Document/sentence similarity solution using open source NLP libraries, frameworks and datasets

Pydata 2022

By Ade Idowu
Credit Suisse

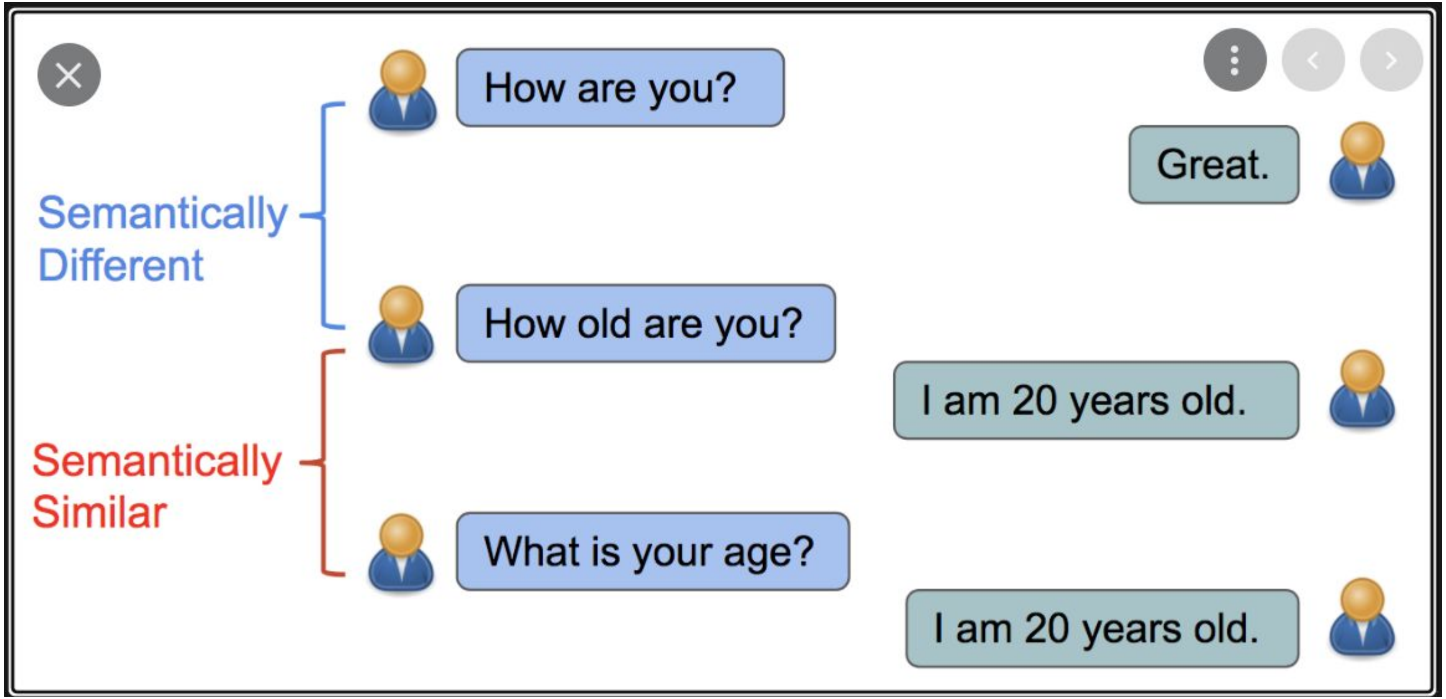
Agenda

- Introduction
- What is STS - Semantic Textual Similarity
- Textural similarity measures
- Sentence pre-processing using NLP
- Feature engineering using Sentence Embedding models
- Visualization/Validation of findings
- Demo Problems & Datasets used in this tutorial
- Q & A

Introduction

- Growing need to develop robust semantic document/text similarity solutions:
  - Recommendation Systems
  - Search Engines
  - News Aggregator Systems
  - Automated Recruitment Systems
  - Biomedical informatics eg. used to compare genes and proteins based on the similarity of their functions
  - And so on...

Introduction (continued..)
- It is all about Semantics!!



What is STS - Semantic Textual Similarity

- STS is a solution that deals with determining how similar two pieces of texts (sentences/phrases/paragraphs) are to each other
- According to wikipedia the distance measured in STS is based on the likeness of their meaning or **semantic content** as opposed to **lexicographical similarity**
- There a number of active ML/NLP work in the area of STS by developers such as Google, Microsoft, HuggingFace, Ubiquitous Knowledge Processing Lab
- There is the popular SentEval toolkit which is actively used by NLP developers to solve common semantic textual similarity tasks i.e. part of the STS benchmark problem suite.

Demo of Solved Problems

- **Simple demo the similarity between book titles**
  - Problem: Use the 5 embedding approaches to encode the titles and compute the top k similarities between the titles
  - Data: Goodreads data sourced [here]
- **Demo a simple Search Engine**:
  - Problem: User provides a query and it is compared (searched) in a corpus of documents to get the top k matches
  - Data: The classic 20 News Group data sourced from [Scikit-Learn dataset module]
- **Demo the performance of the 5 embedding strategies using labelled sentence pair corpus data**:
  - Problem:
    - Embed the sentence pairs from the sentence pair corpus using 5 approaches
    - Compute the similarity between each sentence pair
    - Compute the performance of each approach using Paerson’s correlation coefficient i.e. computing the predicted similarity versus the actual similarity (label)
  - Data: STS Benchmark Sentence Pair data sourced from [here]

Textural similarity measures

- There are a number of textural similarity metrics in ML.
- In this workshop I will explore 3 popular metrics, namely:
  - Jaccard
  - Euclidean
  - Cosine

Textural similarity measures
- Jaccard

- The Jaccard similarity index **compares members for two sets to see which members are shared and which are distinct**.
- It's a measure of similarity for the two sets of data, with a range from 0.0 to 1.0.

Formula

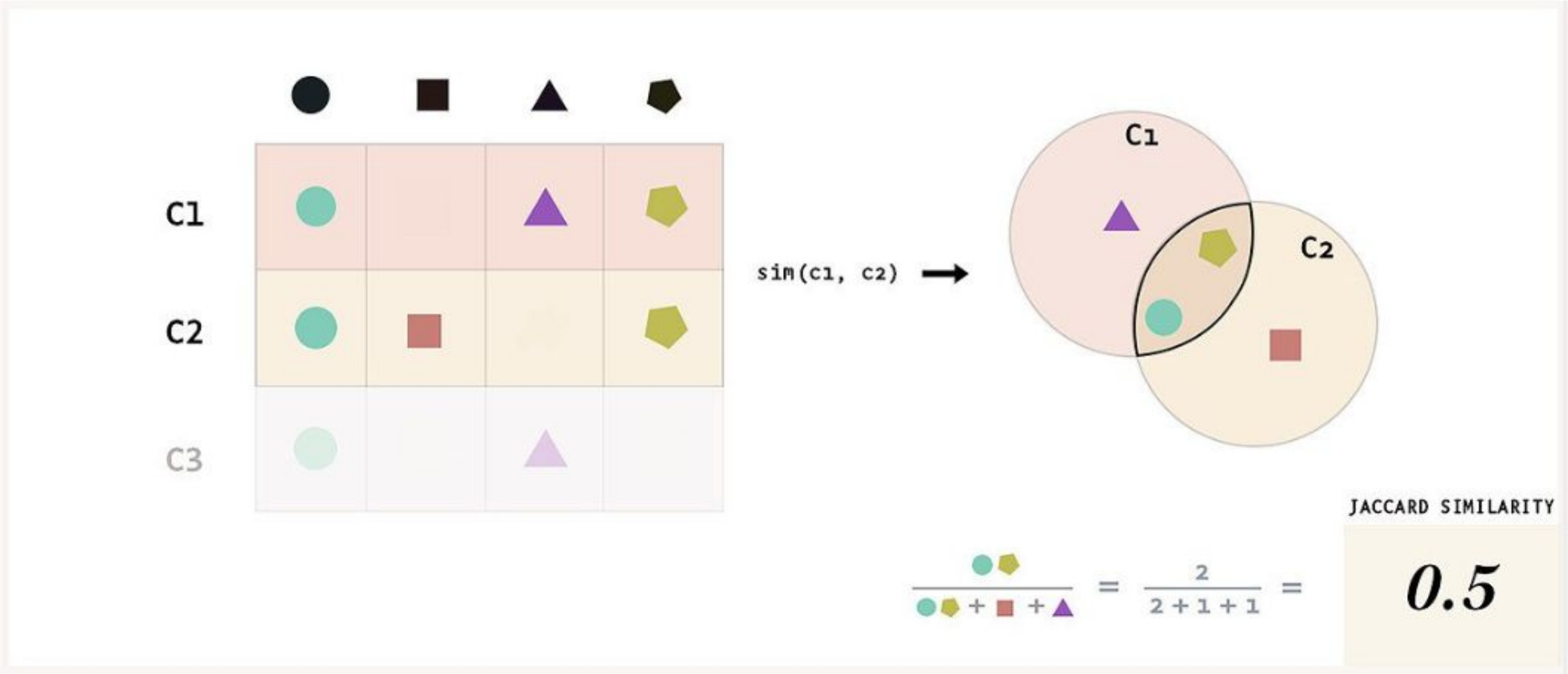
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

