Johns Hopkins Data Science Capstone

Predict Next Word

Anderson Roberto Santos dos Santos

Predict the next word

Author: Anderson Roberto Santos dos Santos

In this project, we developed a shiny application that predicts the next word. To to that, we created a model based on the frequency of the words alone (unigrams) and the frequency of the combination of 2 and 3 words, respectively: bigrams and trigrams.

Steps done to create the model:

1. Download a corpus of text's from: twitter feeds; news and blogs. This corpus is called HC Corpora (www.corpora.heliohost.org).
2. Sample a fraction of the corpus from each source: twitter, news and blogs.
3. Pre-process and clean the corpus.
4. Create the n-gram models
5. Modify the n-gram models in order to reduce the memory usage and processing time to make the predictions

Pré-processing steps:

We removed parts of the text that does not add any useful information for word prediction purposes. We removed:

- Ponctuations
- Numbers
- Profanity words (as we do not want to predict these ones)
- Transform all word to lower-case
- Remove extra white spaces

Why we don't:

- Removed url's: We have created a very good regex to remove them. But it has an unacceptable performance.
  - We addressed this problem with a prunning method that will be showed in later slides.

The prediction model

- Using the pre-processed corpus. We created three n-grams models: unigrams, bigrams and trigrams. To make predictions: We used the bigrams and trigrams with the Kneser-Ney smoothing algorithm (Kneser and Ney, 1995).
  - This prediction smoothing method has a very poor performance because we need to calc all the distinct contexts (previews words) a word could have. To address this, we make this calc previously. Just after creating the ngrams models.
  - Another problem of the Kneser-Ney is that it consider the unigram model as a uniform distribution. So all words in the unigram model will have the same probability of occurring. This is not useful for our purposes. So, we use the trigrams and bigrams with the kneser-ney smoothing formula. If we don't find the previews contexts in this two models, we show the 5 most common words to the user in the unigram model.
- Prunning: As showed by Chen and Goodman (1998), we can remove the words or combinations of words (ngrams) that occur once without losing accuracy of the model in terms of perplexity. So we have cutted all bigrams and trigrams that occurs only once in the sample corpus. Resulting in more than 70% of memory save.

The app, final words and future works

- The app: Just input some text and the app will show the five most probable words.
- The git-hub repository with the app code; the modelling code and this presentation code.

Future works

Unfortunatelly, R is not the best place to create and use such language models. We made it from scratch for didatic purposes. We recomend to use one of the following best toolkits: berkeleylm; SRILM or KenLM. They have several performance improvements, such as: using numbers to represent words instead of strings, with more frequent words represented by smaller numbers; Quatize n-gram probabilities and save in 4-8 bits instead of saving in a double format. Store n-grams in reverse tries; ....

Bibliography used in this presentation

Kneser, R. and Ney, H. (1995). Improved backing-off for M-gram language modeling.

Chen, S. F. and Goodman, J. (1998). An empirical study of smoothing techniques for language modeling.