Ilias Siafakas

ITC6010A1

Natural Language Processing

Dr. Lazaros Polymenakos

6/10/2022

# Requirements:

Python 3.9.5 or later

NLTK ‘Reuters’, ‘Punkt’, ‘Words’ download

# Run instructions:

Execute HW2.py and run setup shown in the terminal. In case of small generated sentences or infinite loading, please restart the program. Maximum expected model building time: 50s

# Goal

The goal of this project is to create a complete language model and then use it to create a functioning spellchecker.

The program developed for HW2 uses the Reuters corpus that has undergone some preprocessing as a base to create its own language models. Two models are created from the corpus, one using bigrams, and a second using trigrams. The program also uses a generic dictionary downloaded from the NLTK library that contains a list of all possible valid words in the English dictionary. Using the language models new sentences are created that contain spelling mistakes, and the user is tasked to identify the correct word out of a list of suggestions, and “fix” the mistakes in the sentence. The idea behind this project is to replicate the functionality behind Microsoft Word’s spellchecker, or Googles “Did you mean X” functionality, based on previous search results or sentences and type patterns.

# Setup

The program requires some user input during startup to define some variables. The variables are:

* Number of sentences to generate: the user should select the number of sentences to be created. Default 1, max 5. In case the system creates small sentences, a large number here will provide more examples for the spellchecker to work, but dramatically increases the total runtime of the program (more mistakes to fix)
* Start new sentence parameter. Input is True/False. In case more than one sentences are asked to be created, an extra option is setup to create all sentences based on the same two starting words, or use them only on the first sentence, and let the randomizer decide which words to use after.
* Error weight. Default ‘2’. Specifies the error frequency for each sentence. Takes values 1-5, 1 being less errors and 5 being more errors. Value is multiplied by 10, and then a randomize function decides the number of errors in the sentence. E.g. picking ‘2’ will create a maximum of 20 errors in the sentence, spread across all words in that sentence.
* Enable Auto mode. Wil enable the automatic spellchecker which will always take the word with the smallest edit distance out of the suggestion list as the ‘correct’ word. Note: this may cause errors and throw the program in infinite loop in some cases.

# Process

After the setup is complete, the program first creates the two models out of the Reuters corpus, and provides a probabilistic model using the probability of co-occurrence for each pair of words in the corpus. This is stored in a dictionary, using the tuple of the first two words as a key, and a new dictionary object as the value, that contains the third word as key, and the probability of co-occurrence as value. The corpus goes through preprocessing, and all words are turned to lowercase. Stop words and symbols are not removed, as they are a vital part of natural sentence forming, and are required by the other functions to form correct sentences.

Then the program tries to create new sentences based on the model, using a fixed first two word starting point, and then randomly choosing from the model what the next word will be. This iterative process ends when the program finds a sentence stopping word or symbol (e.g. full stop) that finishes the sentence. Depending on the user input at the start of the program, one or more sentences are created, and for each one, a function chooses randomly a word in the sentence, and then a random position on the word. It then tries to simulate the addition, deletion or replacement of characters in the word, to provide artificial mistakes or ‘typos’ in the sentence. Words that contain symbols or numbers are skipped from this process.

The sentence is displayed to the user, and a function tries to find the artificial mistakes, and prompt the user to correct them. The function works by splitting the newly generated sentence into trigrams and bigrams again, and cross-checking the probability of co-occurrence with the existing model. If the word is not found, or if the word has very low probability of co-occurring, the function flags this word as ‘incorrect’. By using the edit-distance of that word with the trigrams existing in the model the ‘cost’ for each word is generated.

Example: the newly generated sentence contains the trigram of [1,2][x]. If x is not found in the model or has a very low probability of occurring, then the edit distance with trigrams starting with [1,2][y] for every y in the model is calculated. The top 5 words with smallest edit distance are added to the suggestion list and provided to the user. If the model does not manage to provide enough suggestions to fill the suggestion list, a second model is used, that contains a simple dictionary with all words in the English dictionary. Words that exist in both suggestion lists are merged and only displayed once.

The user then has to select the correct word from the suggestions list, and the program will continue until no more mistakes are identified.

# Future work

A current issue with the program is that during the spellcheck part, if the user selects by mistake a word suggested by the dictionary that does not exist in the active model, it will be flagged again for check on the next pass of the function. This is in part correct, because it flags the word as incorrect based on the probability of co-occurrence. In the real world, a spellchecker might have an option like ‘add to dictionary’ that skips this word and does not flag it as wrong in the future. This can be solved if after every sentence or correction, a new record is added on the model that adds this new trigram to the model and recalculates the probabilities.

A second item that could be worked on is to allow the user to input their own text and not use the automatic generator. This would have very poor results using this model since the Reuters corpus is relatively small and only uses news headlines to build the model. Anything outside similar headlines would throw the model off and it will not make good suggestions.

Finally there are some rules and limitations in place to restrict the error creation function. For example the program needs a starting point to create the trigrams for the spellchecker, so no errors should exist on the first two words. In this version of the program, the first two words are fixed to “today the” since this is the most frequently used bigram across the corpus, to increase the diversity of the sentences created.