Homework #2: CMPT-825

Anoop Sarkar – anoop@cs.sfu.ca

(1) **Language Models applied to TrUeCasIng**

TrueCasing is the process of taking text with missing or unreliable case information, e.g. if the input looks like:

```
as previously reported , target letters were issued last month to
michael milken , drexel 's chief of junk-bond operations ; mr. milken 's
brother lowell ; cary maultasch , a drexel trader ; james dahl , a
drexel bond salesman ; and bruce newberg , a former drexel trader .
```

Then the output of the TrueCasing program should be:

```
As previously reported , target letters were issued last month to
Michael Milken , Drexel 's chief of junk-bond operations ; Mr. Milken 's
brother Lowell ; Cary Maultasch , a Drexel trader ; James Dahl , a
Drexel bond salesman ; and Bruce Newberg , a former Drexel trader .
```

As this example should illustrate, this is not a trivial problem to solve. There are many different ways to attack this problem. In this homework, we will look at the application of language modelling to this problem.

Here are the data files we will be using in the experiments:

Training Data:	`recap.train`	data you will use to create the language model
Test Data (Input):	`recap.input`	lowercased data, input to your TrueCasing program
Test Data (Truth):	`recap.test`	the *real* case information for `recap.input`

These files are available in the directory: `/cs/natlang-a/data/recap/`. The first thing to do is to have a look at the data so that you are familiar with the problem to be solved.

Before we can consider the *model* we will use to solve this problem, let's review some definitions from the lectures. Let $T = s_0, \ldots, s_m$ represent the test data with sentences s_0 through s_m. A *language model* θ computes $P_\theta(T)$:

$$P_\theta(T) = \prod_{i=0}^{m} P_\theta(s_i) = 2^{\sum_{i=0}^{m} \log_2 P_\theta(s_i)}$$

$$\log_2 P_\theta(T) = \sum_{i=0}^{m} \log_2 P_\theta(s_i)$$

where $\log_2 P_\theta(s_i)$ is the log probability assigned by the language model θ to the sentence s_i. Let W_T be the length of the text T in word tokens. The *cross entropy* for T is:

$$H_\theta(T) = -\frac{1}{W_T} \log_2 P_\theta(T)$$

The cross entropy corresponds to the average number of bits needed to encode each of the W_T word tokens in the *test data*. The *perplexity* of the test data T is defined as:

$$PPL_\theta(T) = 2^{H_\theta(T)}$$

Low cross entropy values for test data indicate that the probability distribution of the test data is 'closer' to the probability distribution of the trained language model. Similarly, higher perplexity values indicate a worse fit between test and training data. Thus, we can exploit perplexity values computed by the language model to compare different alternatives.

We will be using the SRI Language Modelling Toolkit to implement the language model. The software is installed in ~anoop/srilm-latest/. The programs to be used are in the bin and bin/i686 sub-directories. You can provide an explicit path or modify your shell PATH variable to access these programs.

The documentation for the programs are HTML files in the man sub-directory: you should at least look at the files ngram-count.html; ngram.html and disambig.html.

We can use the toolkit programs to check the fact that (unseen) lowercased text should have a higher perplexity when compared to (unseen) TrueCased text, because our training data was TrueCased.[1]

In order to do this, create a language model based on the training data by using the command ngram-count.[2]

```
ngram-count -order 3 -text recap.train -lm recap.lm
```

This writes a trigram language model with the SRI LM toolkit's default smoothing to the file recap.lm (check the documentation to find out about the default smoothing method). The language model is stored in a human-readable format called the ARPA LM format (see ngram-format.html).

To compare the test set perplexity between all lowercased text and TrueCased text, we can use the program ngram applies the trigram model θ computed above to any text T to find $PPL_\theta(T)$.[3]

The following command reports the perplexity for the file named <filename>. Compare the reported perplexity of recap.input with recap.test to test the assertion that lowercased text should have a higher perplexity when compared to TrueCased text.

```
ngram -lm recap.lm -ppl <filename>
```

Now let's consider how to apply our language model (trained on TrueCased text) to the problem of restoring TrueCase to lowercase text. Let's consider an example where we want to restore case to the input: "hal 9000". First, let's assume we can collect variants of the TrueCase spelling for each word in the input:

l_w	c_w	$P_m(c_w \mid l_w)$
hal	HAL	$\frac{1}{2}$
hal	Hal	$\frac{1}{4}$
hal	hal	$\frac{1}{4}$
9000	9000	1

m is a model that maps unigrams from lowercase to TrueCase. In this example, since

$$P_m(\texttt{HAL} \mid \texttt{hal}) + P_m(\texttt{Hal} \mid \texttt{hal}) + P_m(\texttt{hal} \mid \texttt{hal}) = 1$$

Only these three TrueCase options are considered for the input "hal". Based on this table, we can now create the following possible TrueCase sentences:

[1] If our training data was lowercased, then unseen lowercased data would have lower perplexity compared to unseen TrueCased text.

