

P2: Machine translation

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.

Introduction

In this project, we will build a machine translation from (almost) scratch! We will translate *from* Spanish (“es”) *to* English (“en”). . . this means that our *channel model* will go from en to es (i.e., backwards) and we’ll have an English language model. The high-level overview of the project is as follows:

1. You will implement IBM Model 1 for generating alignments between en and es sentence pairs that I have provided.
2. You will use these alignments to generate a pfst channel model for translating from en to es (backwards).
3. You will use use this channel model *without* a language model to translate a few sentences.
4. You will use the corpus of raw en text that I have provided to build a bigram (source) language model as a pfsa.
5. You will use the channel model *and* language model to translate the same sentences (things should look better!).
6. You will use alignments that I provide (built using higher order IBM models plus some heuristics) to build a new channel model.
7. You will rerun the test sentences through this channel both with and without the language model.
8. **Extra credit:** Repeat the previous exercise but with a trigram language model instead of a bigram language model.

The files that are relevant to this project are:

1. `es.1c.gz` – a list of 730k parallel Spanish sentences (15.6m words), tokenized, lower-cased and with accents removed. Long sentences have been removed.
2. `en.1c.gz` – a list of 730k English sentences (15.2m words), tokenized, lower-cased and with accents removed. Long sentences have been removed.
3. `es.test` – a list of 20 Spanish sentences that you are supposed to translate to English – these have been selected to be extra-short so that Carmel doesn’t die on you.
4. `alignments.good` – a list of alignments for the 730k parallel sentences, for use in the part where you use my alignments instead of your own.

Part I – IBM Model 1

In this section, you will implement IBM Model 1 and have it spit out a list of alignments. The format of the alignment file should be one sentence-per line (so you should have about 730k lines) and each line will consist of a bunch of aligned word pairs. For instance, an alignment for the first sentence pair might look like:

```
0-0 1-1 2-3 4-3
```

This means that the zero-th Spanish word is aligned to the zero-th English word, the first Spanish word is aligned to the first English word, the second Spanish word is aligned to the third English word and so on. *Null alignments* are not listed, and neither are zero-fertility alignments. The zero-th word really means the first word (this is C- or Perl-style array indexing). See `alignments.good` for an example. Call the output of this `alignments.model1`.

To make life easier on you, we'll run a fixed number of iterations—in this case, 5. This way you don't have to do anything special to judge convergence.

Here is some pseudo-code for Model I:

Initialization step

1. Initialize a ttable, \mathbf{tt}
2. For each English/Spanish sentence pair (\mathbf{n}, \mathbf{s})
 - (a) For each English word $n \in \mathbf{n}$
 - i. For each Spanish word $s \in \mathbf{s}$
 - A. Increment $\mathbf{tt}(n, s)$ by $1/|\mathbf{n}|$
3. Normalize \mathbf{tt} so that for all n , $\sum_s \mathbf{tt}(n, s) = 1$.

Iterations – repeat the following five times

1. Initialize a new ttable $\mathbf{tt2}$
2. For each English/Spanish sentence pair (\mathbf{n}, \mathbf{s})
 - (a) For each English word $n \in \mathbf{n}$
 - i. Initialize an alignment variable z
 - ii. For each Spanish word $s \in \mathbf{s}$
 - A. Set $z_s = p(s | n) \propto \mathbf{tt}(n, s)$
 - iii. Normalize z so that it sums to one
 - iv. For each Spanish word $s \in \mathbf{s}$
 - A. Increment $\mathbf{tt2}(n, s)$ by z_s
3. Normalize $\mathbf{tt2}$ and set $\mathbf{tt}=\mathbf{tt2}$

Complete

1. Repeat the “Iterations” step once more, but instead of doing steps (iii) and (iv), you will compute the Spanish word position that maximizes z_s and print this to the output file.

You should be warned that this will probably take about a half hour to and hour to run if implemented naively in Perl, so don't put it off. Also, *be sure to do something to handle NULL alignments* – this is a simple modification, but you should be sure to do it. **Hand in** your code and the resulting alignments.

Part II – Generating a Channel Model

Now, we will use the alignments you just produced in `alignments.model1` to create a channel model. The channel model will have just one state and will simply transduce English words to Spanish words. We will do this in two steps: the reason for this will become apparent later. In the first step, we'll use our alignments in `alignments.model1` to generate a ttable, called `ttable.model1`.

