COMS 4705 – Natural Language Processing – Spring 2015

Assignment 4

Machine Translation

(version 4.2, 4/9/15)

Due: Friday 4/24/15 at 11:59 PM

Getting Started

The aim of this assignment is to understand and implement various models for machine translation. We will be using the German-English corpus from Comtrans. The corpus contains roughly 100,000 pairs of equivalent sentences, one in German and one in English Our goal will be to determine the alignments between these sentences using different MT models and compare the results.

Assume we are given a parallel corpus of training sentences in two languages. For example, the first pair in the corpus is:

e = Wiederaufnahme der Sitzungsperiode
f = Resumption of the session

For each pair of sentences, we want to determine the alignment between them. Since we do not have any alignment information in our training set, we will need to determine a set of parameters from the corpus, which will enable us to predict the alignments.

For IBM Model 1, we need to determine one parameter set: t(f | e). The q parameters will be initialized to the uniform distribution, meaning that q (j | l, l, m) = 1 / (l + 1). The t parameter represents the translation probability of word f being translated from word e. This results in a simple model that produces alignments based on the most probable translation of each word.

For IBM Model 2, we need to determine two parameter sets: q(j | i, l, m) and t(f | e). The q parameter represents the alignment probability that a particular word position i will be aligned to a word position j, given the sentence lengths l and m. This results in a model that examines both the word translations and the position distortion of a word in the target sentence.

In order to compute these parameters, we use the EM algorithm. The EM Algorithm takes an initial set of parameters and attempts to find the maximum likelihood estimates of these parameters under the model. There are 2 steps: the Expectation step, which computes the the likelihood of the current parameters, and the M step, which updates these parameters to maximize the likelihood. A very detailed explanation with some intuition behind it can be found here:

http://www.isi.edu/natural-language/mt/wkbk.rtf

In this assignment, you will implement a simplified version of the BerkeleyAligner model. This model utilizes the concept of alignment by agreement. Essentially, you will be training two IBM model 2s simultaneously, one from the source language to the target language and one from the target language to the source language. The idea is that intersecting these two models will eliminate some of the errors that unidirectional translation models would produce. In order to accomplish this, the EM algorithm initializes two models and maximizes the combined probability of both models. More information can be found here:

http://cs.stanford.edu/~pliang/papers/alignment-naacl2006.pdf

Tutorial

You will be using many of the alignment tools available in NLTK.

Add NLTK's location to your default path Add the following line of code to your ~/.bashrc file.

```
export PYTHONPATH=$PYTHONPATH:/home/cs4701/python/lib/python2.7/site-packages
```

After, don't forget to run:

```
source ~/.bashrc
```

Create a link to the NLTK files (equivalent to downloading the files yourself) Login to your CLIC account and run the following commands in your home directory.

```
ln –s ~coms4705/nltk_data nltk_data
```

To test that you have successfully linked to the nltk directory, open the python interpreter and run:

```
from nltk.corpus import comtrans
```

This should not produce any errors.

One important object is the AlignedSent object, which are given in the ComTrans corpus object:

```
AlignedSent(['Resumption', 'of', 'the', 'session'], ['Reprise', 'de',
'la', 'session'], Alignment([(0, 0), (1, 1), (2, 2), (3, 3)]))
```

The original sentence can be obtained using: aligned_sent.words. The translated sentence can be obtained with aligned_sent.mots. The alignment array can be obtained with aligned_sent.alignment. Each tuple (i, j) in the alignment indicates that words[i] is aligned to mots[j]. Each sentence in the

An IBMModel1 object can be created with:

```
ibm = IBMModel1(aligned_sents, num_iters)
```

"num_iters" is the number of times you want the EM algorithm to iterate for. The model automatically trains on initialization. You can then call the align method to predict the alignment for an AlignedSent object.

Each AlignedSent object in the corpus contains the correct alignments. In order to evaluate the results of the model, use the AlignedSent.alignment_error_rate() function.

Assignment

The assignment can be found in:

/home/coms4705/Documents/Homework4

It contains the following files:

A.py – contains skeleton code for Part A
B.py – contains skeleton code for Part B
EC.py – contains skeleton code for Part B extra credit
main.py – runs the assignment

[Do not edit the main methods in any of the files.]

You must use the skeleton code we provide you. Do not rename these files. Inside, you will find our solutions with most of the code removed. In the missing code's place, you will find comments to guide you to a correct solution.

