INGI2263 – Computational Linguistics

Assignment 2

Probabilistic Word Prediction

1 Introduction

Whenever you type in a message on a smartphone, you probably noticed that the phone is "smart enough" to guess the next letter or even the next word you want to write. A similar functionality is available in search engines that suggest you possible extensions to the question or to the list of keywords you are currently typing. In this assignment, you are asked to implement such a functionality. To do so, a common and effective approach relies on probabilistic language models, in particular N-grams.

We focus here on a specific corpus of English definitions from Wikipedia. This corpus was partitioned for you in two text files, respectively `definitions_train.txt` and `definitions_test.txt`. You **must start** from those two files, respectively for *training* and *testing* your language models. Otherwise, your results are likely to be incomparable with our own evaluations. This corpus is avalaible as a zip Archive `Definitions.zip` on Moodle for this assignment.

The final part of this assignment is a Shannon game which, in the present context, aims at producing automatically the next word in an English definition. Before getting there, this assignment will let you work through the intermediate steps.

In practice, you are asked to turn in a report with all the elements mentioned in a frame in this document. Note that the first few steps are relatively straightforward. The **last step(s)** however may require quite a bit **more time**, both to implement or to adapt the proposed methods, and to actually run them on the available corpus. It is recommended not to wait the last minute.

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.

The second option is to program some parts or everything yourself. This may be a very good choice as it may be significantly faster than learning and adapting existing softwares. It would also let you go through the bits and pieces of N-gram modeling. Efficient scripting languages (such as PERL, PYTHON, RUBY, *etc*) are particularly convenient for text processing.

As a demonstrative example, consider a very short PERL program to construct and to print a vocabulary (or lexicon) of the distinct word types found in a text corpus:

```
while (<>) {
    @tokens = split;
    $lexicon{$_}++ foreach (@tokens);
}
foreach $word (keys %lexicon) { print "$word\n"; }
```

As a good programmer, you surely noticed that the above program has *several implicit preconditions* such as the possible separator(s) used when tokenizing words in a sentence, or the fact that if specific tags or fancy characters are present in the input text, they are likely to appear in the constructed lexicon. In any case, it should be quite easy (as long as you roughly speak PERL or a similar scripting language) to adapt this illustrative example to fit your needs.

It should also be quite straightforward to extend the above script (or its equivalent in your favorite scripting language) in order to output the words sorted according to their frequency of occurrence in the text, or to build a lexicon of consecutive word pairs, or even more generally consecutive N-grams. If you follow these suggestions, many of the basic steps of this assignment will already be done.

In any case, you are free to reuse software 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).

2 Preprocessing

Get the training data (`definitions_train.txt`) and glance at it. This first analysis should let you realize that this corpus has some peculiarities. It contains relatively short definitions that could be used to design quizz questions. It includes many proper names, some peculiar characters, numbers or dates, punctuation marks, *etc*.

From a statistical language modeling point of view, this is not a critical issue (as a matter of fact, it should illustrate the flexibility of those models). However, this corpus is not huge relative to its lexical diversity (how many word tokens, word types, sentences do you count?) and quite diverse according to several aspects (which ones can you think of?).

Statistical models induce recurrent patterns from some data. For those models to be relevant, the patterns captured should be informative for the task at hand. This observation has some direct consequence on the corpus **tokenization** that you have to perform. The general question is: *how to define the word types on which N-gram models are going to be estimated?* Here is a non-exhaustive list of related questions that you should consider.

- How to deal with punctuation marks? Should we distinguish between variants of such punctuation marks? Should we ignore them?
- Do we need to distinguish between lowercase words, uppercase words or partially capitalized words?

- How to deal with numbers, dates, *etc*?
- *What else?*

Since you want the most relevant part out of this corpus, it is good to make design choices that tend to reduce the lexicon size by mapping several distinct observations to the same type. There is obviously a limit to this reasoning as mapping everything to a single type is always possible but would not result in informative prediction (see the Shannon game). To make things concrete, you are invited to define a lexicon from the training set only (after preprocessing) and by discarding word types occurring strictly less than 3 times in this set.

The test set is assumed to be representative of new data you could analyze or predict, once a model is built from the training only. How are you going to deal with the issue that a word type may appear in the test while it was never observed in the training (or was not kept among the lexicon entries considered)?

Preprocess the training and test data. Build a lexicon from the observed types present 3 times or more in the **training data**, after some adequate **tokenization** and/or **standardization**. Report your lexicon size and a table with the 20 most frequent types in this lexicon. Report the respective frequency of occurrence of those 20 types in the training set. Do these 20 most frequent types match your expectation about the content of this corpus? Are these representative of a general English text?

