the $(n-1)$-order Markov assumption, i.e. that the distribution of words depends only on the previous $n-1$ words.

i. Discuss (briefly) several ways in which this assumption is false for natural language. What properties of language will it not capture?

ii. Despite the above, n-gram models are remarkably good at predicting the next word. Discuss why this might be. What information *is* in the previous word(s) that makes these models perform so surprisingly well?

5. **HMMs (9 pts).** Consider the HMM with two states, X and Y, and an output alphabet {x, y}. The start-state probabilities, state transition probabilities and symbol emission probabilities are all unknown. This HMM is observed to give the output 'x' at $t = 1$, and 'y' at $t = 2$.

(a) We want to try to learn the probability parameters for this HMM from the output sequence observed. To do this, we can use the EM algorithm for parameter estimation. The particular case of the EM algorithm for HMMs is also known as the Forward-Backward algorithm. To use EM, we must start with some initial "guess" values of the parameters, and then run successive iterations of EM with the output sequence observed, to refine our estimates of the parameters. We stop iterating when the parameter values converge.

Use EM to estimate the parameters for this HMM, for each of the following three initial sets of starting parameters (where Π is the start state probabilities, A is the state transition probabilities, and B is the symbol emission probabilities). A good way to do this and see/learn what is going on is to make an Excel spreadsheet that does the calculations for one round of EM. If you then (by hand) copy across the new parameter values, the spreadsheet will then do the next round of EM.

Run EM for 2 iterations, or until the parameters converge, whichever comes first:

(i)

Π		
	X	1
	Y	0

A	X	Y
X	0	1
Y	1	0

B	x	y
X	1	0
Y	0	1

(ii)

Π		
	X	0.5
	Y	0.5

A	X	Y
X	0.5	0.5
Y	0.5	0.5

B	x	y
X	0.5	0.5
Y	0.5	0.5

(iii)

Π		
	X	0.5
	Y	0.5

A	X	Y
X	0.5	0.5
Y	0.5	0.5

B	x	y
X	0.9	0.1
Y	0.2	0.8

(b) What can you observe from these results about the effect of different initialization choices on the outcome of the values of the parameters in EM learning? Discuss with reference to the differences in the three results obtained above.

(c) Now assume that the data sequence is actually being generated by an HMM whose parameters are exactly the ones used in the initialization condition in (5a.i) above. However, when running EM for part (5a.iii), you would notice that not all of the parameters moved closer to the true values of the generating HMM. Explain why this is the case, and what would need to change so that the parameters all improve with successive iterations of EM on this HMM.

(d) Specify all other parameter settings that would assign the same likelihood to the observation sequence as the HMM whose parameters are exactly the ones used in the initialization condition in (5a.i) above.

(e) Does the class of HMM models we are considering define a probability distribution over all strings $s \in (x|y)^*$? (**Hint** Consider the probability mass assigned to strings of different lengths). If not, how can we extend the models to do so? In what applications might it be important that our model define a distribution over observation sequences of all lengths?

(f) Parameter tying is useful for reducing the number of parameters of a model and can be very helpful for improving performance. Consider tying the emission probabilities for states X and Y in the HMM model considered here (tying the parameters means adding in a constraint that they be equal, i.e., $P(x|X) = P(x|Y)$ and $P(y|X) = p(y|Y)$). Write out the expression for the likelihood of the sequence xy in terms of the parameters. Specify the parameters of a maximum likelihood HMM model (there may not be a unique one)..

6. *Practical*. **Corpus munging (6pts).** For this question, consult the handout *What you can do with corpora using just standard Unix command line tools?* and the Ken Church tutorial handout referred to there (and the Unix man pages). The two sets of files to use are:

   /afs/ir/data/linguistic-data/Newswire/wsj/1996/WS96010*
   /afs/ir/data/linguistic-data//ICAME/ace/*

   The first set is Wall Street Journal newswire, while the second is written Australian English material, from a variety of sources.

   (a) Tokenizing the WSJ files: As in the handout, we need to extract text from the body region, and then tokenize it. In his early slides, Church gets words using the tr command, and deleting everything that isn't an alphabetic character. An obvious alternative would be to divide into words on white space.

      i. Give shell command sequences that will read the WSJ text and tokenize in these two ways and write the words one per line to files.

      ii. Compare the files using diff. Give 2 different cases in which both does the wrong thing or something questionable.

   Henceforth, so that we can all get the same answers, suppose we normalize the content of the first 6 files of WSJ text as follows (the stuff below is all one long command line, and assumes that you have gone to the corpus directory by doing cd /afs/ir/data/linguistic-data/Newswire/wsj/1996):

```
sed -e '1,/<TEXT>/d;/<\/TEXT>/,/<TEXT>/d ;/^|/d' WS96010* |
sed -e 's/<p>//g;s/$[0-9.,]*[0-9]/DLLR/g;s/[0-9][0-9.]*%/PRCNT/g;s/[0-9][.,0-9]*/NMBR/g' |
tr -sc 'A-Za-z' '\012' | tr 'A-Z' 'a-z' > ~/wsj.words
```

   This fancier version deletes paragraph marks and lines beginning with '|' (mostly tables), maps numeric tokens onto one of PRCNT DLLR NMBR, and then lowercases all words (including these!)