J = Jaccard distance

A = set 1

B = set 2

Textural similarity measures
- Jaccard



Textural similarity measures - Euclidean

- **Consider two vectors**
  - Rows in the data matrix $X = \langle x_1, x_2, \cdots, x_n \rangle$ $Y = \langle y_1, y_2, \cdots, y_n \rangle$
- **Common Distance Measures:**
  - Manhattan distance:

$$dist(X,Y) = |x_1 - y_1| + |x_2 - y_2| + \cdots + |x_n - y_n|$$

  - Euclidean distance:

$$dist(X,Y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}$$

  - Distance can be defined as a dual of a similarity measure

$$sim(X,Y) = 1 - distance \qquad sim(X,Y) = \frac{1}{dist(X,Y) + \lambda}$$

Textural similarity measures
- Cosine

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

where A_i and B_i are components of vector A and B respectively.

Compare Similarity Measure

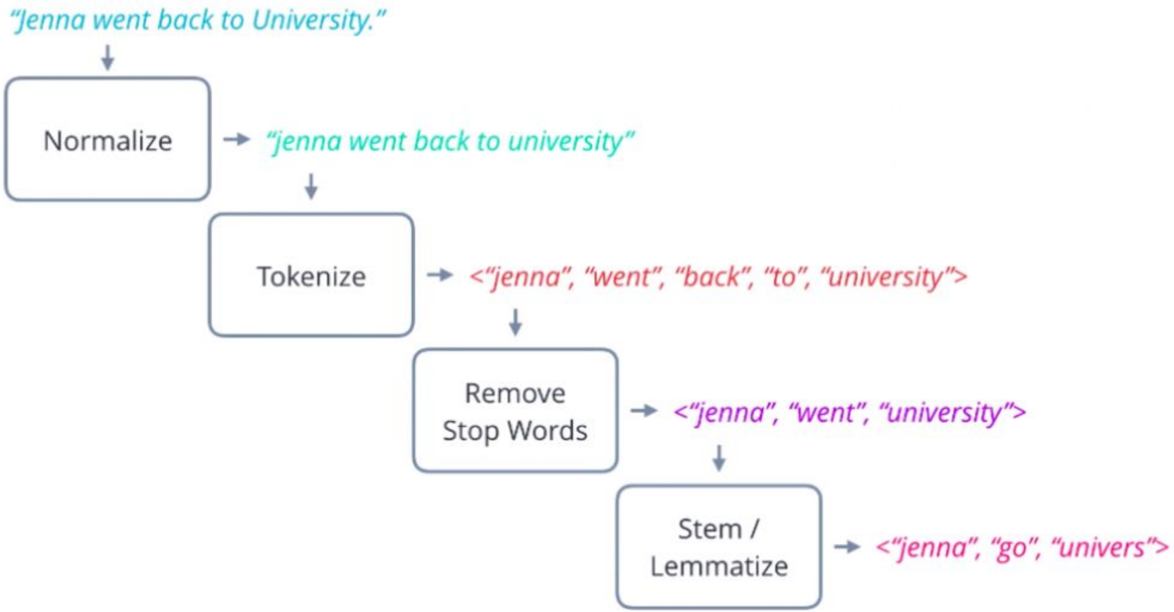
Query = "The Hitchhiker's Guide to the Galaxy (Hitchhiker's Guide to the Galaxy #1)"

corpus_docs	jaccard scores	euclidean_scores	cosine_scores
The Hitchhiker's Guide to the Galaxy (Hitchhiker's Guide to the Galaxy #1)	1.0	1.0	1.0
The Hitchhiker's Guide to the Galaxy (Hitchhiker's Guide to the Galaxy #1)	1.0	1.0	1.0
The Ultimate Hitchhiker's Guide (Hitchhiker's Guide to the Galaxy #1-5)	0.7	0.7544	0.9603
The Ultimate Hitchhiker's Guide to the Galaxy	0.6667	0.6958	0.9342
e: Five Complete Novels and One Story (Hitchhiker's Guide to the Galaxy #1-5)	0.4118	0.5359	0.8054
a Stranger Here Myself: Notes on Returning to America After Twenty Years Away	0.0476	0.3068	0.3021
Bryson's Dictionary of Troublesome Words: A Writer's Guide to Getting It Right	0.1111	0.2878	0.2245
Bill Bryson's African Diary	0.0	0.2838	0.207
Harry Potter Collection (Harry Potter #1-6)	0.0	0.2768	0.1749
Harry Potter and the Order of the Phoenix (Harry Potter #5)	0.0625	0.2744	0.1636
Harry Potter Boxed Set Books 1-5 (Harry Potter #1-5)	0.0	0.2737	0.1607

Sentence pre-processing using NLP

- Raw documents (sentences) are cleaned using standard NLP techniques
- Most of the pre-processing was done using the NLTK library
- Summary of the pre-processing steps include:
    - Normalising i.e. removing unwanted characters
    - Tokenizing
    - Removing stop words
    - Stemming/lemmatization

