

Mini-Project Report On

PROJECT SEARCH ASSISTANT

*Submitted in partial fulfillment of the requirements for the
award of the degree of*

Bachelor of Technology

in

Computer Science & Engineering

By

Aleena Siby (U2003024)

Bibit Sebastian (U2003055)

Celestian Ben Mathew (U2003058)

Christo Mathew (U2003060)

Under the guidance of

Ms. Seema Safar



**Department of Computer Science & Engineering
Rajagiri School of Engineering and Technology (Autonomous)
Rajagiri Valley, Kakkanad, Kochi, 682039**

July 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RAJAGIRI SCHOOL OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS)
RAJAGIRI VALLEY, KAKKANAD, KOCHI, 682039



CERTIFICATE

*This is to certify that the mini-project report entitled "**Project Search Assistant**" is a bonafide work done by **Aleena Siby (U2003024)**, **Bibit Sebastian (U2003055)**, **Celestian Ben Mathew (U2003058)**, **Christo Mathew (U2003060)**, submitted to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (B. Tech.) in Computer Science and Engineering during the academic year 2022-2023.*

Dr. Preetha K. G.
Head of Department
Dept. of CSE
RSET

Dr. Sminu Izudheen
Mini-Project Coordinator
Asst. Professor
Dept. of CSE
RSET

Ms. Seema Safar
Mini-Project Guide
Asst. Professor
Dept. of CSE
RSET

ACKNOWLEDGEMENTS

We wish to express our sincere gratitude towards **Dr. P. S. Sreejith**, Principal of RSET, and **Dr. Preetha K. G.**, Head of Department of Computer Science and Engineering for providing us with the opportunity to undertake our mini-project, "Project Search Assistant".

We are highly indebted to our mini-project coordinators, **Dr. Sminu Izudheen**, Assistant Professor, Department of Computer Science and Engineering for their valuable support.

It is indeed our pleasure and a moment of satisfaction for us to express our sincere gratitude to our mini-project guide **Ms. Seema Safar**, for her patience and all the priceless advice and wisdom she has shared with us.

Last but not the least, we would like to express our sincere gratitude towards all other teachers and friends for their continuous support and constructive ideas.

Aleena Siby

Bibit Sebastian

Celestian Ben Mathew

Christo Mathew

ABSTRACT

The Search Assistant for Document Similarity Comparison is an intelligent tool designed to assist users in comparing the similarity between the user input abstract and already existing projects. Leveraging Natural Language Processing (NLP) techniques, the system preprocesses the input documents by removing stopwords, lemmatizing words, and extracting relevant features. It then utilizes Term Frequency-Inverse Document Frequency (TF-IDF) vectorization to enhance the accuracy of semantic similarity analysis. The system calculates similarity scores using Euclidean Distance to provide users with comprehensive insights into the degree of similarity between the documents. This enables users to efficiently and effectively assess the similarity between projects, making it useful for applications such as plagiarism detection, content retrieval, and document clustering.

Contents

Acknowledgements	ii
Abstract	iii
List of Figures	vi
1 Introduction	1
1.1 Background	1
1.2 Existing System	1
1.3 Problem Statement	2
1.4 Objectives	2
1.5 Scope of the project	2
2 Literature Review	4
2.1 TF-IDF from scratch in python on a real-world dataset[8]	4
2.2 Automatic Exam Correction Framework (AECF) for the MCQs, Essays, and Equations Matching [3]	5
3 System Analysis	8
3.1 Feasibility Analysis	8
3.1.1 Technical Feasibility	8
3.1.2 Operational Feasibility	8
3.2 Hardware Requirements	8
3.3 Software Requirements	8
3.3.1 NLTK (Natural Language Toolkit)	9
3.3.2 Gensim	9
3.3.3 NumPy	9

4	System Design	11
4.1	Architecture Diagram	11
4.2	Sequence diagram	12
5	System Implementation	13
5.1	Module Division	13
5.1.1	Pre processing Module	13
5.1.2	Similarity Calculation Module	15
5.2	Source Code	17
6	Testing	22
7	Results	28
8	Risks and Challenges	32
9	Conclusion	33
	Bibliography	33
	Appendix A: Base Paper	34
	Appendix B: Sample Code	42
	Appendix C: CO-PO and CO-PSO Mapping	47

List of Figures

2.1	Semantic Text Matching. Adapted from [3].	6
4.1	Architecture diagram	11
4.2	Sequence Diagram	12
6.1	Comparison table for similarity scores & aggregate scores	24
6.2	Mean absolute error graph	25
6.3	Root mean square error graph	26
6.4	Mean square error graph	26
7.1	Main Menu interface	28
7.2	Input abstract screen	29
7.3	Result display screen	30
7.4	Project details screen	31

Chapter 1

Introduction

1.1 Background

In academic institutions like our college, students and teachers continually engage in research and project work. However, finding relevant projects or related research for inspiration, collaboration, or building upon existing knowledge can be a challenging task. Traditionally, researchers have relied on keyword-based search engines, which often yield imprecise results due to differences in terminology and language variations. As a consequence, valuable insights, potential collaborators, and opportunities to build upon prior research may be missed. To address this issue, we have developed the Project Search Assistant, a specialized tool tailored to our college's academic environment. The Project Search Assistant aims to streamline the process of discovering similar projects completed by students within the college by accepting project abstracts as input, the system preprocesses and vectorizes the data using Tf-idf representation. Leveraging similarity measures such as Euclidean distance, it identifies projects that share significant thematic or contextual similarities with the given abstract. The search assistant empowers both students and teachers to efficiently explore a curated list of contextually relevant projects, facilitating knowledge sharing, and collaboration, and fostering a vibrant research community within the college.

1.2 Existing System

Manual project comparison: In the existing system, document similarity analysis is often limited by traditional text-matching techniques that fail to capture the semantic meaning and context of text documents accurately. The traditional approaches do not account for the significance of individual words in the context of the entire corpus, and they may struggle to handle synonymous words or variations in word forms, leading to subopti-

mal similarity measurements. Moreover, these methods may not adequately address the issue of information overload, hindering efficient content retrieval and decision-making processes.

Additionally, manual document comparison in the existing system can be time-consuming and impractical, especially when dealing with large volumes of textual data. As a result, researchers and professionals may face challenges in identifying relevant information, detecting plagiarism, or organizing content effectively.

1.3 Problem Statement

The challenge is to create a web application that would assist teachers and students in efficient project exploration. The lack of an automated system for comparing project abstracts hinders efficient project exploration, collaboration, and decision-making. The Project Search Assistant aims to address these challenges by developing an intelligent system that advanced natural language processing to analyze project abstracts, understand the context, and perform similarity analysis to identify projects that closely align with the given input.

1.4 Objectives

The primary objective of our project, the Project Search Assistant, is to facilitate the efficient and accurate discovery of similar projects completed by students within our college's academic environment. The tool aims to simplify the process of finding contextually related projects, enabling both students and teachers to access a curated list of relevant research work. The system's purpose extends beyond simple text comparison, as it is also capable of detecting plagiarism and duplicate content, further enhancing its utility.

1.5 Scope of the project

The project aims to develop a document comparison system utilizing TF-IDF vectorization and NLP techniques. The primary objective is to enable users to compare and contrast different documents by determining their semantic similarity. Advanced text preprocessing techniques like tokenization, lemmatization, and stop-word removal will be employed to enhance data quality. Text data will be transformed into numerical vectors

using TF-IDF vectorization, representing word significance in each document relative to the entire corpus. The system will utilize Euclidean distance as the similarity metric to assess document comparisons. An intuitive user interface will be designed, and performance optimization will ensure efficient handling of large volumes of documents. It is essential to acknowledge that the project's scope may evolve over time based on user feedback, feasibility, and resource availability

Chapter 2

Literature Review

2.1 TF-IDF from scratch in python on a real-world dataset[8]

Author :William Scott

The paper discusses TF-IDF, a technique used to quantify the importance of words in a set of documents. TF-IDF is widely used in Information Retrieval and Text Mining. The paper explains how textual data is typically represented numerically for programming languages to interpret it. Vectorizing documents allows various tasks such as finding relevant documents, ranking, and clustering.

TF-IDF is calculated using the formula:

$$\text{TF-IDF} = \text{TermFrequency}(TF) \times \text{InverseDocumentFrequency}(IDF)$$

Term Frequency measures the frequency of a word in a document, while Document Frequency measures the importance of a word in the whole set of documents. IDF is the inverse of Document Frequency and quantifies the informativeness of a term.

To calculate TF-IDF, the paper describes the process of preprocessing the text data, including converting to lowercase, removing punctuation and stop words, stemming, and converting numbers to words. DF (Document Frequency) is calculated to determine the count of unique documents in which a word appears. To differentiate the importance of words in the document title and body, the paper introduces the concept of different weights (alpha and 1-alpha) for the title and body respectively. The TF-IDF of the title and body is then combined to represent the document's overall TF-IDF score.

The paper also explains how to rank documents using both the Matching Score and Cosine Similarity methods. While Matching Score computes the similarity based on token occurrences, Cosine Similarity considers the angle between document and query vectors,

making it better suited for long queries. The document vectorization process is outlined using numpy arrays to represent the TF-IDF values for both the documents and queries. Finally, the paper presents an analysis comparing Matching Score and Cosine Similarity methods for a sample query and document, demonstrating the superior performance of Cosine Similarity in learning the context and providing better relevance rankings.

2.2 Automatic Exam Correction Framework (AECF) for the MCQs, Essays, and Equations Matching [3]

In this research paper, the authors propose an automated system for measuring similarity between software requirements, which has become increasingly necessary with the rise of online courses (MOOCs) where automatic grading is essential. The system focuses on three main aspects: semantic text matching, multiple-choice questions (MCQs) matching, and equations (expressions) matching.

1. **Semantic Text Matching:** Semantic text matching is the process of finding the similarity percentage between two texts semantically. They pass through text pre-processing, tokenization (i.e. the text can be used as a whole or tokenized into words), feature extraction, and the semantic similarity score is calculated between them. The process is summarized in Figure 1 and discussed below.

Semantic text matching involves finding the similarity percentage between two texts based on their semantic meaning. The process includes text pre-processing, tokenization, and feature extraction. Pre-processing includes steps such as removing whitespaces, converting text to lowercase, removing accents, removing punctuations, removing stop-words, and applying lemmatization. Tokenization breaks the text into components like words and sub-words. Feature extraction involves converting the textual data into numerical feature vectors using techniques like TF-IDF and word embedding. The TF-IDF technique represents the importance of words in a document, while word embedding uses mathematical descriptions of individual words to capture their context.

2. **Calculating Semantic Similarity:** Semantic similarity is a metric that measures the similarity of meaning or content between documents or terms. The paper mentions several methods for calculating semantic similarity, such as Cosine similarity,

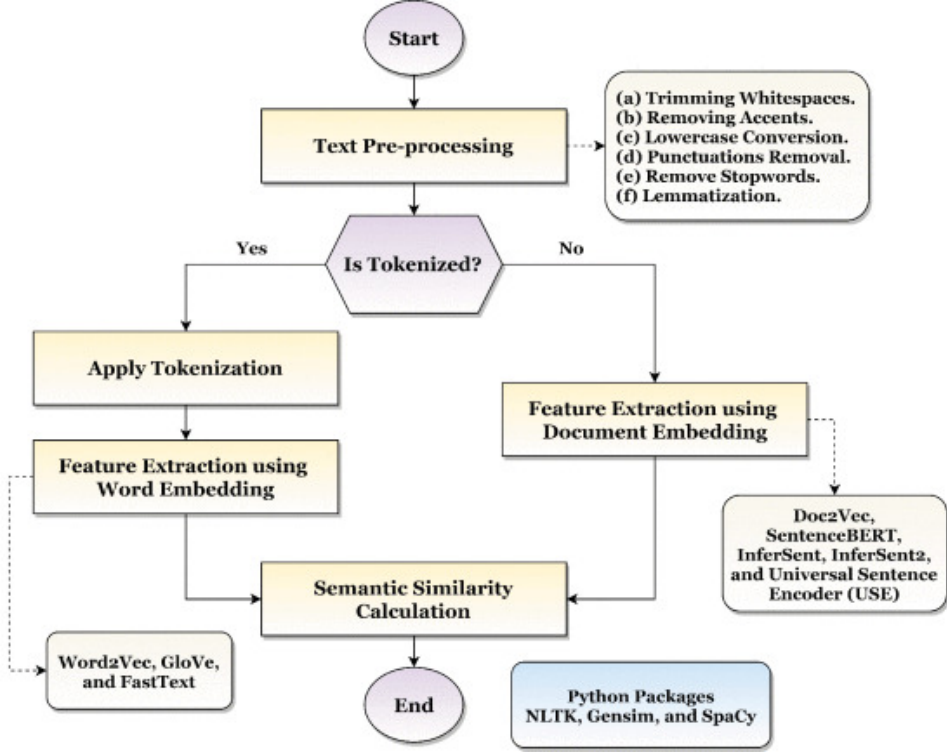


Figure 2.1: Semantic Text Matching. Adapted from [3].

