Project 3: Machine Translation

Due: Friday Mar. 18th, 11:59pm (Eastern Time)

Deliverables

- 5 python files:
  - `ibm-estimator.py` (which computes IBM model one translation parameters using EM)
  - `vdd.py` (which uses the translation parameters learned by `ibm-estimator.py` to perform a simple, direct decoding)
  - `ncd.py` (which uses a noisy channel approach to combine the translation parameters learned by `ibm-estimator.py` as well as the language model probabilities (from `lm.py`) to perform a more reasonable, but still direct decoding)
  - `lm.py` (a bigram language model implementation which will be used as part of `ncd.py`)
  - `fscore.py` (a program which computes the F-score of your translation outputs)
- `writeup.tex` and a corresponding `writeup.pdf` with answers to the Lab questions and which briefly lists the relevant output generated by the program.

All the data you'll need for this is in `/data/cs65-S22/mt/`. Feel free to add your newly generated translation output (e.g. `english-translation-vdd.txt` and `english-translation-ncd.txt`) directly to your GitHub repo but leave the original data in `/data/cs65-S22/mt/`.

1 Parameter estimation for IBM Model 1

Start by building an IBM model 1 parameter estimation program (`ibm-estimator.py`). This program takes as arguments two input files, one for the French half of the parallel corpus, one for the English half. Make sure that your program does not care which is which, so that by switching the order of the files your program will switch between computing $P(e_j|f_k)$ and $P(f_k|e_j)$.

In addition to whatever helper functions you choose to write, make sure that this program includes a function named `get_taus()` which can be called by other Python programs in the future. `get_taus()` should return a data structure (exact details up to you) which contains the proper $\tau_{e,f}$ for each pair of English and French words that your programs believes can serve as translations of one another (or $\tau_{f,e}$ if you flip the order of the input arguments.)

Your program will compute the individual $\tau_{e,f}$s using the EM algorithm that we discussed in class. To initialize EM we assign all translation probabilities the same value, which for consistency and convenience will be 1. Equation 3 tells us how to compute the fractional counts for each word (the e-step), and then at the end of each iteration, Equation 1 does the m-step. Ten iterations or so should be sufficient for all the parameter values to settle down. But you should pick a threshold for convergence using Equation 2.10 from the Charniak reading.

$$\hat{\tau}_{e,f} = \frac{n_{e,f}(a)}{n_{e,\circ}(a)} \tag{1}$$

$n_{e,\circ}(a)$ is of course the number of times that e is aligned to any French word in a (this is not necessarily the same as the number of times e appears in the training corpus).

$$n_{e,\circ}(a) = \sum_f n_{e,f}(a) \tag{2}$$

We derive the fractional counts for each word (during the e-step) by the following:

$$n_{e_j,f_k} += \frac{\tau_{e_j,f_k}}{p_k} \tag{3}$$

$$p_k = \sum_j \tau_{e_j,f_k} \tag{4}$$

After building the parameter estimation program for IBM model 1, you'll need to convince yourself (and me!) that you've implemented things correctly. Part of the assignment is to come up with ways to do that.

Questions

1. Using `English-senate-0.txt` and `french-senate-0.txt` as training data: print out some τs for French words with reasonably obvious English translations. Foremost should be punctuation. If you do not know any French, you can also look at the French for words that look like English words and occur several times— many French words have been borrowed into English.
2. Describe, implement, and report the results here for at least one unit test that gives you confidence that your IBM-model 1 parameter estimation program is working correctly.

2 MT with Very Dumb Decoding

Next write an MT program (`vdd.py`) based upon IBM model 1 and our very dumb decoder, which simply goes through an incoming French sentence and for each word f_i outputs $argMax_{e_j}(P(e_j|f_i))$. This should use the τ parameters estimated via `ibm-estimator.py`. The decoder itself should end up being quite short; likely only a few lines of code.

When your program encounters French words it has never seen before, just pass them through to the English output without change.

Use `English-senate-0.txt` and `french-senate-0.txt` as the training data. We are not tuning any smoothing parameters, so there is no particular need for held-out data.

Questions

3. Use `vdd.py` to translate `french-senate-2.txt` and save your translation to a new file `english-translation-vdd.txt`
4. Describe, implement, and report the results here for at least one unit test that gives you confidence that your Very Dumb Decoder program is working correctly (assuming that your τs from part 1 are correct).

3 MT with a Simple Noisy-Channel Decoder

Now we'll try to do better. For this part of the assignment you'll again be decoding one word at a time, but now you'll compute the compute the reverse translation probabilities, $P(f_k|e_j)$. In addition, you'll adapt the English bigram language model you built during Lab 2.

