Information Technology Course
Module Software Engineering
by Damir Dobric / Andreas Pech

ML 24/25-09 Semantic Similarity Analysis of Textual Data

Senthil Arumugam Ramasamy
senthil.ramasamy@stud.fra-uas.de

Neethu Ninan
neethu.ninan@stud.fra-uas.de

***Abstract*— In the era of big data and natural language processing (NLP), applications such as information retrieval and classification require a deep understanding of semantic similarity in text. This project aims to provide a tool capable of identifying meaningful relationships across words, phrases, and documents. The results show reliable and consistent classification of relevant versus irrelevant content across various domains of respective textual data, aligning well with human judgment. These results confirm the tool's potential for practical deployment and further NLP research.**

Keywords—semantic similarity, textual data

I. INTRODUCTION

Billions of brief text messages are sent on social media every day; of which nearly every tweet is between one and thirty words long. Appropriate information retrieval methods are required in order to access this stream of really brief text fragments. There are varieties of methods available to analyze the meaning of a text. Lexical-Based Similarity (LBS) analysis is the conventional one that relies on calculating the separation between two chains in order to identify their similarities. Character-based and term-based distance measurements are the two categories into which LBS measurements fall. Character-based metrics is used to address typographical problems. Nevertheless, these metrics fail to capture the resemblance with term arrangement problems. Term-based similarity metrics attempt to address this problem. Rather than relying on just character matching, semantic similarity determines how similar two sequences are based on their importance. Lastly, hybrid-based similarity method combines lexical-based and semantic-based similarity metric and hence requires a lot of computational stages [1].

Semantic similarity is a fundamental task in NLP (Natural Language Processing) crucial for applications like information retrieval, document clustering, and recommendation systems. It measures how closely the meanings of texts align, based on direct and indirect semantic relationships. Estimating this similarity remains a challenging research problem due to the complexity and flexibility of natural language, making rule-based approaches difficult. Various techniques have been proposed over time to tackle this issue [2]. There are many methods associated with semantic similarity analysis which falls into three main categories. The classical methods include Latent Semantic Analysis (LSA), uncovers latent structures using a technique called Singular Value Decomposition (SVD) [3], and Explicit Semantic Analysis (ESA), which maps text to structured knowledge sources like Wikipedia [4]. Distributional models such as Word2Vec [5] and GloVe [6] introduced dense vector embeddings based on word context but lacked the ability to capture polysemy and nuanced meaning. The advent of transformer-based models has brought a significant leap by offering context-aware embeddings. The main models include BERT (Bidirectional Encoder Representations from Transformers) [8], GPT (Generative Pre-trained Transformer) and Sentence-BERT (SBERT) [7]. These models excel in understanding semantics at the sentence level by providing a meaningful vector space representation. Both traditional and modern approaches contribute to a robust foundation for semantic analysis, with transformer models leading current advancements. While traditional methods like LSA and ESA are straightforward, they lack contextual awareness. Word embeddings offer improved semantic representation but still struggle with ambiguity as they rely on statistical connections, static embedding generation and require fine tuning and extensive training especially for BERT. Transformer-based models address these issues.

This project utilizes OpenAI GPT (Generative Pre-training Transformer) model and Cosine similarity function to generate and compare embeddings, hence the textual similarity. The text-embedding-3-large model is utilized to generate contextual embeddings that effectively capture the semantic meaning of textual data. This model supports embeddings with a dimensionality of 3072, allowing for a richer and more detailed representation of information. The embedding process begins with tokenizing the input text, which involves converting the text into smaller units (tokens) that the model can interpret and process [9]. After that, each token is transformed into its matching embedding. An

embedding is a list of floating-point numbers that is vectorized which represent various ideas in the respective content [11]. Two vector's distance from one another indicates how related they are. Large distances indicate low relatedness, while small distances indicate high relatedness [10]. The commonly used distance functions are Dot product, Minkowski Distance, Euclidean Distance and Cosine Distance. The application uses cosine similarity to yield a textual similarity metric mainly because it does not need any additional normalization as vectors are automatically normalised by cosine similarity, which yields a more reliable similarity metric that depends only on the angle between the vectors rather than their lengths. Also, it is independent of vector magnitude, in contrast to the dot product and Euclidean distance. Because of this, it is especially helpful for comparing text embeddings, where the direction of the vectors is more important than their length [12]. The cosine similarity between two vectors, A and B, can be expressed mathematically as:

$$\cos\theta = \frac{A.B}{||A||||B||}$$

A.B =dot product
||A|| and ||B|| =vector magnitude

A module for processing the raw input data has been also designed to make the embeddings to focus more on contextual meaning rather than every piece of data or characters in the input. This is implemented in order to extract the relevant context only. The main functionalities in this module include unwanted character removal, conversion to lower case, text normalization, lemmatization/stemming. The generated similarity scores are saved into CSV files, from there an external Python functionality is incorporated to enhance the visualization experience due to the available facilities in Python. By incorporating these approaches, the project implements a robust and well-structured application to analyse the semantic relationship in real world. This enables effective similarity analysis between textual data without requiring extensive pre-training, as openAI GPT model naturally provides context-aware dynamic embeddings. Additionally, the application is designed in a modular way thereby facilitating future research and developments.