We have provided code that creates all output files. Do not modify this code in any way, and pay close attention to the instructions in the comments. If you modify this code it may be very difficult to evaluate your work and this will be reflected in your grade. You may add helper functions if you would like.

The Report

You are required to create a brief report about your work. At the top it should include your

UNI and the time you expect your programs to complete. Throughout the assignment, you will be asked to include specific output or comment on specific aspects of your work.

Part A – IBM Models 1 & 2

In this part, we will be using NLTK to examine IBM models 1 & 2. You will need to implement the methods compute_average_aer(), save_model_output(), create_ibm1(), and create_ibm2().

1) Initialize an instance of IBM Model 1, using 10 iterations of EM. Train it on the training set. Then, save the alignments for the first 20 sentence pairs in the corpus to "ibm1.txt". Use the following format for each sentence pair:

Source sentence	[as given by AlignedSent.words]
Target Sentence	[as given by AlignedSent.mots]
Alignments	[as given by AlignedSent.alignment]
(blank line)	

2) Initialize an instance of IBM Model 2 using 10 iterations of EM. Train it on the training set. Then, save the alignments for the first 20 sentences in the corpus to "ibm2.txt". Use the same format as above.

3) For each sentence pair, compute the alignment error rate (AER). Compute the average AER over the first 50 sentences for each model. Compare the results between the two models. Specifically, highlight a sentence pair from the development set where one model outperformed the other. Comment on why one model computed a more accurate alignment on this pair.

4) Experiment with the number of iterations for the EM algorithm. Determine a reasonable number of iterations (in terms of processing time), which provides a lower bound on the AER. Discuss how the number of iterations is related to the AER.

Part B – Berkeley Aligner

In this part we will improve upon the performance of the IBM models using a simplified version of the BerkeleyAligner Model. In this model, we will train two separate models simultaneously such that we maximize the agreement between them. Here, we will be digging a bit more into the machine translation algorithms, and implementing the EM algorithm.

You will need to complete the implementation of the class BerkeleyAligner, which will support the same function calls as the NLTK implementations of the IBM models.

For the EM algorithm, initialize the translation parameters to be the uniform distribution over all possible words that appear in a target sentence of a sentence containing the source word. Initialize the alignment parameters to be the uniform distribution over the length of the source sentence.

We will be using a simplified quantification of agreement between the two models (in comparison to that proposed in the paper). When you compute the expected counts, use the average expected count with respect to the two models' parameters.

Both model parameters can be stored in the same dictionary. However, the translation and distortion parameters should be stored in separate dictionaries.

1) Implement the train() function. This function takes in a training set and the number of iterations for the EM algorithm. You will have to implement the EM algorithm for this new model. Return the parameters in the following format:

   (translation, distortion)

2) Implement the align() function. This function uses the trained model's parameters to determine the alignments for a single sentence pair.

3) Train the model and determine the alignments for the first 20 sentences. Save the results to "ba.txt". Use the same format as in part A.

4) Compute the average AER for the first 50 sentences. Compare the performance of the BerkeleyAligner model to the IBM models.

5) Give an example of a sentence pair that the Berkeley Aligner performs better on than the IBM models and explain why this is the case.

6) (Extra Credit) Think of a way to improve upon the Berkeley Aligner model. Specifically examine the way we quantify agreement between the two models. In our implementation, we computed agreement as the average expected count of the two models. Implement an improved Berkeley Aligner model that computes agreement in a better way. There is skeleton code in EC.py (same as for B.py) Compute the average AER for the first 50 sentences. Compare to the other models. Again, this part is optional but if your implementation is interesting and shows improved performance, you will be eligible for bonus points.

Deliverables

A.py – methods implemented

Ibm1.txt – contains the first 20 sentence pairs and their alignments for IBM Model 1

Ibm2.txt – contains the first 20 sentence pairs and their alignments for IBM Model 2

B.py – methods implemented

ba.txt – contains the first 20 sentence pairs and their alignments for Berkeley Aligner

EC.py – improved Berkeley Aligner code [optional]

README.txt – writeup of all written portions as well as general comments about your code and how it works or any issues that occur while running it

Make sure running main.py does not fail and that all output is printed correctly. The output of main.py will makeup a large portion of your grade.

Here are some additional problems related to translation:

http://www.nacloweb.org/resources/problems/2012/N2012-C.pdf

http://www.nacloweb.org/resources/problems/2012/N2012-C.pdf

http://www.nacloweb.org/resources/problems/2010/C.pdf