NAME : Hardik Mittal
ROLL NUMBER : 2021114016

2.1) <u>Theory</u>

1. **Explain negative sampling. How do we approximate the word2vec training computation using this technique?**

In the case of fully fledged word2vec, we would apply the Softmax activation function given by

$$p(w|c) = \frac{\exp(h^T v'_w)}{\sum_{w_i \in V} \exp(h^T v'_{w_i})}$$

onto the final output of the MLP, and then use a standard loss metric like Cross Entropy Loss. However, this operation is extremely expensive, since it requires summing over all words in V (the vocabulary), which is generally quite large. Negative Sampling is a technique that approximates the normalization in the denominator of the softmax with a different loss function that is cheap to compute. The loss here is mathematically defined as:

$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$$

The first term tries to maximize the probability of occurrence for actual words that lie in the context window, i.e. they co-occur. The second term tries to iterate over some random words j that don’t lie in the window and minimize their probability of co- occurrence. We sample random words based on their frequency of occurrence:

$$P(w) = \frac{U(w)^{\frac{3}{4}}}{Z}$$

Where U(w) is the unigram frequency of the word w and Z is the normalizing constant. We can see that instead of computing the summation over every word in the vocabulary like in the softmax function, we instead sample only a few, say k , words from the vocabulary based on the above distribution and sum over those instead. This drastically reduces the number of computations we need to perform. However, we should choose k cleverly to get the best approximations. It has been found that it is an effective strategy to choose small values of k for large datasets (say 2-5), while for smaller datasets, a relatively large value is preferred (say 5-20)

2. **Explain the concept of semantic similarity and how it is measured using word embeddings. Describe at least two techniques for measuring semantic similarity using word embeddings.**

Semantic similarity is an important concept in natural language processing and refers to how similar two words or phrases are in meaning. The idea is that words that are similar in meaning should be close to each other in some sort of representation. Word embeddings are one of the most popular ways to represent words and measure semantic similarity.

Word embeddings are dense vector representations of words in a high-dimensional space, where similar words are represented by vectors that are close to each other. These vectors are typically learned through unsupervised learning techniques, such as word2vec or GloVe, by training on large corpora of text data. The resulting word embeddings capture many of the semantic relationships between words and can be used for various natural language processing tasks, including measuring semantic similarity.

To measure semantic similarity using word embeddings, we can use a number of techniques. Two commonly used methods are cosine similarity and Euclidean distance. Cosine similarity is a measure of the similarity between two vectors in a high-dimensional space. Given two word vectors, the cosine similarity is calculated as the cosine of the angle between the vectors. A value of 1 indicates that the vectors are identical, while a value of 0 indicates that the vectors are orthogonal (i.e., they have no similarity).

Euclidean distance, on the other hand, is a measure of the distance between two points in a high-dimensional space. Given two word vectors, the Euclidean distance is calculated as the square root of the sum of the squared differences between the corresponding elements of the vectors. A smaller distance indicates a higher similarity between the two vectors.

Both cosine similarity and Euclidean distance are commonly used to measure semantic similarity between words. In practice, the choice between these two methods depends on the specific task and the particular use case. For instance, cosine similarity is often used in information retrieval and text classification, while Euclidean distance is preferred in clustering and topic modeling tasks.

In conclusion, measuring semantic similarity is an important task in natural language processing and can be done using various techniques, including cosine similarity, Euclidean distance, Word Mover's Distance, and Soft Cosine Similarity. These methods rely on word embeddings, which are dense vector representations of words in a high-dimensional space. By measuring the similarity between word vectors, we can capture many of the semantic relationships between words and use them for various natural language processing tasks.