II. METHODS

The Semantic Similarity analysis of Textual Data is implemented in C# programming language using Microsoft Visual Studio 2022 IDE (Integrated Development Environment) and .Net version 9. This project mainly focusses on the analysis of semantic meaning of textual data by calculating the similarity between them. Fig 1, illustrates the flow of program which represents the end-to-end process of analyzing semantic similarity between textual inputs such as documents or phrases. The process begins with the user providing inputs if documents or phrases have to be compared and whether preprocessing is required. If the preprocessing is not required, the raw text proceeds directly to the embedding stage. However, if preprocessing is needed, the system applies basic natural language processing steps such as the removal of common articles (e.g., "the", "a", "an") and stemming, which reduces words to their root forms. This results in clean and normalized text. The clean text is then passed to the embedding module, where semantic embeddings are generated using OpenAI's embedding API. These embeddings are numerical vector representations of the text that capture its contextual meaning. Once the embeddings are obtained, a cosine similarity calculation is performed to measure the similarity between each pair of input texts. This similarity score is a float value ranging from -1 to 1, where 1 indicates complete similarity, -1 indicates complete dissimilarity, and 0 indicates no semantic relation. The calculated similarity scores are organized and stored in a CSV file for ease of use and analysis. Finally, the system provides a visualization of these results using Python-based plotting libraries, offering an intuitive representation of the semantic relationships between the input texts.

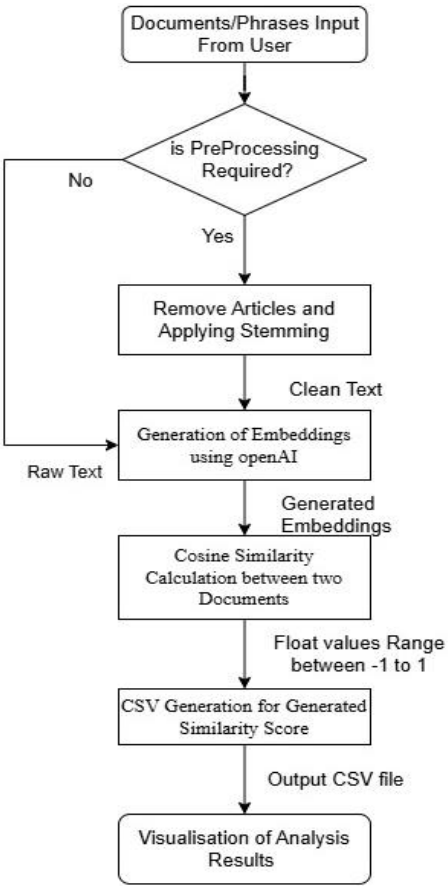

Fig 1: Simplified Block Diagram representing the flow of the project

The project flow as shown in the block diagram can be briefly explained as follows:

1. Creation of Datasets

For words/phrases, around fifty to sixty different words from the same and different domains were selected to utilize for comparison. Also, phrase comparison datasets are formatted as CSV files, which could be modified as required. The application was also designed to support the comparison of documents to show contextual relevance between them.

For any comparison to be performed, both source documents and target documents are required. For example, JobRequirement.txt was treated as the source document, while Job Profile A and Job Profile B are treated as the target documents. The source document contains the core information to compare, while target documents have the information to be compared. Once compared, meaningful results could be obtained. The work is intended to allow the user to set a defined threshold value for better comparison or even classification and a user-specific UI to upload documents. The application supports document comparison dynamically, making it more usable and researchable. Currently just by adding data to source and target documents respectively, but can be enhanced by integrating a UI (User Interface) module in later research.

2. Pre-Processing Interface to Process the Documents

To enable semantic analysis of textual data, the application begins by organizing documents into a predefined folder structure. Two primary folders are maintained; one representing the source documents and another representing the target documents intended for comparison. As illustrated in Fig 2, the source folder referred to as “SourceBasedOnDomains” contains documents that serve as the reference or baseline content. These documents are typically grouped according to specific domains, such as job profiles or academic backgrounds, and represent the context against which other inputs will be compared. The second folder namely “SourceBasedOnRelevance” acts as the target folder and holds documents that need to be evaluated for semantic similarity with respect to the source content.