Jaccard similarity, Euclidean distance, and Manhattan distance. Cosine similarity measures the angle between two vectors projected in a multi-dimensional space, while Jaccard similarity is used for comparing binary vectors or sets. Euclidean distance represents the straight-line distance between two vectors, and Manhattan distance is used when straight-line distance is not available.

$$\begin{aligned}
 \text{Similarity}_{\text{Cosine}} &= \frac{A \bullet B}{\|A\| \bullet \|B\|} \\
 \text{Similarity}_{\text{Jaccard}} &= \frac{|A \cap B|}{|A \cup B|} \\
 &= \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \\
 \text{Distance}_{\text{Euclidean}} &= \sqrt{\sum_{i=1}^n (A_i - B_i)^2} \\
 \text{Distance}_{\text{Manhattan}} &= \sum_{i=1}^n |A_i - B_i|
 \end{aligned}$$

This paper also discusses matching techniques for multiple-choice questions (MCQs) and equations. For MCQs, correction is straightforward and depends on the question type. Similarity between student and reference answers determines the marks

awarded. Equations are represented in infix, prefix, or postfix notation. Expression trees, binary trees with operators and operands, are used for equations matching. The infix to prefix conversion algorithm is summarized, which plays a key role in the proposed equations matching algorithm.

Overall, the paper proposes an approach for automated grading based on semantic text matching, MCQs matching, and equations matching. The semantic similarity calculation plays a vital role in determining the similarity between texts, facilitating the automated grading process. The research paper uses various techniques and mathematical measures to ensure accurate and efficient similarity assessment, making it suitable for grading in online courses where automatic grading systems are crucial.

H. M. Balaha and M. M. Saafan, "Automatic Exam Correction Framework (AECF) for the MCQs, Essays, and Equations Matching," in *IEEE Access*, vol. 9, pp. 32368-32389, 2021, doi: 10.1109/ACCESS.2021.3060940.

Chapter 3

System Analysis

3.1 Feasibility Analysis

3.1.1 Technical Feasibility

The project is technically feasible, as it leverages widely used technologies and can be accessed through standard web browsers, ensuring a large user base. The system's lightweight design and compatibility with various devices make it easily accessible on smartphones and computers. Additionally, cloud-based hosting ensures seamless scalability to accommodate potential growth in users and data volume.

3.1.2 Operational Feasibility

The project is operationally feasible due to its streamlined workflow and user-friendly interface, allowing users to quickly adapt to the system. The search assistant's implementation is straightforward and does not require specialized training or technical expertise, making it accessible to a wide range of users.

3.2 Hardware Requirements

The following are the system requirements to develop the Project Search Assistant.

- Processor: Intel Core i5
- Hard Disk: Minimum 100GB
- RAM: Minimum 8GB

3.3 Software Requirements

The following are the softwares used in the development of the Project Search Assistant.

Operating System: Windows or Linux

3.3.1 NLTK (Natural Language Toolkit)

The Natural Language Toolkit (NLTK) is a powerful Python library widely used for various natural language processing (NLP) tasks. NLTK is employed to preprocess the text data and prepare it for further analysis. The library facilitates tokenization, which breaks the text into individual words or tokens, allowing for better understanding and manipulation of the language data. Additionally, NLTK's lemmatization functionality is utilized to reduce words to their base or root forms, while stemming helps in reducing words to their base stem forms. Furthermore, NLTK provides a set of common stopwords that are filtered out from the text to remove noise and focus on meaningful content. By incorporating these preprocessing techniques, the code ensures that the text data is normalized and ready for semantic similarity analysis.

3.3.2 Gensim

Gensim is another essential Python library utilized in the code for two main purposes: TF-IDF vectorization and Euclidean distance calculation. Gensim is employed to convert text data into numerical vectors using the Term Frequency-Inverse Document Frequency (TF-IDF) approach. It creates a corpus of preprocessed data and a dictionary to map words to numerical indices. By calculating Euclidean distances between query and document vectors, Gensim enables efficient semantic similarity analysis, providing users with relevant search results and an effective search assistant.

3.3.3 NumPy

NumPy serves as a fundamental library for performing efficient numerical computations. The main advantage of using NumPy lies in its ability to handle large arrays and matrices, making it well-suited for tasks involving data manipulation and numerical computations. It is primarily used for calculating the Euclidean distance between the TF-IDF vectors of the user's abstract and existing projects in the database. Its array operations ensure swift and accurate similarity measurements. Additionally, NumPy aids in converting the TF-IDF vectors into dense numerical arrays, facilitating seamless compatibility and

efficient calculations. Overall, NumPy plays a crucial role in enhancing the performance and scalability of the similarity analysis process in this project. [4]

Chapter 4

System Design

4.1 Architecture Diagram

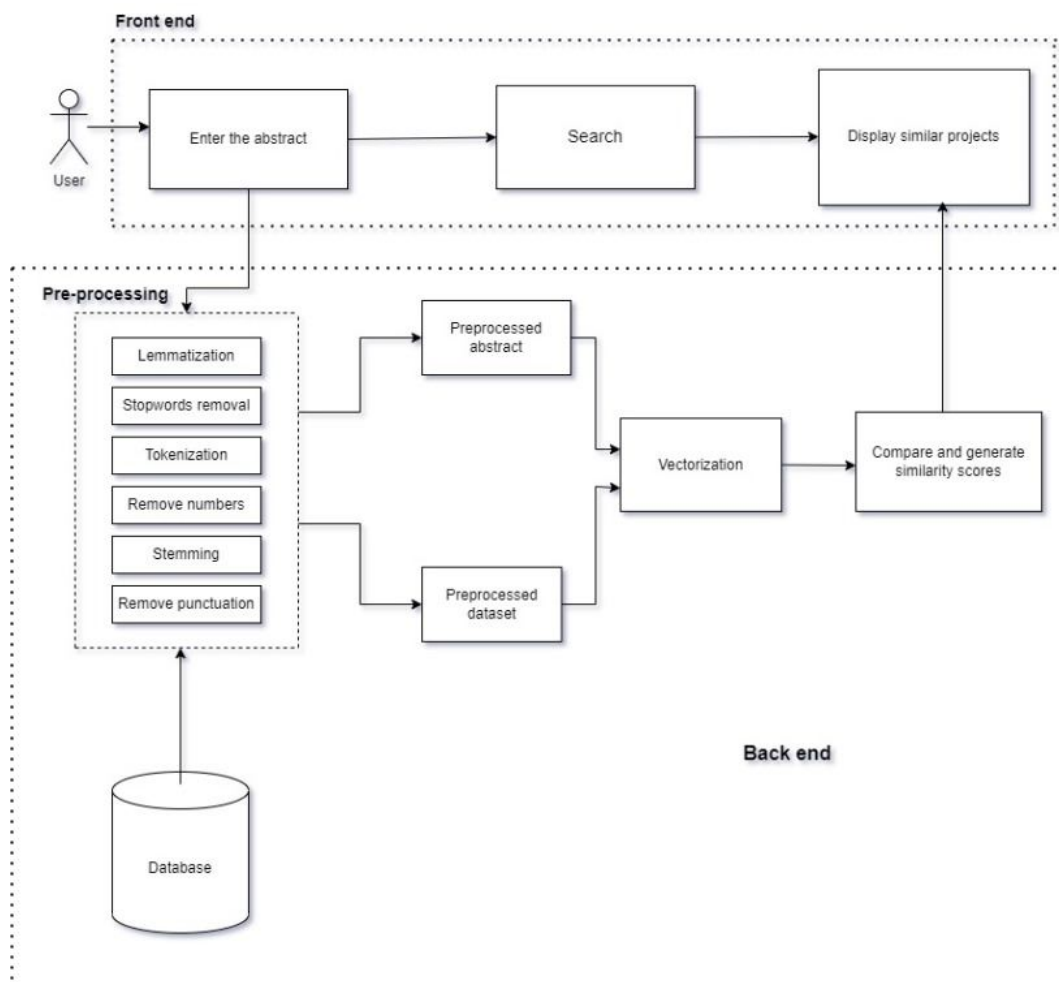


Figure 4.1: Architecture diagram

4.2 Sequence diagram

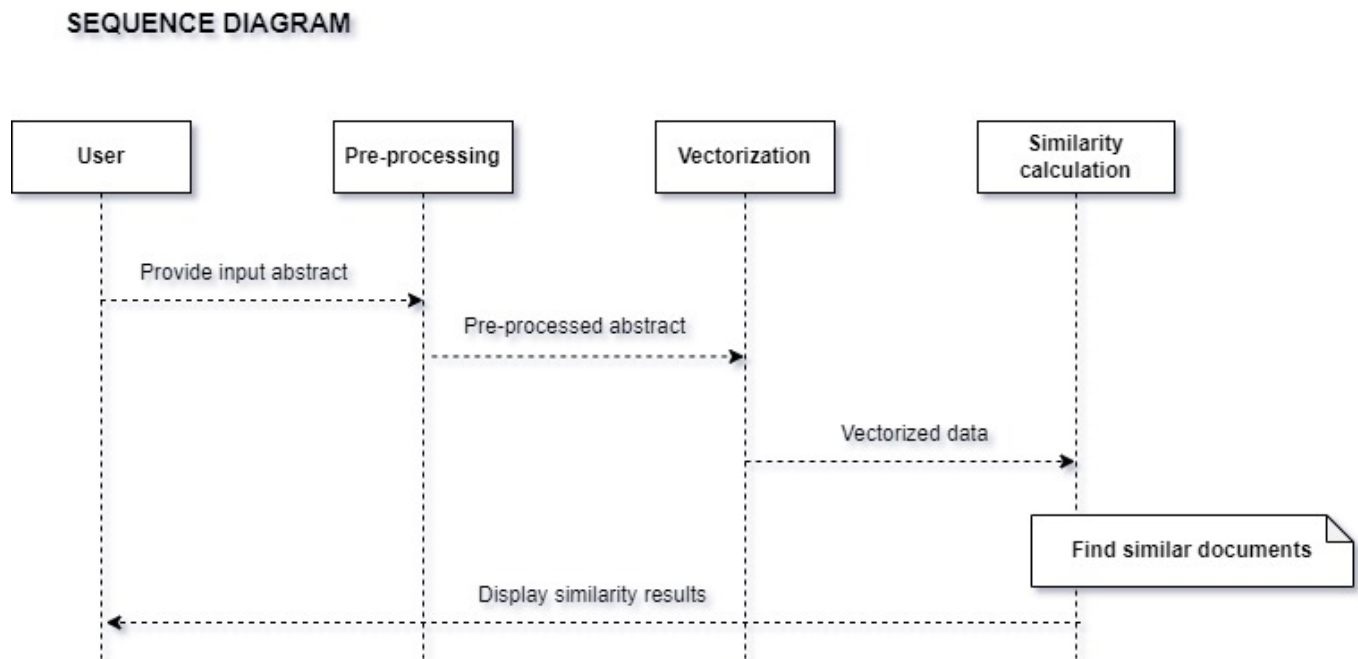


Figure 4.2: Sequence Diagram

Chapter 5

System Implementation

5.1 Module Division

5.1.1 Pre processing Module

The preprocessing begins with data cleaning and standardization to eliminate inconsistencies and ensure a uniform representation of the text data.

- **Lowercasing:** The preprocessing module begins by converting all the text to lowercase. This is done by utilizing Python's built-in string module, Specifically, the `str.lower()` function is applied to the input text, requiring no explicit parameters. This function converts the text to lowercase, ensuring uniformity in letter casing throughout the document.
- **Punctuation Removal:** Any punctuation marks, such as commas, periods, or special characters, are removed from the abstracts. Punctuation is generally not essential for semantic analysis, and removing it streamlines the processing of the text. For this task, we utilize Python's `str.translate()` function along with `str.maketrans()` to remove punctuation marks from the text. The `str.maketrans()` function creates a translation table, mapping punctuation characters to `None`. Subsequently, the `str.translate()` function applies this table to eliminate punctuation from the text, simplifying it for further analysis. This step ensures that the Project Search Assistant focuses on meaningful content while disregarding non-essential punctuation characters, enhancing the accuracy of document comparison.
- **Number Removal:** Number removal is aimed at eliminating numerical digits from the text data. This process is crucial because numerical digits, such as integers and decimals, may not contribute to the overall meaning and semantics of the sentences.

