CS 224N / Ling 237
Natural Language Processing – Spring 2002
Homework 1 (out of total of 30 pts) : Solutions

Out Mon Apr 8
Due Mon Apr 15, 5 pm

The first 4 questions below are purely written questions: a simple probability question to make sure everyone is on the same page, one on Zipf's law, and a couple of questions related to Naïve Bayes text categorization. The remaining questions are designed to give you hands on experience in accessing and using text corpora, and what some of the properties of text viewed in large amounts are. This third part needs to be done on a Unix computer and will probably take more the time. (If you can't access the corpora mentioned, then you're not on our list of officially enrolled people. Send email, and enroll.) We'd like you to hand in written answers to all questions.

1. *(3 pts)* A very famous early example of probabilities done over text was A. A. Markov's (1913) counts of consonants and vowels and their sequencing. Here's the data from Markov's count of *exactly* 20000 letters of the first part of A. Pushkin's novel *Eugène Onégin* – in Russian. We assume that he just ignored spaces.

   Markov defined two events, which he (unmnemonically) called:
   E: a vowel occurred
   F: a consonant occurred

   He then did counts to get the following probabilities, all as relative frequency estimates:

p	estimate of probability that the next letter is an event *E*
p_1	estimate of probability that the next letter is an event *E*, given that the preceding letter was an event *E*
p_2	estimate of probability that the next letter is an event *E*, given that the preceding letter was an event *F*
q	estimate of probability that the next letter is an event *F*
q_1	estimate of probability that the next letter is an event *F*, given that the preceding letter was an event *E*
q_2	estimate of probability that the next letter is an event *F*, given that the preceding letter was an event *F*

   In the text, Markov found 1104 instances of vowel-vowel sequences, while the total number of vowels in the text was 8638.

   Using this information:

a. Express the quantities p, p_1, p_2 , $q, q_1,$ and q_2 in a more mnemonic, intelligible notation (i.e., with events and conditional probabilities).
b. What is the value of $p, p_1,$ and p_2? By how much do p_1 and p_2 differ? What about for $q, q_1,$ and q_2 ?
c. What simple fact of language (true also of English) does the differences between $p, p_1,$ and p_2 (or between $q, q_1,$ and q_2) capture?

Assume the text is circular. There are 8638 vowels, each of which is preceded by something. Since 1104 of them are preceded by vowels, 7534 of them were preceded by consonants. There are 20000 – 8638 = 11362 consonants, and with circular text, there are as many VC pairs as CV pairs, so 7534 were preceded by vowels. The remaining 3828 were then preceded by consonants. Thus, the MLE estimates are (letting V_i be the probability that the i^{th} letter was a vowel and C_i be the probability that the i^{th} letter was a consonant):

$p(V_i) = p = 8638/20000 \approx 0.432$
$p(V_i|V_{i-1}) = p_1 = 1104/8638 \approx 0.128$
$p(V_i|C_{i-1}) = p_2 = 7534/11362 \approx 0.663$
$p(C_i) = q = 11362/20000 \approx 0.568$
$p(C_i|V_{i-1}) = q_1 = 7534/8638 \approx 0.872$
$p(C_i|C_{i-1}) = q_2 = 3828/11362 \approx 0.337$

The difference between p_1 and p_2 is .535, which is also the difference between q_1 and q_2 (obvious consequence of stochastic constraints).

Syllable structure is constrained; it's not the case that vowels and consonants occur independently in natural language. In particular, consonants and vowels tend to like to alternate rather than occur as long sequences of one or the other. The degree of the effect will depend on the phonotactics of the given language.

2. *(3 pts)* 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 Σ be an alphabet of N symbols. Let X be a uniform random process that produces each symbol of Σ, plus a space symbol, with uniform probability $1/(N + 1)$. X will thus generate text where words w over Σ 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. Show that, in the limit, "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 $r'(w)$. What is the slope of the line in the limit?

Let $\lambda = |w|$.