To ensure accurate and consistent comparisons, the system incorporates a preprocessing stage. In this stage, documents from both the source and target folders undergo a series of normalization steps, including lowercasing, removal of stop words and special characters, and lemmatization. These steps are applied to eliminate noise and linguistic variance that may otherwise interfere with the semantic interpretation. As part of this process, two new folders are dynamically created as ProcessedSourceFolder and ProcessedTargetFolder. These folders store the cleaned and preprocessed versions of the original documents.

The preprocessed documents from these two folders are then used in the semantic similarity comparison phase. By standardizing the input through preprocessing and maintaining a clear separation between raw and processed data, the system ensures both reproducibility and flexibility. This structure also facilitates the integration of enhancements or additional preprocessing steps in future development cycles.

The use of this two-tiered folder system with both raw and processed data sets also prepares the application for potential user-facing interfaces. For example, users could upload their own documents (e.g., resumes or statements of purpose) to be automatically preprocessed and compared against predefined templates or expectations, supporting real-time feedback and recommendations.

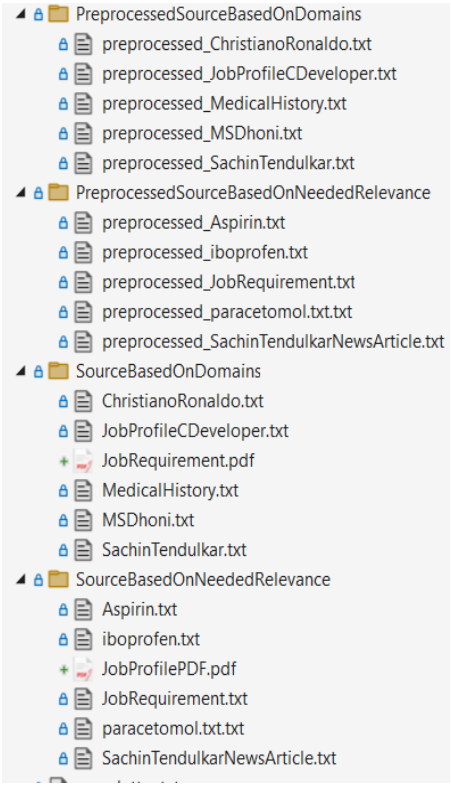

Fig 2: Document folder Structure

Text pre-processing is a fundamental step in our system, aimed at transforming raw textual data into a normalized format suitable for semantic analysis. This process is implemented through the TextPreprocessor class, which adheres to the IPreprocessor interface. It supports processing of various types of input, including words, phrases, and full-length documents. The primary goal of this component is to clean the input text, remove irrelevant information, and standardize the content for effective semantic embedding. Upon receiving an input file, the system first initializes the text data by storing its name, content, and file path. The pre-processing routine begins by converting the entire text to lowercase and removing leading or trailing whitespace. It then applies regular expression-based cleaning to eliminate HTML tags, URLs, and unwanted punctuation or special characters. Additionally, the system expands commonly used contractions such as "can't" to "cannot" and "I'm" to "I am" to maintain consistency in the semantic structure of the text.

A core aspect of the pre-processing pipeline is stop word removal. The system uses a predefined HashSet of stop words commonly occurring but semantically insignificant words like "the", "is", "and", etc. to eliminate noise from the input. This helps in focusing the semantic analysis on content-rich terms. Furthermore, a basic lemmatization mechanism is incorporated using a Dictionary that maps inflected or derived forms of words to their root forms (e.g., "running" to "run", "better" to "good"). This ensures that semantically equivalent terms are treated consistently during embedding generation.

3. *Create Embedding's for the input documents*

The embedding generation process is a key step in the semantic analysis workflow. It involves converting raw or pre-processed textual input into a numerical form that captures the underlying meaning of the text. This is achieved by sending the input data to a pre-trained embedding model, which returns a high-dimensional vector representation for each input. These vectors, known as Embeddings, encode semantic properties that allow for effective comparison between different pieces of text.

In this system, Embeddings are generated for each pair of documents or phrases that need to be compared. The process is performed asynchronously to improve performance and scalability, particularly when dealing with large sets of documents. Once the Embeddings are retrieved, they are stored for further analysis. These stored Embeddings serve as the foundation for subsequent similarity computations, enabling the system to identify and quantify semantic relationships between textual inputs.