By using the regular expression `re.sub(r'+' , "", text)`, we can effectively remove all occurrences of numerical digits from the text. By utilizing this preprocessing step, our project ensures that the text data is optimized for semantic similarity analysis, allowing the subsequent stages to concentrate on capturing the true meaning and semantics of the sentences.

- **Tokenization:** Tokenization refers to the process of breaking down a sentence or text into individual words or tokens. It is a fundamental step in natural language processing that facilitates the analysis of text data on a word-by-word basis. The NLTK library is used for tokenization in this code. Tokenization helps in creating a more meaningful representation of the text, allowing for efficient and accurate processing in subsequent stages.
- **Stopwords Removal:** Stopwords removal filters out common words, such as articles and prepositions, that do not contribute significantly to the meaning of the text. It helps improve efficiency and accuracy by reducing noise and focusing on more meaningful words. The NLTK library is used to access a predefined list of stopwords in English. During preprocessing, the code tokenizes the sentences and then filters out any word tokens that match the words present in the stopwords list. This process effectively removes the stopwords from the text data, leaving behind only the relevant and informative words for further analysis.
- **Lemmatization:** Lemmatization involves reducing words to their base or root form, known as the lemma. The purpose of lemmatization is to group together different inflected forms of a word so that they can be analyzed as a single entity. This process helps in simplifying the text data and extracting the core meaning of words. `WordNetLemmatizer` from the NLTK (Natural Language Toolkit) library is used for lemmatization[4]. During the preprocessing step, each word token in the sentences is lemmatized using the `WordNetLemmatizer`. It first identifies the part of speech (POS) tag for each word, such as noun, verb, adjective, or adverb. Then, the lemmatizer reduces the word to its base form based on its POS tag. By doing so, it consolidates variations of the same word into a single representation, improving the accuracy and consistency of semantic similarity analysis.

- **Synonym Expansion:** The synonym expansion process is performed to enhance the text data by including synonyms of the words present in the sentences. This process helps to increase the coverage of words and capture more variations of meaning, thereby improving the overall analysis of text data. The synonym expansion is achieved using the WordNet module from the Natural Language Toolkit (NLTK) library. WordNet is a lexical database for the English language that organizes words into sets of synonyms called "synsets," which represent different senses of a word. The NLTK library provides easy access to WordNet, allowing the code to find synonyms for each token in the sentences. During the preprocessing, each token in the sentences is checked for synonyms using WordNet's synsets. If synonyms are found for a token, they are added to the token list. This process helps to enrich the vocabulary used in the text data, enabling the model to capture more nuanced and varied meanings of words, which can be particularly beneficial in semantic similarity analysis and other natural language processing tasks.

5.1.2 Similarity Calculation Module

The similarity calculation module quantifies semantic similarity using the TF-IDF vectorization technique, which assigns weights to words based on their importance in the context of the entire corpus. Once the user input abstract and the projects stored in the database are converted into TF-IDF vectors, the module uses the Euclidean distance metric measures similarity between them, with smaller distances indicating higher similarity. The top-ranked documents are presented as the most relevant matches to the input abstract, providing an efficient and effective search assistant tool.

Vectorization

Vectorization is a critical step in natural language processing (NLP) that converts textual data into numerical vectors, enabling machine learning algorithms to process and analyze text effectively.[5] Term Frequency-Inverse Document Frequency (TF-IDF) technique, is a popular method in information retrieval and document analysis. It quantifies the importance of each word in a document relative to a collection of documents (corpus). Tf-idf is calculated using the term frequency (tf), representing the frequency of a word in a document, and the inverse document frequency (idf), measuring the rarity of a

word across the entire corpus. The tf-idf score for a term in a document is the product of its tf and idf. The resulting tf-idf vectors provide a numerical representation of each document. In our project, tf-idf vectorization is applied to both the user's input project abstract and the preprocessed project abstracts stored in the database. The preprocessed input abstract and each document are transformed into TF-IDF vectors using Gensim's models.TfidfModel .This allows for efficient similarity analysis and ranking, aiding the identification of the most contextually similar projects to the user's input abstract. The tf-idf score for a term (t) in a document (d) can be calculated as follows:

$$\text{ValueTFIDF} = TF(t, d) \times IDF(t, D) = TF(t, d) \times \log(N / \{d \in D : t \in D\})$$

where t is the term, d is the current document, D is the documents in the corpus, N is the number of total documents in the corpus, and $\{d \in D : t \in D\}$ is the count of documents in the corpus that contains the term t .

- After calculating the tf-idf scores for all terms in each document, a tf-idf vector is created for each document.
- The tf-idf vector is a numerical representation of the document, with each element in the vector corresponding to the tf-idf score of a specific term.
- The length of the tf-idf vector is equal to the total number of unique terms in the entire corp.

Similarity Score Calculation

The similarity score[2] between the input abstract and each document in the database is calculated using a method called Euclidean distance. Euclidean distance is a way of measuring how similar or dissimilar two points are in a multidimensional space. In this case, the "points" are the numerical representations of the input abstract and each document. The Euclidean distance quantifies how far apart these points are in the space, and a smaller distance means they are more similar. Mathematically, the Euclidean distance between two vectors A and B in an n-dimensional space is calculated using the formula:

$$\text{EuclideanDistance} = \sqrt{(A[1] - B[1])^2 + (A[2] - B[2])^2 + \dots + (A[n] - B[n])^2}$$

The result is a single scalar value that represents the distance or dissimilarity between the two vectors. The Euclidean distance provides a numerical value, but to express simi-

larity, we convert it into a similarity score. The similarity score is calculated by inverting the distance using a simple formula: $\text{similarity} = 1 / (1 + \text{distance})$. This transformation scales the score between 0 and 1, where a score closer to 1 indicates a higher level of similarity. So, the closer the similarity score is to 1, the more similar the document is to the input abstract.

After calculating the similarity scores for each document in the database based on the input abstract, the documents are ranked in ascending order of similarity. The ranking ensures that the most similar and relevant documents appear at the top of the list, making it easy for users to find the most pertinent information quickly. To assist users in exploring the most relevant documents, the top-ranked documents are displayed to the users.

5.2 Source Code

```
1 import os
2 import pandas as pd
3 import string
4 import re
5 import numpy as np
6 from nltk.stem import WordNetLemmatizer, PorterStemmer
7 from nltk.corpus import stopwords, wordnet
8 from nltk.tokenize import word_tokenize
9 from gensim import corpora, models
10 import gensim
11 from api.index import db
12 import time
13
14 def compareData(abstract):
15     begin = time.time()
16
17     def preprocess_data(contents):
18         stemmer = PorterStemmer()
19         wordnet.ensure_loaded()
```



```

20     stop_words = set(stopwords.words('english'))
21     preprocessed_data = []
22     for content in contents:
23         lemmatizer = WordNetLemmatizer() # Define
24         lemmatizer here
25         sentence = content[1].lower()
26         sentence = re.sub(r'\d+', '', sentence) # Remove
27         numbers
28         sentence = sentence.translate(str.maketrans('', '',
29 string.punctuation)) # Remove punctuation
30         tokens = word_tokenize(sentence)
31         tokens = [stemmer.stem(token) for token in tokens if
32 token not in stop_words and token.isalpha()]
33         lemmatized_synonyms = []
34         for token in tokens:
35             synsets = wordnet.synsets(token)
36             lemmas = [lemma for synset in synsets for lemma
37 in synset.lemmas()]
38             if lemmas:
39                 lemmatized_synonyms.extend([lemmatizer.lemmatize(lemma.name())
40 for lemma in lemmas])
41         tokens.extend(lemmatized_synonyms)
42         preprocessed_data.append(tokens)
43         data_time = time.time()
44         print(f"preprocess time2: {data_time-begin}")
45         return preprocessed_data
46
47 def preprocess_query(query):
48     stemmer = PorterStemmer()
49     stop_words = set(stopwords.words('english'))
50     lemmatizer = WordNetLemmatizer() # Define lemmatizer
51     here
52     query = query.lower()

```

```

46     query = re.sub(r'\d+', '', query) # Remove numbers
47     query = query.translate(str.maketrans('', '',
string.punctuation)) # Remove punctuation
48     tokens = word_tokenize(query)
49     tokens = [stemmer.stem(token) for token in tokens if
token not in stop_words and token.isalpha()]
50     lemmatized_synonyms = []
51     for token in tokens:
52         synsets = wordnet.synsets(token)
53         lemmas = [lemma for synset in synsets for lemma in
synset.lemmas()]
54         if lemmas:
55
56     lemmatized_synonyms.extend([lemmatizer.lemmatize(lemma.name())
for lemma in lemmas])
57     tokens.extend(lemmatized_synonyms)
58     print("query time", time.time()-begin)
59     return tokens
60
61 def calculate_euclidean_distance(query, documents):
62     dictionary = corpora.Dictionary(documents)
63     corpus = [dictionary.doc2bow(doc) for doc in documents]
64
65     model = models.TfidfModel(corpus)
66     tfidf_corpus = model[corpus]
67
68     # Convert the query to a tf-idf vector
69     query_bow = dictionary.doc2bow(query)
70     query_tfidf = model[query_bow]
71
72     # Create a matrix of tf-idf vectors for all documents
73     document_matrix =
gensim.matutils.corpus2dense(tfidf_corpus,
num_terms=len(dictionary)).T

```

```

73
74     # Convert the query and document vectors to numpy arrays
75     query_vector =
gensim.matutils.corpus2dense([query_tfidf],
num_terms=len(dictionary)).T
76
77     # Calculate Euclidean distance between the query vector
and each document vector
78     euclidean_distances = np.linalg.norm(document_matrix -
query_vector, axis=1)
79
80     return euclidean_distances
81
82     def read_data():
83         contents = [(int(obj["id"]),obj["sentence"]) for obj in
db.Projects.find()]
84         return contents
85
86     # Read data from mongo
87     data = read_data()
88     print(data[:10])
89     end_read = time.time()
90     # Preprocess data
91     preprocessed_data = preprocess_data(data)
92
93     # Download Word2Vec model
94     # word2vec_model = api.load("word2vec-google-news-300")
95
96     # # Save Word2Vec model
97     # word2vec_model.save(path_word2vec_model)
98
99     # Preprocess query
100     preprocessed_query = preprocess_query(abstract)
101

```

```

102     # Calculate similarity using Euclidean distance
103     query_distances =
calculate_euclidean_distance(preprocessed_query,
preprocessed_data)
104
105     end_eu = time.time()
106     # Rank documents
107     ranked_docs = sorted(enumerate(query_distances, start=1),
key=lambda x: x[1])
108
109     result_docs=[]
110     # Display top-ranked similar documents
111     for doc_id, distance in ranked_docs[:10]:
112         original_doc_id = data[doc_id - 1][0]
113         similarity = 1 / (1 + distance) # Convert distance to
similarity (values closer to 1 are more similar)
114         sim_score= f'{similarity*100:.2f}%'
115         print(f"Doc {original_doc_id}: Similarity
{similarity*100:.2f}%")
116         result_docs.append((original_doc_id,sim_score))
117
118     end = time.time()
119     print(f"read: {end_read-begin}")
120     print(f"eu: {end_eu-begin}")
121     print(f"end: {end-begin}")
122
123     return result_docs

```

Chapter 6

Testing

The similarity calculation phase plays a pivotal role in our project, where we focus on enhancing the accuracy of semantic similarity analysis. In this phase, we aim to develop robust algorithms that can effectively compare and measure the similarity between sentences or text data. The ability to accurately gauge the semantic similarity between sentences is crucial in our project.

To achieve this, we have incorporated several state-of-the-art similarity methods, including Pearson correlation coefficient, Jaccard similarity, Manhattan similarity, Cosine similarity, Euclidean similarity, and a novel combined measure of Cosine and Euclidean similarity. Each method offers unique insights into the comparison of sentences, capturing different aspects of similarity, such as direction, magnitude, and word overlap.

The methods used were:

1. **Pearson Correlation Coefficient:** This method measures the linear correlation between two vectors, providing a value between -1 and 1. [6] A positive value indicates a positive correlation, while a negative value suggests an inverse correlation. A score of 0 implies no correlation.