[2] Ignore any warnings generated by this command.

[3] The SRI LM toolkit actually reports two different perplexity values, what it calls ppl and ppl1. The reason for two values is because the toolkit inserts a begin sentence marker <s> and an end sentence marked </s> to each sentence. The perplexity ppl1 ignores these markers when computing the perplexity. Either value can be used in the comparison. Convince yourself that ppl will usually be lower than ppl1.

1. C_1 = "HAL 9000"
2. C_2 = "Hal 9000"
3. C_3 = "hal 9000"

The most likely TrueCase sentence $\hat{C}$ can be defined as:

$$\hat{C} = \underset{C_i}{\arg\max}\ P_\theta(C_i) \times \prod_{w \in C_i} P_m(c_w \mid l_w) \tag{1}$$

In our example, let's consider our input "hal 9000" and the TrueCase alternative "HAL 9000". We would compute:

$$P_\theta(\texttt{"HAL 9000"}) \times P_m(\texttt{HAL} \mid \texttt{hal}) \times P_m(\texttt{9000} \mid \texttt{9000})$$

P_θ("HAL 9000") can be easily computed using a language model θ. So, in order to attack the TrueCasing problem, all we need are three components: P_θ, P_m and the computation of the arg max in Equation (1). We already know how to use the SRI LM toolkit to compute a trigram model from our training data, so we have P_θ already. Your task is to compute P_m.

Luckily for us, if P_m is available, the SRI LM toolkit has the capability to compute for us $\hat{C}$ using Equation (1). The program `disambig` from the toolkit performs exactly this computation: it maps a stream of tokens from vocabulary V1 to a corresponding stream of tokens from a vocabulary V2 (see `disambig.html`). For our problem, V1 is the input lowercased text, while V2 is the output TrueCased text.

a. Write down this problem of true-casing as a Hidden Markov Model. What are the states and how many of them are there? What are the emissions? Write down the number of free parameters in the HMM needed to solve this task. What are the standard HMM algorithms you could use to solve this task? How could you implement the algorithm in order to make this task more efficient. Compare with the SRI LM solution below.

b. Create the *mapping* model P_m. Create a text file (call it `recap.map`) in the following format:

```
w1 w11 p11 w12 p12 ... w1n p1n
w2 w21 p21 w22 p22 ... w2m p2m
...
```

`w11, w12, ..., w1n` are words from the training data. `w1` is the lowercase form of `w11, w12, ..., w1n` and `p11` is the probability $P_m(\texttt{w11} \mid \texttt{w1})$ which is computed as follows:

$$\texttt{p11} = P_m(\texttt{w11} \mid \texttt{w1}) = \frac{freq(\texttt{w11})}{freq(\texttt{w11}) + freq(\texttt{w12}) + \ldots + freq(\texttt{w1n})}$$

In general, for each word `wij` from the training data where `wi` is the lowercase form of `wij`[4], compute `pij` as follows:

$$\texttt{pij} = P_m(\texttt{wij} \mid \texttt{wi}) = \frac{freq(\texttt{wij})}{\sum_{\texttt{k}} freq(\texttt{wik})}$$

Here are a few lines as an example of what your file `recap.map` should look like:

```
trump Trump 0.975 trump 0.025
isabella Isabella 1
uncommitted uncommitted 1
alberto-culver Alberto-Culver 1
```

[4] In Perl, you can compute the lowercase form of a word using the `lc` function. e.g. `print lc("HAL");` will print `hal`

c. Use the SRI LM toolkit program `disambig` to convert the file `recap.input` to a TrueCased output file (call it `recap.output`). You have to figure out how to use `disambig`. **Hint:** Among the options you should use to `disambig` are: `-order 3 -keep-unk`

d. Using the perl program `scoreRecap.pl` available in `/cs/natlang-a/data/recap` find the error rate of your TrueCasing model. Report this error rate.

```
perl scoreRecap.pl recap.test recap.output
```

e. Can you reduce the error rate further? Two things to try: (1) always uppercase the first letter of the first word in a sentence and (2) always uppercase the first letter of an unknown word (a word not seen in the training data).