Custom method named "PrintScalarValues" is also created to display the output of each document or phrase as embeddings that is to print individual values of every array until the end of the collection size. This method is developed to print the scalar values of embeddings to assist in visualization. By examining these values, the correlation between the generated similarity score and the scalar values in vector space can be better understood.

4. *Calculate Similarity using Cosine Similarity Algorithm based on programmatically generated Embeddings of input documents*

In order to calculate the similarity from the vector embedding, in our project, the implementation of the similarity calculation is done by applying the Cosine Similarity algorithm. The algorithm is capable of returning a single similarity score as a double value, with results ranging from -1 to 1. A value of 1 indicates perfect similarity (identical embeddings), 0 indicates orthogonal vectors (no similarity), and -1 signifies complete dissimilarity (opposite vectors).

The implementation of the Calculate Similarity process involves several steps. First, input validation is performed to ensure that the lengths of the two embeddings are equal. If not, the method returns a score of zero, indicating an invalid comparison. The second step involves Dot Product and Magnitude Calculation, where the dot product between the two embedding vectors is computed, followed by calculating the magnitude of each embedding vector separately using the formula: Magnitude = sqrt (Σ embedding[i]^2).

The third step is normalization, where the cosine similarity score is derived by dividing the dot product by the product of the magnitudes, using the formula: Cosine Similarity = Dot Product / (Magnitude1 * Magnitude2). This normalization ensures that the similarity score remains within the valid range of -1 to 1. Finally, the implementation includes Error Handling to address scenarios where the magnitude of any vector is zero, which would result in some invalid calculations. In such cases, an error is raised to notify the user/developer, and if any other unexpected errors occur, the method returns a similarity score of zero and logs the error. This structured approach ensures that the similarity calculation is robust, reliable, and capable of handling a wide range of input scenarios.

5. *Output Generation and Semantic Analysis*

The application is designed to generate its output in the form of structured CSV files. These files store the results of semantic similarity comparisons between pairs of documents or phrases. Each record in the CSV captures key details such as the two items being compared, their respective domains or contexts, and the similarity score derived from their semantic embeddings. This structured format allows for easy storage, retrieval, and analysis of the generated data.

The output data is further utilized in external visualization tools to graphically represent the relationships and patterns observed between the textual inputs.

Interactive plots have been incorporated into the results section to provide a clearer and more intuitive explanation of the outcomes. These include phrase similarity classifications by domain, document similarity across source and target files, and scalar embedding visualizations. The interactive nature of these plots allows for detailed inspection of individual data points and supports a more comprehensive understanding of the semantic relationships being analyzed.

External Sources for viewing sample live interactive plots are deployed at our repository GitHub pages are given below for a better understanding:

Phrase Comparison Plots:

Phrase Comparison Analysis Plots

Document Comparison Plots:

Documents Similarity between Source and Target Documents

Scalar Generation Plots:

Plotting Embedding As Scalar Plots

6. *Test Case Implementations*

Several test cases have been implemented using Microsoft.VisualStudio.TestTools.UnitTesting, and the test cases are executed through Visual Studio Test Explorer inorder to ensure that all code and methods are covered when executed through the Test Explorer. This coverage includes both positive and negative scenarios, such as handling exceptions and null values. Mock data has not been used in

the current implementation, as this application serves as the initial setup for analysis purposes. The application's "data" folder, which was defined for maintaining datasets, has been copied into the test project. These files are read dynamically by the program to successfully maintain code coverage and execute the test cases. The test cases have been separated by creating individual test classes for each defined service, based on their functionalities. Since the functionalities have been segregated into four classes, corresponding test classes have been created, as illustrated in Fig. 3.

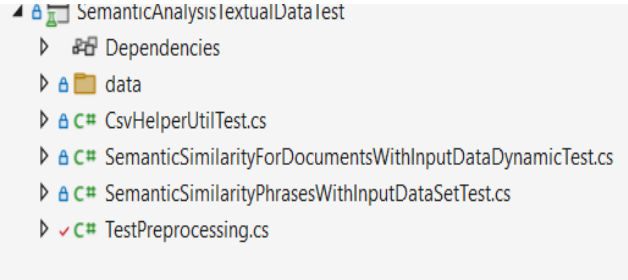

Fig 3: Folder Structure for Test Project

The implemented test cases in each categories are briefly expained as follows:

1) Test for Preprocessor Method: This test ensures all methods in the preprocessing module function correctly. It is crucial for confirming accurate embedding generation.

2) Basic Test Framework Validation*:* This test verifies the setup of the test framework by creating an instance of the class and performing a simple assertion check.