2.2) Implementation

Model 1 - Using Singular Value Decomposition

- The model is trained using Singular Value Decomposition on Co-Occurrence matrix to create the embeddings for all the words in the training corpus

Hyperparameters used

S.No.	Name	Value	Reason
1	Number of Reviews Used	40,000	Computational Constraints, else could have much bigger corpus
2	Window Size	30	The bigger the window size, the more generalized meaning is contained in the vectors
3	Embedding Size	300	So that the vectors contain enough information about the word, but at the same time keeps computational resources in mind as well
4	Minimum Frequency of Tokens	5	To remove the unnecessary words

`

- **Corpus :**
    - The corpus used is a bit different from the given corpus (it is a subset of what was given) since anyways the whole data was not being used.
    - Only the `reviewText` part of the corpus is being used to train the model.
    - Very basic cleaning of the corpus is being done, like lowering the case of words & removing the punctuations.
    - Afterwards, indices and vocab are created from that data.

- **Co-occurrence Matrix & Creating Embeddings**
    - The co-occurrence matrix does exactly what it sounds : it is a matrix containing the words and the times they are occurring with other words in the range of `WINDOW_SIZE`
    - Using SVD on the co occurrence matrix, the vectors are created for all the words in the training corpus

- **Plotting of the Graphs and Finding the Closest Similar Words**
    - I have used cosine on the embeddings of the words to find their similarity. The word having maximum cosine similarity with the target word is ideally the closest to it semantically.
    - Since the embedding size is 300, which cannot be plotted on a graph, I have used T-SNE to reduce the dimensions from 300 to 2, for easier visualization.

2.2) Implementation

Model 2 - Using Continuous Bag of Words (CBOW)

- The model is trained using CBOW which is a prediction based method to create the embeddings for all the words in the training corpus

Hyperparameters used

S.No.	Name	Value	Reason
1	Number of Reviews Used	40,000	Computational Constraints, else could have much bigger corpus
2	Window Size	30	The bigger the window size, the more generalized meaning is contained in the vectors
3	Embedding Size	300	So that the vectors contain enough information about the word, but at the same time keeps computational resources in mind as well
4	Minimum Frequency of Tokens	5	To remove the unnecessary words
5	Batch Size	64	Decent tradeoff between computation power and the accuracy
6	Negative Sample Size per batch	30	Since there are 60 positive samples per word
7	Learning Rate	0.001	Neither to fast nor too slow
8	Number of Epochs	5	Lack of computational power

`

- **Corpus :**
  - The corpus used is a bit different from the given corpus (it is a subset of what was given) since anyways the whole data was not being used.
  - Only the `reviewText` part of the corpus is being used to train the model.
  - Very basic cleaning of the corpus is being done, like lowering the case of words & removing the punctuations.
  - Afterwards, indices and vocab are created from that data.
  - At the end, `dataset` is being created which is a list of tuples, where each tuple contains a list of the context of a target word and the target word

- **Model :**
  - A very basic neural model having an embedding layer, and a linear layer is being used.
  - A `log_softmax` function is used at the end of the linear layer to predict the probabilities

- **Finding the Closest Similar Words**
    - I have used cosine on the embeddings of the words to find their similarity. The word having maximum cosine similarity with the target word is ideally the closest to it semantically.

2.3) Analysis

2.3.1) . Display the top-10 word vectors for five different words (a combination of nouns, verbs, adjectives, etc.) using t-SNE (or such methods) on a 2D plot.

The 5 words chosen are :

- Horror
- Good
- Kill
- Hero
- Action

The results for the SVD model were as follows : (trained on 40,000 sentences)

- horror : [('scary', 0.109581), ('genre', 0.101445), ('gore', 0.090937), ('fans', 0.082512), ('slasher', 0.068409), ('classic', 0.061048), ('halloween', 0.059283), ('suspense', 0.056013), ('flick', 0.053359)]

- good : [('pretty', 0.072421), ('excellent', 0.055329), ('bad', 0.049266), ('decent', 0.042857), ('fine', 0.041185), ('much', 0.041035), ('guys', 0.036309), ('actually', 0.030077), ('cool', 0.025375)]