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

2. **Jaccard Similarity:** Jaccard similarity calculates the intersection over union of two sets. It is particularly useful for comparing text data,[1] treating sentences as sets of words. The score ranges from 0 to 1, with 1 indicating identical sets.

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

3. **Manhattan Similarity:** Also known as City Block Distance, this method calculates the sum of the absolute differences between corresponding elements of two vectors.

It measures the distance between two points along the axes at right angles. Lower values indicate greater similarity.

$$D(A, B) = \sum_{i=1}^n |A_i - B_i|$$

Where: - $D(A, B)$ represents the Manhattan distance between vectors A and B . - $|A_i - B_i|$ calculates the absolute difference between the i -th element of vector A and the i -th element of vector B . - n is the number of elements in the vectors A and B , which corresponds to the dimensions of the vectors.

4. Cosine Similarity: Cosine similarity calculates the cosine of the angle between two non-zero vectors. It measures the similarity in direction rather than magnitude. The score ranges from -1 to 1, with 1 indicating perfect similarity[7]. The cosine similarity is a measure of similarity between two vectors in a multi-dimensional space. For two n -dimensional vectors A and B , the cosine similarity $\text{cosine}(A, B)$ is calculated as the cosine of the angle between the two vectors. Mathematically, it is represented as:

$$\text{cosine}(A, B) = \frac{A \cdot B}{\|A\| \|B\|}$$

Where: - $\text{cosine}(A, B)$ represents the cosine similarity between vectors A and B . - $A \cdot B$ denotes the dot product of vectors A and B . - $\|A\|$ and $\|B\|$ represent the Euclidean norms (magnitudes) of vectors A and B respectively.

5. Euclidean Similarity: Euclidean similarity computes the Euclidean distance between two vectors in n -dimensional space. It calculates the straight-line distance between two points. Smaller distances indicate higher similarity.

The Euclidean distance is a measure of distance between two points in a coordinate system. For two n -dimensional vectors A and B , the Euclidean distance $D(A, B)$ is computed as the square root of the sum of the squared differences between their corresponding elements. Mathematically, it is represented as:

$$D(A, B) = \sqrt{\sum_{i=1}^n (A_i - B_i)^2}$$

Where: - $D(A, B)$ represents the Euclidean distance between vectors A and B. - A_i and B_i are the i -th elements of vectors A and B, respectively. - n is the number of elements in the vectors A and B, which corresponds to the dimensions of the vectors.

6. Combined Cosine and Euclidean Similarity: We developed a novel approach that combines the cosine similarity and euclidean similarity scores to capture both direction and magnitude information. This combined measure allows for a more comprehensive evaluation of semantic similarity.

We compare these similarity scores with the already calculated aggregated similarity score, which serves as a benchmark or ground truth. This aggregated similarity score is obtained through a meticulous process that incorporates the strengths of various similarity methods and provides a comprehensive assessment of the semantic similarity between the two sentences.

By comparing the individual similarity scores with the aggregated score, we gain valuable insights into the performance and effectiveness of each similarity method in capturing semantic similarity. This comparative analysis allows us to identify which methods are more accurate and reliable for our specific use case, enabling us to make informed decisions about the most suitable approach for measuring sentence similarity in our project.

	A	B	C	D	E	F	G	H	I	J	K	L
1	caption1	caption2	agg_score	sampling	sent1	sent2	pcc_score	jac_score	euc_score	man_score	cos_score	csec_score
2	411553	5142	5	c2c_isim	A red surf	A red surf	5	5	5	5	5	4.999999
3	411760	601619	3.1	c2c_isim	A person i	The man i	4.777415	2.260274	3.136914	4.370005	4.557171	4.562959
4	415990	699092	2.7	c2c_isim	A man ridi	A man on	4.150215	1.410256	2.41882	3.930216	3.300785	3.322995
5	463283	560719	1.29	c2c_isim	A group of	a table set	3.462353	0.03012	1.946896	3.890209	1.940251	1.980244
6	702799	225429	2.6	c2c_isim	A pizza wi	A pizza sit	3.665254	0.442177	1.886405	3.686896	2.322484	2.357482
7	451090	169361	2.1	c2c_isim	A large bu	A street sc	3.756957	0.531915	1.873053	3.073142	2.482737	2.51564
8	653035	498205	1.87	c2c_isim	Two surfe	There is a	3.611811	0.178571	1.932234	3.404155	2.237181	2.273294
9	793599	756017	1.9	c2c_isim	A bathroo	A small ba	3.793241	0.714286	2.058193	3.623577	2.590217	2.621715
10	794391	652989	2.71	c2c_isim	A toilet wi	a bathroo	3.79716	0.261194	2.057159	3.705059	2.595024	2.626459
11	794856	428325	2.2	c2c_isim	The toilet	A white to	4.640994	1.801802	3.045991	4.455859	4.283241	4.292609
12	795708	461455	2.08	c2c_isim	A clean to	Toilet with	4.062065	0.59375	2.350594	4.16715	3.115565	3.140196
13	797103	670117	1.18	c2c_isim	a bathroo	A view of	4.137929	1.038961	2.456452	3.971705	3.274924	3.297472
14	153540	400350	1.18	c2c_isim	A small eka	head sh	3.947754	0.058594	2.553474	4.093925	2.899304	2.926762
15	463644	218195	2.58	c2c_isim	there are	A plate to	3.837943	0.163934	2.408592	3.921602	2.636822	2.667711
16	328052	208244	2.2	c2c_isim	Two Giraf	Two giraff	3.618769	0.326087	2.226666	3.693498	2.243764	2.279791
17	484200	533774	0	c2c_isim	Elephants	a white co	4.059701	0.4	2.414264	3.845846	3.118032	3.142631
18	13957	795328	1.1	c2c_isim	A young b	A woman	4.269998	0.514019	2.66591	4.222003	3.536696	3.555822
19	16225	437221	1.1	c2c_isim	A boy abo	two tennis	3.607234	0.05102	2.035954	3.541056	2.215564	2.251959

Figure 6.1: Comparison table for similarity scores & aggregate scores

In the process of selecting the most effective similarity calculation method, we utilized three evaluation metrics: Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Mean Square Error (MSE). These metrics provided a quantitative assessment of the performance of each similarity method in comparison to the aggregated similarity score, which serves as the ground truth.

The MAE measures the average absolute difference between the individual similarity scores and the aggregated score. A lower MAE indicates that the method's predictions are closer to the ground truth, implying higher accuracy.

The RMSE and MSE are additional error metrics that provide a more comprehensive evaluation of the prediction errors. RMSE calculates the square root of the average squared differences between the individual similarity scores and the aggregated score, while MSE computes the mean of these squared differences.

For each similarity method, we obtained the MAE, RMSE, and MSE values and plotted them on separate graphs. These graphical representations allowed us to visualize and compare the performance of each method across different error metrics. By analyzing the trends and patterns in the graphs, we were able to identify the similarity method that achieved the lowest error values, indicating its superior accuracy in predicting semantic similarity.

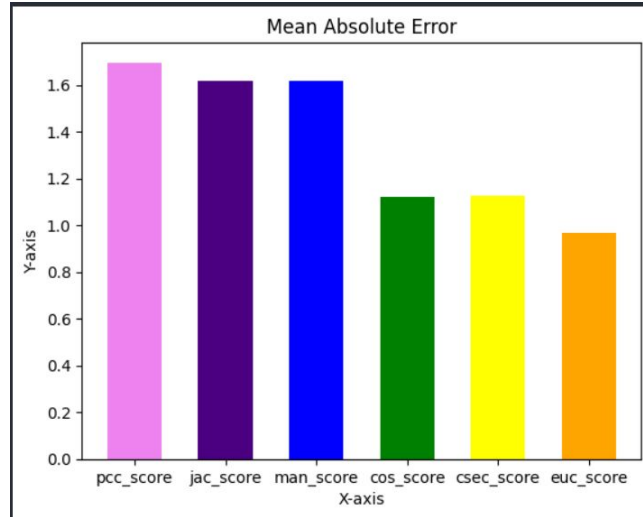


Figure 6.2: Mean absolute error graph

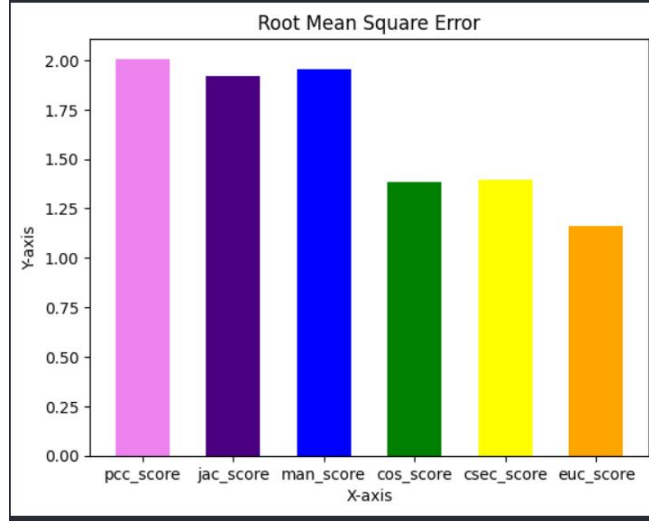


Figure 6.3: Root mean square error graph

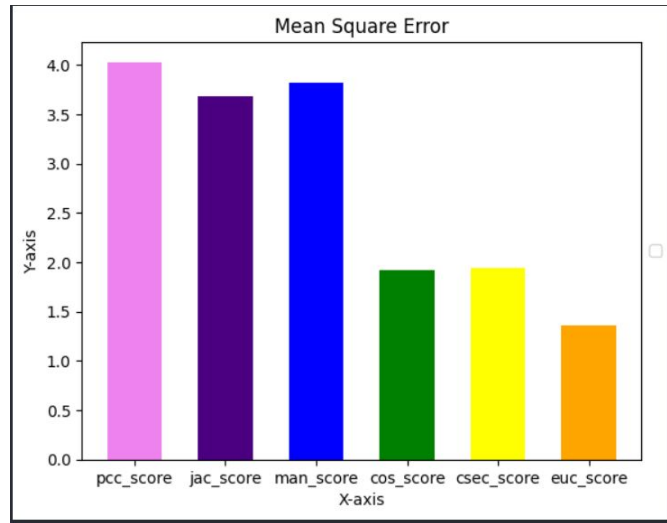


Figure 6.4: Mean square error graph

After conducting rigorous testing and evaluation of various similarity calculation methods, we have concluded that the Euclidean method is the most suitable choice for our project. The Euclidean method consistently demonstrated superior performance across the evaluation metrics, including Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Mean Square Error (MSE). Its ability to accurately predict semantic similarity and its relatively low error values make it the optimal solution for our specific use case.

By choosing the Euclidean method, we can confidently ensure that our project's similarity analysis produces reliable and precise results. This selection aligns with our project's

objectives and helps us achieve the desired level of accuracy in comparing sentences and determining their semantic similarity.

Chapter 7

Results

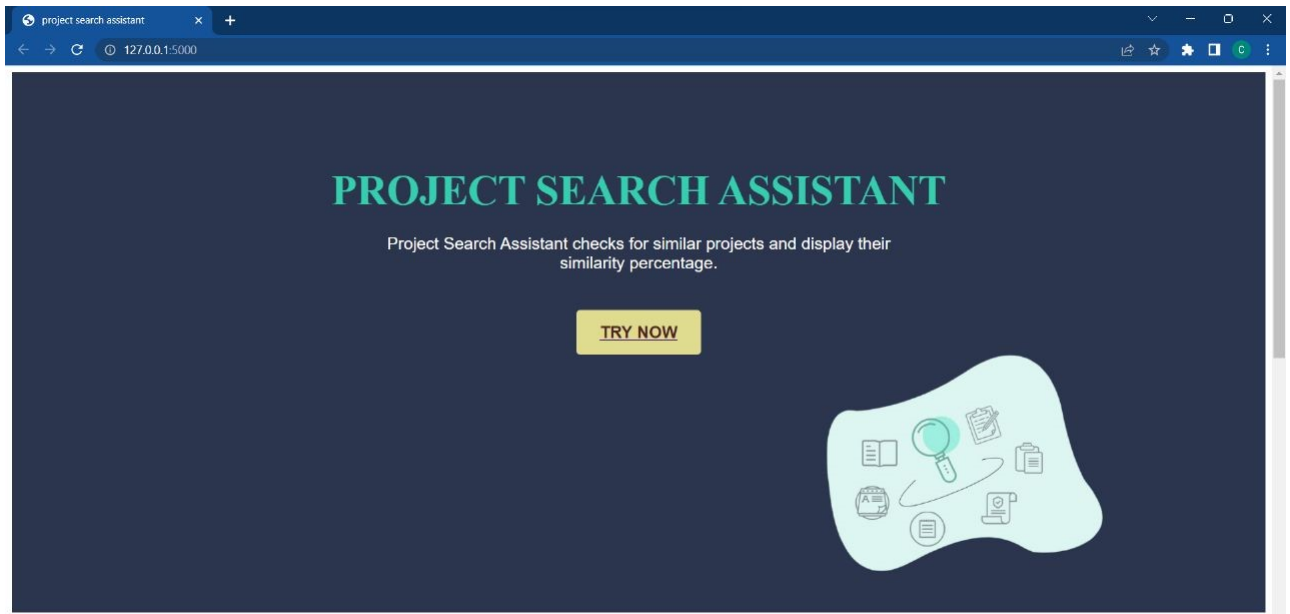


Figure 7.1: Main Menu interface

The homepage of the Project Search Assistant presents a clear and inviting interface, offering users a prominent "Try Now" button. Upon clicking this button, users are swiftly directed to a designated page where they can effortlessly input their abstract or desired text for semantic similarity analysis. This streamlined approach ensures a seamless and user-friendly experience, enabling users to quickly initiate the text analysis with their preferred input.

