Indian Institute of Technology

Gandhinagar

Capgemini Internship

:Internship Report:
Group 2 - Team TriGypsy

Team Members	**Email-id**
Siddharth Soni	siddharth.soni@iitgn.ac.in
Janvi Thakkar	janvi.thakkar@iitgn.ac.in
Kumar Ayush Paramhans	ayush.kumar@iitgn.ac.in

Supervisor	**Mentors**
Mr. Ashish Buch	Mr. Rajkumar Koradiya
	Ms. Bhavani Kajuluri
	Ms. Rinkal Amin

Contents

1 Problem Statement

The problem statement for our team is as follows:

Impact analysis of software change leveraging ML, NLP models from huge number of test cases and requirement documents written in English using Python, TensorFlow / Elmo or an equivalent model.

2 Various models and their Approach

There are various language models available today. Our team had done an enormous amount of research on the existing models, their advantages, disadvantages and drawbacks before finalising our model.

The various models that we went through have been discussed below:

2.1 WORD2VEC

This model is useful in processing unstructured text data. It converts every input element into vectors on which we can perform mathematical operations to predict the next word.

Some of its advantages are mentioned below:

- It converts unlabelled raw data into labelled data. This data can be passed as an input to the model with very little memory through online methods.
- It tries to establish a sub-linear relationship between target and context words, like, king:: man and queen:: woman.

Some of its advantages have been discussed below:

- However, such relationships are defined only for a few words.
- It is unable to distinguish between similar words used in a different context.

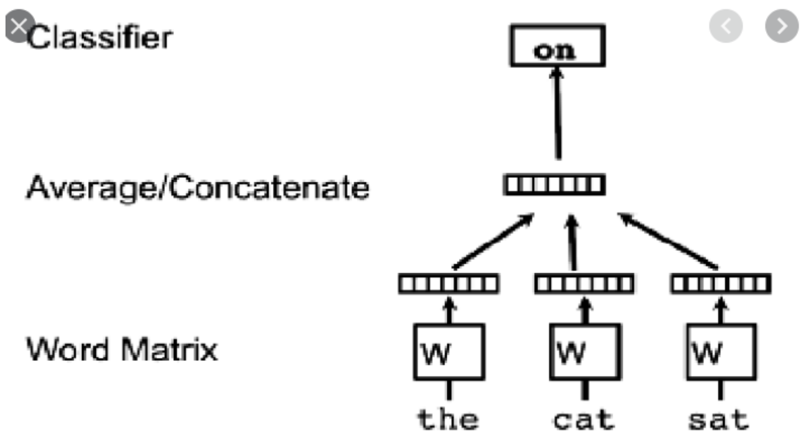

Figure 1: Framework of the vectors of Word2Vec(1)

Figure 2: Mathematical operations on word vectors(2)

2.2 GLOVE

The model derives its name from Global Vectors. This model trains its vectors to learn from the frequency of the different words occurring together in a large document. It is a count/frequency-based model.

Some of its advantages are discussed below:

- It improves upon Word2Vec by using word vectors to establish sub-linear relationships.
- Some words like 'a, an, is, the, for, etc.' (also known as stopwords) are of not much use. They are given lower weights to increase the efficiency of the model.

The disadvantages of the model are mentioned below:

- GLOVE uses a co-occurrence matrix for word vectors. This consumes a lot of memory. Besides, if we change either of the parameters, we will have to reconstruct the matrix again.
- The model is context-independent, i.e., this model is also unable to distinguish between words based on context.

2.3 ELMo

ELMo briefly stands for Embeddings from Language Models. This model is one of the greatest advancements on the pre-existing models.

The main advantages of this model are:

- It is able to distinguish between words based on context. It is better suited for text similarity. It uses character convolution which helps the model to handle out of vocabulary words.
- It can perform bi-directional training of the word vectors and it uses LSTM architecture to train the model.
- LSTM models do not have any sentence length limits as opposed to Transformer architecture in BERT.

Some of the disadvantages of this model are:

- In the LSTM approach, the sentence is processed word by word, whereas in the Transformer approach, the sentence is processed as a single unit. So, while back-tracing the error in the LSTM approach, it takes time as the model needs to check the error word by word.