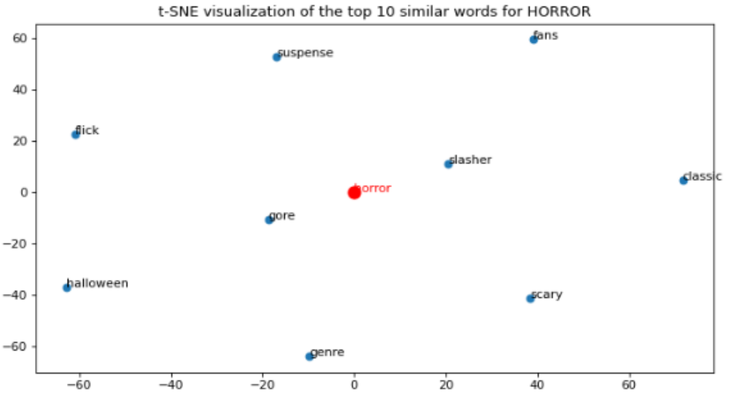
- kill : [('freddy', 0.081079), ('peck', 0.076839), ('gregory', 0.065182), ('shark', 0.057481), ('atticus', 0.045754), ('body', 0.043652), ('mockingbird', 0.043337), ('finch', 0.036733), ('killed', 0.035828)]

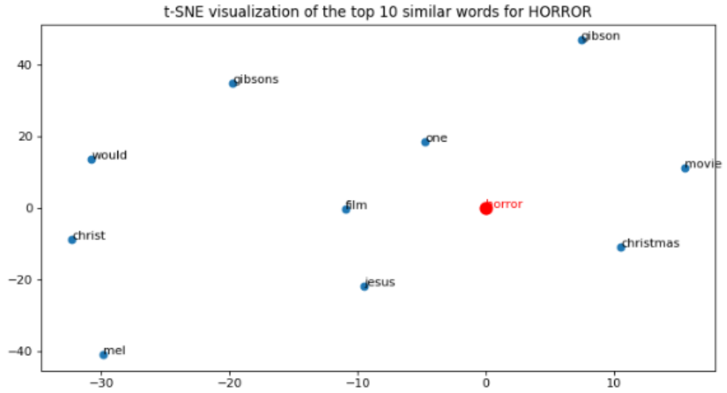
- hero : [('action', 0.078851), ('zorro', 0.043116), ('travis', 0.034361), ('guy', 0.034308), ('last', 0.030787), ('sequences', 0.030557), ('driver', 0.026989), ('arnold', 0.025081), ('slater', 0.024916)]

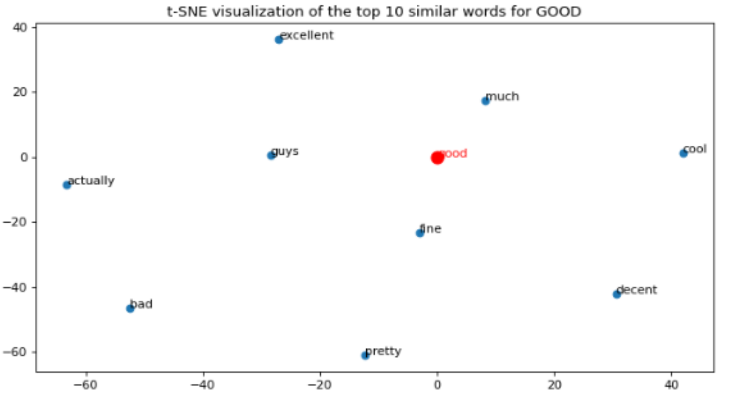
- action : [('sequences', 0.091087), ('deleted', 0.089136), ('drama', 0.086863), ('hero', 0.078851), ('suspense', 0.063428), ('adventure', 0.054727), ('plenty', 0.046997), ('lots', 0.04572), ('blade', 0.044186)]