To generate a specific word w of length λ, X must generate the λ letters in w and then a space. Thus the probability of this is

$$p(w) = \frac{1}{(N+1)^{\lambda+1}}$$

Now what is the average rank $r'(w)$ for w? There are N^k many words of any length k. Thus, throwing in a length zero word to simplify matters, before any words of length λ there are $N+\ldots N^{\lambda-1}$ many shorter words on our list. On average, then, w will be preceded by all of these and by half of the words of length λ. So

$$\begin{aligned} r'(w) &= 1+N+\ldots N^{\lambda-1} + \frac{1}{2}N^{\lambda} \\ &= \frac{(N^{\lambda}-1)}{N-1} + \frac{N^{\lambda}}{2} \\ &= \frac{N^{\lambda+1} + N^{\lambda} - 2}{2N-2} \end{aligned}$$

Therefore, $\log p(w) = -(\lambda+1)\log(N+1)$ **and**

$$\log r'(w) = \log[N^{\lambda+1} + N^{\lambda} - 2] - \log(2N-2).$$

We are interested in the limit as λ goes to infinity of log $r'(w)$. If λ is very large, then $\log[N^{\lambda+1} + N^{\lambda} - 2] - \log(2N-2) \approx \log[N^{\lambda+1} + N^{\lambda}] \approx \log[N^{\lambda+1}] = (\lambda+1)\log N$.

Therefore, $\log p(w) = -\frac{\log(N+1)}{\log(N)} * \log r'(w)$

Therefore, log $p(w)$ and log $r'(w)$ become linearly related in the limit as Mandlebrot's law states, with the slope being $-\frac{\log(N+1)}{\log(N)}$

For N = 26, this is already very close to Zipf's law.

3. *(7 pts)* In this question, you are asked to trace the Naïve Bayes algorithm for classification on some toy sentences, and then to consider some variants. You should use the (multinomial) version of the algorithm from the extract of Tom Mitchell's *Machine Learning*.

   The following five sentences are the training data:

(the number in parentheses indicates the category that each sentence belongs to)

The dog barked. (1)
The mouse ran up the clock. (2)
The dog ran away. (1)
The collie dog barked. (1)
A mouse ate the cheese. (2)

The following three sentences are the test data:

A dog barked loudly.
The dachshund ran up the hill.
The rodent nibbled the cheese.

a. To which category does the algorithm assign each of the test sentences? Please show your calculations.
b. Suppose that the correct classifications of the test data are 1, 1, 2 respectively. On this basis, for each category, state the precision, recall, and F measure of the classifier, using $\alpha = 0.5$.
c. Suppose we omit the prior term. Does this affect the results on this test data (why/why not?)?
d. Suppose that instead of multiplying the factors that we *add* them. Does this make a difference in the decisions here? (Does it depend on whether the prior is included or not?) Comment in general on the qualitative effect on decisions of the model that would be caused by adding rather than subtracting (assuming a naïve Bayes model trained on a more reasonable amount of data).
e. It's hard to make probabilistic sense of the model in part d., but a well-founded way in which we can add probabilities is to combine factors P(c) and P(c|w) by averaging them: this gives a mixture model estimate for P(c), with equal weights on each mixture component. Examine the predictions of this model on this test data (please show your calculations). Comment in general on the qualitative differences between using such a decision model rather than the standard naïve Bayes model, or the model of part d.

Size of vocab = 12 ; P(1) = 3/5, P(2) = 2/5

P(the|1) = 4/23 ; P(dog|1) = 4/23 ; P(barked|1) = 3/23 ; P(ran|1) = 2/23
P(away|1) = 2/23 ; P(collie|1) = 2/23

P(the|2) = 4/23 ; P(mouse|2) = 3/23 ; P(ran|2) = 2/23 ; P(up|2) = 2/23
P(clock|2) = 2/23 ; P(a|2) = 2/23 ; P(ate|2) = 2/23 ; P(cheese|2) = 2/23

Sentence 1:
P(1) P(a|1)P(dog|1)P(barked|1) = 3/5 x 1/23 x 4/23 x 3/23 = 5.9 x 10-4

P(2) P(a|2)P(dog|2)P(barked|2) = 2/5 x 2/23 x 1/23 x 1/23 = 6.6 x 10-5

Sentence 2:
P(1)P(the|1)P(ran|1)P(up|1)P(the|1) = 3/5 x 4/23 x 2/23 x 1/23 x 4/23 = 6.9 x 10-5
P(2)P(the|2)P(ran|2)P(up|2)P(the|2) = 2/5 x 4/23 x 2/23 x 2/23 x 4/23 = 9.1 x 10-5

Sentence 3:
P(1)P(the|1)P(the|1)P(cheese|1) = 3/5 x 4/23 x 4/23 x 1/23 = 7.9 x 10-4
P(2)P(the|2)P(the|2)P(cheese|2) = 2/5 x 4/23 x 4/23 x 2/23 = 1.1 x 10-3

a. The algorithm would assign 1 to sentence 1, 2 to sentence 2, and 2 to sentence 3.