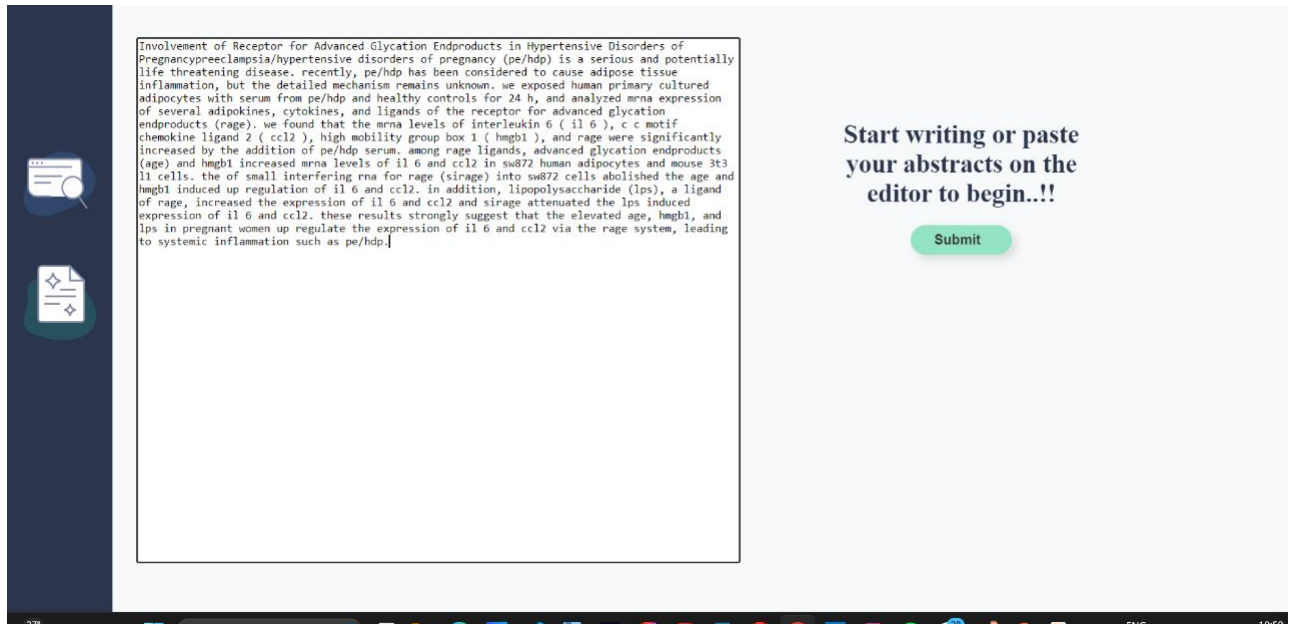



Figure 7.2: Input abstract screen

Once the user clicks on the "Try Now" button, they are redirected to a new page featuring a user-friendly text box. In this text box, users can conveniently enter their project abstract or any desired text they wish to compare with the existing project abstracts. To initiate the comparison process, the page includes a "Check Similarity" button. By clicking this button, the system efficiently calculates and presents the similarity scores, enabling users to gain valuable insights into the semantic similarity between their input and the existing project abstracts.



Id	Project Title	Similarity Score	Project Details
6862609	Involvement of Receptor for Advanced Glycation Endproducts in Hypertensive Disorders of Pregnancy	100.00%	Details
6873610	Abnormal Distribution and Function of Circulating Monocytes and Enhanced Bacterial Translocation in Major Depressive Disorder	44.03%	Details
6827222	Lower serum expression of miR 181c 5p is associated with increased plasma levels of amyloid beta 1-40 and cerebral vulnerability in normal aging	43.95%	Details
6833146	Associations of plasma high sensitivity C reactive protein concentrations with all cause and cause specific mortality among middle aged and elderly individuals	43.77%	Details
6826184	Serum Apolipoprotein A I Combined With C Reactive Protein Serves As A Novel Prognostic Stratification System For Colorectal Cancer	43.74%	Details
6853912	The Vitamin D status is associated with serum C reactive protein and adhesion molecules in patients with renal cell carcinoma	43.68%	Details
6856491	Inhibition of NADPH Oxidase Derived Reactive Oxygen Species Decreases Expression of Inflammatory Cytokines in A549 Cells	43.45%	Details
6887538	The AAA+ ATPase RUVBL2 is essential for the oncogenic function of c MYB in acute myeloid leukemia	43.44%	Details
6842560	Genome wide identification and characterization of the MADS box gene family in Salix suchowensis	43.36%	Details
6838094	Agmatinase promotes the lung adenocarcinoma tumorigenesis by activating the NO MAPKs PI3K/Akt pathway	43.36%	Details

Figure 7.3: Result display screen

Upon completion of the similarity calculation process, the user is directed to the last page, where the results of the comparison are presented. This page displays a comprehensive list of similar projects, sorted in descending order based on the similarity score in percentage. For each matched project, the page showcases the project title as well as the corresponding similarity score, which is expressed as a percentage. Alongside each project entry, an option labeled "Details" is available. By clicking on this option, users can access comprehensive details about each specific project.



Figure 7.4: Project details screen

Upon clicking the "Details" option for a specific project on the results page, users are directed to an additional page that showcases the title and abstract of the project. This page specifically corresponds to the project that shares a similarity with the user-provided input abstract. The content displayed on this page offers a quick glance at the essential information of the matched project, allowing users to better assess its relevance and potential interest.

Chapter 8

Risks and Challenges

1. Data Quality : High quality data ensures that the input data is accurate, relevant, and properly labeled to avoid biased or misleading results. Biased or misleading data can significantly impact the accuracy of similarity scores and, consequently, the quality of our semantic similarity analysis. It requires careful selection and curation of text data to avoid introducing unintended biases or inaccuracies in the results.
2. Size of dataset : Handling large datasets is a challenge for our project as it requires significant computational resources for processing and analyzing text data. With the dataset size increasing, the processing time and resource constraints can impact system efficiency and scalability.
3. Similarity calculation method : Selecting the best algorithm for similarity calculation poses a significant challenge in our project. With multiple methods available determining the most suitable approach requires thorough testing and evaluation. Factors like accuracy, computational complexity, and robustness must be considered to ensure reliable and meaningful similarity scores for project recommendations.

Chapter 9

Conclusion

In conclusion, our project, the Search Assistant, aims to provide an efficient and accurate solution for comparing project abstracts and identifying similar projects. By leveraging advanced natural language processing techniques such as TF-IDF vectorization and Euclidean distance calculation, we have developed a robust system capable of handling large datasets and offering meaningful insights into project similarities. The Search Assistant's user-friendly web interface enables users to conveniently input their project abstracts, view relevant project matches, and explore detailed information about each similar project. Through rigorous testing and evaluation, we have determined that the Euclidean distance algorithm yields the best results for our specific use case. Our project not only enhances research and project exploration but also showcases the power of leveraging NLP techniques in real-world applications. As we continue to enhance and expand the capabilities of our project Search Assistant, we are confident in its potential to revolutionize project comparisons and research exploration.

Bibliography

- [1] S. K. Ag, S. K. Kumar, and M. K. Tiwari. An efficient recommendation generation using relevant jaccard similarity. *Inf. Sci.*, 483:53–64, May 2019.
- [2] O. Ahian, M. Treutwein, P. Estellé, S. Wongwises, D. Wen, G. Lorenzini, A.S. Dalkilic, W.-M. Yan, and A.Z. Sahin. Measurement of similarity in academic contexts. *Publications*, 5:18, 2017.
- [3] Hossam Magdy Balaha and Mahmoud M. Saafan. Automatic exam correction framework (aecf) for the mcqs, essays, and equations matching. *IEEE Access*, 9:32368–32389, 2021.
- [4] Balakrishnan and E. Lloyd-Yemoh. Stemming and lemmatization: A comparison of retrieval performances. In *Proc. SCEI Seoul Conf.*, April 2014. [online] Available: <http://eprints.um.edu.my/13423/>.
- [5] M. P. Agus Christian and D. Suhartono. Single document automatic text summarization using term frequency-inverse document frequency (tf-idf). *ComTech Comput. Math. Eng. Appl.*, 7(4):285–294, 2016.
- [6] Rada Mihalcea, Courtney Corley, and Carlo Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *Proc. AAAI*, volume 6, pages 775–780, 2006.
- [7] A. Saini, A. Bahl, S. Kumari, and M. Singh. Plagiarism checker: text mining. *International Journal of Computer Applications*, 134(3):8–11, 2016.
- [8] William Scott. Tf-idf from scratch in python on a real-world dataset. *Medium*, February 2019.

Appendix A: Base Paper

An Automated System for Measuring Similarity between Software Requirements

¹Fatma A. Mihany, ²Hanan Moussa, ³Amr Kamel, ⁴Ehab Ezzat, ⁵Muhammad Ilyas

^{1,2,3,4}Cairo University, Faculty of Computers and Information, Cairo, Egypt

⁵Department of Computer Science & IT, University of Sargodha, Sargodha, Pakistan

{¹fatmaabdeldayem, ²h.moussa, ³a.kamel, ⁴e.ezat}@fci-cu.edu.eg}, ⁵m.ilyas@uos.edu.pk

ABSTRACT

Recently, usage of text similarity has increased rapidly to be involved in different areas such as document clustering, information retrieval, short answer grading, text summarization, machine learning and natural language processing. Lexical-based similarity and semantic-based similarity are the two main categories of text similarity. Reusability of software components increases productivity and quality. In this paper, we propose that there is some linkage between text similarity and software reusability. In an organization, whenever a new incoming project is received, similarity test can be done to identify some similar projects and therefore some components to be reused such as design, code and test cases instead of starting building software from scratch. In this paper, we present an interactive system to measure the lexical similarity between a new incoming project and a set of completed projects exist in the repository and therefore identify some components to be reused.

CCS Concepts

• **Software and its engineering** → **Software creation and management** → **Designing software** → **Requirements analysis**

Keywords

Text similarity; Measurement; Requirements; Reusability

1. INTRODUCTION

When a new incoming project is received in an organization to be produced, so it should go through a specified process called software development life cycle (SDLC). This process is a systematic approach for building software. Planning, analysis, design, development, testing and maintenance are the main phases of SDLC. There are various models of SDLC such as spiral, iterative, agile and waterfall. Based on the selected model the procedure of building the software differs [1,2].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permission@acm.org.

AMECSE '16, May 28-29, 2016, Cairo, Egypt

© 2016 ACM. ISBN 978-1-4503-4293-3/16/05\$15.00

DOI: <http://dx.doi.org/10.1145/2944165.2944173>

In recent years, requirements engineering gets more attention due to its great importance. Requirements engineering (RE) consists of several phases such as requirements elicitation, analysis, specification, validation, and management. The main goal of the requirements engineering process is to generate a set of requirements that satisfy some criteria such as completeness, consistency, correctness and so on. Mainly, software requirements are categorized into functional requirements and non-functional requirements. The requirements produced from the requirements engineering process should meet customer needs unless it needs some iterations [1,2,3].

There are two ways to start development of a new computer-based system; first building all of its components from scratch, second reusing some existing components. Reusability is defined as using some existing components to solve other problems. Software reusability increases productivity, validity and quality. Also, reusability may save effort, time and cost [2,3,4]. The decision to reuse or start development from scratch is not an easy task. Sometimes the decision maker needs some support before taking the decision. Also identifying reusable components needs some effort. Reusability decision can be taken based on some similarity measurements. In other words, if a new project is similar to some existing projects, so reusability can be performed if not starting development from scratch is a must as shown in figure 1.