The format of the t-table should be a *long* list of translation probabilities between English words and Spanish words. Each line should be formatted as “*probability en-word s-word*”, where *en-word* is an English word, *es-word* is a Spanish word and *probability* is the probability of translating *en-word* to *es-word* (remember the channel goes backwards). Here's a random subset of lines from the one I created:

```
0.0232558139534884      begs      macartney
0.00578034682080925    fundamentally  enmiendas
0.122994652406417      intensive  intenso
0.0149253731343284     survey     intergubernamental
0.0625                  phoney     realidad
0.000300480769230769   see        evidente
```

To do this, we will need to write a program that will loop through all lines in `alignments.model1`, `en.lc.gz` and `es.lc.gz`. Essentially, what we'll do is count how many times an English word n is aligned to a Spanish word s . The translation probability from n to s will then be proportional to the count (properly normalized). Be sure to properly handle null alignments and zero-fertility words! You should account for these in the ttable by words called `*e*` (not surprisingly...).

Now, we will build a small script to convert `ttable.model1` to a pfst. The reason we're doing this in two steps is because it will kill Carmel if we try to build a pfst for the entire ttable. So what we'll do is build a pfst that only contains the subset of the ttable that we care about.

So, what you should do is write a program that reads in the text `es.test` and remembers which Spanish words it has seen (plus `*e*`). Next, probabilities stored in `ttable.model1` but only store those that have to do with Spanish words contained in `es.test`. Based on these, you should generate your pfst. The pfst should only have a single state and should be able to both null-generate and have zero-fertility. Write the pfst to `channel1.pfst`.

Hand in all of your code, `alignments.model1`, `ttable.model1`, and `channel1.pfst`.

Part III – Translating without a Language Model

We can translate the sentences in `es.test` using the following command:

```
% carmel -qbrIk 1 channel1.pfst es.test
```

(The `-q` means quiet – if you want to see what's going on, remove it.)

Run this and write down the translations. Look at the first five in particular. What seems to be wrong with them (if anything)? What about the last one?

Hand in the actual out and an answer to the above questions.

Part IV – Building a Language Model

Now we will build a language model based on `en.1c.gz`. First, we will build a unigram language model and then build a bigram language model. These should be called `lm1.pfsa` and `lm2.pfsa`, respectively. The only catch is that we can't build complete language models—we can only include those words that we care about.

So, what we'll have to do is actually read in `channel1.pfst` and find out what are the possible English words that it can produce and then only include these in the language model.

For the unigram language model, there should be only one state. It should have as many self-arcs as there are vocabulary items on the English side of `channel1.pfst`. The self-arc probability should be the unigram probability of that word in `en.1c.gz`.

Once you've build the unigram language model, you should be able to run:

```
% carmel -qbrIk 1 lm1.pfsa channel1.pfst es.test
```

Write down the translations your produce. How do these translations compare to the translations we had before? Explain any differences you see. Does this look better or worse?

Now, build a bigram model following the same rubric. Call it `lm2.pfst` and run:

```
% carmel -qbrIk 1 lm2.pfsa channel1.pfst es.test
```

Again, write down the translations and analyze them.

There's one glaring problem with our model so far: it doesn't allow permutations of the order of the words! We could try to build a channel model that will permute words, but this is a pain. So we'll let `carmel` do it for us using the `-P`. This only really matters for the bigram language model because the unigram language doesn't care about order anyway. Run:

```
% carmel -qbrIPk 1 lm2.pfsa channel1.pfst es.test
```

And write down the results. If something is taking too long, you can abort it. You should *at least* be able to get the first seven or eight outputs though. How do these look in comparison to those without permutations?

(Addition: *You may have noticed that these outputs use epsilons WAY too much. Create a copy of `channel1.pfst` called `channel1-eps.pfst`. Either manually or automatically go through and replace all source epsilon probabilities with $1e-6$. In other words, any time an epsilon can be created on the English side, make it's probability artificially small. Once you've done this, rerun with `lm1` and `lm2` and compare the outputs.*)

Handin the code to build the language models, the language models themselves, the outputs and answers to the above questions.

Part V – Using My Alignments

In this section, you will just rerun some of the previous experiments, but using my alignments instead. Basically, rerun the code you used to generate the `ttable.model1` and `channel1.pfst` but using `alignments.good` as the alignments. Call the new files `ttable.good` and `channel2.pfst`. Then, rebuild the unigram and bigram language models as necessary and call them `lm1-good.pfsa` and `lm2-good.pfsa`. Then run the three experiments from Part IV and compare the translation outputs. **(Addition:** *Finally, do the same as before with epsilon English words.*) Hand in all the files you generate, the translations, and a description of how things seem to have changed.

Part VI – A Trigram Language Model

Repeat Part IV but with a trigram language model instead of a bigram language model. This part is not required – it is extra credit.