Sentence pre-processing using NLP

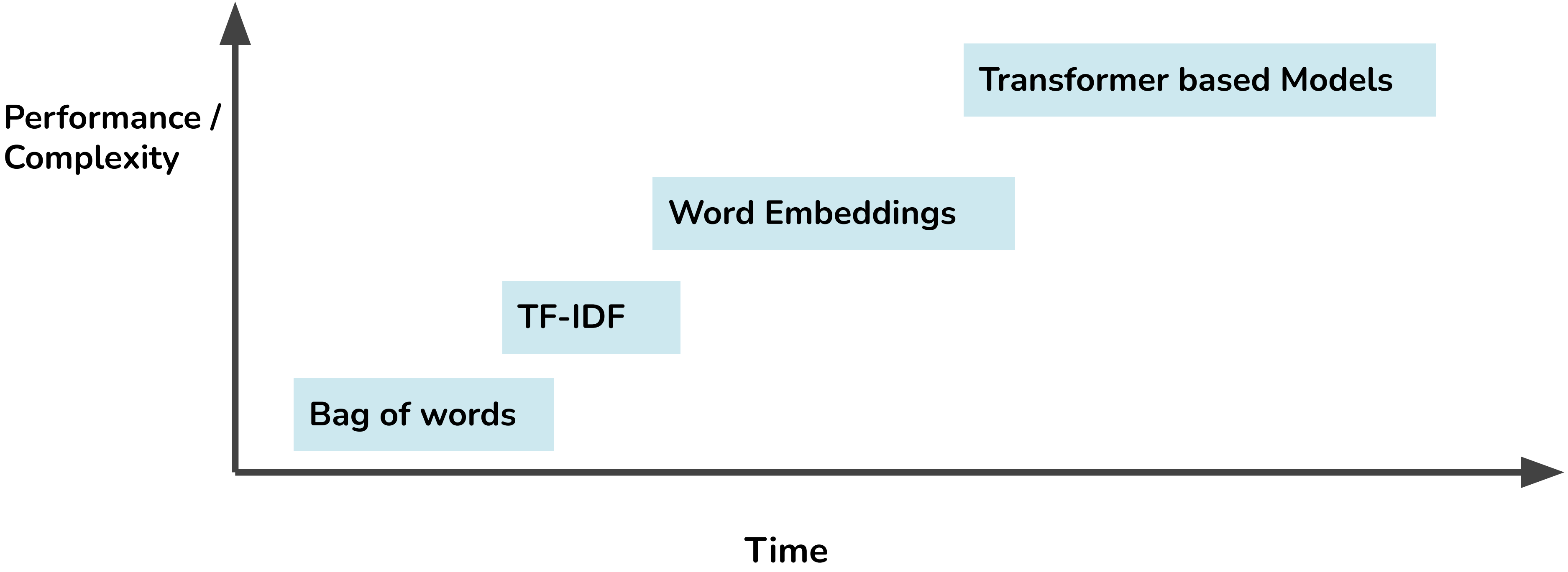


Feature engineering using Sentence Embedding techniques

Embedding techniques explored include:

- TF-IDF
- Word embedding with averaging
- Word embedding with SIF (Smooth Inverse Frequency)
- Google’s USE - (Universal Sentence Encoder)
- S-Bert Encoder

Evolution of Text Embedding Models



Sentence Embedding techniques

- TF-IDF

- TF-IDF (term frequency-inverse document frequency) is **a statistical measure that evaluates how relevant a word is to a document in a collection of documents**
- tf-idf(t, d) = tf(t, d) * idf(t,d)
  - tf(t, d): Term frequency count of t in d / number of words in d
  - idf(t): Inverse document frequency = log(N/(df + 1))
  - df(t): Number documents containing t

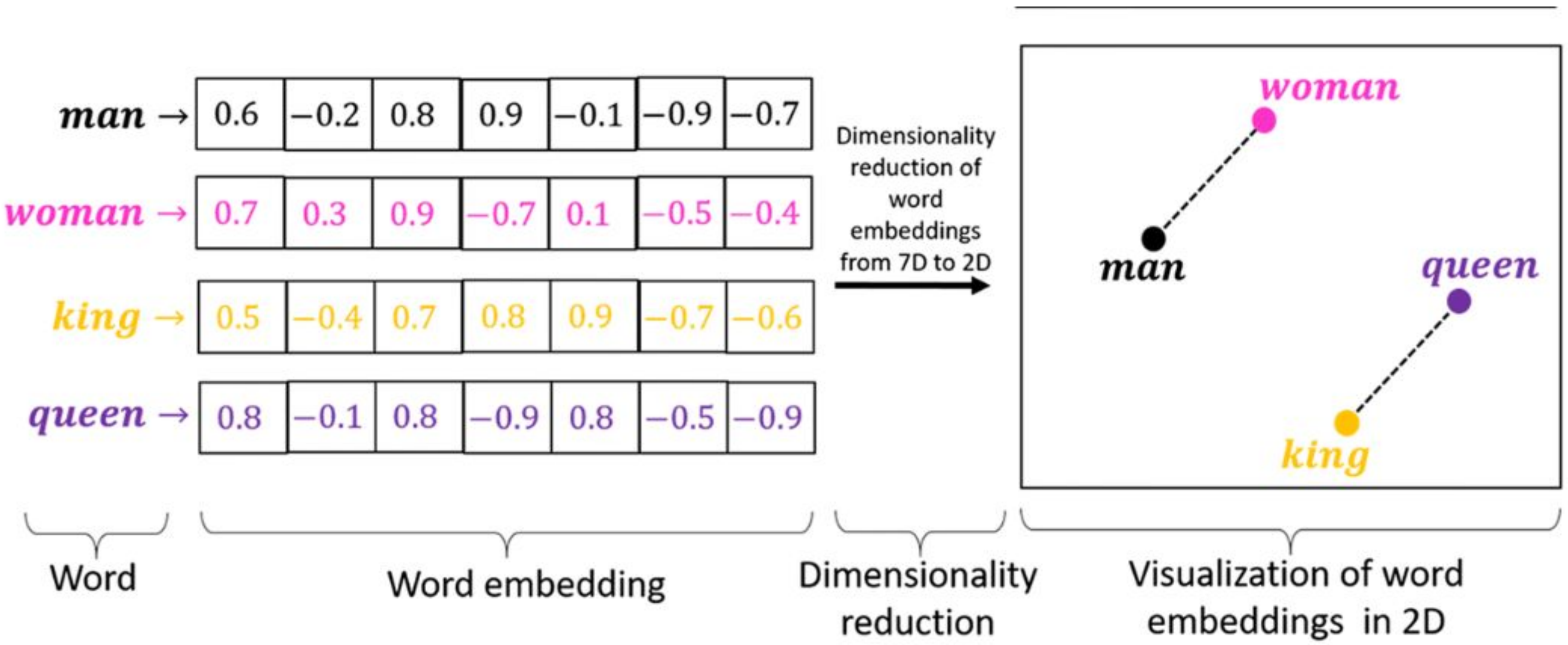
Sentence Embedding techniques

- Word Embedding

Word embedding is the representation of words as a real-valued vector that encodes the meaning of the word i.e. words that are closer in the vector space are expected to be similar in meaning

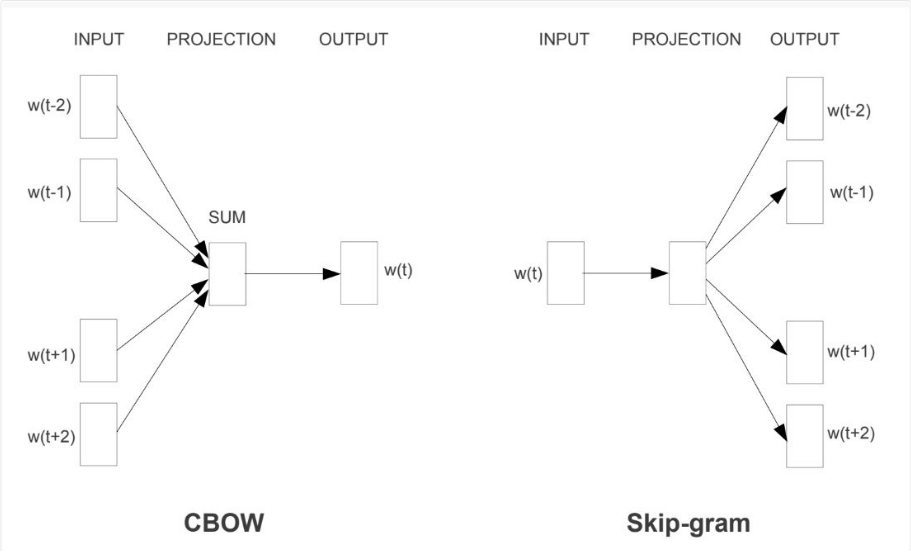
- Word embedding is a shallow neural net prediction-based approach
- Generally provides semantic meaning rather than frequency based approaches such as:
  - Count Vectorizer (bag of words)
  - TFIDF
- Word embedding vectors are low/dense dimensions such as 100, 200, 500, etc
  - Contrast to very large/sparse vectors produced by frequency-based embeddings
- Popular approaches to word embedding are:
  - Word2Vec (Google)
  - GloVe (Stanford Uni)
  - Fasttext (Facebook)

Word Embedding



Word Embedding - Word2Vec

- CBOW - Continuous bag of words
- Continuous Skip Gram



Sentence Embedding - by averaging word embeddings

S = “Hello participants of Pydata 2022!!”

