Computer Science 401 — 10 February, 2010
St. George Campus — University of Toronto

Homework Assignment #2
Due: Friday, 5 March, 2010 at 12 PM, IN CLASS

Letter-substitution Ciphers

TA: Jackie Cheung

1 Introduction

This assignment will give you experience in working with hidden Markov models, bigram models, and Baum-Welch re-estimation.

Your task is to build a hidden Markov model (HMM) to decipher texts that have been encoded with a letter-substitution cipher, without the use of annotated data.

2 Decipherment

What does this say?

> NKaIcAvXaOAX AV3aoAlibUKAfwAJ3ibAiIAaKI3i2KbARKaaiLLAE124YAv3OiI3LAR3aoKIJA
> IXAJKLLAiIJANKaIcAetDiOnK2IAlK2I3LAvXaOAD2iI
> 5YKaKAiJA2XAOaKJJi2UA2KKbAIXAJKLLAIYKAD2iIATDIAqKA3aKAbXi2UAiIAJXAqKA4
> 32A4X24K2Ia3IKAX2AXDaA4XaKATDJi2KJJAaK2Ii2UA3DIXnXTiLKJAi2AIYKABsA32bA3Ta
> X3bAJ3ibA8iLLi3nAsLibKaANKaIcAJAKmK4DIiGKAGi4KAOaKJibK2I
> 8KA3aKAX2L1AUXi2UAIXAJKLLA3IAIYKAaiUYIAOai4KA NKaIcAetDiOnK2IAY3bAXOKa3Ii

Give up? It's the beginning of the file, */u/cs401/Texts/test_data_plain.txt*. This entire document is enciphered as */u/cs401/Texts/test_data.txt*.

Cryptographic protocols today use very sophisticated functions for encoding data, but there was a time in antiquity when simply substituting one character for another was considered secure. These functions are called *letter-substitution ciphers*. They make fun parlour games now, and they are still of some interest among people who study *archaeological decipherment*, in which ancient writing systems that use unknown scripts are translated — these are special because generally the author did not mean for them to be unreadable!

In this assignment, we'll be considering text with upper and lower case letters, as well as digits and interword spaces — no punctuation. We'll take the upper and lower case variants of a letter to be distinct symbols, however – no special relationship can be assumed in how they are encoded. We'll also assume that space is no different than any other character, in that interword spaces can be encoded by and stand for another character, but that sentence boundaries are always indicated with a newline. This is an idealization: there are few writing systems that have absolutely no punctuation, a great many writing systems that do not indicate word or end-of-sentence boundaries, but probably not so many that would use interword spaces to indicate an actual sound value.

We'll also assume that we have a very definite hypothesis of the language that the enciphered text is written in: English. Not only that, but we'll assume that the enciphered text is derived from a letter-substitution that has been applied to standard English spelling (as opposed to a phonetic transcription), and that the letter-substitution is *bijective*. This means that every letter of plain text is encoded by only one letter of cipher text, and *vice versa*. As a result, there is a unique inverse to any letter-substitution function that is used to encipher text. It is your job to find this inverse, given a lot of plain-text English training data, and a sample of enciphered text. The inverse can then be used to decipher the test data, as well as other documents that are enciphered using the same function.

Warning! We are not just asking you to decipher this one document. When we test your code, we'll test it on different documents that have been enciphered with different letter-substitution functions, and it should work. Do *not* hard-wire specific details of the function used to encode */u/cs401/Texts/test_data.txt* into your implementation. All that you can assume is the general information stated here.

3 Your tasks

1. Preprocess the texts [5 marks]

Write a Python program that translates text (enciphered or plain) into sequences of numbers. You may assume that punctuation has already been stripped out of the text. Map every character, including interword spaces, but not including newlines, into a fixed positive integer. Different occurrences of the same character, even on different lines or in different files (training and test data, for example), should receive the same number. Your program should map newlines to newlines. Make sure that your output separates each integer with a space.

This step is necessary because the HMM utility which you will be using can only work with positive integers, not other symbols. Call this program, `w2id.py`.

2. Estimate initial [5 marks] and bigram [5 marks] probabilities

You should build a model of conditional bigraph probabilities using the plain text in */u/cs401/Texts/training_data.txt*. A bigraph is like a bigram, but at the character level. You will need to run `w2id.py` on */u/cs401/Texts/training_data.txt* for this task.

There are actually two parts to this. Write one program that estimates the probability of a sentence (input line) of plain text beginning with a particular character. Call this program `get_init.py`. Write another program that estimates, for each pair of characters A and B, the probability of seeing B in English plain text after having seen an A, i.e., $P(B \mid A)$. Make sure that these probabilities are normalized over contexts A. Call this program `get_cond.py`.

