Word2Vec and FastText Word Embedding with Gensim in Python

Business Objective

The biggest challenge in the NLP (Natural Language Processing) domain is to extract the context from text data, and word embeddings are the solution that represents words as semantically meaningful dense vectors. They overcome many of the problems that other techniques like one-hot encodings and TFIDF have.

Embeddings boost generalization and performance for downstream NLP applications even with fewer data. So, word embedding is the feature learning technique where words or phrases from the vocabulary are mapped to vectors of real numbers capturing the contextual hierarchy.

General word embeddings might not perform well enough on all the domains. Hence, we need to build domain-specific embeddings to get better outcomes. In this project, we will create medical word embeddings using Word2vec and FastText in python.

Word2vec is a combination of models used to represent distributed representations of words in a corpus. Word2Vec (W2V) is an algorithm that accepts text corpus as an input and outputs a vector representation for each word. FastText is a library created by the Facebook Research Team for efficient learning of word representations and sentence classification.

This project aims to use the trained models (Word2Vec and FastText) to build a search engine and Streamlit UI.

Data Description

We are considering a clinical trials dataset for our project based on Covid-19. The link for this dataset is as follows:
Link:https://dimensions.figshare.com/articles/dataset/Dimensions_COVID-19_publications_datasets_and_clinical_trials/11961063

There are 10666 rows and 21 columns present in the dataset. The following two columns are essential for us,

- Title
- Abstract

Aim

The project aims to train the Skip-gram and FastText models for performing word embeddings and then building a search engine along with a Streamlit UI.

Tech stack

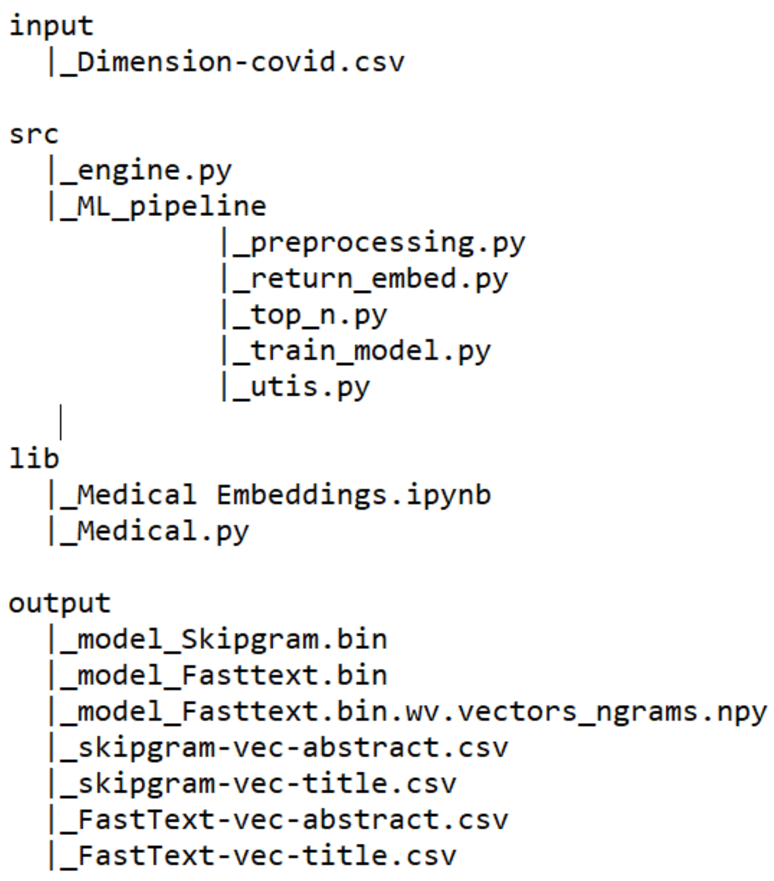
- Language - Python
- Libraries and Packages - pandas, numpy, matplotlib, plotly, gensim, streamlit, nltk.

Environment – Jupyter Notebook

Approach

1. Importing the required libraries
2. Reading the dataset
3. Pre-processing
    - Remove URLs
    - Convert text to lower case
    - Remove numerical values
    - Remove punctuation.
    - Perform tokenization
    - Remove stop words
    - Perform lemmatization
    - Remove '\n' character from the columns
4. Exploratory Data Analysis (EDA)
    - Data Visualization using word cloud
5. Training the 'Skip-gram' model
6. Training the 'FastText' model
7. Model embeddings – Similarity
8. PCA plots for Skip-gram and FastText models
9. Convert abstract and title to vectors using the Skip-gram and FastText model
10. Use the Cosine similarity function
11. Perform input query pre-processing
12. Define a function to return top 'n' similar results
13. Result evaluation
14. Run the Streamlit Application

Modular code overview



```
input
  |_Dimension-covid.csv

src
  |_engine.py
  |_ML_pipeline
            |_preprocessing.py
            |_return_embed.py
            |_top_n.py
            |_train_model.py
            |_utis.py
   |
lib
  |_Medical Embeddings.ipynb
  |_Medical.py

output
  |_model_Skipgram.bin
  |_model_Fasttext.bin
  |_model_Fasttext.bin.wv.vectors_ngrams.npy
  |_skipgram-vec-abstract.csv
  |_skipgram-vec-title.csv
  |_FastText-vec-abstract.csv
  |_FastText-vec-title.csv
```

Once you unzip the modular_code.zip file you can find the following folders within it.

1. input

2. src

3. output

4. lib

1. Input folder **-** It contains the data that we use for our analysis. A clinical trials dataset is considered for our project, which is based on Covid-19.
   - Dimension-covid.csv.
     There are 10666 rows and 21 columns present in the dataset.
2. Src folder - This is the most important folder of the project. This folder contains all the modularized code for all the above steps in a modularized manner. This folder consists of:
   - engine.py

   - ML_pipeline
     The ML_pipeline is a folder that contains all the functions put into different python files, which are appropriately named. These python functions are then called inside the engine.py file.

3. Output folder **–** The output folder contains the best fitted model that we trained for this data. This model can be easily loaded and used for future use and the user need not have to train all the models from the beginning.

   Note: This model is built over a chunk of data. One can obtain the model for the entire data by running engine.py by taking the quality data to train the models.

4. Lib folder **-** This is a reference folder. It contains,
   - The original ipython notebook that we saw in the videos.
   - The Medical.py notebook which will be used for running our Streamlit UI
   - The ppt used in the videos is also present in this folder.

Project Takeaways

1. Understanding the business problem
2. Understanding the architecture to build the Streamlit application
3. Learning the Word2Vec and FastText model
4. Importing the dataset and required libraries
5. Data Pre-processing
6. Performing basic Exploratory Data Analysis (EDA)
7. Training the Skip-gram model with varying parameters
8. Training the FastText model with varying parameters
9. Understanding and performing the model embeddings
10. Plotting the PCA plots
11. Getting vectors for each attribute
12. Performing the Cosine similarity function
13. Pre-processing the input query
14. Evaluating the results
15. Creating a function to return top ‘n’ similar results for a given query
16. Understanding the code for executing the Streamlit application.
17. Run the Streamlit application.