Homework #1: 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: ∼`anoop/cmpt825/hw1/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 per-word log probability for T is:

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

The *perplexity* of the test data T is defined as:

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

The lower the perplexity, the better the language model. This enables comparison of different language models.

We will be using the SRI Language Modelling Toolkit to implement the language model. The software is installed in ∼anoop/cmpt825/sw. The programs to be used are in the appropriate subdirectories for x86-64/SRILM/bin or i386/SRILM/bin. 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. If our training data was lowercased, then unseen lowercased data would have lower perplexity compared to unseen TrueCased text.

In order to do this, create a language model based on the training data by using the command ngram-count. Ignore any warnings generated by this command.

```
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 $PPL_\theta(T)$.[1]

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] 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:

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

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

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

$P_\theta(\texttt{"HAL 9000"})$ 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.
Once P_m is available, the SRI LM toolkit has the capability to compute $\hat{C}$ in 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. Report the perplexity value of `recap.input` versus `recap.test`. Which is higher?
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:

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

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

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

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
```

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` to find the error rate of your TrueCasing model. Report this error rate. Also report the new perplexity score.

```
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). Does the perplexity always decrease when the error rate decreases?

(2) **Chinese Word Segmentation**

In many languages, including Chinese, the written form does not typically contain any marks for identifying words (compared to the space character which delimits words in the English script). Given the Chinese text "北京大学生比赛", a plausible segmentation would be "北京(Beijing)/大学生(university students)/比赛(competition)" (Competition among university students in Beijing). However, if "北京大学" is taken to mean Beijing University, the segmentation for the above character sequence might become "北京大学(Beijing University)/生(give birth to)/比赛(competition)" (Beijing University gives birth to competition), which is less plausible.

Word segmentation refers to the process of demarcating blocks in a character sequence such that the produced output is composed of separated tokens and is meaningful. It is an important task that is prerequisite for various natural language processing applications such as machine translation.

In the directory ∼`anoop/cmpt825/hw1/cnwseg` you are provided with a file `wfreq.utf8` which contains a list of words and their frequencies collected from a corpus that was segmented by hand by experts. We will use this data as training data to build a word segmentation system.

Let the input $\mathbf{c}$ be a sequence of characters $c_0, \ldots, c_m$. The space of possible outputs is given by a set $\mathcal{W}$. A sequence of segmented words $\mathbf{w} \in \mathcal{W}$ where $\mathbf{w} = w_0, \ldots, w_n$ for some $n \leq m$. Note that $\mathcal{W}$ can contain segmented outputs of varying length.

One way to use the training data given to you is to find the most likely segmentation provided by a unigram model.

$$\mathbf{w}^* = \underset{\mathbf{w}=w_0,\ldots,w_n \in \mathcal{W}}{\operatorname{argmax}} P(w_0, \ldots, w_n) = \underset{\mathbf{w}=w_0,\ldots,w_n \in \mathcal{W}}{\operatorname{argmax}} \prod_{i=0}^{n} P(w_i)$$

Implement the dynamic programming algorithm that captures the above model based on the unigram probabilities $P(w_i)$ estimated from the unigram frequencies provided to you. If any character in the test data `test.data` is missing from the training data segment them into a sequence of one character words, i.e. assume that any unseen character has a count of one since the model will naturally prefer a shorter segmentation (with fewer words).

Use the file `score.perl` to find the accuracy of your word segmentation system. As input to `score.perl` you will need the following files:

- A dictionary file that contains the words observed in the training data. This is used to provide the in-vocabulary (IV) performance versus the out-of-vocabulary (OOV) performance to highlight the impact of unknown words. The dictionary file can be easily created by stripping the frequencies from the file `wfreq.utf8`.
- The output of your system on the input file `test.data`
- The human annotated output on the same data `gold.data` which is used to measure the performance of your system.

The performance is evaluated in terms of the number of insertions and deletions required in terms of the words produced by your system to match the gold data. The final figures reported are the precision P (how many words produced were matched in the gold data) and recall R (how many words in gold data were produced). The F_1 score is the harmonic mean of the

precision and recall ($\frac{2PR}{P+R}$). The evaluation script also provides statistics on how well the system performed on IV and OOV words.

A sample output of the evaluation program is:

```
TOTAL INSERTIONS: 22126
TOTAL DELETIONS: 3186
TOTAL SUBSTITUTIONS: 21416
TOTAL NCHANGE: 46728
TOTAL TRUE WORD COUNT: 154864
TOTAL TEST WORD COUNT: 173804
TOTAL TRUE WORDS RECALL: 0.841
TOTAL TEST WORDS PRECISION: 0.749
F MEASURE: 0.793
OOV Rate: 0.230
OOV Recall Rate: 0.540
IV Recall Rate: 0.931
```

(3) **Language model interpolation**

A language model computes the probability for a given sentence. This is typically done by using the Markov assumption that only a finite history is used to compute the joint probability of all the words in a sentence. For example, the following is a unigram language model that ignores the history altogether:

$$\Pr(s) = \Pr(w_0, \ldots, w_n) \approx P(w_0) \cdot P(w_1) \cdot \ldots \cdot P(w_n)$$

The maximum likelihood estimate for a unigram model is:

$$P_{\text{ML}}(w) = \frac{f_t(w)}{\sum_w f_t(w)} \text{ where } f_t(\cdot) \text{ is the frequency observed in training data}$$

Similarly for a bigram model $\Pr(w_0, \ldots, w_n) \approx P(w_0) \cdot P(w_1 \mid w_0) \cdot \ldots \cdot P(w_n \mid w_{n-1})$:

$$P_{\text{ML}}(w \mid w_{-1}) = \frac{f_t(w_{-1}, w)}{\sum_w f_t(w_{-1}, w)} = \frac{f_t(w_{-1}, w)}{f_t(w)}$$

And similarly for a trigram model:

$$\Pr(w_0, \ldots, w_n) \approx P(w_0) \cdot P(w_1 \mid w_0) \cdot P(w_2 \mid w_0, w_1) \ldots \cdot P(w_n \mid w_{n-2}, w_{n-1})$$

An *interpolated* language model combines the estimates from different n-gram probability estimates. For example, for an interpolated language model for trigrams:

$$P_{\text{INT}}(w \mid w_{-2}, w_{-1}) = \lambda_1 P_{\text{ML}}(w \mid w_{-2}, w_{-1}) + \lambda_2 P_{\text{ML}}(w \mid w_{-1}) + \lambda_3 P_{\text{ML}}(w)$$

where:

$$\sum_i \lambda_i = 1 \text{ and } \lambda_i \geq 0 \text{ where } i \in \{1, 2, 3\} \text{ for trigrams} \tag{2}$$

The values of λ_i can be found using the Expectation Maximization (EM) family of algorithms. In this case, we will maximize the expected value of trigrams in some *held-out* data. We obtain the held-out data by subtracting some text from the training data such that the frequency we get from training is $f_t(\cdot)$ as above, but now we can also obtain frequencies in the held-out data which we will refer to as $f_h(\cdot)$, e.g. $f_h(w_{-2}, w_{-1}, w)$ is the frequency of a trigram in the held-out data.

Using EM we aim to find the values of $\lambda_1, \lambda_2, \lambda_3$ such that $L(\lambda_1, \lambda_2, \lambda_3)$ is maximized for a given held-out set, where:

$$L(\lambda_1, \lambda_2, \lambda_3) = \sum_{w_{-2}, w_{-1}, w} f_h(w_{-2}, w_{-1}, w) \cdot P_{\text{INT}}(w \mid w_{-2}, w_{-1})$$

The EM algorithm for such a setting is defined as follows:

1: Initialize $\lambda_1, \lambda_2, \lambda_3$ such that Condition (2) is satisfied.

2: Compute log-likelihood, for each sentence s_i in the held-out set h:

$$L(h) = \sum_i \log P_{\text{INT}}(s_i)$$

3: Find expectation:

$$\begin{aligned} c_1 &= \sum_{w_{-2}, w_{-1}, w} \frac{f_h(w_{-2}, w_{-1}, w) \lambda_1 P_{\text{ML}}(w \mid w_{-2}, w_{-1})}{P_{\text{INT}}(w \mid w_{-2}, w_{-1})} \\ c_2 &= \sum_{w_{-2}, w_{-1}, w} \frac{f_h(w_{-2}, w_{-1}, w) \lambda_2 P_{\text{ML}}(w \mid w_{-1})}{P_{\text{INT}}(w \mid w_{-2}, w_{-1})} \\ c_3 &= \sum_{w_{-2}, w_{-1}, w} \frac{f_h(w_{-2}, w_{-1}, w) \lambda_3 P_{\text{ML}}(w)}{P_{\text{INT}}(w \mid w_{-2}, w_{-1})} \end{aligned}$$

4: Re-estimate $\lambda_1, \lambda_2, \lambda_3$:

$$\begin{aligned} \lambda_1 &= \frac{c_1}{c_1 + c_2 + c_3} \\ \lambda_2 &= \frac{c_2}{c_1 + c_2 + c_3} \\ \lambda_3 &= \frac{c_3}{c_1 + c_2 + c_3} \end{aligned}$$

5: Compute new log-likelihood, for each sentence s_i in the held-out set h:

$$L^+(h) = \sum_i \log P_{\text{INT}}(s_i)$$

6: **if** $|L(h) - L^+(h)|$ is small enough (say 10^{-4}) **then**

7: stop and return current value of $\lambda_1, \lambda_2, \lambda_3$

8: **else**

9: set $L(h) = L^+(h)$ and go to Step 3

10: **end if**

The training data `austen-train.txt` and held-out data `austen-heldout.txt` are both available in the directory `~anoop/cmpt825/hw1/interp`

a. Create the unigram, bigram and trigram maximum likelihood models based on the frequencies of the n-grams from the training data.

   Replace all unigrams that have a count of one in the training data with a special token, `_UNK_`. This ensures that the unigram probability will return an estimate for unknown words in the held-out set (where tokens unseen in training are replaced with `_UNK_`).

b. Implement the Expectation-Maximization (EM) algorithm to find the interpolation parameters by using the counts of the trigrams in the held-out set. For initial values of $\lambda_1, \lambda_2, \lambda_3$ use $\frac{1}{3} + r$, where r is sampled from a normal distribution with a mean of 0.01 and then re-normalize to ensure that $\sum_i \lambda_i = 1$ and $\lambda_i \geq 0$.

c. Plot the value of log-likelihood and perplexity on the held-out set for each iteration of the EM algorithm. You can run the EM algorithm until the log-likelihood converges (difference between the current iteration and previous one is less than 10^{-4}) or simply for a fixed number of iterations.

d. Try different initialization schemes for $\lambda_1, \lambda_2, \lambda_3$ and run EM to convergence. Plot the various runs in one graph.