The outputs of both programs should use negative base-10 logarithms of probabilities instead of probabilities themselves. The output of `get_init.py` should be a sequence of pairs, one per line, of the form *ID val*, where *ID* is the id number assigned to a particular character by `w2id.py`, and *val* is the negative base-10 logarithm of the probability that that character will begin a sentence. Sort the output by ID number, and call it *init.txt*.

The output of `get_cond.py` should be a sequence of triples, one per line, of the form *AID BID val*, where *AID* is the ID number assigned to the context character A above by `w2id.py`, *BID* is the ID number of the predicted character B above, and *val* is the negative base-10 logarithm of the probability that B will follow an occurrence of A. To conserve space, do not generate any line for which *val* would be infinity (corresponding to probability zero). Sort the output by *AID*, and within each value of *AID* by *BID*. Call this file *cond.txt*.

You do not need to perform any smoothing on either estimate.

3. Build an HMM [25 marks]

Design an HMM that will allow you to decipher a text without annotated training data using the models from step (2) and Baum-Welch re-estimation. There is no specific topology (an arrangement of states and edges) required here — just find the best one that you can for this task.

Write a program called `run_bw.py` that initializes your HMM, and runs N iterations of Baum-Welch re-estimation on the test data, for a command-line input parameter N.

`run_bw.py` should call the utility, */u/cs401/sbw/sbw*, which runs one iteration of Baum-Welch on the model. The Appendix contains information on the input/output formats for this utility. You may write temporary files, but upon termination `run_bw.py` should have deleted all of them. *It is your responsibility to ensure that your temporary files will not exhaust the TA's disk space when he runs your code.*

Call the final output matrix of the HMM, after all iterations have completed, *output.txt*. This file should be formatted just as */u/cs401/sbw/sbw* expects the initial output matrix to be.

4. Decipher the test data [10 marks]

Using *output.txt*, map */u/cs401/Texts/test_data.txt* to the file *test_data_deciphered.txt*. You do not have to implement a Viterbi decoder (although you may) — use the output matrix however you like. Make sure that you translate the id numbers back into characters, using the same function that `w2id.py` used. Call this program `decipher.py`. In your discussion, be certain to tell us the value of N that you used to obtain this output. Give us only your best *test_data_deciphered.txt*, and the *output.txt* for it.

5. Compute error [15 marks]

Write a Python program, `error.py`, that compares your output to the correct decipherment. Do this in two different ways. First, compare every character of *test_data_deciphered.txt* to the corresponding character of */u/cs401/Texts/test_data_plain.txt*. Count the number of characters that you got wrong. A lower score is better in this measure.

Second, compare your output matrix, *output.txt*, to the unique output matrix that contains only 0s and 1s and makes no errors. The actual matrix depends on your HMM topology — give us the one for your HMM as *goldoutput.txt*. Explain how you performed this comparison in your discussion. **Hint:** use Kullback-Leibler divergence.

This program should generate a report with these computations on standard output. Be sure to generate enough text along with the two error measures so that we know which they are.

6. Perform steps (3)-(5) again [25 marks]

This time, however, write a different program, called `run_mbw.py`, which runs Baum-Welch in a very strange way. After each round of re-estimation, `run_mbw.py` will discard the new transition matrix and initial probability vector, and run the next round using the new output matrix, but the original transition matrix, and the original initial probability vector. So Baum-Welch will make changes in each iteration, but you throw out all of the changes except the new output matrix. Call the final output matrix *moutput.txt*, and the resulting decipherment *test_data_mdeciphered.txt*.

You may experiment with different HMM topologies, but you should submit a version of `run_bw.py` and `run_mbw.py` that that use the same topology. Submit whichever topology gives you the best overall performance in either version.

7. Discussion [10 marks]

Why did you choose the HMM topology that you did? What others did you try (if any)? What value of N did you need to use for Baum-Welch? What value of N did you need with the modified training regimen?

Did the modification seem to help? Why do you think that it did (not)? What error measures did your best attempts receive? How did you compare output matrices? Looking through your output, were there particular characters that your model had trouble with? Why?

Please limit your answer to 500 words or less.

4 General specification

When we test your code, we will call it like this:

```
python w2id.py /u/cs401/Texts/training_data.txt > training_data_id.txt
python get_init.py training_data_id.txt > init.txt
python get_cond.py training_data_id.txt > cond.txt

python w2id.py /u/cs401/Texts/test_data.txt > test_data_id.txt
python run_(m)bw.py test_data_id.txt init.txt cond.txt N

python decipher.py test_data_id.txt (m)output.txt > test_data_(m)deciphered.txt
python error.py /u/cs401/Texts/test_data_plain.txt test_data_(m)deciphered.txt (m)output.txt
goldoutput.txt
      > report
```

where N is a number of iterations that we choose. `run_(m)bw.py` should create *(m)output.txt*.