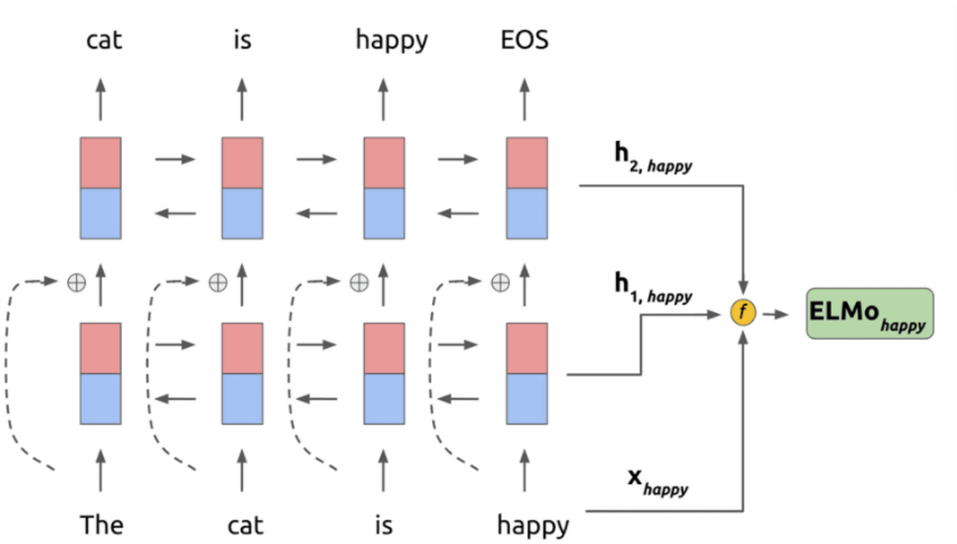

Figure 3: Framework of ELMo(4)

2.4 BERT

BERT(Bidirectional Encoder Representations from Transformers) uses the bidirectional LSTM / Autoencoder (AE) language model.

Some of its advantages are given below:

- The AE language model can use the words from both forward and backward directions at the same time to predict the next set of words.

Similarly, the disadvantages of this model are:

- This model uses the concept of 'MASK' to hide some words from the input data in the pretraining phase. This masked/hidden word is to be predicted in the next phase. However, it causes discrepancy during the time of fine tuning as such masks are absent in the real data set. Another disadvantage is that it assumes the predicted words (masked tokens) are independent of each other.

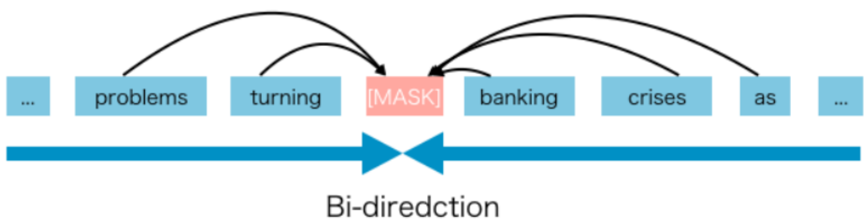

Figure 4: Ideology behind the architecture of BERT(3)

2.5 XLNet

XLNet is the latest development in the NLP models. This model has additional features than BERT.

The advantages possessed by this model are:

- It is an Autoregressive language model (AR model) which uses the context word (this can come from either forward or backward direction) to predict the next word. Hence, it is good at sentence prediction.

The disadvantages of using this model are:

- This model cannot use both forward and backward context words at the same time to predict the next word.

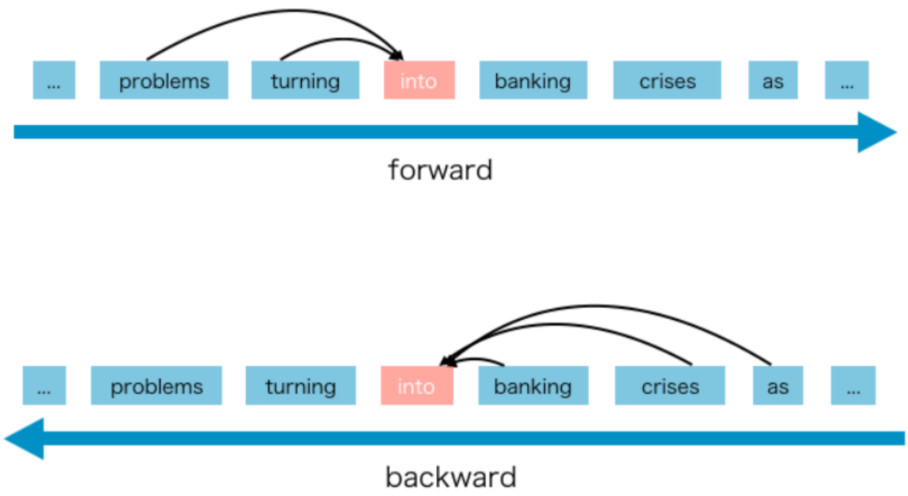

Figure 5: Ideology behind the architecture of XLNet(3)

The language model is based on permutation.