W_1 = “hello”, W_2 = “participants”, W_3 = “of”, W_4= “pydata” , W_5 = “2022”

$$\begin{array}{c} W_1 \\ \begin{bmatrix} W_{11} \\ W_{12} \\ \\ \\ W_{1n} \end{bmatrix} \end{array} + \begin{array}{c} W_2 \\ \begin{bmatrix} W_{21} \\ W_{22} \\ \\ \\ W_{2n} \end{bmatrix} \end{array} + \quad .... \quad + \begin{array}{c} W_n \\ \begin{bmatrix} W_{n1} \\ W_{n2} \\ \\ \\ W_{nn} \end{bmatrix} \end{array} = \begin{array}{c} D \\ \begin{bmatrix} \frac{W_{11}+W_{21}+\ldots+W_{n1}}{n} \\ \\ \\ \frac{W_{1n}+W_{2n}+\ldots+W_{nn}}{n} \end{bmatrix} \end{array}$$

Sentence Embedding - by using Smooth Inverse Frequency (SIF)

- SIF was developed by *Sanjeev Arora, Yingyu Liang, Tengyu Ma* in a seminal paper titled: **A Simple but Tough-to-Beat Baseline for Sentence Embeddings**
- Sentence is embedded by a weighted average of the word vectors, and then modified a bit using PCA/SVD

Algorithm 1 Sentence Embedding

Input: Word embeddings $\{v_w : w \in \mathcal{V}\}$, a set of sentences $\mathcal{S}$, parameter a and estimated probabilities $\{p(w) : w \in \mathcal{V}\}$ of the words.

Output: Sentence embeddings $\{v_s : s \in \mathcal{S}\}$

for all sentence s in $\mathcal{S}$ **do**

$\quad v_s \leftarrow \frac{1}{|s|} \sum_{w \in s} \frac{a}{a+p(w)} v_w$

end for

Form a matrix X whose columns are $\{v_s : s \in \mathcal{S}\}$, and let u be its first singular vector

for all sentence s in $\mathcal{S}$ **do**

$\quad v_s \leftarrow v_s - uu^\top v_s$

end for

Sentence Embedding example

Sentence =

“The Hitchhiker\'s Guide to the Galaxy (Hitchhiker\'s Guide to the Galaxy #1)”

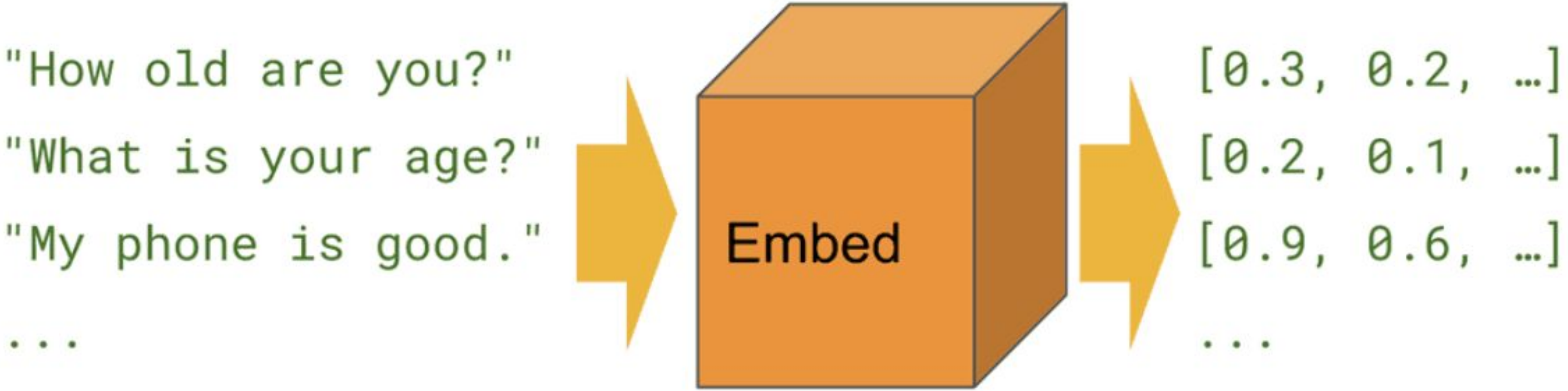
Embedding =

[-1.8593e-02, -3.0173e-04, 3.1182e-02, -2.5220e-03, -6.8647e-03,-7.5697e-03, -1.9542e-02, 3.5058e-02, 6.2249e-02, 1.7444e-02, 6.2431e-02, 2.9910e-03, 2.3374e-02, -2.0799e-02, -2.2098e-02,-3.1721e-02, -1.6348e-02, -4.7030e-02, 2.5878e-02, -5.6786e-03, -6.3790e-02, 8.3778e-02, -1.8895e-02, 2.1439e-02, 1.6235e-02,-4.9725e-02, 7.4626e-02, 1.9593e-02, -6.2797e-02, -4.6862e-02, -3.8502e-02, 9.1784e-02, -4.3534e-02, 2.2962e-02, -8.6189e-03,-8.3778e-02, 3.5653e-02, -6.1582e-02, 1.2460e-03, -5.9104e-02, .

 …………]

Google’s USE embedding

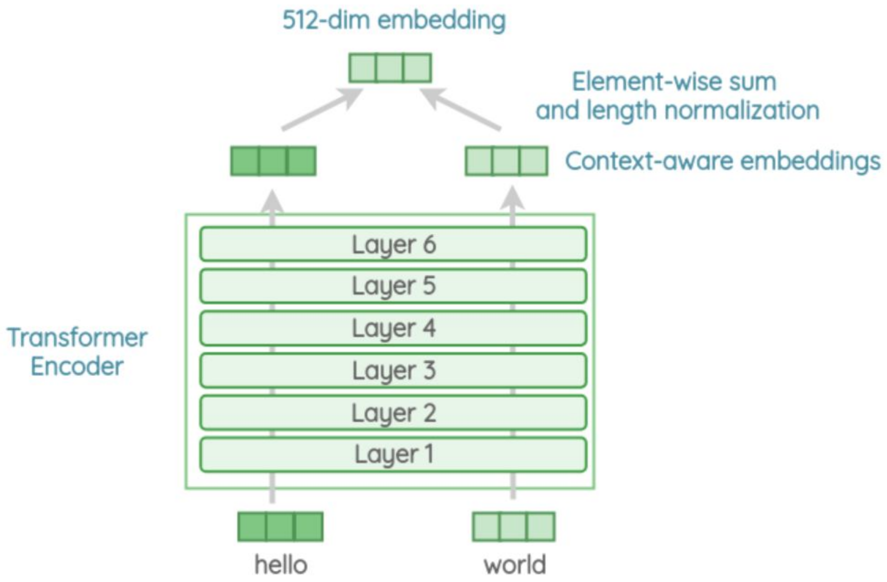
Google’s Universal Sentence Encoder provides a very easy solution to convert/encode sentences to embedding vectors