3) Exception Handling in Document Comparison: This test sets up directories and files, invokes the CompareDocumentsAsync method, and checks for valid results, ensuring proper exception handling.

4) Validity of Similarity Score Calculation: Reads content from source and target files, calculates similarity scores using the CalculateEmbeddingAsync method, and confirms that scores are greater than zero.

5) Service Provider Configuration Validation: Validates the configuration of services returned by the ConfigureServices method, ensuring it matches the expected type.

6) Source and Target File Retrieval: Tests the GetSourceAndTargetFiles method, ensuring the correct retrieval of source and target files with valid file extensions.

7) Scalar Value Printing Validation: Captures and validates console outputs from the PrintScalarValues method, ensuring correct scalar value printing.

8) Accuracy of Similarity Score Calculation: Evaluates similarity scores between different pairs of embeddings, ensuring correctness.

9)Handling of Different Length Embeddings: Confirms that the CalculateSimilarity method returns zero for embeddings of different lengths.

10) Handling of Empty Inputs: Ensures the CalculateEmbeddingAsync method returns a similarity score of 0 when given empty input strings, verifying robustness and reliability for edge cases.

11) Handling of Invalid File Paths: Validates that the document comparison service properly detects and handles invalid or non-existent file paths by raising appropriate exceptions.

12) Phrase Processing and Result Saving: Confirms the correct processing of phrases and saving of results using the CsvHelperUtilTest class.

III. RESULTS

To fulfill the goal of understanding the semantic analysis of textual data between documents and generating meaningful outputs that correlate with real-time use cases, Python was employed as an external development tool to generate graphical charts. The application is designed to read the output CSV file, which is dynamically generated and placed in the root directory of the Python application. Currently, our project requires manual effort to place the output CSV file. However, this can be overcome with further developments in this area.

```
SimilarityScore,Phrase1,Phrase2,Domain,Context
0.6479362993424067,Marathon,Boston Marathon,Sports,Event
0.25625848711100535,Cardiology,Stent,Medical,Device
0.49747645303261157,Ecosystem,Biodiversity,Environment,Balance
0.4561009677421827,E-sports,DOTA 2,Technology,Gaming
0.21914239454245363,Angela Merkel,Government,Politics,Leadership
0.10346757405300674,Cristiano Ronaldo,Government,Sports,Irrelevant
0.6275231159758883,Dog,Cat,Animals,Pets
0.3517908733108561,Medicine,Paracetamol,Medical,Medical
0.41384088925191015,Cricket,Sachin Tendulkar,Sports,Sports
0.5085530528447204,Cricket,Sports,Sports,Sports
0.37765978608302875,Cricket,Yorker,Sports,Bowling
```

Fig 4: Representing Similarity Score generated for Phrases

```
FileName1,FileName2,domain,SimilarityScore
preprocessed_ChristianoRonaldo.txt,preprocessed_Aspirin.txt,Unknown,0.08791521703854203
preprocessed_ChristianoRonaldo.txt,preprocessed_iboprofen.txt,Unknown,0.09176405752194787
preprocessed_ChristianoRonaldo.txt,preprocessed_JobRequirement.txt,Unknown,0.1691789017744734
preprocessed_ChristianoRonaldo.txt,preprocessed_paracetomol.txt.txt,Unknown,0.07036064090718619
preprocessed_ChristianoRonaldo.txt,preprocessed_SachinTendulkarNewsArticle.txt,Unknown,0.3962180875445941
preprocessed_JobProfileCDeveloper.txt,preprocessed_Aspirin.txt,jobvacancy,0.002868180085283813
```

Fig 5: Representing Similarity Score generated between Documents

Both Fig 4 and Fig 5, indicates the generated similarity scores for phrases and documents respectively. As documents requires preprocessing, it is shown with tag "preprocessed"

in from of each document name. The corresponding output CSV files created by the program are saved in the release/data folder inside the project.

Three types of plots are designed to support analysis the data distribution from the generated scores.

