HUGH HAN, PEIYI ZHENG

600.465 Natural Language Processing **Fall 2016**
Homework #3 **Due:** 27 September 2016, 5:00pm

1. The log probability for each sample corpus, training on the `switchboard-small` corpus, is calculated as follows.

```
-12111.3   speech/sample1
-7388.84   speech/sample2
-7468.29   speech/sample3
```

To calculate the perplexity of each corpus, we can simply raise 2 to the power of each *negated* log probability. That is, we can simply do the following.

$$\begin{aligned} &2^{12111.3} && \text{for} && \texttt{speech/sample1} \\ &2^{7388.84} && \text{for} && \texttt{speech/sample2} \\ &2^{7468.29} && \text{for} && \texttt{speech/sample3} \end{aligned}$$

To calculate the perplexity per word, we first need to calculate the cross-entropy by dividing each negated log proability by the number of words in its respective corpus.

The number of words in each corpus is calculated as follows.

```
1686   speech/sample1
 978   speech/sample2
 985   speech/sample3
```

Then we see that the perplexity per word of each file is the following.

$$\begin{aligned} &2^{\frac{12111.3}{1686}} && \text{for} && \texttt{speech/sample1} \\ &2^{\frac{7388.84}{978}} && \text{for} && \texttt{speech/sample2} \\ &2^{\frac{7468.29}{985}} && \text{for} && \texttt{speech/sample3} \end{aligned}$$

We obtain the following approximated perplexity-per-word values.

$$\begin{aligned} &145.36 && \text{for} && \texttt{speech/sample1} \\ &188.06 && \text{for} && \texttt{speech/sample2} \\ &191.61 && \text{for} && \texttt{speech/sample3} \end{aligned}$$

Training on the larger `switchboard` corpus, we get the following log probabilities for the following sample corpuses, respectively:

```
-12561.5   speech/sample1
-7538.27   speech/sample2
-7938.95   speech/sample3
```

Note that each of the log-probabilities grew more negative for its respective sample corpus. Thus, the each perplexity must also be smaller for its respective sample corpus. The reason for this is because if we increase the size of our training data, then we introduce a greater number of possible sentences. That is, the probability that the sentences in our specific test files appear would be much smaller, in that there is a greater variety of possibilities to choose from.

2. `IMPLEMENTATION PROBLEM`

3. We chose to do spam detection.

   When classifying the sample files containing genuine messages, we received the following output:

```
    178 looked more like gen_spam/train/gen (98.89%)
    2 looked more like gen_spam/train/spam (1.11%)
```

   When classifying the sample files containing spam messages, we received the following output:

```
    66 looked more like gen_spam/train/gen (73.33%)
    24 looked more like gen_spam/train/spam (26.67%)
```

   That is, we had the following error rates for the different data sets.

$$\begin{aligned} \texttt{gen} &: \quad \text{error-rate} = 1.11\% \\ \texttt{spam} &: \quad \text{error-rate} = 73.33\% \end{aligned}$$

   (a) Each classification decision is based on probability. That is, whichever training corpus yields a higher cross-entropy with each test file is the training corpus to which that test file will be classified.

   However, we cannot necessarily deduce that the lowest cross-entropy occurs with the lowest error rate. With a low error rate, it could just be that the opposite incorrect corpus has a higher cross-entropy than the correct corpus, instead of the correct corpus having the lowest possible cross-entropy.

   Thus, we are forced to do some type of guess-and-check to find the value of the lowest cross-entropy. We proceeded by doing a human version of binary search.

   Using our binary search, we find the following two values of the approximate smallest log-probabilities.

$$\begin{aligned} \texttt{gen} &: \quad \text{log-probability} \approx -423681 \\ \texttt{spam} &: \quad \text{log-probability} \approx -280362 \end{aligned}$$

   Using these values, we can obtain the approximate smallest cross-entropies.

$$\begin{aligned} \texttt{gen} &: \quad \text{log-probability} \approx -423681 \\ \texttt{spam} &: \quad \text{log-probability} \approx -280362 \end{aligned}$$

The λ values that were used in finding these minimum cross-entropies were as follows.

$$\begin{aligned} \texttt{gen} &: \quad \lambda = 0.01357 \\ \texttt{spam} &: \quad \lambda = 0.0083 \end{aligned}$$

(b) We can do binary search again, but let's think of how we can be a little smarter this time. We already have

$$\begin{aligned} \lambda_{\texttt{gen}} &= 0.01357 \\ \lambda_{\texttt{spam}} &= 0.0083 \end{aligned}$$

so we can use them as critical points. We see that:

(i) all $\lambda < \lambda_{\texttt{spam}}$ cause increasing log-probabilities

(ii) all $\lambda > \lambda_{\texttt{gen}}$ cause increasing log-probabilities

Then the following must be true.

$$\lambda_{\texttt{spam}} \le \lambda^* \le \lambda_{\texttt{gen}}$$

So now we can do a smarter guess-and-check binary search, and we find the following value of λ^*.