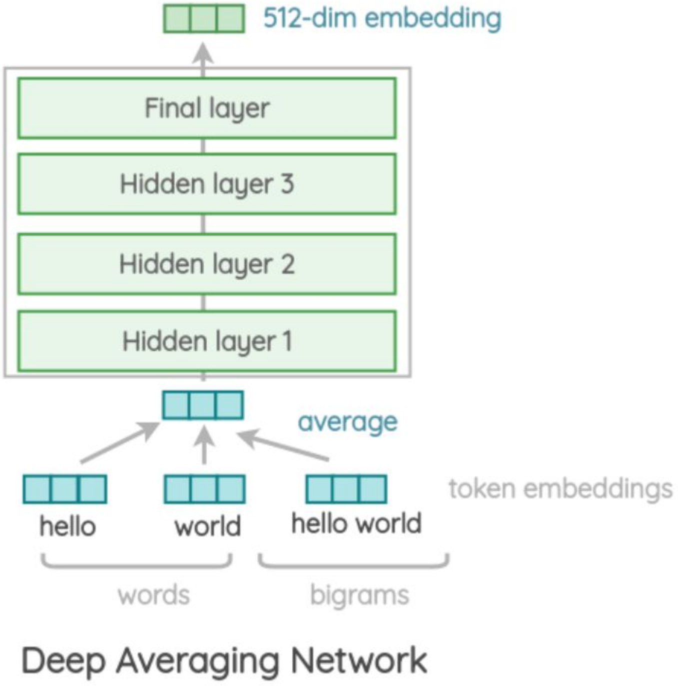
USE architecture

USE is provided in 2 variant architectures:

- Transformer
- DAN - Deep Averaging Network



USE architecture

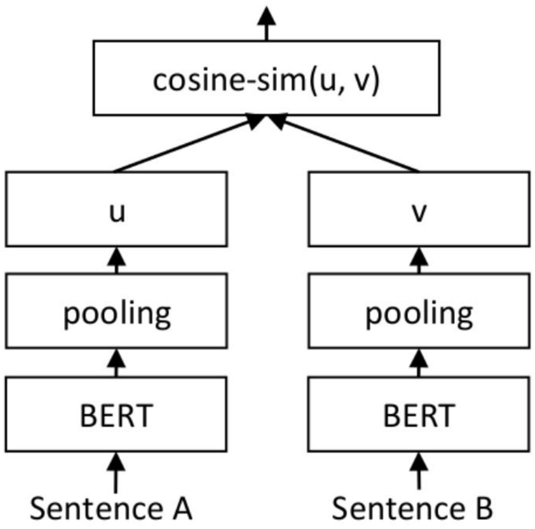


USE library/API details

```
import tensorflow as tf

import tensorflow_hub as hub

USE MODEL URL =
"https://tfhub.dev/google/universal-sentence-encoder/4"

def embed(input):
  return model(input)

sentence = "I am a sentence for which I would like to get
its embedding."

sentence_embeddings = embed(sentence)
```

S-Bert Encoder

- SBert.Net developed a very robust sentence encoder that utilizes a Siamese BERT-Networks
- Details of this solution can be found in the paper titled: Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks



S-BERT library/API details

```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer('all-MiniLM-L6-v2')


sentences = ['This framework generates embeddings for each
input sentence','Sentences are passed as a list of
string.', 'The quick brown fox jumps over the lazy dog.']

sentence_embeddings = model.encode(sentences)
```

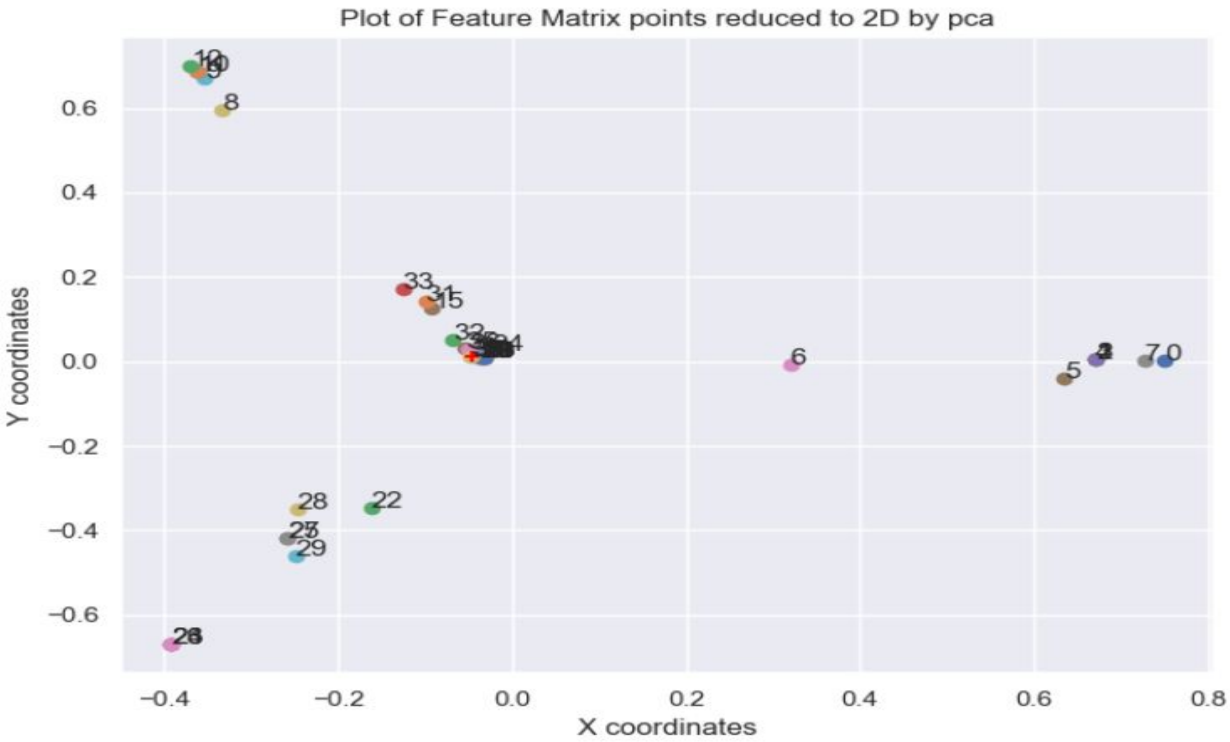
Visualizing Sentence embeddings

- 2-D Visualization of the sentence embeddings will provide an insight to the STS of the processed sentences
- Scatter plots of the embeddings and heatmaps of similarity matrix can be very useful
- But we will need to reduce the embedding dimensions to 2-D
- There are number of techniques to reduce the dimensions:
  - PCA - Principal Component Analysis
  - t-SNE - t-distributed Stochastic Nearest Embedding
  - MDS- MultiDimensional Scaling
  - UMAP - Uniform Manifold Approximation & Projection