Remember that we will test your code on different documents that use different letter-substitution functions. Do not hardwire the substitution for */u/cs401/Texts/test_data_plain.txt* into your code. You may hardwire the character-to-id mappings in `w2id.py` and `decipher.py` if you wish. We do not care which IDs you assign to which characters.

As part of grading your assignment, the grader may run your programs using Unix scripts. It is therefore important that your each of your programs precisely meets all the specifications and formatting requirements, including program and file names. **A program or HMM that cannot be evaluated because it varies from specifications will receive zero marks.**

If a program uses a file or helper script name is specified within the program, it must read it either from the directory in which the program is being executed, or it must read it from a subdirectory of */u/cs401* whose path is completely specified in the program. Do **not** hardwire the absolute address of your home directory within the program; the grader does not have access to this directory.

All your programs must contain adequate internal documentation to be clear to the graders. External documentation is not required.

5 What to hand in

Except where noted, your assignment should be submitted both on paper (so that the graders can easily view the results and scribble on them) and electronically (so that the graders can run your programs and grep your results if they need to). You should submit:

1. All your code for `w2id.py`, `get_init.py`, `get_cond.py`, `run_bw.py`, `run_mbw.py`, `decipher.py`, and `error.py` (including helper scripts, if any).

2. A brief description of your HMM, with an indication of what the states correspond to. Be precise; a reader should be able to reimplement your model from the information you provide. You must indicate how many states you had to use in your model.

3. The files *init.txt* and *cond.txt* (electronic only — no paper copy).

4. Your output files, *output.txt*, *moutput.txt*, *goldoutput.txt*, *test_data_deciphered.txt* and *test_data_mdeciphered.txt* for the best HMM topology you tried, and for the best value of N (electronic only — no paper copy). The versions you submit for the modified training regimen in step (6) can be for a different N, but must have used the same topology.

5. Your discussion (paper only—no electronic copy).

On paper: Your paper submission must be in an *unsealed* 9×12-inch envelope. On the outside of the envelope, or on a sheet attached to the outside of the envelope, provide the following information:

- your first and last name — underline your last name
- your student number,
- your CDF login id,
- your preferred contact email address,
- whether you are an undergraduate or graduate
- this statement with your signature below it: *I declare that this assignment, both my paper and electronic submissions, is my own work, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters and the Code of Student Conduct.*

Envelopes missing any of this information, and electronic submissions without timely paper submissions will not be marked.

Electronically: The electronic submission must be made from the CDF submission site. Do not `tar` or `compress` your files, and do not place your files in subdirectories.

6 Working at home

If you want to do some or all of this assignment on your home computer, you will have to do the extra work of downloading and installing the requisite software and data. You take on the risk that your home computer might not be adequate for the task. You are strongly advised to upload regular backups of your work to CDF, so that if your home machine fails or proves to be inadequate, you can immediately continue working on the assignment at CDF. When you have completed the assignment, you should try your programs out on CDF to make sure that they run correctly there. **A submission that does not work on CDF will get zero marks.**

Appendix: sbw

We have provided a utility, */u/cs401/sbw/sbw*, to help you with Baum-Welch re-estimation. This utility comes with no documentation of its own, apart from this appendix.

The input format is as follows:

```
******************************
N K M
pi1 ... piN
#cols1 a11 ... aN1 (sparse, transposed)
#cols2 a12 ... aN2
...
#colsN a1N ... aNN
#cols1 b11 ... bN1 (sparse, transposed)
#cols2 b12 ... bN2
...
#colsK b1K ... bNK
#cols1 o1 ... oT1 (the first observed sequence/sentence)
#cols2 o1 ... oT2
...
#colsM o1 ... oTM (the M-th observed sequence/sentence)

Notation:
N = number of states
K = number of output symbols
M = number of output observed sequences
Ti = length of the i-th observed sequence/sentence
******************************
```

All of the variables referred to here are positive integers, even the actual characters of the observed sequence(s).

Note well: the transition and output matrices are transposed. This means that they have been rotated so that their columns are rows and their rows are columns.

Note extra well: you need not – and should not — specify probabilities that are 0. Each aij and bij is actually a pair, $r : v$, where r is the row number (of the actual matrix, column number of the transposed matrix) and v is the value at that position. Values are negative base-10 logarithms of probabilities, and are assumed to be row-normalized (in the transposes, column-normalized).

The output of `sbw` to `stderr` consists of the number of processed sentences and the average log-likelihood of the observed sequences. The output to `stdout` consists of new Π, A, and B parameters in the same order and format as the input. The parameters N, K and M are not repeated, nor are the observed sequences.