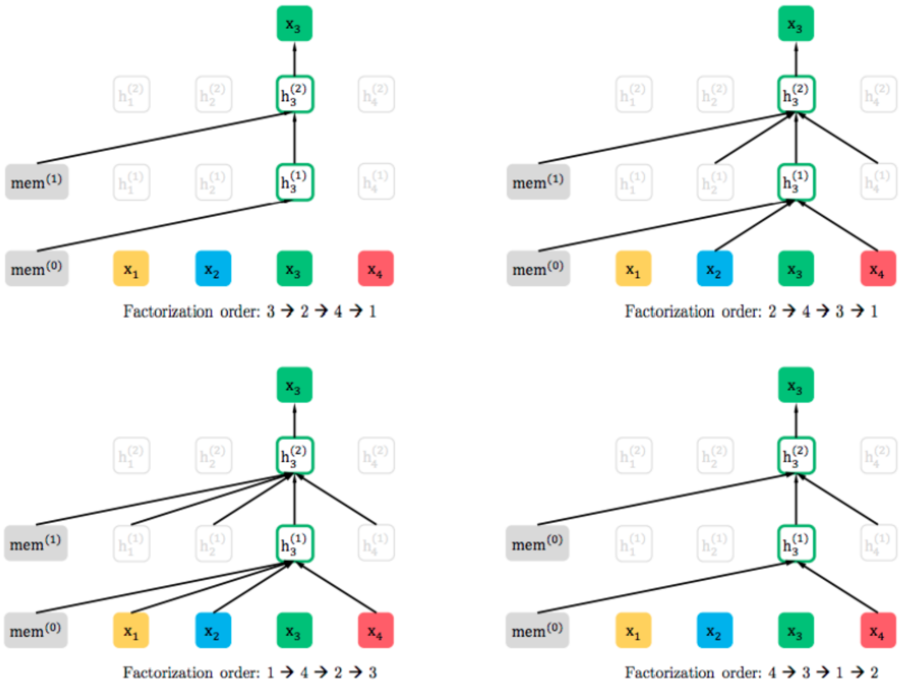

Figure 6: Permutation in the architecture of XLNet(3)

2.6 USE

For the USE model, please refer to the next section for a descriptive detail.

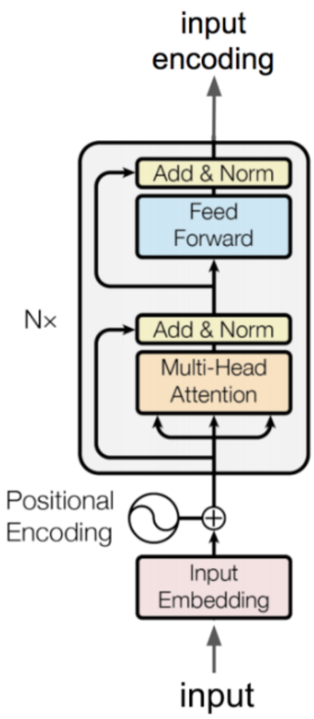

Figure 7: Transformer architecture of USE(5)

3 Why USE Model ?

Universal Sentence Encoder (USE) uses the transformer architecture while Embeddings from Language Models (ELMo) uses LSTM architecture. Transformer architecture is made for comparing the similarity between two texts whereas LSTM architecture focuses on predicting the next words in a sentence by analyzing the previous words. Tasks related to contextual similarity prefers to use the USE model upon ELMo. Further, in LSTM approach a sentence is processed word by word whereas in transformer, a sentence is processed as a single unit. therefore, the contextual meaning can be analyzed by a machine more accurately in a complete sentence processing.

The claim that USE provides better results in comparing the similarity between sentences was verified by using a sample data containing some simple English sentences.