(b) Do something similar for the ACE files. Use all files in that directory, extract the contents between <sample> and </sample> but then additionally delete all text within some other SGML element (such as "<X> The Australian </X>" or "<bl> HEATHER McKENZIE </bl>"). Special thing to note: this corpus used a + sign to indicate original line hyphenation points! Get rid of those. Then tokenize and lowercase as for the WSJ text. Give the command(s) you used to do all this.

(c) Under the definition of 'word' that results from the commands given above, how many word tokens are there in these two corpora, and how many word types are there? (See the wc and uniq commands.)

(d) What are the 15 most frequent words in each corpus? What differences of any note are there between the two lists? (Use uniq and sort.)

(e) Make dictionaries for the words used in the two corpora (using sort and uniq). Examine how the usage of word types in the two corpora differs. (Use the comm command.) How many word types appear uniquely in our WSJ and ACE corpora? How many appear in both corpora? Note that this has little to do with differences between American and Australian English. Quite common simple words appear in only one or the other (e.g., *cough* only in WSJ and *coughed* only in ACE).

(f) What are the 10 most common word bigrams and 10 most common word trigrams in the two corpora (see Church's Bigrams slide)? – give lists. Give the commands you used to generate the WSJ trigrams starting from the wsj.words file above.

(g) Finally, let's validate Zipf's Law on these texts. With uniq one can get counts of words. You can keep just the counts with the cut command: "cut -f 1". One can then sort these counts and then one can use uniq again to get count-counts. Finally, one can number them with ranks using the nl (number lines) command.

   i. Give the command(s) that will produce a file with a column of ranks and a column of frequencies for one of these corpora.

   ii. For doing the rest of this, one could use a Unix graphing/math package if you know one (Matlab, Splus, GnuPlot, etc.), but I think the easiest thing to do is work with Excel. Import the rank-frequency file. (There are various ways to do that: either with a text file, and importing it with File|Open, or using cut and paste, and then the Data|Text to columns... option.) Make new columns that are the log of the value of those columns, and then graph the log-log values as an XY scatterplot. Include a plot.

   iii. Right click on a data point, and do "Add Trendline". Ask for a linear regression model, and for the equation to be printed. What is the slope for the two corpora? Are they similar? Do the data show the kind of curvature suggested by Mandelbrot's law?

7. *Practical.* **POS tagging (3pts).** We have installed in

   /afs/ir/class/cs224n/src/icopost-0.9.1

   a part of speech tagger by Ingo Schröder – well, there's actually a family of 3 part of speech taggers: an HMM tagger, a maximum entropy (loglinear model) tagger, and a transformation-based learning tagger, but we'll just use the trigram tagger. There's some documentation for it in the html subdirectory or at:

http://nats-www.informatik.uni-hamburg.de/~ingo/icopost/

We put a large amount of training text (extension .dat – this corresponds to what the documentation calls a "cooked" file), an English ngram and lexicon file in the data directory for use with the tagger. The data here comes from the Penn Treebank, and is tagged with the Penn Treebank tagset (see chapter 4 of M&S). You can use it to tag myfile.txt in your home directory with a command like the following if you are sitting in the data directory (see the documentation for details):

```
../bin/t3 -r 3 -v 3 english-wsj-train-0-18.ngram english-wsj-train-0-18.lex $HOME/myfile.txt
> $HOME/myfile.tagged
```

We also installed in

/afs/ir/class/cs224n/src/brill-tagger

Eric Brill's well-known transformation-based learning tagger (see M&S ch. 10 for discussion). If you go to the Bin_and_Data directory of it, you should be able to run it with the command:

```
./tagger LEXICON /afs/ir/class/cs224n/src/icopost-0.9.1/data/test-article.txt BIGRAMS
LEXICALRULEFILE CONTEXTUALRULEFILE > ~/article-tagged.txt
```

(where that should all be one long line). If you're in another directory, you need to give paths to the data files.

With these long command lines, you might want to define some aliases or little scripts to run the taggers!

Run both these taggers on the two test data files:

/afs/ir/class/cs224n/src/icopost-0.9.1/data/test-article.txt
/afs/ir/class/cs224n/src/icopost-0.9.1/data/test-novel.txt

Compare their output (you should be able to usefully use commands tr, diff, and wc).

(a) Where do they disagree? Give your judgment of which one is right in those cases, with some justification. (It isn't always easy to decide which is right: as well as the material in Chapter 3 of the textbook, you might look at:

/afs/ir/data/linguistic-data/Treebank/3/docs/tagguid1.ps

which particularly gives instructions for tagging with the Penn Treebank tag set.

(b) Are there any places that you notice where both are wrong?

(c) For at least two errors, give an explanation of why the tagger probably made a mistake (e.g., "this probably happened because *can* is most commonly used as a modal verb, and in this situation there isn't strong contextual evidence such as a preceding article to show that it is being used as a noun"). Are there any inexplicable errors?