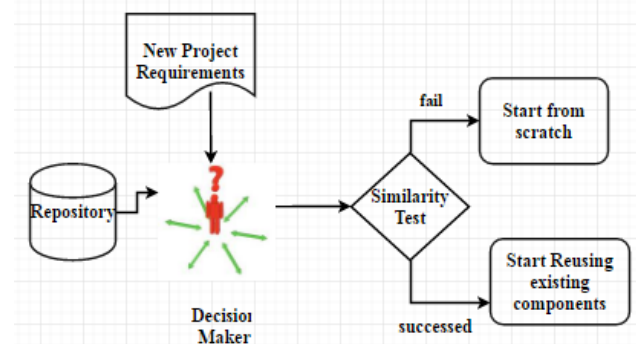


Figure 1. Reusability Decision

In this paper, we present an automated system to measure the similarity between requirements to facilitate reusability. After determining some similar projects, some components can be identified to be reused. Our system helps decision maker to determine whether to start development from scratch or reusing some existing components as shown in figure 1.

This paper is structured as follows: In section 2, we explain text similarity techniques. In section 3, we describe in details our proposed system with its phases and implementation. Preliminary

results are presented in section 4. In section 5, we analyze and evaluate the results of the proposed system. Section 6 presents our conclusions and future work.

2. TEXT SIMILARITY

Text similarity defined as specifying how much some text close to each other. Text similarity can be measured by some stylistic information such as sentence length, word variation and so on. There are many similarity techniques such as syntax-based, semantic-based and hybrid-based techniques [5]. Measuring similarity between words, sentences, paragraphs and documents is an important component in various research areas such as information retrieval, document clustering, word-sense disambiguation, automatic essay scoring, short answer grading, machine translation and text summarization. Document similarity is also another usage of text similarity. Document similarity measurement is an important technique for categorizing and clustering documents [5,6,7]. Simple text similarity techniques taxonomy is shown in figure 2.

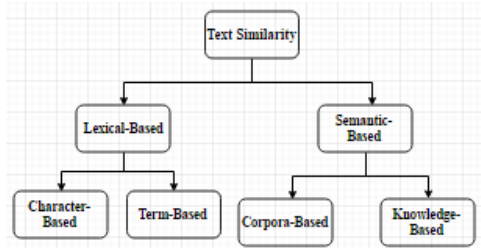


Figure 2. Text similarity Taxonomy [6]

2.1 Lexical-Based Similarity

Lexical similarity is concerned with syntax only. In other words, it is concerned with what is written or sequence of characters and how much these words are similar to each other regardless of their meaning. Mainly Lexical similarity measures are divided into two main categories: character-based similarity and term-based similarity. There are a lot of algorithms which are used in lexical similarity such as Longest Common SubString (LCS), Damerau-Levenshtein, Jaro, Jaro-Winkler, and others [6,8].

An algorithm is used to measure pairwise similarity between documents by representing all documents in a matrix view, the rows represent the tokens or words which are extracted from each document and the columns represent all documents and every cell in the matrix will be filled in with a Boolean value (1 indicates that token exist in the specified document and 0 indicates the opposite) as shown in table 1 [7,9]. If we are dealing with a bigger number of documents for example multinational company which has requirements documents for several years, the computations will be very huge. So, some aggregations should be made and logical relations can be used to represent the relation between documents. $R(d_1, d_2, v)$ represent some relation between two documents where d_1 refers to the first document, d_2 refers to second document and v indicates the number of common tokens in both documents [9]. Similarity between d_1 and d_2 documents can be computed using the following equation:

$$Sim(d_1, d_2) = \frac{|d_1 \cap d_2|}{|d_1 \cup d_2|} \quad (1)$$

Table 1. Documents Representation Matrix

	Doc1	Doc2	Doc3	Doc n
Token1	0	1	0			1
Token2	1	0	1			1
Token3	0	1	1			0
Token4	1	1	0			0
...						
...						
...						
...						
...		1	0			1
...						
...						
...						
Token m	1	1	0			0

2.2 Semantic-Based Similarity

The semantic similarity has emerged in the 1990s into some psychological experiments [10]. It is considered as a basic concept for interpreting and organizing objects. Semantic similarity measures how the words are similar to each other by comparing the meaning behind each one. For example, 'gift' and 'present' lexically are not similar at all, but the two words are similar semantically. Corpus-based similarity techniques measure the semantic similarity between words based on huge corpora. The corpora defined as a large set of text used for research purposes. Semantic similarity is extensively used in query answering systems to help the user to find what he or she means regardless of the sequence of characters written [6,7,8]. Ontologies are also used for similarity purposes. Ontologies offer organized structures and clear representation of knowledge by connecting conceptualization with semantic pointers [9,11].

3. FOUR-PHASE SIMILARITY SYSTEM (FPSS)

In a previous work [12], we introduced a framework called "Four-Phases Similarity" framework, the main goal of the framework is to measure the similarity between software requirements targeting reaching reusability. We extended our previous work in this paper by presenting in details all phases of the framework with its implementation and preliminary results. The main objective of the proposed system (FPSS) is to measure the similarity between software requirements to identify some components to be reused. FPSS allows calculating a measurement which indicates how close two requirements to each other. The computed values will be analyzed to determine which components to be reused. Design components, code components, and test cases can be reused in case of a successful similarity test. The proposed system has two major objectives: one for measuring similarity and another one for identification of reusable components. Every phase of the framework will be described in details with its implementation in

next section. All phases in our system should work sequentially to get the desired output.

FPSS is an interactive system which waits for some inputs from a user and based on them, it returns the expected result as shown in figure 3. The user should type some information about the incoming project to be saved in the repository. The user should fill in the form shown in figure 4 by selecting project name of the new project and the other required fields. The purpose behind different phases of our system is to measure the similarity between projects and therefore suggesting some components to be reused.

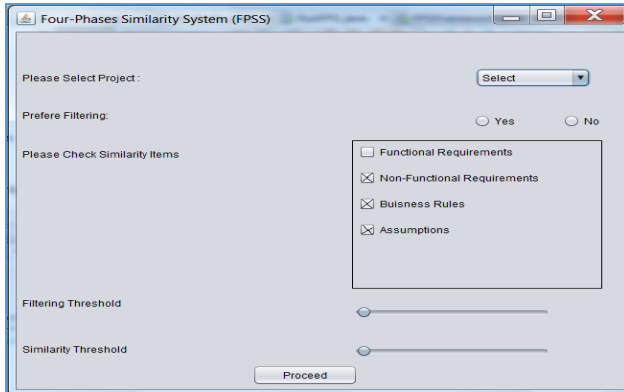


Figure 3. A snapshot for FPSS

3.1 First Phase: Annotation

When a new incoming project is received in an organization, the requirements are collected and analyzed to be documented and saved in the repository. There is some general information about the project and other information about functionalities (functional requirements) and others for non-functional requirements. In our implementation of FPSS, we designed a template for project general information as shown in figure 4. There are five sections highlighted in figure 4 which are project name, business domain, application type, programming language and delivery year, these five sections are considered as an annotation for a project. In other words, every project in our repository is described or annotated according to those five sections. Based on other previous studies, we have designed this template for project annotation. Based on our experiments we concluded that every project is well described according to the highlighted five sections. The annotation will be used in the next phase.

Figure 4. Add Project Form (highlighted project annotation)

3.2 Second Phase: Filtering

The main purpose of this phase in our system is to minimize search scope of projects in the repository based on certain conditions. As described in the previous phase every project is annotated based on five sections (shown in figure 4). Filtering phase is an initial similarity test, it measures the similarity between projects based on their annotations only. If a specific project passed this initial test, it will join the next phases of the proposed system; otherwise, it will be excluded out of the overall process. There are five conditions should be checked corresponding to the five sections of the annotation as shown in figure 5.

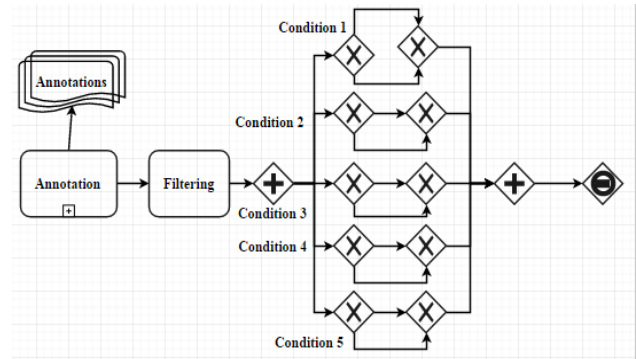


Figure 5. Filtering Phase

Condition 1: regarding project name

Condition 2: regarding business domain

Condition 3: regarding application type

Condition 4: regarding programming language

Condition 5: checks whether the difference between starting date (year only) of the incoming project and delivery date (year only) of the old project exceeds specific threshold or not.

One person can decide whether two projects may have similar components or not just by viewing their annotations as an initial test without going through their requirements. By considering an example shown in figure 6, the two projects seems not have similar components. From our point of view there are five reasons for that: project names have different characters and seems not related at all, the two projects do not relate to similar business domains, the type of application of both projects is different, the programming language is not the same and there are long period of time between both projects. According to these five reasons we can guarantee or with high probability that the two projects are not similar. This manual process has been automated in our system which helps the decision maker to filter some projects out of overall similarity test as those projects seems not similar from the beginning. Some threshold should be specified according to user needs to filter based on it and it can be an input from the user as shown in figure 3.

Project Name: Online Shopping	Project Name: Banking System
Business Domain: Sales	Business Domain: HR
Application Type: Web App	Application Type: Desktop App
Programming Language: PHP	Programming Language: Java
Start Date: 2015	Start Date: 1995

(a)

(b)

Figure 6. Example of annotations of two different projects

3.3 Third Phase: Requirements Similarity Measurement Phase

The main objective of this phase is to measure the similarity between a new incoming project and a set of projects exist in the repository. The work in this phase may depend on the output of the previous phase or not based on some inputs from a user as shown in figure 3. If the user prefers to filter some projects, so the previous phase should be done first and the system takes output to be an input to current phase. If the user does not prefer filtering, so the current phase works on all projects in the repository. According to our system, it compares between requirements of the projects to determine whether the projects are similar or not. Our system works on a predefined template for every requirement which is shown in figure 7.

FPSS system takes the description of every requirement and convert it into some sentences and therefore to some words. After this step, a similarity measurement can be used for calculating a similarity value between requirements by using a similarity measurement. There are various measurements for calculating lexical similarity as described earlier in section 2.1. Dice, Jaccard, Euclidean distance, Cosine, Jaro, Block distance are some examples of those measurements. We used only three similarity measures in our work which are Dice [13], Jaccard [14] and Cosine similarity measurements. The formulas for the three measurements are defined as formula (2), (3) and (4):

$$\text{Cosine Similarity} = \frac{|words_{r_1} \cap words_{r_2}|}{\sqrt{|words_{r_1}| |words_{r_2}|}} \quad (2)$$

$$\text{Dice Similarity} = \frac{2|words_{r_1} \cap words_{r_2}|}{|words_{r_1}| + |words_{r_2}|} \quad (3)$$

$$\text{Jaccard Similarity} = \frac{|words_{r_1} \cap words_{r_2}|}{|words_{r_1}| + |words_{r_2}| - |words_{r_1} \cap words_{r_2}|} \quad (4)$$

Figure 7. Requirements Template

The system produces a three similarity measurements values according to the three used measures Dice, Jaccard and Cosine after applying the following algorithm:

Algorithm 3.3 Calculate Requirements Similarity Algorithm

```

Input: RP // a repository of existing projects
Input: NP // a new project
Output: s // similarity value
Variables: OP // a single old project exists in RP
               r_new //list of requirements for NP
               r_old //list of requirements for one OP

1: OP = 0, r_old = '', r_new = '';
2: r_new=getAllrequirements (NP)
3: for every OP in RP
4: r_old=getAllrequirements (RP [OP])
5: for every requirement in r_new
6: r_new_description=getDescription (NP)
7: r_old_description=getDescription (r_old)
8: r_new_description =convertToWords (r_new)
9: r_old_description =convertToWords (r_old)
10: Sim (r_new_description, r_old_description)
11: End for
12: getMax (Sim (r_new_description,r_old_description))
13: End for
14: Average (Max(Sim(r_new_description,r_old_description)))
15: End for
16: Return s

```

3.4 Fourth Phase: Reusability

The main objective of this phase is to facilitate or enhance software reusability by suggesting some components to be reused. In the previous phase of our system, we measure the similarity between projects. Our ultimate goal is to facilitate reusability to increase productivity and quality of software. The proposed system in this paper does not handle reusability issues, but we will work on implementing this phase and integrating it with the whole system in our future work.