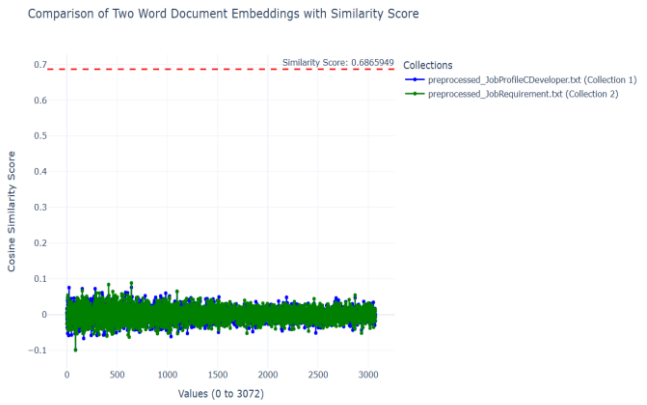

Fig 6: Scalar Values vs. Similarity Score Plot for One Comparison

As shown in Fig 6, the chart illustrates the contextual relevance between two documents by plotting similarity scores on the Y-axis against scalar values corresponding to the sequence of embeddings, ranging from 0 to 3052, on the X-axis. The example shown represents a comparison of generated embedding values between two documents: *preprocessed_JobRequirement.txt* (used as the source document) contains sample job requirement data and *preprocessed_JobProfileCDeveloper.txt* (used as the target document with a sample candidate resume).

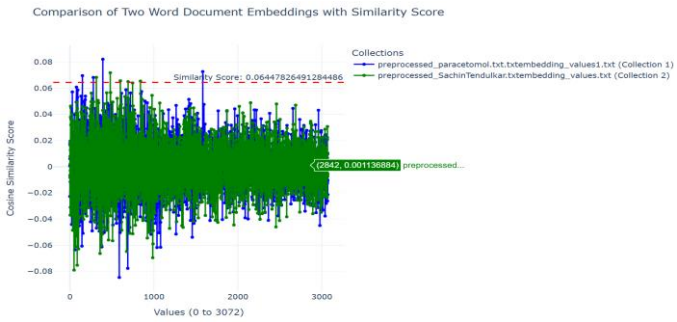

Fig 7: Scalar Values vs. Similarity Score Plot for One Comparison for two dissimilar documents

As shown above, Fig 7 was also designed to illustrate how contextual relevance is generated by plotting **similarity scores** on the Y-axis against **scalar values** generated as collection of embeddings (ranging from 0 to 3052) on the X-axis and this plot is plotted between two dissimilar documents to observe if embeddings overlap between documents is less which is adding some meaning to understand the relevance. The example shown represents a comparison of generated embedding values between two documents: "*preprocessed_Paracetamol.txt"* (used as the source information contains sample data about paracetamol medicine) and "*preprocessed_SachinTendlkar.txt"* (used as the target document contains sample information about famous cricketer in India).

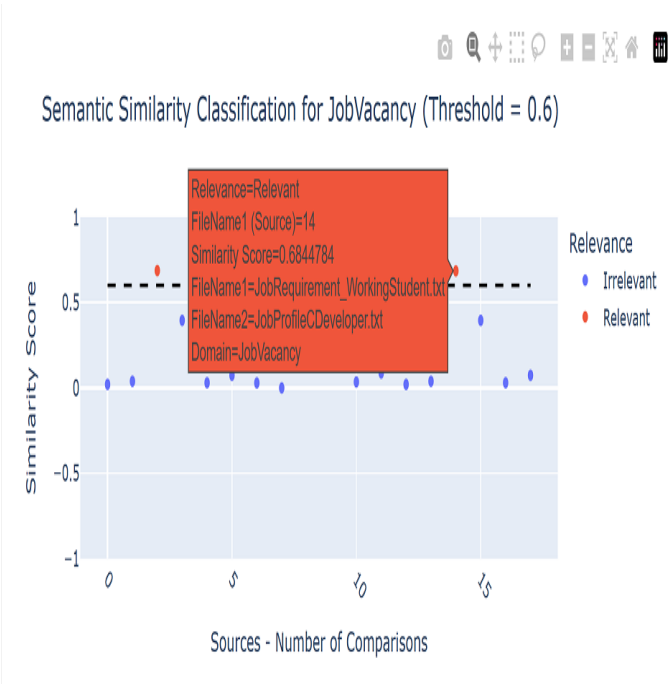

Fig 8: Total Number of Documents Compared VS Similarity Score- Job Vacancy

The chart shown in Fig 8; provides a visual representation of document similarity comparisons within the provided dataset. The X-axis displays the defined document comparison pairs, while the Y-axis indicates their corresponding similarity scores. Each dot on the chart represents a single comparison between two documents. For example, one such comparison involves "*JobRequirement_WorkingStudent.txt"* the source document contains sample job requirement data and "*JobProfileCDeveloper.txt"* as the target document with a sample candidate resume, resulting in a similarity score of **0.6844784**. A threshold value of **0.6** is marked with a dotted line. Similarity scores above this threshold, shown as red dots, are considered relevant, while those below the threshold, shown as blue dots, are considered less relevant.