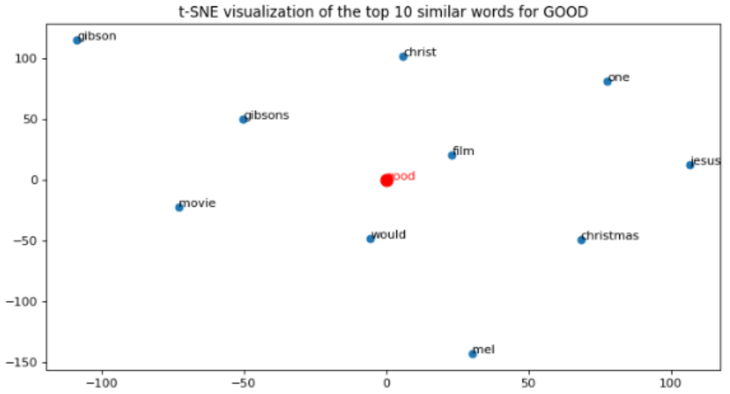
Trained on 40, 000 reviews

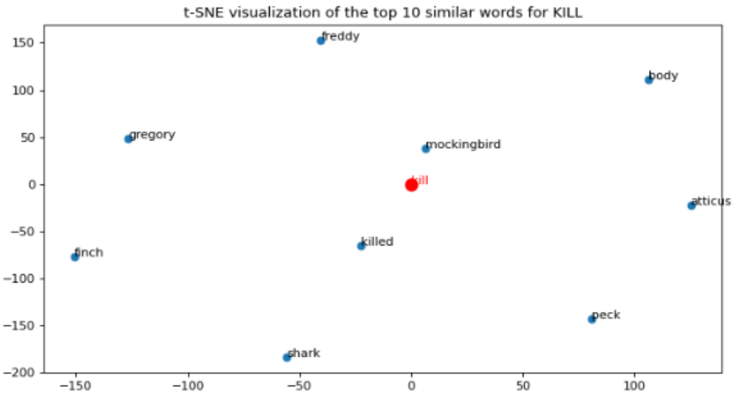
Trained on 500 reviews

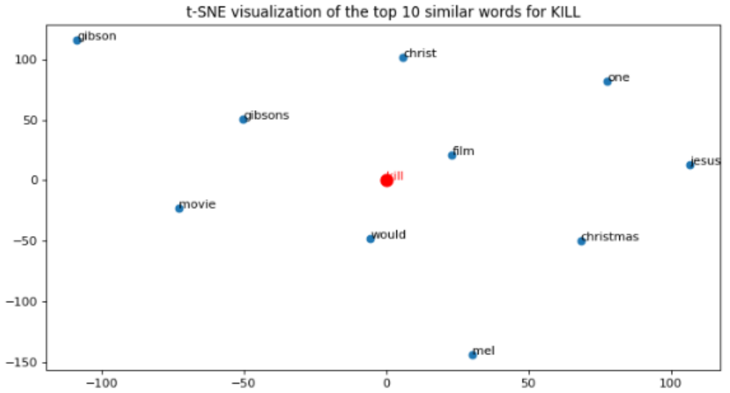