b. The precision for class 1 is 1/1 = 100%. The precision for class 2 is 1/2 = 50%. The recall for class 1 is 1/2 = 50%. The recall for class 2 is 1/1 = 100%. The F-measure for class 1 is 2(1)(0.5)/(1 + 0.5) = 2/3. The F-measure for class 2 is 2(0.5)(1)/(0.5 + 1) = 2/3.

c. If we omit the prior term:
Sentence 1: No change (12/23^3 vs 2/23^3)
Sentence 2: No change (32/23^4 vs 64/23^4)
Sentence 3: No change (16/23^3 vs 32/23^3)

d. If we add instead of multiplying, we will not different results if we do not include the prior. If we include the prior, however, then the classifications for sentence 2 and sentence 3 will become 1, rather than 2.
(Sentence 2: 124/115 vs 106/115 ; Sentence 3: 114/115 vs 96/115)

In general, adding rather than multiplying will have the result that the category with the highest prior will almost always be returned. This is because the number of categories is typically much smaller than the number of vocabulary items, so the prior terms are orders of magnitude larger than the other factors.

e. If we take the average of P(c) and P(c|w):

$$P(c|w) = \frac{P(c)*P(w|c)}{P(w)}$$

$$P(w) = P(w|c1)*P(c1) + P(w|c2)*P(c2)$$

Sentence 1: $P(c1|w) = \frac{5.9 \times 10\text{-}4}{5.9 \times 10\text{-}4 + 6.6 \times 10\text{-}5} = 0.899$

[P(c1|w) + P(c1)]/2 = [0.899 + 3/5]/2 = 0.750

P(c2|w) = 1 – P(c1|w) = 0.101

[P(c2|w) + P(c2)]/2 = [0.101 + 2/5]/2 = 0.251

Sentence 2: $$P(c1|w) = \frac{6.9 \times 10-5}{6.9 \times 10-5 + 9.1 \times 10-5} = 0.431$$

[P(c1|w) + P(c1)]/2 = [0.431 + 3/5]/2 = 0.516

P(c2|w) = 1 – P(c1|w) = 0.569

[P(c2|w) + P(c2)]/2 = [0.569 + 2/5]/2 = 0.484

Sentence 3: $$P(c1|w) = \frac{7.9 \times 10-4}{7.9 \times 10-4 + 1.1 \times 10-3} = 0.418$$

[P(c1|w) + P(c1)]/2 = [0.418 + 3/5]/2 = 0.509

P(c2|w) = 1 – P(c1|w) = 0.582

[P(c2|w) + P(c2)]/2 = [0.582 + 2/5]/2 = 0.491

In this case, we will always choose class 1 – the higher prior makes a much greater difference. Notice that it is NOT okay to use the values for P(c)*P(w|c) in place of P(c|w) – you must normalize with P(w). (For example, suppose one document had a prior of 2/5 and a P(w|c) term of 5 x 10-4, while another had a prior of 3/5 and a P(w|c) of 5 x 10-5 – in this case, you would get different results depending on if you used P(c|w) or P(c)*P(w|c).)

Doing this may seem strange (and is not necessarily recommended – see question 4 for another alternative), but is one way of ensuring that a prior distribution is washed out. Because of the false independencies of the Naive Bayes model, the context probabilities are often much too sharp, differing by many orders of magnitude, with the effect that the prior probability term scarcely has any effect (as in part c.). This can be bad if the sense distribution is very skewed: Naive Bayes classifiers commonly underpropose the common sense. Doing what is given in this question would mean that such a common sense would almost always be chosen, unless the difference in the context probabilities is very large.

4. *(3pts)* Suppose a Multinomial Naive Bayes text classifier when tested on some independent data generates the following confusion matrix. Such a matrix shows counts of the number of times that the test item was of category *x*, and the classifier judged it as of category *y* (where a, b, c, d are the four categories here).

		predicted			
		a	b	c	d
	a	50	26	20	12
Act	b	3	8	2	0
-ual	c	0	0	4	0
	d	0	1	0	1

Now suppose we modify the classifier so that the prior term is raised to some power t. The formula is now: $\text{argmax}_c\ [P(c)^t \times \text{Product}_f\ P(f|c)]$.