3 Word and N-grams counts

Consider here the training set after the preprocessing described above. In particular, any word type that is no longer part of the defined lexicon should have been replaced by the `<UNK>` tag, representing *unknown* words.
Extract the unigram, bigram and trigram counts from such training set. Report a plot of the histogram of those counts. Use a different color for each distinct model order. Both axes in your plot should be in log scale. Do you observe the Zipf's law for unigrams? Is it the same for bigrams or trigrams? Add any comment you would consider relevant from this analysis.

4 N-gram estimation

In this part of the assignment, you should estimate N-gram models from the training set, after your preprocessing and specific lexicon definition. Depending on the flexibility of your implementation, you will be able to consider, more or less easily, different model orders, i.e. different values for N. Your implementations (or the software you use) should at least deal with $N \leq 3$. Maximum likelihood (ML) estimation should be straightforward from the counts you already computed before. You should however *smooth* the ML estimates (do you remember why?). You are expected to use *Laplace smoothing* as a baseline (it is extremely simple but offers a poor smoothing). You are expected to use another more advanced smoothing technique (pick the one you prefer) and report comparative results. When implementing a smoothed N-gram model it is **very strongly recommended** to check that the model consistency equation **always** applies.

$$\sum_{w} P(w|h) = 1, \forall h$$

where the sum runs over all[1] word types w in the lexicon and h denotes an history or N-gram context $w_{i-1}w_{i-2}\ldots$ Since running this check on all possible histories may take some prohibitively large computing time (how many such histories would you have to consider as a function of N?), you are expected to run this check at least on a significant fraction of the observed histories in the training **and** test data[2]. This check should also be repeated when you change the model order.

If the above equation is not satisfied, wrong perplexity results will be reported in your evaluation below (Can you see why? How to artificially increase or decrease the perplexity if the above equation is not fulfilled?).

Estimate N-grams models from the training data with Laplace smoothing, *i.e.* the add 1 method. Report a table with the training (left column) and test (right column) perplexities obtained with your models. The rows in your table should correspond to different model orders. Report the perplexities at least for $1 \leq N \leq 3$, ideally $1 \leq N \leq 5$.
Repeat the same experiments while smoothing N-grams either by linear interpolation or using back-off smoothing. Report a new perplexity table as before.
In each case, always report as well the out-of-vocabulary (OOV) rate, that is the number of out-of-vocabulary events divided by the number of predictions used to compute the perplexities. Add any comment you would consider relevant about those comparative results.

[1] Some care need to be taken with the boundary markers <s> and </s>. Indeed <s> should never be predicted, hence the sum over w does not include <s> as one possible w. However, <s> may very well be part of a useful history h. In contrast, </s> is never included in some history since we do not predict after this marker but </s> is a possible word w to be predicted.

[2] An even more rigorous protocol would not use the test data at this point. It is however important to run this check on some data which was not used to estimate the model (can you figure out why?). One possible way to do so consists in partitioning the training set into two folds (typically 90%, 10%), estimate a model on the first fold and check it on both folds. Once the consistency checked is performed, a new model is built on the whole training data. A final check can be performed later while computing the real test set perplexity.

5 Let's play the Shannon game

After all the above steps, you are about to make a big impact on the smartphone market! Before dreaming about your future gross income, it might be good to estimate first whether your language models built from training data are actually predicting something valuable. Here is the beginning of a text fragment. Use your language models to predict the next possible word. For standardization reason, this message fragment is using uppercase letters only. If you made a different choice during the preprocessing steps, you can safely translate it to the form that best suits your choice.

`THIS IS A MUNICIPALITY IN THE ...`

Which word (if any) is most likely to replace the dots in the above text fragment? What is the second, third, fourth,... most likely alternative?

Use the language models estimated on the preprocessed training set with the best smoothing technique you have implemented and evaluated in section 4. Consider increasing model order, at least $1 \leq N \leq 3$ or ideally $1 \leq N \leq 5$.
For each model order, report a table with the 10 most likely predictions to complete the above text fragment, in **decreasing order of probability**. The first column should mention the word predicted and the second column its corresponding probability.
If not already present in your table, what would be the rank of the word `COUNTY` and what would be its probability? In other words, what is the minimal number of rows your table should have to include the word `COUNTY` and what would contain the corresponding row?
Are you satisfied with the prediction of your models? What would you consider as the optimal model order, based on this game and the results of section 4? Add any comments you estimate relevant to convince an investor of the strength of your NLP technology.

6 Practical information

For any question related to this assignment, you are invited to contact the teaching assistant: charlotte.hansart@uclouvain.be

In order to submit your work (in the form of a PDF report), you must be member of a group on the Moodle website for this course http://moodleucl.uclouvain.be/course/view.php?id=7865