Practical Session: Language Models.

David Abadia, University of Zaragoza
dabadia@unizar.es

Introduction

Modelling the human language is a scientific challenge with multiple potential applications; e.g. speech recognition, machine translation, OCR, handwriting recognition, automatic text generation and text completion. The set of grammar rules in each language gives us conditions on the *legal* sentences allowed, but only a subset of meaningful ones are naturally used by the speakers. Language models are probabilistic in essence, as different words can follow a specific one with different probabilities given the context. Graphical models are an excellent framework for this problem, and specifically Markov chains given their temporal nature.

Objectives

1. Understand the n-gram model for natural language.
2. Build simple bigram and trigram models in python and use them for automatic text generation.

1 Background

1.1 Problem Definition

The probability of a sentence $w_1^n = w_1 \dots w_k \dots w_n$ composed of n words w_k is given by the chain rule

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)\dots P(w_n|w_1^{n-1}) = \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \tag{1}$$

N-gram models assume that the probability of a word only depends on the $N-1$ previous words. If we assume that the probability of a word only depends on the previous one (the Markov property), the language model is called 2-gram or bigram model

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k|w_{k-1}) \tag{2}$$

And in general, the N-gram approximation is as follows

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k|w_{k-N+1}^{k-1}) \tag{3}$$

1.2 Training the model

The N-gram conditional probabilities can be extracted from raw text based on the relative frequency of word pairs

$$P(w_k|w_{k-1}) = \frac{count(w_{k-1}w_k)}{count(w_{k-1})} \tag{4}$$

Words that do not appear in the training data including an explicit symbol for unknown word <UNK>. As the probability depends on the length of a sentence, there is a measure called *perplexity* ($PP(w_1^n)$) of how well a model fits the data

$$PP(w_1^n) = \sqrt[n]{\frac{1}{P(w_1 w_2 \dots w_n)}} \tag{5}$$

In order to model word combinations that do not appear in the training data, the probabilities should be smoothed to assign a non-zero probability to non-present combinations.

2 Practical session

The goal of the practical session is to build a very simple bigram as well as a trigram language model from the raw text in a book. Such model will be used to generate text automatically. The practical will comprise of understanding how the trigram works and afterwards to enrich proposed code and to propose also the bigram aproach. Part of the material for this practical session was taken from:

- `https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-language-model-nlp-python-code/`
- `https://nlpforhackers.io/language-models/`

More information:

- `https://www.nltk.org/api/nltk.html`
- `https://web.stanford.edu/~jurafsky/slp3/` (–> Language Modeling with N-Grams)

2.1 Load the text

The first step is to load the raw text. For that purpouse we will use the Project Gutenberg [1], one of the oldest projects with the aim of digitize cultural works, and will download a book in raw text format. As an example, Quijote by Cervantes has an appropriate size for the implementation we are going to build. For the practical session you should use at least two different texts, as different as possible (for example of different centuries), and compare the automatic text generated by both.

[1] `https://www.gutenberg.org/`

2.2 Trigram model

A jupyter notebook is provided with an example of a trigram model, you r goal is to understand how it works to explain it in the inform, as well as to improve what you consider. As an example a text is provide to make easier the understaning of the trigram model, please feel free to update the text to really understand how it works and then consider text from at least two books. (Note: The text of the books should be prepared to get the pure text, e.g. by deleting the initial heads of the Gutenberg project)

2.3 Bigram model

Once you have understood how the trigram model works, you should implement the bigram model.

2.4 Generate random text

The jupyter notebook provides a first version of automatic text generation by means of the trigram model. It should be extended also to the bigram model.

3 Report

As a result of this practical session you should provide a report which should contains:

- draw of the graphical model for a bigram and trigram models.
- explanation of the trigram model by answering the questions of the jupyter notebook
- explanation of the considerations followed to implement the bigram model
- results comparing the different generated sentenecs between at least two books of different styles. this analysis should be carried out of the trigram as well as for the bigram model
- analysis of the comparion of the obtained results with trigram and bigram model, which model seems to have better performance? why?

The focus of the report should be the theoretical aspects of the practical. The results should be presented as formally and graphically as possible.

On the other hand a jupyter notebook should also be sent, which should implement the bigram model as well as the required functionalities previously described for the trigram model. Comments explaining the implemented code should be provided within the jupyter notebook.