$$\lambda^* \approx 0.0104$$

(c) First, let's try to classify the test files containing genuine messages.

```
347 looked more like gen_spam/train/gen (96.39%)
13 looked more like gen_spam/train/spam (3.61%)
```

Next, let's try to classify the test files containing spam messages.

```
50 looked more like gen_spam/train/gen (27.78%)
130 looked more like gen_spam/train/spam (72.22%)
```

So then we get the following error rates.

$$\begin{aligned} \texttt{gen} &: \quad \text{error-rate} = 3.61\% \\ \texttt{spam} &: \quad \text{error-rate} = 27.78\% \end{aligned}$$

(d) For this problem, we can all of the possible file lengths into 10 buckets of file length intervals. For instance, the first bucket would contain the classification results from the files with lengths in the range [0, 40), the second bucket would contain those in the range [40, 80), and so on. We can reserve our last bucket as a "special" bucket. That is, this special bucket will contain the classificatioon results of all of the files with lengths greater than 400. Now note that the first bucket can serve as a baseline, which represents the lowest accuracy of 75%. The reason for this is that small files containing less information, so they could be more difficult to classify.

(e) It is easy to see that as the training data size increases, the classification accuracy also increases. But increasing the size of the training set does not necessarily solve all classification accuracy issues. At some point, it will level off. For example, it was noted that when we switched from `gen-times4` to `gen-times8`, the classification accuracy was improved only slightly.

4. (a) Let's say that we take $V = 19{,}999$.

   i. Using the UNIFORM estimate, if we see a word that is out of the vocabulary, we would have no choice but to assign that OOV word a probability of 0, due to it being the 20,000th word of a vocabulary of size 19,999. That is, the sum of the probabilities of all other words already sum to 1, and we cannot have a probability greater than 1.

   $$\sum_{i=1}^{V} \frac{1}{V} \quad = \quad 1$$

   However, this is an bad estimation, simply because no word can have a probability of 0, regardless of whether that word was observed in the training data.

   ii. Using the ADDL estimate,

   (b) If we let $\lambda = 0$, then we are not modifying our probability at all. That is, we get the following from our probabiliy estimate.

   $$\begin{aligned} \hat{p}(z \mid xy) &= \frac{c(xyz) + \lambda}{c(xy) + \lambda V} \\ &= \frac{c(xyz) + 0}{c(xy) + 0} \\ &= \frac{c(xyz)}{c(xy)} \end{aligned}$$

   (c) i. First, let's think about the case when $c(xyz) = c(xyz') = 0$.

   We see that

   $$\hat{p}(z \mid xy) \quad = \quad \frac{\lambda V \cdot \hat{p}(z \mid y)}{c(xy) + \lambda V}$$

   Then the difference between $\hat{p}(z \mid xy)$ and $\hat{p}(z' \mid xy)$ is the possible differing values of $\hat{p}(z \mid y)$ and $\hat{p}(z'|y)$. However, we cannot guarantee that $\hat{p}(z \mid y) = \hat{p}(z' \mid y)$.

   For example, we might never see either of the phrases "kiss the tree" and "kiss the Jupiter". However, when we backoff, it is possible that we have seen both of the phrases "the tree" and "the Jupiter".

   And in English, it is much more likely that the phrase "the tree" is more common than "the Jupiter", which demonstrates a case in which that $\hat{p}(z \mid y) \neq \hat{p}(z' \mid y)$.

   ii. Now let's think about the case when $c(xyz) = c(xyz') = 1$.

   (d)

5. (a) `IMPLEMENTATION PROBLEM`

   (b) Using $\lambda^* = 0.0104$, as calculated in problem **3**(b), we can calculate the cross-entropies for the switchboard corpora and analyze the text categorization error rates for `gen/spam`. The following generalizations are made with the assumption that $\lambda^* = 0.0104$.

   cross-entropies for the switchboard corpora

   Below contains output using ADDL for switchboard corpora.

```
WE MUST  /usr/local/data/cs465/hw-lm/speech/sample1
CALCULATE  /usr/local/data/cs465/hw-lm/speech/sample2
THESE VALS /usr/local/data/cs465/hw-lm/speech/sample3
```

   Below contains output using BACKOFF_ADDL for switchboard corpora.

```
-10040.1 /usr/local/data/cs465/hw-lm/speech/sample1
-6039.31 /usr/local/data/cs465/hw-lm/speech/sample2
-6413.95 /usr/local/data/cs465/hw-lm/speech/sample3
```

   TODO : Make generalization

   text categorization error rates for gen/spam

   Below contains output using BACKOFF_ADDL for `gen` test files.

```
209 looked more like gen_spam/train/gen (58.06%)
151 looked more like gen_spam/train/spam (41.94%)
```

   Below contains output using BACKOFF_ADDL for `spam` test files.

```
7 looked more like gen_spam/train/gen (3.89%)
173 looked more like gen_spam/train/spam (96.11%)
```

   From these outputs, we can see that the error rates calculated using BACKOFF_ADDL were lower than the error rates calculated using ADDL as in **3**(c) for `spam` test files.

