Spring 2025

EDS 6397 – Natural Language Processing

Assignment #2 – Measuring document similarity using embeddings

Use of artificial intelligence assistant such as ChatGPT in developing the code for this assignment is allowed but discouraged. However, the report needs to be completely written by you. Use of grammar correction tools is disallowed as well.

Assignment Overview

The objective of this assignment is to familiarize you with word vectorization (embedding) techniques TF-IDF and Word2Vec which are sparse and dense vector representation for tokens respectively. You will create word vectors and use them for document similarity assessment as part of a movie recommender system. You will learn how to form TF-IDF sparse vectors. You will also learn how to train dense Word2Vec embeddings and use them for the same document similarity task. You will also learn how to use pretrained Word2Vec dense embeddings.

Assignment Description

Make sure you follow the instructions below very carefully. Four tasks are given, and two Jupyter notebooks are required along with one PDF report.

Input Data

One csv file is provided named 'assignment2_data.csv'. This data is downloaded from Kaggle and holds some metadata about 4800+ movies. The columns of this dataset ae "Title", "Popularity", "Tagline", and "Overview".

Task 1 - Preprocessing

Your objective is to write a code in Python (in a Jupyter notebook) to do the following:

- Read the input document into a Pandas dataframe.
- Clean up the input data. This includes removing records (rows) that have missing data under "Tagline" **and** "Overview". If even one of those columns has a value, do not remove that record.
- Create a second dataframe named "data" using "Title" and "Popularity" columns of the original data. The third column in this dataframe should be formed by merging contents of "Tagline" and "Overview" columns with a space character in between. Name this column "Full_Overview".
- Perform regular expression to remove punctuation and symbols and special characters from the title and full overview columns of "data".

- Tokenize your input documents using Spacy's small English words (en_core_web_sm)
- Remove stop words from the full overview column (use Spacy's stop words list).
- You may choose to perform lemmatization at this point. The choice is yours, however whether you decide to use it, you need to write you justification in the report. You may try the rest of the assignment with and without lemmatization and then write you justification as to whether or not you decided to use / not use it.

Task 2 – Similarity using Sparse Vectors

Here the objective is to create a TF-IDF representation from "full overview" text of all movies. Each row of the dataframe under full overview is treated as a "**document**".

- Perform TF-IDF sparse vectorization to create vector embeddings for all words in the **documents**. (Use TfidfVectorizer function from scikit-learn)
- Next create a function to calculate document similarities between a given query document and all the nearly 4800 documents in "data" (meaning the full overview column). This function should take one input, which is the index of the query document. Then this function calculates the cosine distance between the TF-IDF representation of the document identified by the input index with all the other documents and returns the top 5 with highest scores (Use cosine_similarity function from scikit-learn).
- Finally, use the function created at the previous step to find most similar movies to query movies below. Sort the top 5 similar documents using popularity scores (from highest to lowest) and recommend 5 movies given an input movie.
  Query Movies: "Taken", "Pulp Fiction", "Mad Max", "Rain Man", "Bruce Almighty"

Questions to answer for task 2

1- What is the vector size of TF-IDF vectors?
2- What is the vocabulary size of TF-IDF vectors?
3- For each query and top 5 recommended movies, read the full overviews and state whether or not you agree with TF-IDF-based recommender system. Note that you do not need to read all the 4800+ overviews to come up with your answer, just judge whether the top 5 picks for each query movie are fair.

Task 3 – Similarity Using Dense Vectors

3.1 training dense vectors

- Using the Gensim package, train a dense Word2Vec vector representation for your input documents. Set the vector size by trial and error to a number between 150 and 300 (not a thorough search, just test 3-4 values and keep the best). Make sure you give a window size between 7 and 13 for training. Justify your choice. Set the minimum number of appearances for a token to 1, use skip-gram training method, and set maximum iterations to a number between 10 and 20. Justify your choice.