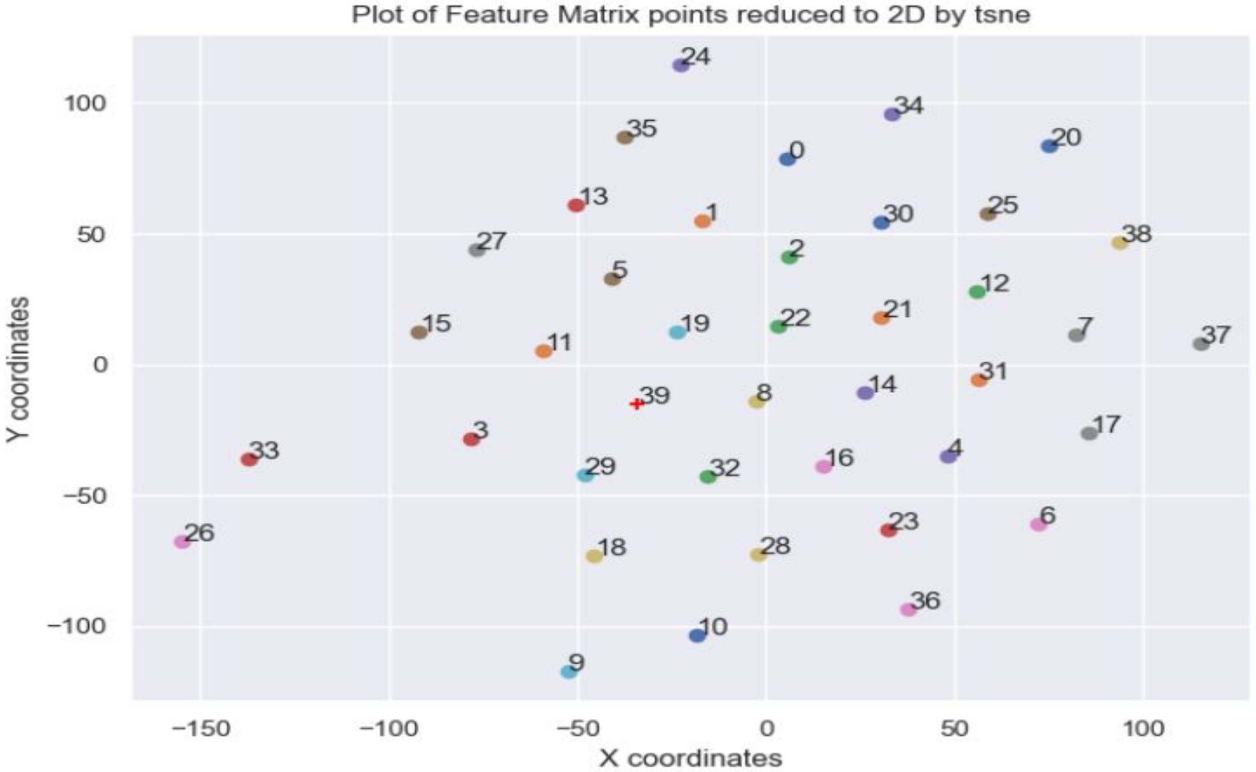
Dimension Reduction library/API details

- PCA:
  - sklearn.decomposition.PCA
    - sklearn.decomposition.PCA(n_components=2)
- t-SNE:
  - sklearn.manifold.TSNE
    - sklearn.manifold.TSNE(n_components=2)
- MDS:
  - sklearn.manifold.MDS
    - sklearn.manifold.MDS(n_components=2)
- UMAP
  - umap
    - umap.UMAP(n_components=2)
- All four models use fit/transform api call:
  - <model>.fit_transform(<embedding>)

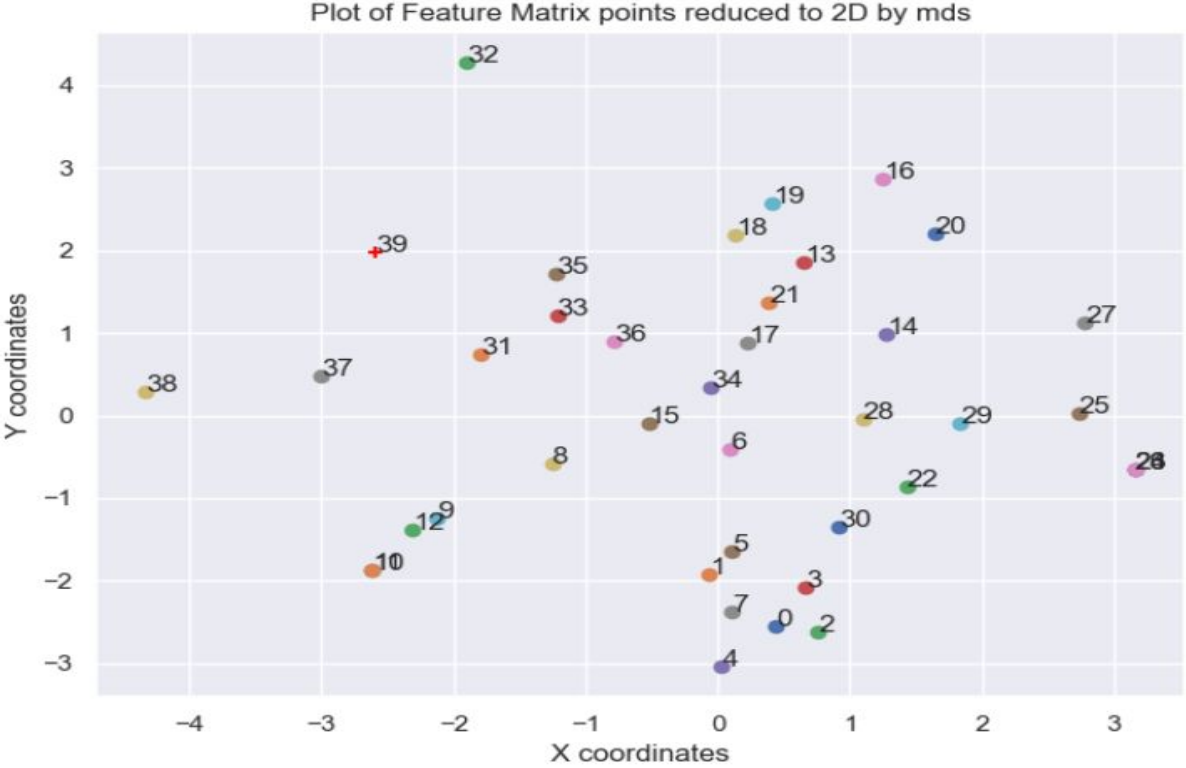
Example of PCA Visualization (tf-idf)



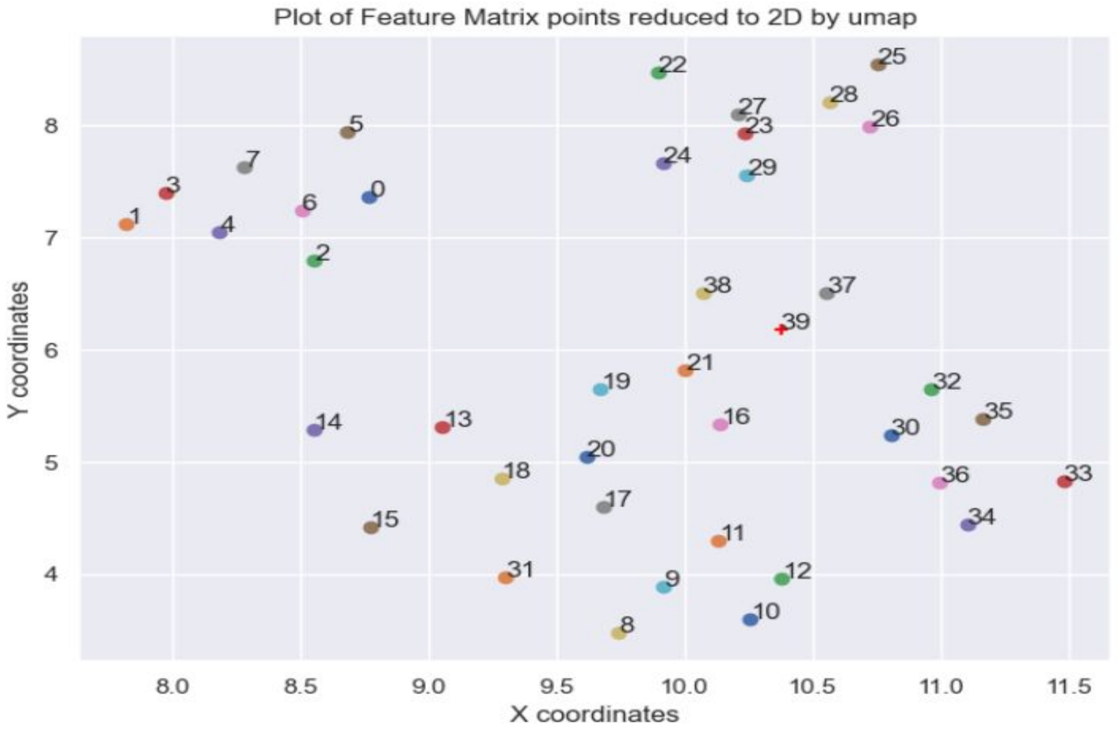
Example of t-SNE Visualization (sif)