In this version (`ncd.py`) instead of maximizing $P(e_j|f_k)$, you should maximize $P(e_j|e_{j-1})P(f_k|e_j)$.

Since we're going from left to right one word at a time, we will know the previous English word ay each step (In the model, you should set the zeroth word to be *START* but don't actually print this to the output).

Just like for the Very Dumb Decoding, use `English-senate-0.txt` and `french-senate-0.txt` as the training data. When your program encounters French words it has never seen before, just pass them through to the English output without change.

Questions

5. Use `ncd.py` to translate `french-senate-2.txt` and save your translation to a new file `english-translation-ncd.txt`
6. Describe, implement, and report the results here for at least one unit test that gives you confidence that your noisy channel program is working correctly (assuming that your τs from part 1 are correct).

4 MT Evaluation

Now that you have two different working MT system we'd like to compare them. Since we only have one reference translation for each original sentence we'll do that directly according to *F-score*. As we discussed in class, F-score is a standard measure of accuracy which is defined as the *harmonic mean* of *precision* and *recall*:

$$F = 2 * \frac{precision * recall}{precision + recall} \tag{5}$$

Precision is the number of correct results divided by the number of all returned results. While recall is the number of correct results divided by the number of results that should have been returned. In this case, we are counting the number of individual word tokens translated correctly. You may consider a translation of the word "pain" in `french-senate-2.txt` to "bread" correct if "bread" occurs anywhere in the corresponding sentence in `english-senate-2.txt`. The total number of returned results is, of course, the word-token count of your translated output file (which will be exactly the same as that of `french-senate-2.txt`), and the number of results that should have been returned is the word-token count of `english-senate-2.txt`

Write a program (`fscore.py`) which prints out the F-score of an input translation file in comparison to the correct reference translation file `english-senate-2.txt` just for sentences of length 10 or less. Your translation output will include cases of sentences longer than that, but simply exclude them from the calculation of the resulting F-score.

Questions

7. Use `fscore.py` to report the F-score for both your `english-translation-vdd.txt` and `english-translation-ncd.txt` and outputs.
8. Subjectively, which of your two translations looks better to you? Which gives the better F-score, and why? What does this result tell us about the two different decoders? And finally, what does this tell us about our chosen method of evaluation?

Implementation notes

Both of your translators should be producing results in the range of 50% to 60%, or perhaps a bit higher. That isn't very good in the grand scheme of things, but it is far better than simply choosing words at random

As you start working on programming your solutions, keep in mind that these systems will take a long time to run if implemented poorly. In order to allow your projects to be graded in a reasonable amount of time, I require that all handins run in less than 30 minutes or else I won't be able to grade it. (I don't expect your solutions will take this long, but it's just an upper-bound).

If your program is taking a long time to run, you should consider applying the following optimization: after you finish running the EM algorithm, but before you begin decoding, throw away any τ values less than some fairly small value epsilon. You can experiment with different values of epsilon, but you can get good results even with seemingly large values. Of course, if your F-Scores seem too low, you should make sure that this optimization is not what is causing the problem (e.g., run the program once without the optimization, and see whether you get better results).

Also, remember that storing a 2-dimensional array with every possible pair of English and French words will require more memory than a department computer can realistically provide. Thus, you will need to use a sparse mapping of some sort (I recommend a nested dictionary structure) to store your τ and n values.

5 Feedback

Please provide answers to these in `writeup.tex` just like the rest of the questions. There are no right or wrong answers here of course :)

Questions

7. Approximately how long did you spend on this assignment?

8. Which aspects of this assignment did you find most challenging? Were there any significant stumbling blocks?

9. Which aspects of this assignment did you like? Is there anything you would have changed?

6 Extra Credit Ideas

If there's something cool you'd like to do, go ahead and try it! Here's two possible ideas:

- See if you can come up with and implement a decoding scheme that works more effectively than the two you are required to implement as part of the lab.
- Try and visualize the effect of some of our design decisions on both EM convergence and translation output. Try making a plot of the change in likelihood at each iteration of EM (you should find that it asymptotes pretty quickly). Or try making a plot of the effect of different minimum τ cutoffs on the resulting F-score. Perhaps you can come up with a way to visualize the among of space savings vs. performance losses here!

For any extra credit ideas you choose to do, simply add your code to the repo, and describe the design and relevant results in the writeup. I will award up to 20% extra credit on the lab depending on the scope of exactly what bonus material you complete.