- Next create a function to calculate document similarities between a given query document and all the nearly 4800 documents in "data" (meaning the full overview column). This function should take one input, which is the index of the query document. Then this function calculates the cosine distance between the dense vector representation of the document identified by the input index with all the other documents and returns the top 5 with highest scores (Use cosine_similarity function from scikit-learn). The only difference with TF-IDF is that here you have a vector representation for each token, not document. You need to aggregate token representations. Use centroid for aggregation.
- Next, use the function created at the previous step to find most similar movies to query movies below. Sort the top 5 similar documents using popularity scores (from highest to lowest) and recommend 5 movies given an input movie.
  Query Movies: "Taken", "Pulp Fiction", "Mad Max", "Rain Man", "Bruce Almighty"

3.2 using pretrained dense vectors

- Use gensim.downloader to download 'word2vec-google-news-300' which is a **pre-trained word2vec model**. Note that the first call to this downloader takes a long time to run as it is downloading a large model. All subsequent calls to the function however take less time as they load the already downloaded model, not downloading it from the internet again.
- Finally, using the similarity function and the pretrained embeddings, sort the top 5 similar documents using popularity scores (from highest to lowest) and recommend 5 movies given an input movie.
  Query Movies: "Taken", "Pulp Fiction", "Mad Max", "Rain Man", "Bruce Almighty"

Questions to answer for task 3

1- For each query and top 5 recommended movies, read the full overviews and state whether or not you agree with Word2Vec-based recommender system. Note that you do not need to read all the 4800+ overviews to come up with your answer, just judge whether the top 5 picks for each query movie are fair.
2- Comment on training Word2Vec vectors versus using pretrained ones. Which one worked better? Do you recommend training dense vectors for a task like this or is it better to use pretrained vectors?
3- Based on the results so far, rank sparse vectorization, task-specific training of dense vectors, and using pretrained dense word vectors for the document similarity task given in this assignment. Comment on this ranking. Is the result what you expected?

Analysis Required for Task 4

- Use a second Jupyter Notebook for this task.
- Use the same pretrained Word2Vec model as last task: 'word2vec-google-news-300'.
- Create a function (that calls a collection of other functions you develop) to take a word analogy input, calculate top 5 most similar outputs, then plot the original 3 words, plus the top 5 most similar words, plus 20 randomly sampled words from Spacy vocabulary (see Note 2 below for the reason) using t-SNE plot. Make sure you draw the analogy lines for the 4 words of the analogy; *3 of these words are given to you below, the fourth is the top most similar word.* Make sure to only draw two lines for each analogy. Do not plot 3 or 4 lines to represent an analogy.

Here are the analogies you should run. Include each plot in your Jupyter notebook:

king - man + woman ≈ ?

doctor - man + woman ≈ ?

france - spain + madrid ≈ ?

florida - texas + austin ≈ ?

Note 1: out of the 28 words to plot, there might be some duplicates. Write a function that can remove duplicates before plotting. Your plot would then have less than 28 words, but that is fine.

Note 2: the vocab of the Word2Vec model that we loaded earlier contains 3 million tokens most of which are complex hyphenated words. When you randomly sample 20 words out of it, there will be issues when creating t-SNE plots. To avoid this, load the 'en_core_web_sm' language package of spacy and randomly sample 20 words out of it.

Questions to answer for task 4

1. Do all analogies return the correct answer as the top most similar word? (make sure you Google search here if you are not sure)
2. You are instructed to randomly sample 20 words from spacy vocab to add to the plot. Can you guess why?
3. t-SNE does not preserve distances perfectly. Does re-running the plot change the alignment?

What to submit:

1- Your Python Jupyter Notebooks should include plenty of comments and markup cells to ensure that your code is easy to read and understand. The Notebooks should also contain the results of each cell executed by you. The two notebooks should be named as follows: **[Last_name]_HW2_tasks1_2_3.ipynb** and **[Last_name]_HW2_task4.ipynb**

2- A report (No longer than 3 pages) **in PDF format** that briefly discusses the assignment and the results of all tasks. Make sure your answer to questions of both tasks are included in the report. The name of the file should be **[Last_name]_HW2.pdf**

Late submissions will be penalized at a rate of 1 point per hour. We round up the time to the nearest hour.

Make sure you click on submit so your assignment is "Turned In". Uploading the files doesn't mean you turned the assignment in. If your assignment status is not "Turned In" on our end, you might face a penalty.

Due Date: February 27, 2024 by 11:59 PM (submit through Teams)