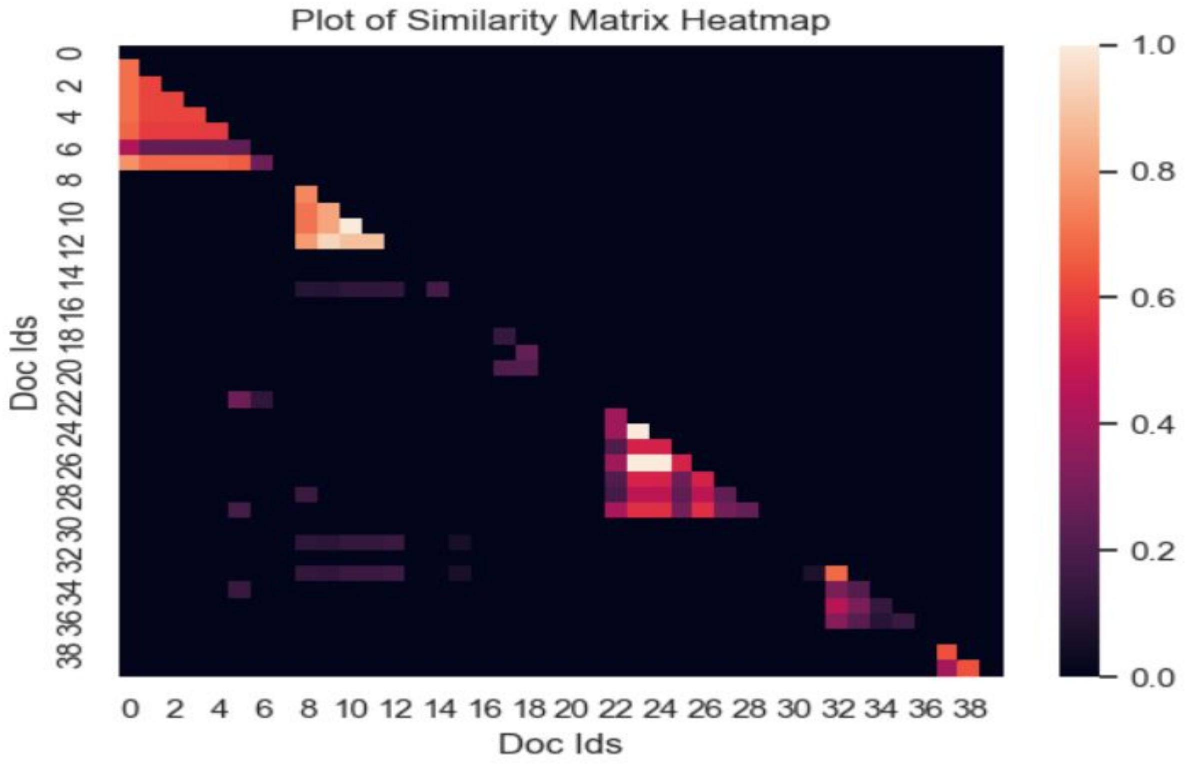
Example of MDS Visualization (use)



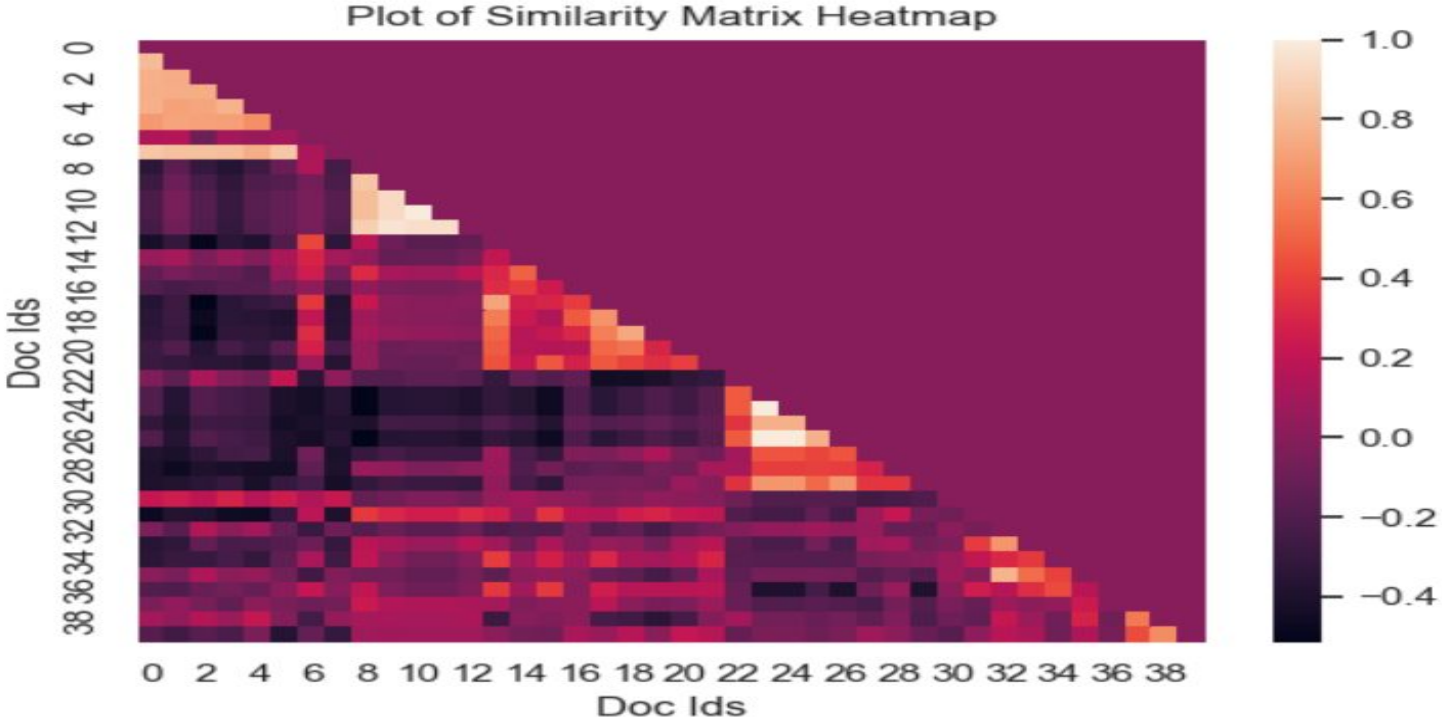
Example of UMAP Visualization (s-bert)