Requirements traceability is defined as an approach that helps user to traverse between requirements artifacts through different phases of software development lifecycle. Traceability relationship describes some relationship between project components. Some linkage will be modeled to specify all traceability relationships. There is a popular technique called traceability matrix used in requirements traceability era. Traceability matrix is a table captures requirements. It helps traversal starting from requirement to its testing component. Also, one can move and traverse between project components based on the template of the traceability matrix [15,16].

4. RESULTS

We implemented 'Four-Phases Similarity system (FPSS)' by java programming language and we have used NetBeans IDE 8.0.2. As mentioned earlier in this paper, our proposed system helps decision makers to measure the similarity between requirements

of an incoming project and a set of already completed projects for facilitating reusability. Our system is an interactive system as shown in figure 3. It takes some parameters from a user to generate different results based on the inputs. Our system works according to the following flow chart in figure 8.

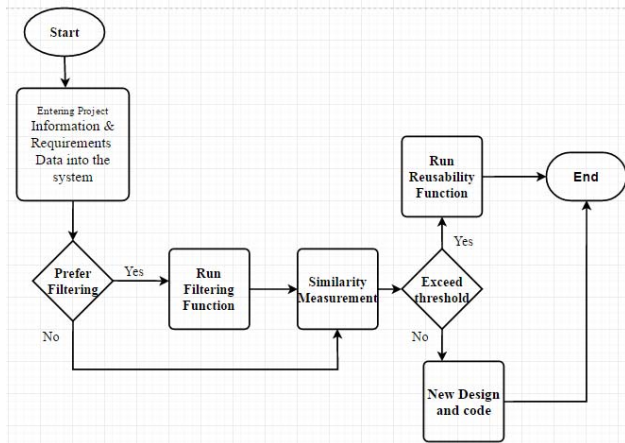


Figure 8. Flow Chart for FPSS

4.1 Without Filtering

When the incoming project is selected, it will be tested according to all projects exist in the repository and therefore, the search area will be huge, especially if we deal with a multinational organization which has a big number of already developed projects. As stated earlier, we focus in this paper on measuring the similarity between projects by comparing between requirements of different projects with each other. We tested the proposed system on a set of projects and some results are shown in figure 9, figure 10 and figure 11. In the example of figure 9, the incoming project has been compared with all existing projects in the repository. Every requirement will be compared to every requirement from an old project and then choose maximum values and finally get an average of all maximum similarities as explained in the algorithm in section 3.3.

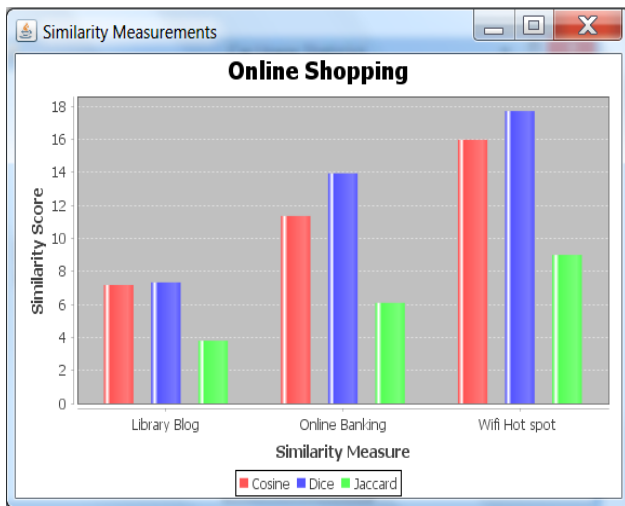


Figure 9. Similarity measurements values (without filtering), online shopping is the name of the incoming project and it is compared to other existing projects and for every combination a three similarities are computed.

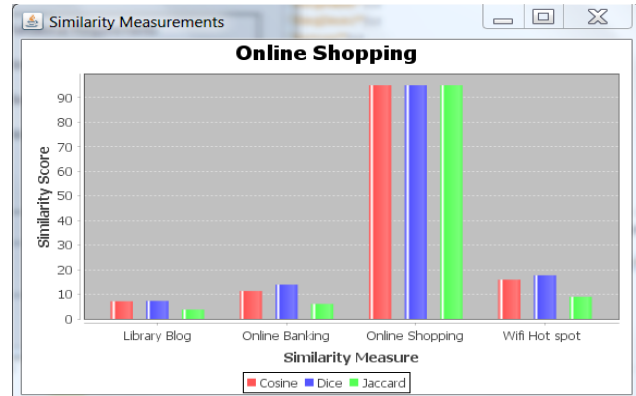


Figure 10. Similarity measurements values (without filtering), online shopping is the name of the incoming project and it is compared to itself and other existing projects.

4.2 With Filtering

When the incoming project is selected, it will be tested according to only some selected projects from the filtering phased and therefore, the search area will be minimized as shown in figure 10. In this example, the incoming project has been compared with only some existing projects in the repository which passed filtering phase. Before starting running our own similarity algorithm (explained in section 3.3), the filtering process should run first. According to the five sections of the annotation template (shown in figure 3) some similarity test will be done. Only projects that passed this test will go through other phases of the system. Filtering is done as shown in figure 4. After that similarity algorithm will run without any changes as explained in section 3.3.



Figure 11. Similarity measurements values (with filtering) Library Blog is compared to other existing projects.

5. DISCUSSION

We faced some problems when trying to get real life requirements of real systems due to some privacy issues and this point is one of our limitations, but we will overcome this problem in our future work. By running the system with no filtering, the new project will be compared to all existing projects as shown in figure 9 and figure 10. All similarities computed according to comparing functional requirements only. We used three different similarity measures which have been explained earlier and defined in formulas 2, 3, and 4 in section 3.3. Based on our results, we concluded that variation between cosine and dice measurements is insignificant, but the variation with Jaccard is noticeable. In figure

8 and figure 10, the incoming project is compared to some other projects whatever all projects or smaller set of projects. We have done a simple verification test by including the project itself in the similarity test as shown in figure 9. The similarity values from the mentioned three measures approaching 100% similarity when the project is compared to itself. We will verify the system by making some manual testing and tracking outputs of the automated system.

SimReq framework is presented in [17], it measures the similarity between software requirements of the running project with the requirements of already completed projects. Their software takes the textual requirement and compares it with a requirement of another project. There are two main differences between our own work and their work; first SimReq framework compares requirement-to-requirement but our system compares a whole project with another whole old project. In our system, we do a two level similarity; first level, annotation of the projects and make an initial similarity test to measures the similarity based on the annotations, second requirements similarity. There is only one level similarity for requirements in SimReq framework. The same similarity measures are used in FPSS and simreq system. It is concluded in the other work that Cosine similarity measurement is the best one as it gets greater similarity values in their experiments. One of the findings in our work is that Jaccard similarity measurements produce greatest similarities compared to the other two measures. Based on our preliminary results, we think that accuracy of the measurement differs according to the nature of the data set being used.

6. CONCLUSION AND FUTURE WORK

In this paper, we have presented an automated system that measures the similarity between software requirements to enhance software reusability as an extension of a previous work. In our previous work, we have presented a framework for measuring the similarity between requirements. We have proposed that there is a connection between text similarity and software reusability. In this work, we used lexical text similarity to measure the similarity between a new project with a set of already completed projects, therefore suggesting some reusable components. Software reusability increases productivity and saves cost, time and cost. Design components, code components, and test cases can be reused. Our proposed system in this paper is composed of four main phases which are annotation, filtering, similarity measurement, and reusability phase.

In our future work, we will test our system on a bigger number of projects; also, it will be tested with some real life projects. Some different similarity measurements can be used. We will compare all measurements results on same data set to conclude which one will be the best. The system will be well tested and verified with reasonable error rate. We will determine a better pre-defined template for requirements and annotation to overcome the problem of informal style of writing. In the future, we will work on the reusability phase and will implement some traceability techniques to identify the components to be reused. We can go through how the reusable components can be customized to save effort and time instead of building them from scratch. Reusability module will be integrated with our system to be a comprehensive system for measuring the similarity of requirements components and facilitating software reusability.

7. REFERENCES

- [1] Apoorva Mishra and Deepty Dubey. 2013. A comparative study of different software development life cycle models in different scenarios. *International Journal of Advance research in computer science and management studies*.
- [2] Swarnalatha k s, GN Srinivasan, Meghana Dravid, Raunak kaseria, Kopal Sharma. 2014. A Survey on Software Requirement Engineering for Real Time Projects based on Customer Requirement. *International Journal of Advanced Research in Computer and Communication Engineering*.
- [3] Pierre Bourque and Richard E. Fairley. 2014. Guide to the software engineering body of knowledge (SWEBOK (R)):Version 3.0. *IEEE Computer Society Press*.
- [4] Michael Keating. 2012. Reuse Methodology Manual for System-On-A-Chip Designs. *Springer Science & Business Media*.
- [5] Ralf Steinberger, Bruno Pouliquen, Johan Hagman. 2002. Cross-lingual document similarity calculation using the multilingual thesaurus eurovoc. *Computational Linguistics and Intelligent Text Processing*.
- [6] Wael H. Gomaa, Aly A. Fahmy. 2013. A survey of text similarity approaches. *International Journal of Computer Applications*.
- [7] Papias Niyigena, Zhang Zuping, Weiqi Li and Jun Long. 2015. Efficient Pairwise Document Similarity. *Computation in Big Datasets. International Journal of Database Theory and Application*.
- [8] Rada Mihalcea, Courtney Corley and Carlo Strapparava. 2006. Corpus-based and knowledge-based measures of text semantic similarity. *Association for the Advancement of Artificial Intelligence*.
- [9] David Sánchez, Montserrat Batet, David Isern and Aida Valls. 2012. Ontology-based semantic similarity: A new feature-based approach. *Expert Systems with Applications*.
- [10] Goldstone, R. L. (1994). Similarity, interactive activation, and mapping. *Journal of Experimental Psychology: Learning, Memory, and Cognition*.
- [11] Monika Lanzemberger, Jennifer Sampson. 2008. Making Ontologies Talk: Knowledge Interoperability in the Semantic Web. *IEEE Intelligent Systems*.
- [12] Fatma A. Mihany, Hanan Moussa, Amr Kamel, Ehab Ezat. 2016. A Framework for Measuring Similarity between Requirements Documents. *10th International Conference on Informatics and Systems*.
- [13] Lee R. Dice. 1945. Measures of the amount of ecologic association between species. *Ecology*.
- [14] Paul Jaccard. 1901. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*.
- [15] Duraisamy, Gunavathi, and Rodziah Atan. 2013. Requirement traceability matrix through documentation for scrum methodology. *Journal of Theoretical & Applied Information Technology*.
- [16] Dean Leffingwell and Don Widrig. 2002. The role of requirements traceability in system development. *The Rational Edge*.
- [17] Muhammad Ilyas, Josef Küng. 2009. A Similarity Measurement Framework for Requirements Engineering. *Fourth International Multi-Conference on Computing in the Global Information Technology*.