When the ELMo model was used to build the embeddings of the sentences, the following results were obtained. When the keyword "how old are you?" was searched for the similarity, the similarity score was 0.65 with the sentence "what is your age?".

```python
from sklearn.metrics.pairwise import cosine_similarity
search_string = "how old are you?"

print("SEARCH STRING: ", search_string)
try:
  no = int(search_string)
  if no in test_ids:
    embeddings2 = [test_ids[int(search_string)]]
  else:
    print("You entered an invalid test ID")
except:
  embeddings2 = embed(tf.convert_to_tensor([search_string]))

# print("Enter the number of results expected")
results_returned = 10                             # showing 5 best matched test cases
# if results_returned >= count:
#   results_returned = count

output =[]
cosine_similarities = pd.Series(cosine_similarity(embeddings2, embeddings).flatten())
print("ID  ", "SENTENCE   ", "SIMILARITY INDEX")
for i,j in cosine_similarities.nlargest(int(results_returned)).iteritems():
  print(index_keeper[i],"  ",  text_d[index_keeper[i]],"  ", j)
```

```
SEARCH STRING:  how old are you?
ID   SENTENCE    SIMILARITY INDEX
15    how old are you?    1.000000238418579
18    where is your younger brother?    0.6567085981369019
17    what is your age?    0.6529829502105713
19    your are older than your brother.    0.6238949298858643
16    you must be the elder one, here is your toy.    0.6157844662666321
6    How about getting a new phone as it do not work accordingly.    0.6002612709999084
7    Your cellphone looks great, what model is it?    0.5886045694351196
5    I think that my cell phone is better than yours.    0.5665239095687866
11    If you don't mind can we have another cup of soup.    0.5321239233016968
4    I like my phone and it is the best.    0.5279172658920288
```

Figure 8: ELMo results

whereas USE model showed the similarity score of 0.90.

```python
from sklearn.metrics.pairwise import cosine_similarity
search_string = "how old are you?"

print("SEARCH STRING: ", search_string)
try:
  no = int(search_string)
  if no in test_ids:
    embeddings2 = [test_ids[int(search_string)]]
  else:
    print("You entered an invalid test ID")
except:
  embeddings2 = embed(tf.convert_to_tensor([search_string]))

# print("Enter the number of results expected")
results_returned = 10                             # showing 5 best matched test cases
# if results_returned >= count:
#   results_returned = count

output =[]
cosine_similarities = pd.Series(cosine_similarity(embeddings2, embeddings).flatten())
print("ID  ", "SENTENCE   ", "SIMILARITY INDEX")
for i,j in cosine_similarities.nlargest(int(results_returned)).iteritems():
  print(index_keeper[i],"  ",  text_d[index_keeper[i]],"  ", j)
```

```
SEARCH STRING:  how old are you?
ID   SENTENCE    SIMILARITY INDEX
15    how old are you?    1.0
17    what is your age?    0.9015963673591614
19    your are older than your brother.    0.4381609261035919
18    where is your younger brother?    0.33963343501091003
16    you must be the elder one, here is your toy.    0.2782174050807953
7    Your cellphone looks great, what model is it?    0.270521879196167
6    How about getting a new phone as it do not work accordingly.    0.1835881471633911
12    Collect your plaything firtborn.    0.17248454689979553
5    I think that my cell phone is better than yours.    0.13325488567352295
14    This toy is for your big brother.    0.10688968002796173
```

Figure 9: USE results

We know that the two compared sentences are same in semantic meaning. Therfore, USE model performs better than the ELMo in considering the similarity between two texts.

