CS224N/Ling 237 Homework #3

Due: Wed, Apr 21, 2004

This assignment covers topics related to n-gram language models, starting with a couple of written questions, then moving to doing some practical language model construction (using the srilm package).[1] It's intended to not be too large, so you can also start thinking about the WSD homework.

Written questions

1. **Smoothing probability models (6 pts).**

   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 estimated count of seen words is discounted by a certain percentage, for example $\alpha = 5\%$:

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

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

   (a) Show that absolute discounting yields a probability model. In particular show that $\sum_{w_n \in V} P_{\text{abs}}(w_n|w_1 \cdots w_{n-1}) = 1$, 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 estimating $P(w_3|w_1, w_2)$ in the *comes across* __ example used in the class handout)? Explain your answer with a concrete example.

[1] The practical part is based on an assignment by Anoop Sarkar.

2. **Linguistic and mathematical issues (6 pts).**

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

      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? In particular, what kinds of grammatical information do they capture?

   (b) *I know there was some reason why I learned calculus.* 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 $p_H = \frac{N_H}{N}$.

      i. Prove that this is the value of p_H which makes D most likely.

      ii. 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 probabilities, or make use of a prior distribution.

Practical Questions

1. **Language Models applied to TrUeCasIng (8 pts).** TrueCasing is the process of taking text with missing or unreliable case information, and fixing it. For example, 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 .

   Note that we've used a deterministic tokenizer to chop words, and that it in general keeps punctuation separate from words.

   TrUeCasIng has various uses:

- Older folk will remember that there used to be telegrams written in all uppercase. TrUeCasIng could be used to turn them into something more readable.
- Younger folk will know that they often get email or IM messages in all lowercase. TrUeCasIng could be used to turn them into something more readable.
- If speech recognition only gives back a list of recognized words, you want to do an operation which includes TrUeCasIng (but also, the insertion of punctuation, etc.) to transform that text into nicely edited text in a word processor.
- Under various common conventions, text sometimes has initial letters capitalized or is in all caps, even when this is not the usual capitalization of the words (for example, in headings or book or movie titles). For many other NLP purposes (such as parsing and interpreting these sequences), it would be easier to do it on text that is capitalized in 'the normal fashion'. For example, in English, exploiting the fact that, in non-initial position, capitalized things are usually names of some sort, is very helpful for name recognition.

As the above example illustrates, 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.gold` the real case information for recap.input

These files are available in the directory: /afs/ir/class/cs224n/hw3 . The first thing to do is to have a look at the data so that you are familiar with the problem to be solved.

Recall that a language model gives the probability of a sequence of words:

$$P_\theta(w_1 \cdots w_n) = \prod_{i=1}^{n} P(w_i | w_1 \cdots w_{i-1})$$

$$\log_2 P_\theta(w_1 \cdots w_n) = \sum_{i=1}^{n} \log_2 P(w_i | w_1 \cdots w_{i-1})$$

(Here θ is representing all the parameters in your model: the probabilities you give to different words and words following other words.)

The per-word cross entropy for $W = w_1 \cdots w_n$ is:

$$H(W, \theta) = -\frac{1}{n} \log_2 P_\theta(W)$$

The perplexity of the test data T is defined as:

$$PPL(W, \theta) = 2^{H(W,\theta)}$$

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

We will be using the SRI Language Modelling Toolkit to implement the language model. The software is installed in /afs/ir/class/cs224n/srilm/ . For SUN Sparc's (such as all the elaine's), the programs to be used are in the following directories: /afs/ir/class/cs224n/srilm/bin and /afs/ir/class/cs224n/srilm/bin/sparc-elf. We haven't compiled the software for other architectures. So use an elaine. You can provide an explicit path or modify your shell PATH variable to access these programs. For example, if using (t)csh, you would say:

setenv PATH /afs/ir/class/cs224n/srilm/bin:/afs/ir/class/cs224n/srilm/bin/sparc-elf:$PATH

The documentation for the programs are HTML files in the following location:

/afs/ir/class/cs224n/srilm/man/html

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.

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`, which applies the trigram model computed above to any text W to find $PPL(W, \theta)$.[3]

The following command reports the perplexity for the file named filename.

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:

[2] Ignore the warning message 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 (here, really "paragraph", but most newswire paragraphs are only one sentence long). 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.

l_w	c_w	$P_m(c_w\|l_w)$
hal	HAL	1/2
hal	Hal	1/4
hal	hal	1/4
9000	9000	1

Here, m is a model that maps unigrams from lowercase to TrueCase. It enumerates possibilities for TrUeCasIng each word, that is, here

$$P_m(HAL|hal) + P_m(Hal|hal) + P_m(hal|hal) = 1$$

Only these three TrueCase options are considered for the input "hal" (because that's all we saw in the training data). Based on this table, we can now create the following possible TrueCase sentences:

(a) C1 = "HAL 9000"

(b) C2 = "Hal 9000"

(c) C3 = "hal 9000"

The most likely TrueCase sentence C can be defined as:

$$\hat{C} = \arg\max_{C_i} P_\theta(C_i) \times \prod_{w \in C_i} P_m(c_w|l_w)$$

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

$$P_\theta(\text{"HAL 9000"}) \times P_m(\text{HAL}|\text{hal}) \times P_m(9000|9000)$$

P ("HAL 9000") can be easily computed using a language model . So, in order to attack the TrueCasing problem, all we need are three components: P_θ, P_m, and the computation of the $\arg\max$ above.

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 $\hat{C}$. 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) Compare the perplexity (ppl) of recap.input with recap.gold to test the assertion that lowercased text should have a higher perplexity when compared to TrueCased text. Report the perplexity values (ppl) thaty ou obtain.

(b) Create the mapping model P_m (this will require a little programming or scripting. . .). 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
    . . .

Here, w11, w12, ..., w1n are words from the training data. w1 is the lowercase form of w11, w12, ..., w1n and p11 is the probability $P_m(w11|w1)$ which is computed as a relative frequency estimate.

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
black black 0.73 Black 0.21 BLACK 0.06

(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 Report the command that you used.

(d) Score the accuracy (or, the opposite, the error rate) of your TrueCasing model. Report this error rate.

(e) Look at a few of the errors the model makes, and include a brief discussion of why it goes wrong (never seen the word before, misleading distribution, needs more context, ...).

(f) Bonus: Can you reduce the error rate further? Three things to try: (1) building a different, better language model by exploring the options to the ngram-counts program; (2) always uppercase the first letter of the first word in a sentence,[4] and (3) always uppercase the first letter of an unknown word (a word not seen in the training data). If you can do better, report the method and the results. (Building the language model based on the test data *isn't* a good solution to propose.[5])

[4] You should be able to do this with `sed`!

[5] Why not?