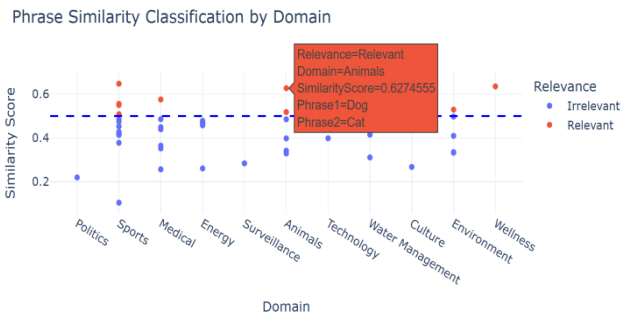

Fig 9: Phrases Comparison (X-axis) vs. Similarity Score (Y-axis)

The chart, shown in Fig 9, presents a visual representation of phrase similarity classification across

various domains. The X-axis represents different domains such as Politics, Sports, Medical, Energy, and others, while the Y-axis indicates the similarity scores of compared phrase pairs. Each point on the chart corresponds to a specific phrase comparison, with red dots indicating scores above the defined threshold of 0.6, classified as relevant, and blue dots representing scores below the threshold, classified as irrelevant.

In the highlighted example from the "Animals" domain, the phrases **"Dog"** and **"Cat"** are compared, resulting in a similarity score of **0.6274555**, which is above the threshold and thus marked as relevant.

In both Fig. 8 and Fig. 9, the analysis is presented on a single, interactive chart to provide a clear and consolidated view of all similarity values. This approach enables users to hover over the red and blue dots to view detailed information, including the similarity scores and associated comparison data. These interactive elements are part of the live plot functionality, and the corresponding details are documented in the project documentation.

Run	Document Key	Document To be Compared	Similarity Score	isPreProcessing Enabled
Run Without Pre Processing For One Sample	JobProfileCDeveloper.txt	JobRequirement.txt	0.70290482	FALSE
Run With Pre Processing For One	preprocessed_JobProfileCDeveloper.tx t	preprocessed_JobReq uirement.txt	0.70199654	TRUE

Fig 10: Observation From with and Without Pre-Processing

As shown in Fig 10, a comparison was made between semantic similarity scores of two documents *JobProfileCDeveloper.txt* (used as the source document contains sample information about random applicant resume) and *JobRequirement.txt* (used as the target document contains sample information about sample job posting) with and without preprocessing. When no preprocessing was applied, the similarity score was **0.70290482**, whereas with preprocessing enabled, the score was **0.70199654**. The similarity score is marginally higher in the run without preprocessing; however, the difference is minimal (approximately 0.0009). This indicates that, in this particular case, preprocessing had a negligible impact on the overall similarity result.

The test cases has been successfully implementated as illustrated by Fig 11, where it is demonstrated that all test cases have been executed, and unit testing has been completed with satisfactory code coverage. This ensures that the implemented methods for the system is robust ,hence a finely tuned semantic similarity analysis system.

The test method results confirms two very important aspects that the SaveResultsPhrase method in the CsvHelperUtil class correctly saves a list of PhraseSimilarity objects to the CSV file. It verifies that the CSV file is created and contains the correct number of records, while also ensuring the CSV file is created with the correct number of records. Additionally, the test checks whether the content of both files matches the original data. This validation is crucial for confirming the functionality of the SaveResultsPhrase method, ensuring it accurately writes data to the specified file formats. Such accuracy is essential for applications that rely on exporting data for further analysis or sharing.

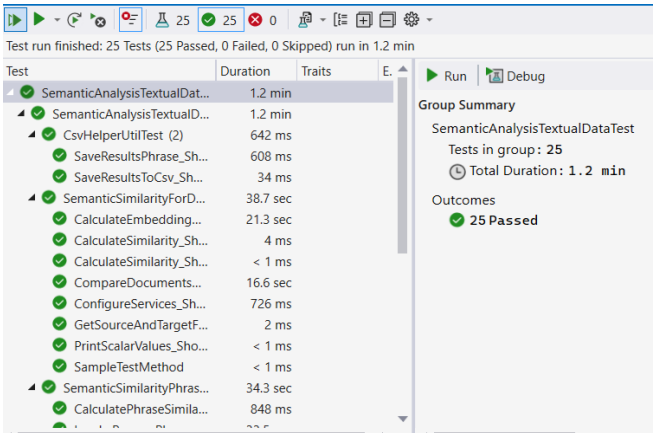

Fig 11: Representation of Test Run Results of All Test Cases

IV. Discussion

From the project, it is observed that preprocessing techniques like removing stop words, lemmatization, and eliminating articles can slightly affect the similarity scores of generated embeddings indicates efficient token usage when working with OpenAI's GPT model, making it a cost-effective by eliminating unwanted tokens. The observation also confirmed that OpenAI Embedding effectively achieves contextual alignment, and the results indicate that by applying specific improvements to preprocessing analysis across different use cases, better results can be achieved. The preprocessor module gain insight into the contextual variations caused by small changes in documents, even when those documents discuss the same topic or belong to the same domain.