(a) What changes would you expect in the confusion matrix for different values of t?
(b) State in particular why it might be useful to include such a "prior boost" factor in a naïve Bayes model

a) If t > 1, then this will cause us to more strongly weight the prior distribution of the documents into classes. That is, more documents will be put into the a priori most common class. Assuming uniformity of distribution between the training and test data, we see that in the actual distribution, most documents are in class a., and so the effect would be to increase the tendency to classify more documents as class a. Given the results above, this effect could very well improve the classifier. If t = 1, nothing changes. If t < 1, then the effect would be to decrease the effect of the prior. This is likely to have only a small effect, but would tend to mean that slightly fewer documents are classified as class a., for this example, which might tend to worsen performance.

b) Adding the prior boost allows the user to tune the algorithm by adjusting how much weight it should give to the prior. If you know that your data is split into one very common category and several rare categories, but you want to have high recall on the rare categories, you may want to use this prior boost (using t < 1) to reduce the number of false negatives for the rare categories. Using t > 1 will give more weight to the prior, which is often helpful in a Naive Bayes model, as discussed in question 3.

For the remainder of the questions, 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.

5. *(3 pts)* 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.

- a. 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.
- b. Compare the files using diff. Give 2 different cases in which both does the wrong thing or something questionable.
- c. It isn't possible to do perfect tokenization with tr – one needs more complex tokenization rules to get an even heuristically good tokenizer (written, with e.g., lex or flex), but give a command for an improved tokenizer that does a bit better than either of these.

a)

```
sed -e '1,/<TEXT>/d;/<\TEXT>/,<TEXT>/d;/^|/d' WS96010* | tr –sc ‘A-Za-z’ ‘\012’
sed -e '1,/<TEXT>/d;/<\TEXT>/,/<TEXT>/d;/^|/d' WS96010* | tr -s '[:space:]' '\012'
```

b) Dividing on whitespace leaves punctuation attached to the words: e.g. "losses," . It also treats contractions as one word: e.g. isn't

Keeping only alphabetic characters separates hyphenated words: e.g. "double-digit" and abbreviations like "U.K.". It also deletes all numbers, which we might want to keep.

c) Various answers are acceptable here – we were looking for something that perhaps handled apostrophes or hyphens in a more intelligent way.

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:

```
sed -e '1,/<TEXT>/d;/<\/TEXT>/,/<TEXT>/d ;/^|/d ' WS96010* |
sed -e ' s/<p>//g; s/$[0-9.,]*[0-9]/DOLLAR/g;s/[0-9][0-9.]*%/PERCENT/g;s/[0-9][.,0-9]*/NUMBER/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 PERCENT DOLLAR NUMBER, and then lowercases all words (including these!)

6. *(4 pts)* Continuing with two corpora:
    - a. 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 you used to do all this.
    - b. Under this definition of 'word' how many word tokens are there in these two corpora, and how many word types are there? (See the wc and uniq commands.)

c. What are the 15 most frequent words in each corpus? What differences of any note are there between the two lists? (See uniq and sort.)
d. 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 other (e.g., *cough* only in WSJ and *coughed* only in ACE).

a)

```
sed -e '1,/<sample>/d;/<\/sample>/,/<sample>/d' ace_* | sed -e
's/<[a-zA-Z][^<]*<\/[a-zA-Z]*>//g;s/<[/a-zA-Z]*>//g;s/$[0-9.,]*[0-9]/DOLLAR/g;
s/[0-9][0-9.]*%/PERCENT/g;s/[0-9][.,0-9]*/NUMBER/g;s/+//g' | tr -sc 'A-Za-z' '\012' |
tr 'A-Z' 'a-z'
```

b) There are 756,118 tokens in the ACE corpus and 728,630 tokens in the WSJ corpus.
(wc –l ace.words)
There are 36,499 types in the ACE corpus and 23,874 types in the WSJ corpus
(sort < ace.words | uniq | wc –l)

c) Fifteen most frequent words:
(sort < ace.words | uniq -c | sort -nr | sed 15q)

ACE corpus	WSJ corpus
the	the
of	of
and	to
to	a
a	in
in	number
number	and
is	s
that	for
for	that
it	is
was	on
with	it
on	dollar
as	percent

d) From part b, we know that there are 36,499 types in the ACE corpus and 23,874 types in the WSJ corpus.

There are 21437 unique words in ACE and 8812 unique words in WSJ; therefore, there are 15062 words in common.

(First make dictionaries: sort < ace.words | uniq > ace.dict
sort < wsj.words | uniq > wsj.dict
Then get words that are unique to ace.dict