   At the same time, the error rates calculated using BACKOFF_ADDL were higher than the error rates calculated using ADDL as in **3**(c) for `gen` test files.

   We can conclude that switching from ADDL to BACKOFF_ADDL causes the error rates for classifying spam files to decrease extraordinarily, but at the cost of spiking up error rates for classifying genuine files.

   (c) *Extra credit.*

6. (a)

   (b)

   (c)

   (d)

   (e) *Extra credit.*

   (f)

(g) i.
ii.
iii.
iv.
v.
vi.
vii.

7. Let w denote an arbitrary word. When we want to do the classification, we may want to consider the probability $P(w \text{ is spam} \mid w)$. According to Bayes's Theorem, we can write the following equation.

$$\begin{aligned} P(w \text{ is } \texttt{spam} \mid w) &= \frac{P(w \mid w \text{ is } \texttt{spam}) \cdot P(\texttt{spam})}{P(w)} \\ &= \frac{P(w \mid w \text{ is } \texttt{spam}) \cdot P(\texttt{spam})}{P(w \mid w \text{ is } \texttt{spam}) \cdot P(\texttt{spam}) + P(w \mid w \text{ is } \texttt{gen}) \cdot P(\texttt{gen})} \end{aligned}$$

$P(w \text{ is } \texttt{spam} \mid w)$ and $P(w \mid w \text{ is } \texttt{gen})$ are the probabilities generated separately from the two models we train using the `spam` and `gen` corpora.

Now we know the following.

$$P(\texttt{spam}) = \frac{1}{3}$$

Then the following naturally follows.

$$\begin{aligned} P(\texttt{gen}) &= 1 - P(\texttt{spam}) \\ &= \frac{2}{3} \end{aligned}$$

We can rewrite then rewrite the following.

$$P(w \text{ is } \texttt{spam} \mid w) = \frac{1}{1 + \frac{2P(w \mid w \text{ is } \texttt{gen})}{P(w \mid w \text{ is } \texttt{spam})}}$$

By doing this, we can calculate the probability $P(w \text{ is } \texttt{spam} \mid w)$. Obviously we don't need *priori* when training the model.

In order to implement this method, we need to

i. train the `gen` model and store the probabilities for all test files,
ii. train the `spam` model and store the probabilities for all test files,
iii. iterate through all $P(w \mid w \text{ is } \texttt{gen})$ and $P(w \mid w \text{ is } \texttt{spam})$, so that we can calculate $P(w \text{ is } \texttt{spam} \mid w)$.

Extra credit: We implemented this change in `problem7.py`. We tested our implementation using an extremely small value for λ (which was `add0.00001` in our case). From this specific test, the program successfully classified approximately 33.3% of the test data to be spam. That is, the result is extremely close to *priori*.

8. (a) Using Bayes Theorem, we can express the probability $P(\vec{w} \mid U)$ in the following equation.

$$P(\vec{w} \mid U) \quad = \quad \frac{P(U \mid \vec{w}) * P(\vec{w})}{P(U)}$$

Now in order to maximize $P(\vec{w} \mid U)$, we can think about maximizing $log_2(P(\vec{w} \mid U))$. Using the rules of logarithms, note that the following statement is true.

$$log_2(P(\vec{w} \mid U)) \quad = \quad log_2(P(U \mid \vec{w})) + log_2(P(\vec{w})) - log_2(U)$$

Since $log_2(U)$ remains constant for all of the possible $\vec{w}$ being considered, we can ignore the final term on the right side of the equation when maximizing $log_2(P(\vec{w} \mid U))$.

That is, we only need to worry about maximizing the two other terms, $log_2(P(U \mid \vec{w})) + log_2(P(\vec{w}))$. Luckiily, $log_2(P(U \mid \vec{w}))$ is provided to us in the input files. Then all we need to do is calculate $log_2(P(\vec{w}))$ using our trigram model, which will measure the extent to which it looks like English.

So we iterate all 9 candidates and compute $log_2(P(U \mid \vec{w})) + log_2(P(\vec{w}))$, and can thus find the candidate with the highest probability $P(\vec{w} \mid U)$.

(b) `IMPLEMENTATION PROBLEM`

(c) Some trigrams in unrestricted data are very uncommon due to the existence of words like "uh", "um", or "huh" intertwined inside of the sentences.

Therefore in order to overcome this problem we need to utilize backing off. In pratice, we set the value of λ to be 0.01 according to our experiments on development data. We can see the overall error rates in the table below.

	test/easy	test/unrestricted
3-gram model	0.141	0.380
2-gram model	0.159	0.396
1-gram model	0.210	0.408

9. *Extra credit.*

10. *Extra credit.*

(a)

(b)

(c)

(d)

(e) i.

ii.

iii.

(f)

(g)

11. *Extra credit.*