INGI2263 – Natural Language Processing

Assignment 2

HMM Part-of-Speech Tagging

1 Introduction

In this assignment you will be asked to estimate and to use a statistical Part-of-Speech tagger, in particular based on Hidden Markov Models, for tagging a large corpus. We focus here on an English corpus of press articles, fiction extracts, *etc* collected at the Brown University, USA. A local copy of this data is available on Icampus:

`Document > data > brown_corpus`

You are free to use it for this assignment but you are **not** supposed to redistribute it or to make any commercial use of it. This corpus was preprocessed and partitioned for you in two text files, respectively `brown_train.gz` and `brown_test.gz`. You **must start** from those two files, respectively for *training* and *testing* your POS tagger. Otherwise, your results are likely to be incomparable with our own evaluations.

In practice, you are asked to turn in a report with all the elements mentioned in a frame in this document.

You are free to use or to adapt existing softwares for this assignment, **provided** that you cite explicitly any software you use. In this case, you should describe the necessary adaptations you had to implement (if any) or any parameter tuning you had to go through. In other words, we would like to know the necessary steps you went through to reuse existing softwares. We recommend in particular to rely on your favorite scripting language to go through the first steps of this assignment (process the corpus, build a lexicon and estimate a tagset, *etc*) and to rely on some existing software[1] to apply Viterbi decoding for the actual tagging of the test sentences.

In any case, you are free to reuse software and/or to program yourself as long as you tell us what you did. You are not asked to turn in the program sources but a description of your implementation choice(s).

[1]For instance, the `HMM` or `hmm.discnp` R packages offer the necessary functions to estimate and to use HMMs. In case you find or you design a better alternative, please let us know.

2 Tags and words statistics

Get the training data (`brown_train.gz`) and uncompress it. Look at the data and notice that this corpus is already **standardized**. In particular, all words and tags are in upper case. The **tokenization** is also already performed and you are **not** supposed to change it.

Essentially, the corpus is made of a set of *segments* with one segment per line. Most segments correspond to actual *sentences*, ending with an explicit period mark. Some segments however do not end with a period mark because they correspond, for instance, to the header of an article. In the sequel, the tagging will be performed at the segment level, that is **line per line**. In other words, the tagging process is applied on each segment individually with no contextual information across consecutive segments. In the sequel, we will use the common term *sentence* both to refer to an actual sentence (ending with a period) or any other kind of segment. You are not supposed to edit the segments by adding a period mark when there is none or to modify the line breaking convention. This would also be irrelevant from a linguistic viewpoint in several cases.

Each sentence is a sequence of pairs of tokens `WORD/TAG`. The first element of the pair is the `WORD`, the second element is its corresponding `TAG` and the delimiter between them is the single character `/`. Note that such a simple rule needs to be considered with some care when dealing with real data. For instance, nothing prevents a word itself to contain the character `/` or others non-alphabetic characters like in: `1/4-INCH/NN` where the "word" `1/4-INCH` is tagged as `NN`. Note also that punctuation marks are included in this corpus with their respective tags. Do not remove them as they may convey some useful information for tagging new sentences. Some tags are also compound tags as they label specific tokens such as `IT'S/PPS+BEZ` where the token `IT'S` is labeled as the concatenation of a *3-person singular pronoun* `PPS` and `BEZ` the associated form of the verb *to be* at the present indicative. All the above peculiarities can be dealt with two general rules such as:

1. `WORD/TAG` pairs are separated by white space(s).
2. In a `WORD/TAG` pair, everything appearing before (resp. after) the last occurrence of `/` is the `WORD` (resp. the `TAG`).

You are invited to refine the above rules if necessary when processing the corpus.

The vocabulary (or lexicon) will be built from the training set **only**. To save some computing time, you should consider a restricted lexicon made of the 5,000 most frequent word tokens observed in the training set. An additional special token `<UNK>` should be reserved to match the out-of-vocabulary words, in the training and test sets. The set of tags to be considered are all tags appearing in the **training set** and nothing more.

- Build a lexicon from the 5,000 most frequent[2] words appearing in the training set (`brown_train.gz`). Report the 10 most frequent words in the training. Replace each occurrence of word tokens not included in your lexicon with the special token `<UNK>`, both in the training and the test set.
- Build the set of tags from the observed tags in the training. Report a table with the 10 most frequent tags and their respective number of occurrences. Report also all tags appearing the least frequently and their respective number of occurrences.
- Report how many `WORD/TAG` tokens (not types) and how many sentences (actually segments) you find respectively in the training and test sets.

3 Baseline tagger

The training set is used to estimate the parameters of a tagger. The test set is representative of a new tagging task, which are sequences of words that need to be tagged. The file `brown_test.gz` actually contains tagged sentences in order to allow you to compute the performance of the taggers you will design during this assignment.

Process the test corpus to extract the sequences of words **without** their respective tags. Store the `WORD/TAG` test pairs for further performance analysis.

The simplest (but already statistical) tagger assigns the most likely tag to each test word based on the frequency of `WORD/TAG` association for this word in the training. This tagger will serve as your baseline. What do you expect its accuracy will be? Hint: how many words are associated always to the same tag in the training set? Does this figure define the baseline test accuracy, an upper or lower bound to it, an approximation to it?

- What is the most likely tag for the word `THROUGH`? How many times was this word tagged in the training with this most likely tag? How many times was it tagged with a different tag and which one(s)?
- Report the *tagging test error rate* of your baseline tagger. Explain whether or not it matches your expectation and why. What is the average error rate per tag?
- What is the tagging accuracy for the test set words which should actually be tagged `JJS`? With which 2 others tags is `JJS` most often confused? What is the relative frequency of such errors?
  What is the tagging accuracy for the test set words which should actually be tagged `NP`? With which 2 others tags is `NP` most often confused? What is the relative frequency of such errors?
  All the questions in this last paragraph are related to a confusion matrix between tags estimated on the test set.

4 HMM Tagger

In this final part of this assignment, you will estimate and use a HMM tagger.

The description is short but the work to be done is likely to be somewhat more time consuming than the previous steps. Computing time may also be an issue according to your actual implementation. In any case, it is recommended to check your implementations first on toy examples (both limited training and test sets and small vocabulary of words and tags). It is also recommended to monitor the computation per individual test sentence to get a sense of the overall computing time.

We consider here a standard HMM tagger having the structure of a first-order Markov chain and no specific modeling of an end-of-sentence. In practice, this means that you don't need to consider a specific marker `</s>`. Note that we are not predicting the next word here but we are tagging observed sequences.

- Compute the necessary counts from the training set to build a HMM tagger. Build a discrete HMM model from them. Consider some appropriate smoothing of those counts. Tag the test sentences accordingly.
- Report the *tagging test error rate* of your HMM tagger. What is the average error rate per tag? Check that your HMM tagger outperforms the baseline. Otherwise, explain why and/or check your evaluations of your baseline and HMM models.
- What is the tagging accuracy, of your HMM tagger, for the test set words which should actually be tagged `JJS`? With which 2 others tags is `JJS` most often confused? What is the relative frequency of such errors?
  What is the tagging accuracy for the test set words which should actually be tagged `NP`? With which 2 others tags is `NP` most often confused? What is the relative frequency of such errors? Do you observe significant differences between the baseline and the HMM taggers with respect to those confusion statistics?

Turn in your assignment on Icampus as a report in PDF format in due time.