4 End-to-End architecture

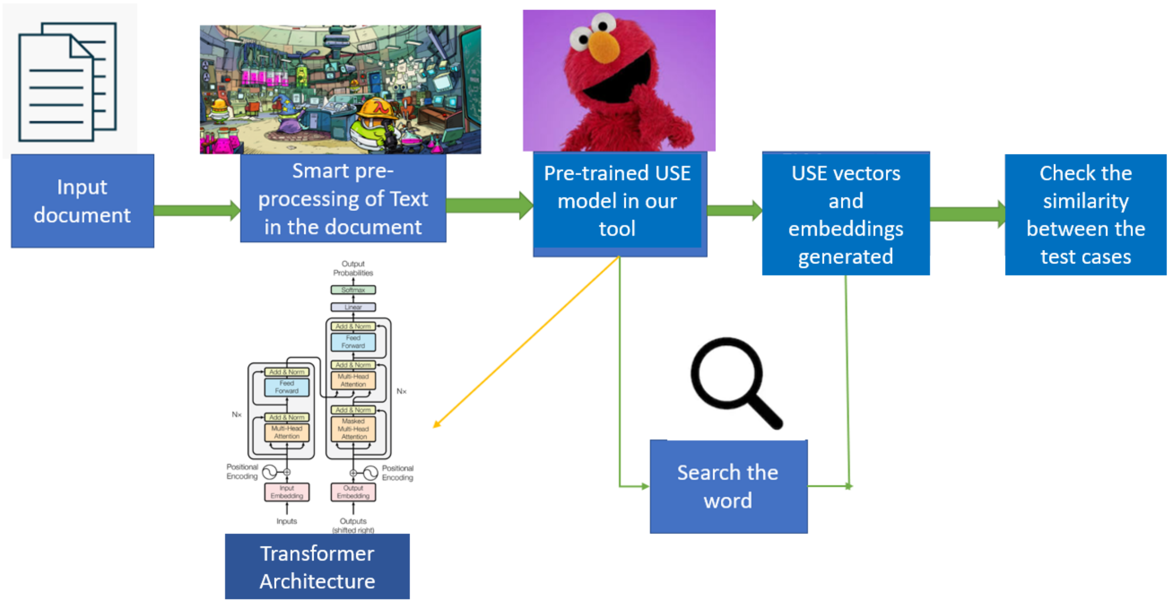

Figure 10: End-to-End architecture(6)

4.1 Methodology

There are basically three main steps in our model.

- Pre-processing of the documents (unique to the format of a document).
- Creating their embeddings using Universal Sentence Encoder(USE) model.
- Searching the most semantic similar document based on phrases or Test IDs.

4.2 Working

- We take a .docx document as our input (containing hundreds of Test IDs) and separate each Test ID by creating unique .txt file.
- We then create embeddings by reading all .txt files and save the embeddings in **embeddings.npz** file and their corresponding ids in **id.npz** file.
- The search engine involves searching for the Test IDs most similar to the given search criteria. The search engine allows the user to search using 3 different methods:
  - **Based on Test ID:** This will return all the id's similar to this Test ID based on the threshold set by the user.
  - **Based on Keyword:** This will return all test id's with most semantic similarity to the keyword.

  – **Based on Test ID+Keyword:** This search allows user to search all test cases similar to particular test id and then allow user to search based on keyword from all the test id's returned from the first test id search. In brief, this allows the user to search among only those test cases which user thinks to be relevant at first. This increases accuracy and also decreases the amount of time as we are narrowing our search criteria.

4.3 Python files involved in Model

- In **pre_process.py**, we are creating a .txt file for every unique test ids and also using USE model to create the embeddings for the test id and saving it in **embeddings.npz** file and their ids in **id.npz** file.
- In **update_emb.py**, we are updating only particular embedding, based on the file name, that is unique id.txt it finds the id from **id.npz** and change the embeddings in **embeddings.npz** file based on that particular index.
- In **automate_cre.py**, we are monitoring the directory **main_document**, this allows user to create embeddings automatically. While this file is running, the user can add all the main files in the directory, when the process terminates, **mainfile.data** will be generated which contains the path to the directory.
- In **convert_to_csv.py**, this file stores all the test ids returned in .csv file, this refrains users to search the same keyword again.
- In **graph_plot.py**, the embeddings or precisely the 'vectors' of the test cases (formed by using the USE model) are reduced to 2 components of dimensionality. The dimensionality reduction is done by using Principal Component Analysis (PCA) algorithm. The 2-dimensional embeddings are plotted on a graph for getting a look of similarity between different test cases.
- In **search_engine.py**, we have two types of the search engine:
  – For **keyword/test-id**: The user can enter either a keyword or a test-id to search the semantically matching test cases. The search engine asks the user to set a minimum threshold value (in the range of 0-1) for the search. It then uses the concept of cosine similarity to find the matching test cases. The output contains the matching test cases in decreasing order of similarity from starting towards the end. If the output is empty, the user gets the option to still find a given number of matching test cases irrespective of the threshold.
  – For **keyword+test-id**: This is the second search engine. The user needs to first enter a keyword, then enter the test-id in the next line for the search. The user gets the option of threshold here as well. If the output is empty, then the user has the option to obtain the matching test cases based on either a keyword or a test-id.