Appendix B: Sample Code

```
import os
import pandas as pd
import string
import re
import numpy as np
from nltk.stem import WordNetLemmatizer, PorterStemmer
from nltk.corpus import stopwords, wordnet
from nltk.tokenize import word_tokenize
# from sklearn.feature_extraction.text import TfidfVectorizer
# from gensim.models import Word2Vec
from gensim import corpora, models
```

```
# import gensim.downloader as api
import gensim
from api.index import db
import time
```

```
def compareData(abstract):
    begin = time.time()
    # path_word2vec_model = 'word2vec.model'
```

```
def preprocess_data(contents):
    stemmer = PorterStemmer()
    wordnet.ensure_loaded()
    stop_words = set(stopwords.words('english'))
    preprocessed_data = []
    for content in contents:
        lemmatizer = WordNetLemmatizer() # Define lemmatizer here
        sentence = content[1].lower()
        sentence = re.sub(r'\d+', '', sentence) # Remove numbers
        sentence = sentence.translate(str.maketrans("", "", string.punctuation)) # Remove
punctuation
        tokens = word_tokenize(sentence)
        tokens = [stemmer.stem(token) for token in tokens if token not in stop_words and
token.isalpha()]
        lemmatized_synonyms = []
        for token in tokens:
            synsets = wordnet.synsets(token)
            lemmas = [lemma for synset in synsets for lemma in synset.lemmas()]
            if lemmas:
```

```

        lemmatized_synonyms.extend([lemmatizer.lemmatize(lemma.name()) for lemma in
lemmas])
        tokens.extend(lemmatized_synonyms)
        preprocessed_data.append(tokens)
    data_time = time.time()
    print(f"preprocess time2: {data_time-begin}")
    return preprocessed_data

def preprocess_query(query):
    stemmer = PorterStemmer()
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer() # Define lemmatizer here
    query = query.lower()
    query = re.sub(r'\d+', "", query) # Remove numbers
    query = query.translate(str.maketrans("", "", string.punctuation)) # Remove punctuation
    tokens = word_tokenize(query)
    tokens = [stemmer.stem(token) for token in tokens if token not in stop_words and
token.isalpha()]
    lemmatized_synonyms = []
    for token in tokens:
        synsets = wordnet.synsets(token)
        lemmas = [lemma for synset in synsets for lemma in synset.lemmas()]
        if lemmas:
            lemmatized_synonyms.extend([lemmatizer.lemmatize(lemma.name()) for lemma in
lemmas])
    tokens.extend(lemmatized_synonyms)
    print("query time",time.time()-begin)
    return tokens

def calculate_euclidean_distance(query, documents):
    dictionary = corpora.Dictionary(documents)
    corpus = [dictionary.doc2bow(doc) for doc in documents]

    model = models.TfidfModel(corpus)
    tfidf_corpus = model[corpus]

    # Convert the query to a tf-idf vector
    query_bow = dictionary.doc2bow(query)
    query_tfidf = model[query_bow]

    # Create a matrix of tf-idf vectors for all documents
    document_matrix = gensim.matutils.corpus2dense(tfidf_corpus,
num_terms=len(dictionary)).T

```

```

# Convert the query and document vectors to numpy arrays
query_vector = gensim.matutils.corpus2dense([query_tfidf], num_terms=len(dictionary)).T

# Calculate Euclidean distance between the query vector and each document vector
euclidean_distances = np.linalg.norm(document_matrix - query_vector, axis=1)

return euclidean_distances

def read_data():
    contents = [(int(obj["id"]),obj["sentence"]) for obj in db.Projects.find()]
    return contents

# Read data from mongo
data = read_data()
print(data[:10])
end_read = time.time()
# Preprocess data
preprocessed_data = preprocess_data(data)

# # Download Word2Vec model
# word2vec_model = api.load("word2vec-google-news-300")

# # Save Word2Vec model
# word2vec_model.save(path_word2vec_model)

# Preprocess query
preprocessed_query = preprocess_query(abstract)

# Calculate similarity using Euclidean distance
query_distances = calculate_euclidean_distance(preprocessed_query, preprocessed_data)

end_eu = time.time()
# Rank documents
ranked_docs = sorted(enumerate(query_distances, start=1), key=lambda x: x[1])

result_docs=[]
# Display top-ranked similar documents
for doc_id, distance in ranked_docs[:10]:
    original_doc_id = data[doc_id - 1][0]
    similarity = 1 / (1 + distance) # Convert distance to similarity (values closer to 1 are more
similar)
    sim_score= f'{similarity*100:.2f}%'
    print(f'Doc {original_doc_id}: Similarity {similarity*100:.2f}%')

```

```
    result_docs.append((original_doc_id,sim_score))

end = time.time()
print(f"read: {end_read-begin}")
print(f"eu: {end_eu-begin}")
print(f"end: {end-begin}")

return result_docs
```

Appendix C: CO-PO And CO-PSO Mapping

COURSE OUTCOMES:

After completion of the course the student will be able to

SL. NO	DESCRIPTION	Blooms' Taxonomy Level
CO1	Identify technically and economically feasible problems (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO2	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO3	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions of minimal complexity by using modern tools & advanced programming techniques (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO4	Prepare technical report and deliver presentation (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO5	Apply engineering and management principles to achieve the goal of the project (Cognitive Knowledge Level: Apply)	Level 3: Apply

CO-PO AND CO-PSO MAPPING

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
CO1	3	3	3	3		2	2	3	2	2	2	3	2	2	2
CO2	3	3	3	3	3	2		3	2	3	2	3	2	2	2
CO3	3	3	3	3	3	2	2	3	2	2	2	3			2
CO4	2	3	2	2	2			3	3	3	2	3	2	2	2
CO5	3	3	3	2	2	2	2	3	2		2	3	2	2	2

3/2/1: high/medium/low

JUSTIFICATIONS FOR CO-PO MAPPING

MAPPING	LOW/ MEDIUM/ HIGH	JUSTIFICATION
100003/CS6 22T.1-PO1	HIGH	Identify technically and economically feasible problems by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS6 22T.1-PO2	HIGH	Identify technically and economically feasible problems by analysing complex engineering problems reaching substantiated conclusions using first principles of mathematics.
100003/CS6 22T.1-PO3	HIGH	Design solutions for complex engineering problems by identifying technically and economically feasible problems.
100003/CS6 22T.1-PO4	HIGH	Identify technically and economically feasible problems by analysis and interpretation of data.
100003/CS6 22T.1-PO6	MEDIUM	Responsibilities relevant to the professional engineering practice by identifying the problem.
100003/CS6 22T.1-PO7	MEDIUM	Identify technically and economically feasible problems by understanding the impact of the professional engineering solutions.
100003/CS6 22T.1-PO8	HIGH	Apply ethical principles and commit to professional ethics to identify technically and economically feasible problems.
100003/CS6 22T.1-PO9	MEDIUM	Identify technically and economically feasible problems by working as a team.
100003/CS6 22T.1-PO10	MEDIUM	Communicate effectively with the engineering community by identifying technically and economically feasible problems.
100003/CS6 22T.1-PO11	MEDIUM	Demonstrate knowledge and understanding of engineering and management principles by selecting the technically and economically feasible problems.
100003/CS6 22T.1-PO12	HIGH	Identify technically and economically feasible problems for long term learning.
100003/CS6 22T.1-PSO1	MEDIUM	Ability to identify, analyze and design solutions to identify technically and economically feasible problems.
100003/CS6 22T.1-PSO2	MEDIUM	By designing algorithms and applying standard practices in software project development and Identifying technically and economically feasible problems.
100003/CS6 22T.1-PSO3	MEDIUM	Fundamentals of computer science in competitive research can be applied to Identify technically and economically feasible problems.
100003/CS6 22T.2-PO1	HIGH	Identify and survey the relevant by applying the knowledge of mathematics, science, engineering fundamentals.

100003/CS6 22T.2-PO2	HIGH	Identify, formulate, review research literature, and analyze complex engineering problems get familiarized with software development processes.
100003/CS6 22T.2-PO3	HIGH	Design solutions for complex engineering problems and design based on the relevant literature.
100003/CS6 22T.2-PO4	HIGH	Use research-based knowledge including design of experiments based on relevant literature.
100003/CS6 22T.2-PO5	HIGH	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes by using modern tools.
100003/CS6 22T.2-PO6	MEDIUM	Create, select, and apply appropriate techniques, resources, by identifying and surveying the relevant literature.
100003/CS6 22T.2-PO8	HIGH	Apply ethical principles and commit to professional ethics based on the relevant literature.
100003/CS6 22T.2-PO9	MEDIUM	Identify and survey the relevant literature as a team.
100003/CS6 22T.2-PO10	HIGH	Identify and survey the relevant literature for a good communication to the engineering fraternity.
100003/CS6 22T.2-PO11	MEDIUM	Identify and survey the relevant literature to demonstrate knowledge and understanding of engineering and management principles.
100003/CS6 22T.2-PO12	HIGH	Identify and survey the relevant literature for independent and lifelong learning.
100003/CS6 22T.2-PSO1	MEDIUM	Design solutions for complex engineering problems by Identifying and survey the relevant literature.
100003/CS6 22T.2-PSO2	MEDIUM	Identify and survey the relevant literature for acquiring programming efficiency by designing algorithms and applying standard practices.
100003/CS6 22T.2-PSO3	MEDIUM	Identify and survey the relevant literature to apply the fundamentals of computer science in competitive research.
100003/CS6 22T.3-PO1	HIGH	Perform requirement analysis, identify design methodologies by using modern tools & advanced programming techniques and by applying the knowledge of mathematics, science, engineering fundamentals.
100003/CS6 22T.3-PO2	HIGH	Identify, formulate, review research literature for requirement analysis, identify design methodologies and develop adaptable & reusable solutions.

100003/CS6 22T.3-PO3	HIGH	Design solutions for complex engineering problems and perform requirement analysis, identify design methodologies.
100003/CS6 22T.3-PO4	HIGH	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS6 22T.3-PO5	HIGH	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools.
100003/CS6 22T.3-PO6	MEDIUM	Perform requirement analysis, identify design methodologies and assess societal, health, safety, legal, and cultural issues.
100003/CS6 22T.3-PO7	MEDIUM	Understand the impact of the professional engineering solutions in societal and environmental contexts and Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS6 22T.3-PO8	HIGH	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions by applying ethical principles and commit to professional ethics.
100003/CS6 22T.3-PO9	MEDIUM	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS6 22T.3-PO10	MEDIUM	Communicate effectively with the engineering community and with society at large to perform requirement analysis, identify design methodologies.
100003/CS6 22T.3-PO11	MEDIUM	Demonstrate knowledge and understanding of engineering requirement analysis by identifying design methodologies.
100003/CS6 22T.3-PO12	HIGH	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS6 22T.3-PSO3	MEDIUM	The ability to apply the fundamentals of computer science in competitive research and prior to that perform requirement analysis, identify design methodologies.
100003/CS6 22T.4-PO1	MEDIUM	Prepare technical report and deliver presentation by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS6 22T.4-PO2	HIGH	Identify, formulate, review research literature, and analyze complex engineering problems by preparing technical report and deliver presentation.

100003/CS6 22T.4-PO3	MEDIUM	Prepare Design solutions for complex engineering problems and create technical report and deliver presentation.
100003/CS6 22T.4-PO4	MEDIUM	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions and prepare technical report and deliver presentation.
100003/CS6 22T.4-PO5	MEDIUM	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and Prepare technical report and deliver presentation.
100003/CS6 22T.4-PO8	HIGH	Prepare technical report and deliver presentation by applying ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
100003/CS6 22T.4-PO9	HIGH	Prepare technical report and deliver presentation effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS6 22T.4-PO10	HIGH	Communicate effectively with the engineering community and with society at large by prepare technical report and deliver presentation.
100003/CS6 22T.4-PO11	MEDIUM	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work by prepare technical report and deliver presentation.
100003/CS6 22T.4-PO12	HIGH	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by prepare technical report and deliver presentation.
100003/CS6 22T.4-PSO1	MEDIUM	Prepare a technical report and deliver presentation to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas.
100003/CS6 22T.4-PSO2	MEDIUM	To acquire programming efficiency by designing algorithms and applying standard practices in software project development and to prepare technical report and deliver presentation.
100003/CS6 22T.4-PSO3	MEDIUM	To apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs by preparing technical report and deliver presentation.
100003/CS6 22T.5-PO1	HIGH	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS6 22T.5-PO2	HIGH	Identify, formulate, review research literature, and analyze complex engineering problems by applying engineering and management principles to achieve the goal of the project.

100003/CS6 22T.5-PO3	HIGH	Apply engineering and management principles to achieve the goal of the project and to design solutions for complex engineering problems and design system components or processes that meet the specified needs.
100003/CS6 22T.5-PO4	MEDIUM	Apply engineering and management principles to achieve the goal of the project and use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS6 22T.5-PO5	MEDIUM	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and to apply engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PO6	MEDIUM	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities by applying engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PO7	MEDIUM	Understand the impact of the professional engineering solutions in societal and environmental contexts, and apply engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PO8	HIGH	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice and to use the engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PO9	MEDIUM	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings and to apply engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PO11	MEDIUM	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team. Manage projects in multidisciplinary environments and to apply engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PO12	HIGH	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change and to apply engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PSO1	MEDIUM	The ability to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas. Apply engineering and management principles to achieve the goal of the project.

100003/CS6 22T.5-PSO2	MEDIUM	The ability to acquire programming efficiency by designing algorithms and applying standard practices in software project development to deliver quality software products meeting the demands of the industry and to apply engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PSO3	MEDIUM	The ability to apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs thereby evolving as an eminent researcher and entrepreneur and apply engineering and management principles to achieve the goal of the project.

