Data Science Capstone Project - Shiny App
NLP, Text Mining, R, Data Science

author: Rajiv Sharma

Date: September 10th, 2017

Additional Info Documentation - Shiny App - Predicting The Next Word

Main Features

This is a Shiny app that accepts an user input of a single word or a combination of words and predicts (suggests) the next most probable word(s).

The project was developed using Natural Language Processing (NLP) techniques and Text Mining (TM) using the R programming language. The data for this application comes from a Corpus from https://web-beta.archive.org/web/20160930083655/ - Helios Data

A variety of well-known R packages have been used to practice text mining and natural language processing, such as:

1) TM at "https://cran.r-project.org/web/packages/tm/vignettes/tm.pdf" Text Mining in R

2) quanteda: href="https://cran.r-project.org/web/packages/quanteda/quanteda.pdf" Quantitative Analysis of Textual Data in R

Product Development

Tasks we needed to perform: sampling, data cleaning, exploratory analysis, predictive modelling and some creative exploration.

The Helios Data was, initially, reduced by 15% using a systematic sampling process. This sample was cleaned by conversion to lowercase, removing punctuation, links, extra white spaces, numbers and all kinds of special characters. The sample was reduced by removing stopwords. It results in a more handy and meaningful dataset.

The next step was assembling tables of the most frequent n-grams (1-gram, 2-gram and 3-gram). Each table was sorted according to the frequency of n-grams (decreasing order).

(What is 'n-gram'? "https://en.wikipedia.org/wiki/N-gram" - From Wikipedia: In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sequence of text or speech.

In addition to using test methods in 'quanteda' and 'tm' packages, we preferred developing our own functions to execute this task. It is programmed in the following source files:

"download_import_sample.r"
"n_grams.R"
"KB_nlp.r"
"letsdoit.R"

Those aggregated uni-, bi- and tri-term frequency matrices have been passed to data frames and saved in 3 .rds format files.

The n-gram files were modified to facilitate analysis based on their raw probabilities and also their stupid-backoff counts and probabilities.

The resulting data frames are used to predict next words in consonance with the text input by the user of this Shiny App, and also taking account the backoff probability of the n-grams occurrences.

The Algorithm

The main strategy to predict the next word(s) is based on "https://en.wikipedia.org/wiki/Katz's_back-off_model" -

Katz Backoff and Stupid Backoff Algorithms

Despite the Katz Backoff method being more attractive from a computational standpoint, we implemented a more simplified approach known as "Stupid Backoff"- it always takes the prediction from the highest

order n-gram first and works backwards to a lower gram token to use for prediction:

It uses a 3-grams data frame; the first two words of which are the last two words of the sentence input by the user and that we are using to predict the next possible set of single-word occurrences.

The 3-gram data frame is already sorted from highest to lowest frequency.

The entered words are matched with the all the 3-grams and the ones matching all the first 2 words are shown on the basis of their frequencies.

If no 3-gram is found, we back off to the 2-grams df (first one word of bigram = last one word of the sentence you typed in).

Recursively, If no 2-gram tokens are found, we back off to a 1-gram (single word of unigram last word of the sentence).

Other issues and limitations

We also added bos (beginning of the sentence) and eos (end of the sentence) to refine the model more to take into account occurrences of words at the beginning and the end of a sentence. If the user inputs more than three words, only the last two are considered as an initial parameter of predicting the model function.

The Algorithm was designed to convert the text by removing stopwords as it yielded more meaningful and accurate words. But if the user only enters two or three sets of words that are all stopwords then the Algorithm switches to using a pre-processed data set that does not remove stopwords.

Sampling the Capstone Dataset we could decrease the size of 583 MBytes (.txt files) to 7.6 MB - four .rds data files (almost 99% of reduction). It allowed that the Shiny application to load enough of the database to process an algorithm that produces a fair prediction, with good performance (times less than 1 second).
Possible Improvements

Implement, test and compare with other modelling ways to get machine prediction, like interpolated Kneser-Ney Smoothing, pure Katz Back-Off or Good-Turing discounting. (Good material to get deep into this is a lecture from Cornell, cited in References).

References

“https://class.coursera.org/dsscapstone-006" - Coursera.org - Johns Hopkins University - Data Science Capstone

“https://class.coursera.org/nlp/lecture" - Coursera.org - Stanford - Natural Language Processing.

“ h t t p : / / w w w . m o d s i m w o r l d . o r g / p a p e r s / 2 0 1 5 / Natural_Language_Processing.pdf">Natural Language Processing: A Model to Predict a Sequence of Words.

Text Mining Infrastructure in R (pdf file)

“https://en.wikipedia.org/wiki/N-gram">Wikipedia

"http://english.boisestate.edu/johnfry/files/2013/04/

Bigrams and Trigrams - Boise State University
"https://en.wikipedia.org/wiki/Katz's_back-off_model">Wikipedia - Katz's back-off model.

“http://www.aclweb.org/anthology/D07-1090.pdf” - Large Language Models in Machine Translation (Stupid Backoff).

“http://www.cs.cornell.edu/courses/cs4740/2014sp/lectures/smoothing+backoff.pdf" Smoothing, Interpolation and Backoff - Cornell University.

“https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip" - Capstone Dataset SwiftKey.