4.4 GUI - User Interface

Apart from the option of using the command-line interface for searching through the document, the user also gets an access to GUI as well. The GUI is for all the three types of searches, viz., keyword or test-id or keyword+testid. In case, if the user forgets to enter any threshold value, then the search engine will take the default value from the 'threshold.csv' file. The GUI is shown below:

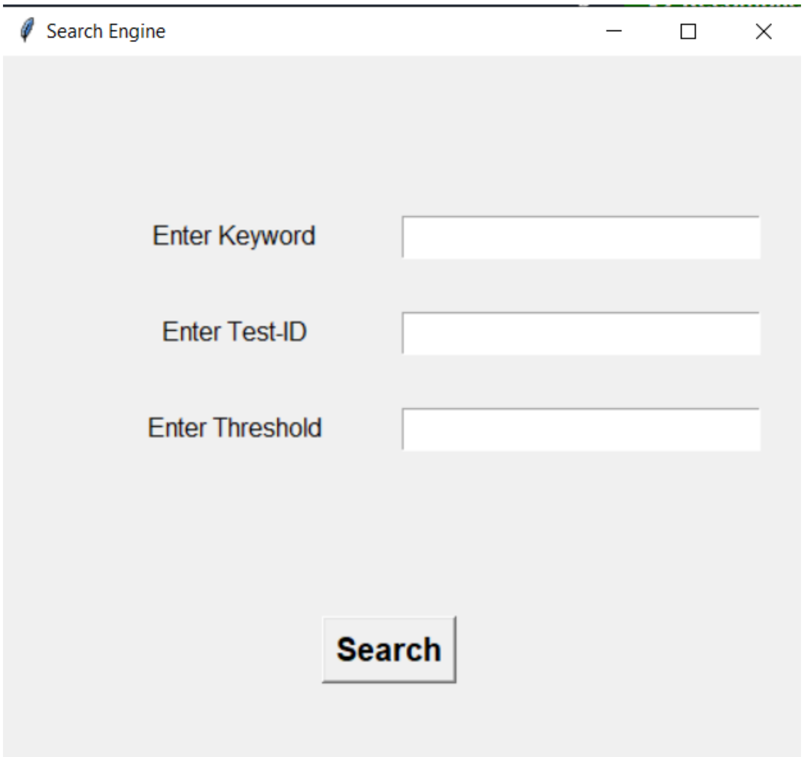

Figure 11: The GUI for end-user

4.5 How to set the optimum threshold value

As described in section 3 we can deduce some optimum values of threshold for the three different types of searches (only keyword, only Test-ID and Both).

- Keyword: All the test cases have a large number of words which are not comparable to some entered keywords. So a good threshold is in the range of 0.01 - 0.05.
- Test-ID: The words in the entered string (a test case itself) and other test cases are comparable so a good or optimum threshold is in the range of 0.4 - 0.7.
- Searching for Both: The optimum threshold ranges from 0.03 - 0.08.

Note: If the output provided contains a large number of test-ids, then increase the threshold in all the three cases to get top similar results.

4.6 Plot and its features

The user gets the feature of a graph, where he can observe the similairty between the cases, using the concept of the search engine. If the user wants to observe one particular ID in the graph along with its surroundings, then he can just enter the test-id in the textbox and view the particular test case with its similar matching test cases. A view of the graph is shown below:

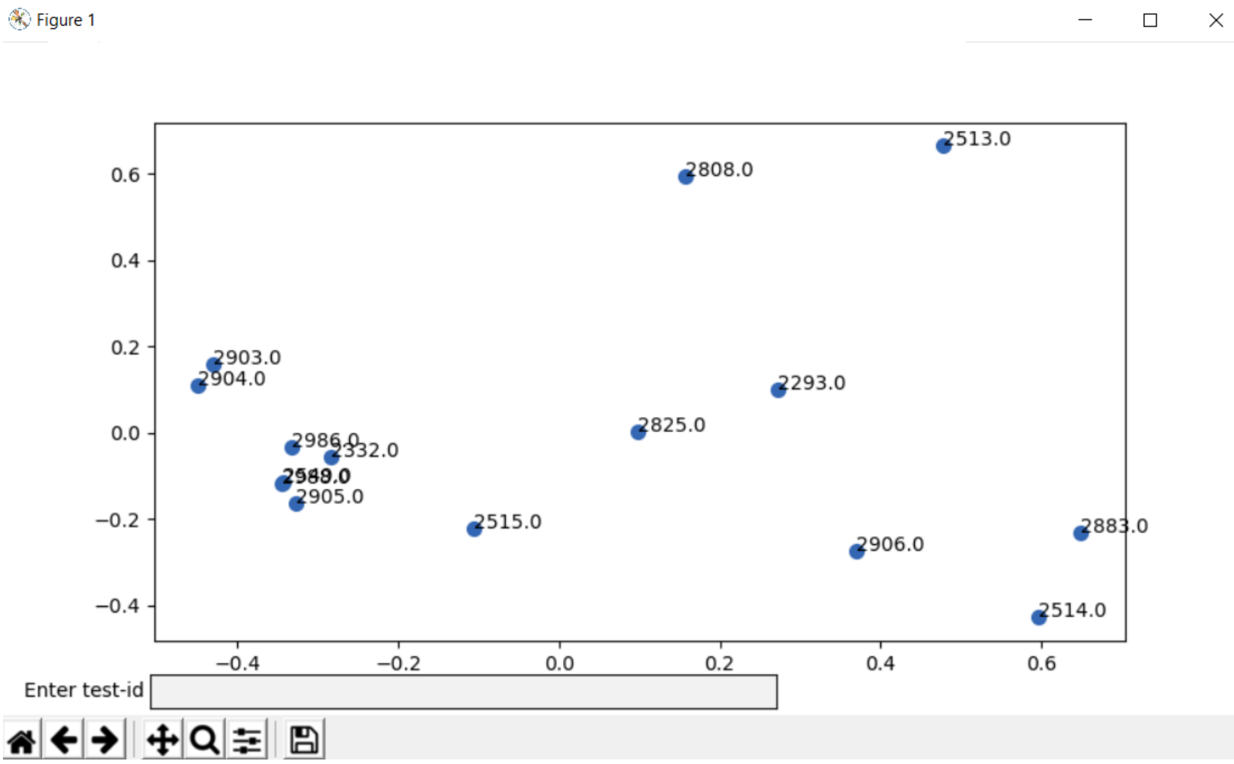

Figure 12: The graph for end-user

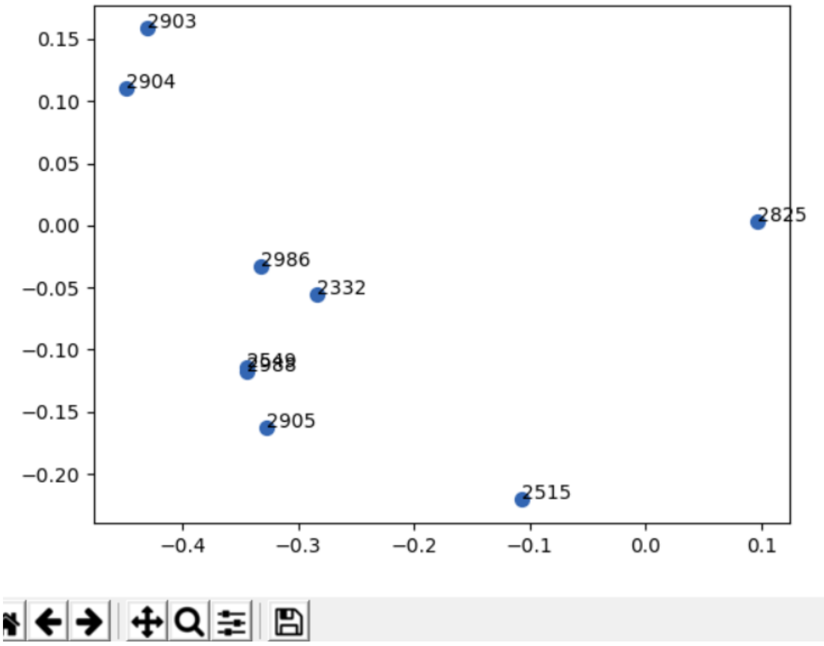

Figure 13: Graph showing similar test cases to test-id 2905

5 Steps to install the product on your device

The link appended below contains the steps to install the tool on your local environment. It is in the 'README.md' section.

Link: `https://github.com/SoniSiddharth/Impact_analysis`

The link also contains the code files associated with the tool.

6 Explanation of different python modules

6.1 Numpy

Numpy is a library for the Python. it is used for the operations which involves multi-dimensional arrays and analysis. it involves high dimensional arrays and matrices along with high-level mathematical functions to operate on these. However, in this project its use was very much limited.

6.2 Pandas

Pandas is a library for Python programming language. it is used for data manipulation and analysis. it is used for the cleaning of data (data pre-processing phase). it is majorly used for Machine Learning and Natural Language Processing in form of dataframes.

6.3 Tensorflow

Tensorflow is one of the most famous libraries used for Machine learning applications, for example: neural networks. tensorflow has different functions and classes for high-level mathematical calculations.

6.4 Tensorflow-Hub

Tensorflow-hub is a library used for loading different machine learning models. The library is for the consumption of reusable parts of ML models.

6.5 Spacy

Spacy is a library written in Python and Cython, it is used for advanced Natural Language Processing. it can also be used for Deep learning and neural networks.