The project also showcases by selecting an appropriate threshold value can significantly influence the effectiveness of the application in semantic analysis. For the analysis, the project uses a threshold value of **0.6** to differentiate between relevant and less relevant comparisons based on semantic similarity. The result of cosine similarity, if closer to 1 indicate stronger similarity. So, its evident that the threshold of 0.6 offers a balanced approach as it is sufficiently close to 1 to identify meaningful relationships while allowing for some flexibility in the analysis.

Currently, the placement of the output CSV file requires manual effort by developers or the application administrator. However, this limitation has been identified, and improvements are planned for later stages, either by the developers themselves or by providing opportunities for other developers to enhance the implementation.

In conclusion, the project demonstrates how OpenAI embeddings and similarity algorithms may be used practically to connect theory to real-world applications. By

simply switching datasets, it provides flexibility for use cases including job matching or university admissions by efficiently computing phrase and document similarities with understandable visualisations. Future enhancements, such automated CSV integration, wider dataset compatibility, and an easier-to-use interface, are also made possible by the framework. All things considered, it supports NLP applications and functions as a flexible instrument with significant room for future study and advancement.

This project serves as a foundational tool for multiple applications, including NLP tasks like text classification, sentiment analysis, and entity recognition by providing clean, preprocessed text. Its data cleaning capabilities enhance consistency and quality in data science workflows. It can also support SEO by optimizing content for search engines, integrate with CMS platforms to ensure high-quality publishing, and aid academic research by preparing textual data for studies such as linguistic or social media analysis.

V. REFERENCES

[1] A. Aboelghit and T. Hamza, "Textual Similarity Measurement Approaches: A Survey," The Egyptian Journal of Language Engineering, vol. 10, 2020, doi: :10.21608/ejle.2020.42018.1012

[2] G. Majumder, P. Pakray, A. Gelbukh, and D. Pinto, "Semantic Textual Similarity Methods, Tools, and Applications: A Survey," Computacion y Sistemas, vol. 20, pp. 647–665, 2016, doi:10.13053/CyS-20-4-2506

[3] P. Hastings, "Latent Semantic Analysis," 2004. Available: https://www.researchgate.net/publication/230854757_Latent_Semantic_Analysis

[4] E. Gabrilovich and S. Markovitch, "Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis," in Proceedings of the IJCAI International Joint Conference on Artificial Intelligence, vol. 6, 2007. Available: https://www.researchgate.net/publication/200042392_Computing_Semantic_Relatedness_using_Wikipedia-based_Explicit_Semantic_Analysis

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," in Proceedings of the 1st International Conference on Learning Representations, *ICLR 2013, Workshop Track Proceedings*, Scottsdale, Arizona, USA, May 2-4, 2013. Available: https://www.researchgate.net/publication/234131319_Efficient_Estimation_of_Word_Representations_in_Vector_Space

[6] J. Pennington, R. Socher, and C. Manning, "GloVe: Global Vectors for Word Representation," in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 2014, pp. 1532–1543, Association for Computational Linguistics,doi: 10.3115/v1/D14-1162

[7] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Hong Kong, China, 2019, pp. 3982–3992, Association for Computational Linguistics, doi: 10.18653/v1/D19-1410

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota, 2019, pp. 4171–4186, Association for Computational Linguistics, doi: 10.18653/v1/N19-1423

[9] OpenAI, "New Embedding Models and API Updates," 2024[Online]. Available:https://openai.com/index/new-embedding-models-and-api-updates/.

[10] Microsoft, "Vector embeddings in Azure Cosmos DB," *Microsoft Learn*, Dec. 3, 2024[Online]. Available: https://learn.microsoft.com/en-us/azure/cosmos-db/gen-ai/vector-embeddings.

[11] OpenAI, "Introducing text and code embeddings," *OpenAI*, Jan. 25, 2022[Online]. Available: https://openai.com/index/introducing-text-and-code-embeddings

[12] D. P. Yadav, N. K. Kumar, and S. K. Sahani, "Distance Metrics for Machine Learning and its Relation with Other Distances," *Mikailalsys Journal of Mathematics and Statistics*, vol. 1, no. 1, pp. 15–23, 2023. Available: https://doi.org/10.58578/mjms.v1i1.1990.