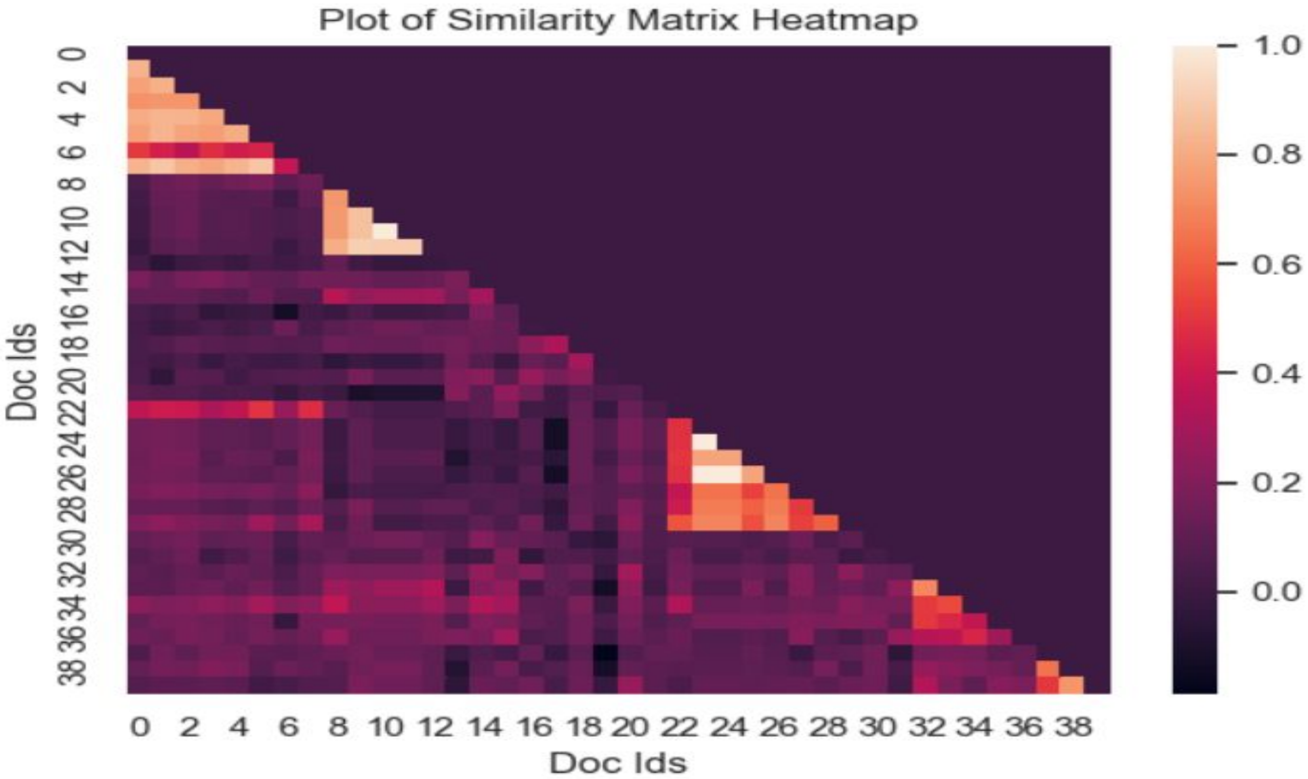
Heatmap of TF-IDF computed similarity



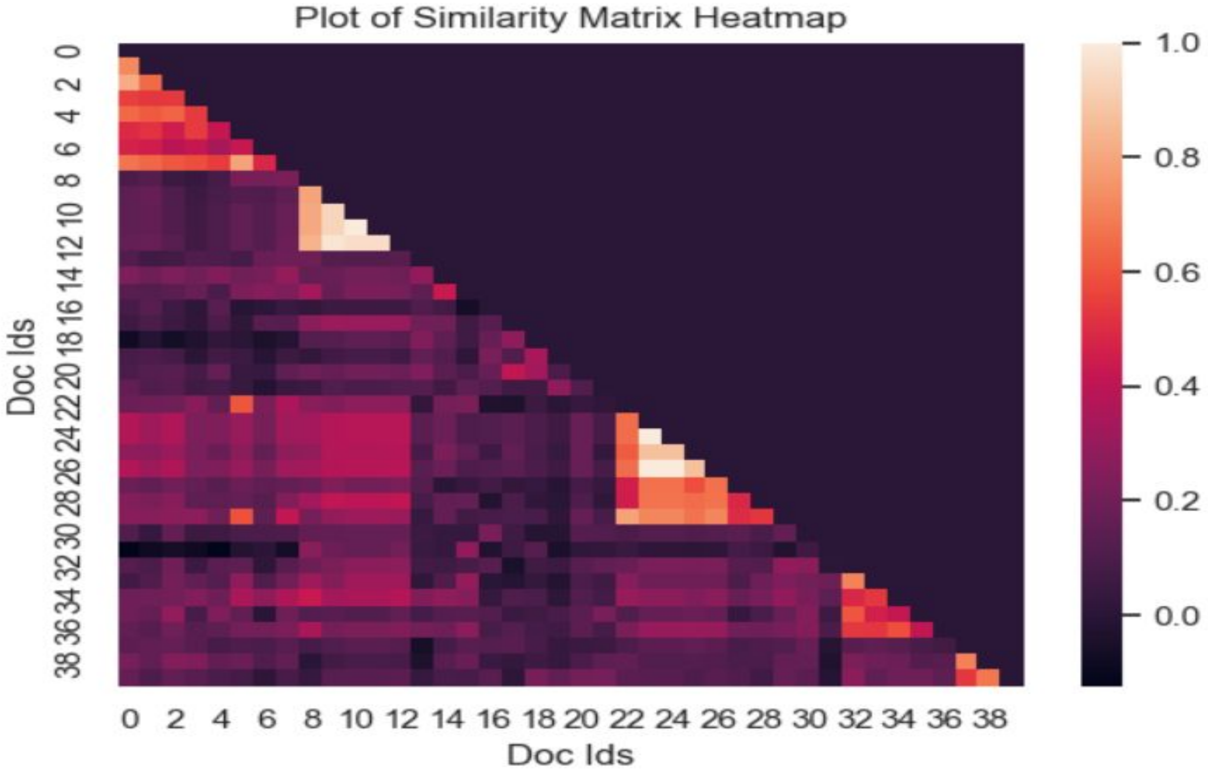
Heatmap of SIF Word Embedding computed similarity



Heatmap of USE computed similarity



Heatmap of S-BERT computed similarity



Validation of Similarity Results

- The similarity results for the embedding methods were validation using Pearson Correlation metric
- Pearson correlation metric computes the linear correlation between two sets of data.
- This metric was used to compute the correlation between the ‘actual’ similarity (label) against the predicted similarity using any 1 of the 5 embedding approaches

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y}$$

where:

- cov is the covariance
- σ_X is the standard deviation of X
- σ_Y is the standard deviation of Y

Comparison of Embedding approachs

Embedding type	Pearson correlation	p-value
sbert	0.851	0.0
use	0.7912	0.0
sif_word_embedding	0.7145	0.0
average_word_embedding	0.6431	0.0
tfidf	0.6273	0.0

Demo of Solved Problems

- **Simple demo the similarity between book titles**
    - Problem: Use the 5 embedding approaches to encode the titles and compute the top k similarities between the titles
    - Data: Goodreads data sourced here
- **Demo a simple Search Engine**:
    - Problem: User provides a query and it is compared (searched) in a corpus of documents to get the top k matches
    - Data: The classic 20 News Group data sourced from Scikit-Learn dataset module
- **Demo the performance of the 5 embedding strategies using labelled sentence pair corpus data**:
    - Problem:
        - Embed the sentence pairs from the sentence pair corpus using 5 approaches
        - Compute the similarity between each sentence pair
        - Compute the performance of each approach using Paerson’s correlation coefficient i.e. computing the predicted similarity versus the actual similarity (label)
    - Data: STS Benchmark Sentence Pair data sourced from here

Program Library requirements

fse == 1.0.0,   gensim == 4.2.0

matplotlib == 3.3.4, nltk == 3.7

numpy == 1.22.3, pandas == 1.2.4

scipy == 1.6.2, seaborn == 0.11.2

sentence_transformers == 2.2.0, sklearn == 0.0

tensorflow == 2.4.0, tensorflow_hub == 0.12.0

torch == 1.11.0, tqdm == 4.59.0

umap_learn == 0.5.3

Demos with Q & A