6.6 Watchdog

Watchdog is an open source library of Python used for keeping a track of operations occurring in a directory. It's function is the same as its name suggests.

6.7 Textract

Textract is a library in Python used for the processing of texts and data.

6.8 Matplotlib

Matplotlib is a library in Python to plot various types of graphs.

6.9 Scikitlearn

Scikitlearn is a module in Python for Machine learning. It is an extension of Scipy library in python.

7 Type of License for the Python Modules (11)

The license type of the python modules involved in the project are as follows:

Citation: The description given in the tables below has been taken from the output of a command used in python to get the license description. The relevant sources have been mentioned in the 'Bibliography section' as reference number '10' (mentioned in the heading also).

The versions mentioned in the table are the latest versions as on 25/05/2020. The version number may vary depending on the date a user has installed the python module.

Name	Version	License	Description
Numpy	1.18.4	BSD	Fundamental package for array computation with Python
Pandas	1.0.3	BSD	Data structures for data analysis, time series and statistics
TensorFlow	2.2.0	Apache License 2.0	Open-source software for high performance numerical computation
TensorFlow Hub	0.8.0	Apache 2.0	To foster, publication, discovery and consumption of reusable parts of ML models
spaCy	2.2.4	MIT	Library for advanced NLP in Python
Watchdog	0.10.2	Apache License 2.0	Filesystem events monitoring
Textract	1.6.3	MIT	Extract text from any document
Matplotlib	3.2.1	PSF	Library for creating visual graphs with Python
Scikit-learn	0.23.1	OSI approved	Library for ML operations
random2	1.0.1	PSF	It is python 3 version of random module of Python 2.7

8 Vote of Thanks

We would like to give our vote of thanks to **Mr. Ashish Buch** for his constant support throughout the project. Besides, we would also like to appreciate Mr. Raj, Ms. Bhavani, Ms. Rinkal for providing us with the tasks every week to be implemented in our model. They, assisted us in our model by providing the user requirements from their side.
We would also like to thank our end-users Mr. Mihir and Mr. Mayur for their feedback on the tool.

9 Bibliography

References

[1] Figure for Word2Vec: https://towardsdatascience.com/deep-transfer-learning-for-natural-language-processing-text-classification-with-universal-1a2c69e5baa9

[2] Information on Word2Vec vectors: https://www.linkedin.com/pulse/short-introduction-using-word2vec-text-classification-mike

[3] Information on XLNet and BERT: https://towardsdatascience.com/what-is-xlnet-and-why-it-outperforms-bert-8d8fce710335

[4] Figure for framework of ELMo: https://medium.com/saarthi-ai/elmo-for-contextual-word-embedding-for-text-classification-24c9693b0045

[5] Figure for the architecture of USE: https://www.dlology.com/blog/keras-meets-universal-sentence-encoder-transfer-learning-for-text-data/

[6] Figure for End-to-End architecture:
- https://group.thinkproject.com/en/solutions/project-collaboration/document-management/
- https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908
- https://www.lightbox.co.nz/series/sesame-street-count-on-elmo
- https://www.shutterstock.com/search/searching
- http://proofthatblog.com/2013/07/01/the-question-is-what-happened-to-the-question-mark/
- https://ireneli.eu/2018/12/17/elmo-in-practice/
- https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04

[7] Advantages and Disadvantages of Word2Vec and Glove: https://www.quora.com/What-are-the-advantages-and-disadvantages-of-Word2vec-and-GloVe

[8] Information on GLOVE: https://medium.com/sciforce/word-vectors-in-natural-language-processing-global-vectors-glove-51339db89639

[9] Information on ELMo: https://www.quora.com/What-are-the-main-differences-between-the-word-embeddings-of-ELMo-BERT-Word2vec-and-GloVe

[10] Proximity_plotter. py by Mr. Mihir Darji

[11] License type of python files with description:
- https://pypi.org/project/pip-licenses/
- https://pypi.org/project/textract/
- https://pypi.org/project/tensorflow/
- https://pypi.org/project/spacy/
- https://pypi.org/project/matplotlib/

- https://pypi.org/project/scikit-learn/
- https://pypi.org/project/random2/

[12] LateX table generator: https://www.tablesgenerator.com/

[13] https://towardsdatascience.com/elmo-contextual-language-embedding-335de2268604

[14] All the resources that we have referred till date can be found in the references section of each of the Weekly Reports that has been shared with the mentors.