```
comm ace.dict wsj.dict | cut –f 1 | sort | uniq | wc –l
Then get words that are unique to wsj.dict
comm wsj.dict ace.dict | cut –f 1 | sort | uniq | wc –l)
```

7. *(3 pts)* What are the 10 most common word bigrams, trigrams, and fourgrams 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.

Bigrams:

ACE Corpus	*WSJ corpus*
of the	of the
on the	in the
to the	dollar million
on the	for the
and the	number number
to be	u s
for the	in number
at the	on the
with the	to the
from the	the company

Trigrams:

ACE corpus	*WSJ corpus*
one of the	the u s
number per cent	number number number
number and number	number to number
a number of	dollar million in
some of the	number cents a
number number number	the company s
as well as	one of the
part of the	cents a share
the end of	of dollar million
i don t	in new york

Fourgrams:

ACE corpus	*WSJ corpus*
number per cent of	number cents a share
the end of the	in the u s
at the end of	to number number number
at the same time	new york stock exchange
the rest of the	number days percent number
on the other hand	number number to number
one of the most	number to number number
to number per cent	the new york stock
as a result of	number number number to
on the basis of	number or percent to

Command to get trigrams:

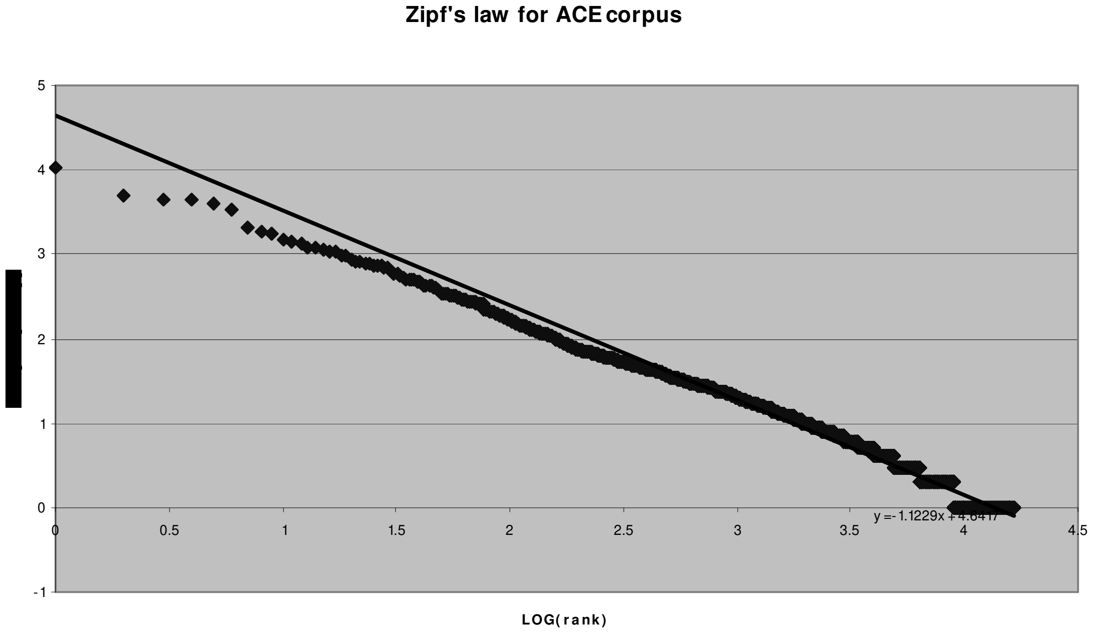
```
tail +2 wsj.nextwords > wsj.nextnextwords
paste wsj.words wsj.nextwords wsj.nextnextwords | sort | uniq -c > wsj.trigrams
sort -nr < wsj.trigrams | sed 10q
```

8. *(4 pts)* 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 number them with ranks using the nl (number lines) command.

    a. Give the command that will produce a file with a column of ranks and a column of frequencies for one of these corpora.
    b. 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, make new columns that are the log of the value of those columns, and then graph the log-log values as an XY scatterplot. 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?

a) sort wsj.words | uniq –c | cut -f 1 sorted_list | sort –n –r | nl > zipf

b) Slope for ACE: -1.1229
Slope for WSJ: -1.39

The slopes are similar for the two corpora. The data shows the curvature suggested by Mandelbrot's law.