

P1: Language modeling and tagging

Note: Please submit the written aspects of assignments in postscript or PDF format only. I highly recommend you use \LaTeX to prepare the assignments. A solution will be posted here after the due date. See <http://www.cs.utah.edu/classes/cs5964/handin.html> for handin instructions.

ALL QUESTIONS ARE EQUALLY WEIGHTED.

Introduction

In this project we will build a model for entity tagging based on the noisy channel framework, using the Carmel toolkit. A binary for Carmel exists in `~cs5964/bin/carmel`. Documentation is on the class webpage.

The data for the project is in `~cs5964/p1/`. There are three files here, `train`, `dev`, `test`. The `train` and `dev` files are labeled, but the `test` file is unlabeled. The first line in the training file looks like:

```
Peter_I-PER Blackburn_I-PER
BRUSSELS_I-LOC 1996-08-22_0
The_0 European_I-ORG Commission_I-ORG said_0 on_0 Thursday_0 it_0 disagreed_0 with_0 \
  German_I-MISC advice_0 to_0 consumers_0 to_0 shun_0 British_I-MISC lamb_0 until_0 \
  scientists_0 determine_0 whether_0 mad_0 cow_0 disease_0 can_0 be_0 transmitted_0 \
  to_0 sheep_0 ._0
```

As we can see, each sentence is on its own line. Words are separated from their tags by an underscore, and tags are of the form “B-X”, “I-X” or “O”. The task is to add these tags to the `test` file.

We will, of course, be solving this problem with the noisy channel model. In particular, we will estimate $p(\text{word} \mid \text{tag})$ and a language model on tags.

Part A – The Channel Model

In this section, we build a very simply channel model. We will improve on the channel model later, but this will get something running.

In this case, the channel model will simply be just of the form $p(\text{word} \mid \text{tag})$, where “word” is the lower-cased version of the word.

Question 1: Assume that there are only four tags, “O”, “PER”, “ORG”, “LOC.” Draw a sketch of the finite state transducer corresponding to this ~~source model~~ **channel model**. You needn’t include probabilities. It may be useful to use epsilon transitions (but you obviously don’t have to).

When computing the actual pfst, we will need to deal with smoothing and unknown words. We will do this as follows. First, we will use simple “add- α ” smoothing, where α is a parameter we will vary later (for now, just use $\alpha = 0.1$). Next, we will ignore words that appear strictly fewer than five times. This will give us a vocabulary of ~~4631 words~~ **4630 words plus the unknown word**. Any word that does not fall on this list you should consider to be the word “*UNK*”.

Question 2: Write a program that reads in `train` and generates the channel model just described. Be sure to normalize the probabilities appropriately. Draw the section of the ~~psa~~ **pfst** corresponding to the PER label with the top 10 transitions (as rated by probability). Save this model in a file called ~~channel1.psa~~ **channel1.pfst** and be sure to hand it in.

Part B – The Language Model

In this part, we build a language model pfsa over tags (not words). We will use a bigram language model for this, again using “add- α ” smoothing.

Question 3: Draw the full fsa.

Question 4: Write a program that read in `train` and generates a bigram language model over tags. Store this in `language1.pfsa` and be sure to hand it in.

Part C – Adjusting the Hyperparameters

Before we adjust the α s that we used for smoothing, let’s get this thing up and running.

Question 5: Write a program that converts the `dev` file into a fsa that just generates the corresponding strings (you’ll need to strip off the labels). Using `carmel`, compose this with `channel1.pfsa` and `language1.pfsa`. This will give you output labels. You can use the script:

```
~/cs5964/bin/compare_tags.pl <file1> <file2>
```

To evaluate the performance of the tagger. Assuming that you generate a file in the same format as `dev`, this will give you the error rate of your tagger. Compute the error rate and write it down.

Question 6: Now, we’ll try varying the α parameters. In particular, for each possible value of $\alpha \in \{0.1, 1, 2, 5\}$, regenerate your pfsa and rerun the `compare_tags.pl` script. Note that you’ll have to run 16 times, since there are two “ α ”s used above. Report all scores and which combination works best. Can you provide any intuition for why that might be?

Part D – A Better Channel Model

One of the big problems with our original channel model (Part A) was that it ignores capitalization. But including all capitalization is overkill.

Here, we use a slightly different channel model. Essentially, we want to keep capitalization information for common words, ignore it for less common words, and then treat unknown words differently if they have different capitalization patterns.

Here’s what we’ll do.

Look at the lowercased version of the data. Any word that appears fewer than ten times we will consider as unknown. For example, our old friend “abortion” would be just under the cutoff (it appears once as “Abortion” and eight times as “abortion”). It would thus be considered unknown. We will have four different unknown-word tokens. One, which we’ll call “*UNK-lc*” will be for words that are entirely made up of lower-case letters and hyphens. Next, “*UNK-ic*” (ic = initial cap) will be for words that have the first letter capitalized and are followed by a non-empty sequence of lower-case letters and hyphens. Next, “*UNK-num*” will be for any unknown word that contains a number 0-9. Finally “*UNK-other*” will be for anything else.

Next, we consider non-unknown words. For these, we will maintain capitalization information. In particular, we will use the full word form, rather than the lower-cased version. The only exception is a form that appears fewer than twenty times. For instance, the lowercased word “with” appears as “with” (840 times), “With” (17 times) and “WITH” (10 times). For this word, we would keep the form “with” and replace both of the others with “with-*OTHER*”. On the other hand, for the word “the” (7243 times), “The” (1127 times) and “THE” (20 times), we would keep all three versions since they all appear more than twenty times. Finally, for the word “Washington” (42 times) and “WASHINGTON” (19 times), we would keep the first and replace everything else with “WASHINGTON-*OTHER*”.

I realize that this is a bit confusing, but hopefully the examples will help.

Question 7: Build a channel model according to this specification (again with add-0.1 smoothing) and save

it as `channel2.pfsa`.

Question 8: Using add-0.1 smoothing for both the channel model and language model, rerun on the dev data, run the evaluation script, and report your accuracy. Does this more complicated channel model help?

A Better Language Model

Question 9: Replace the bigram language model with a trigram language model. Save this as `language2.pfsa`. Rerun with both `channel1.pfsa` and `channel2.pfsa`. Report results. Does it help?

Wrapping Up

Question 10: Using $\alpha = 0.1$ in all models, run all four combinations of the channel/language models on the test data. Be sure to put the results back in the “word_tag” format. Call each `testXY`, where X,Y is either 1 or 2 and is the number of the channel model and language model used, respectively. Hand all four in.