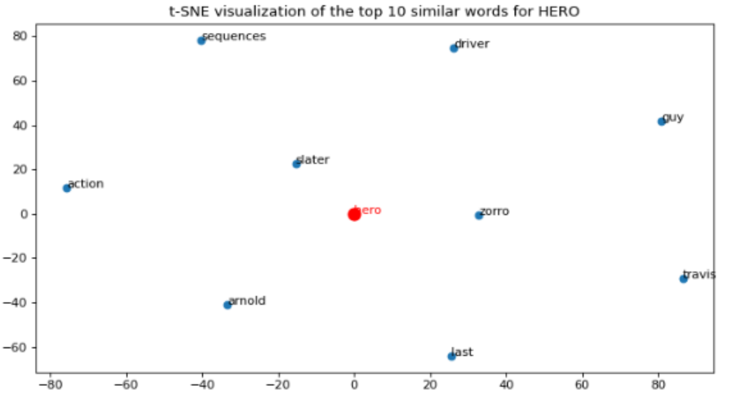


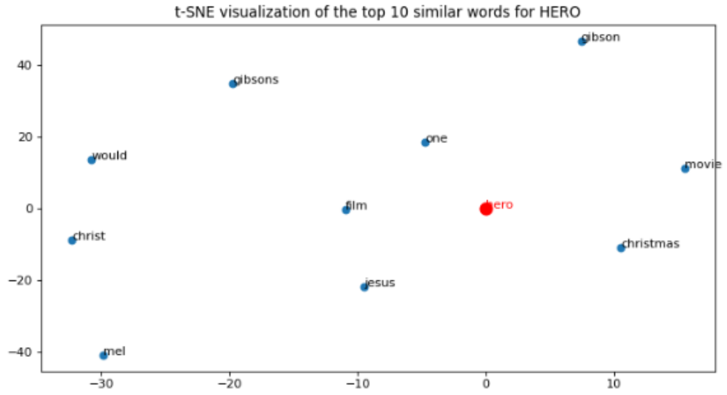


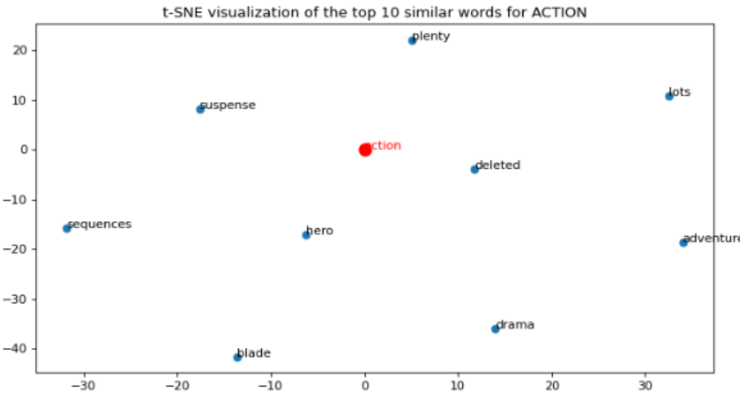


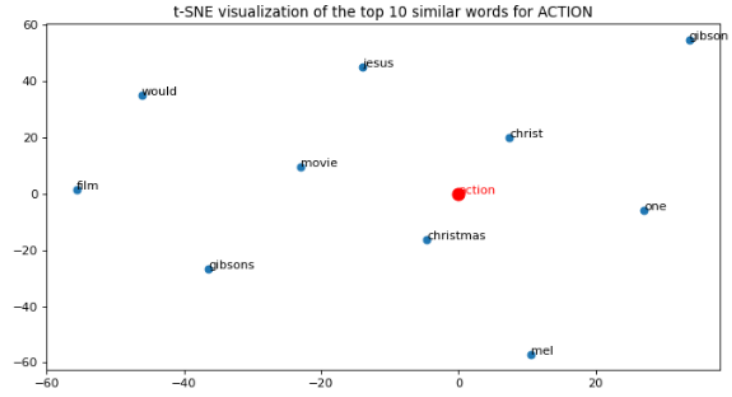






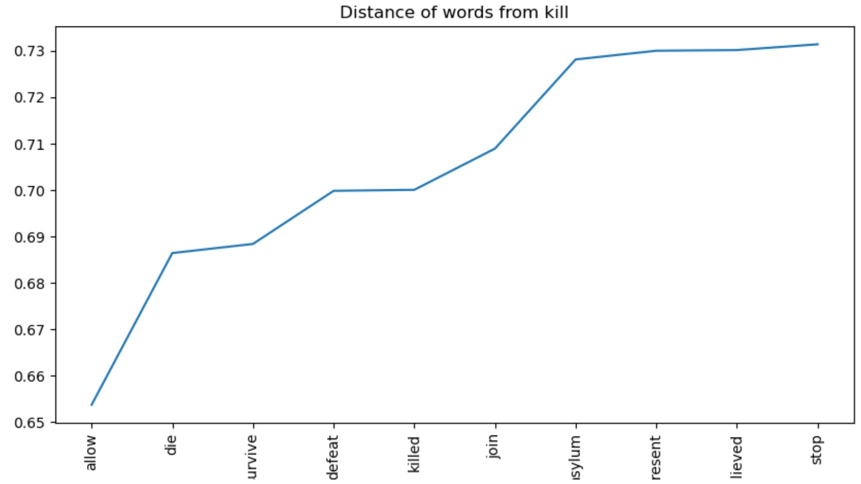


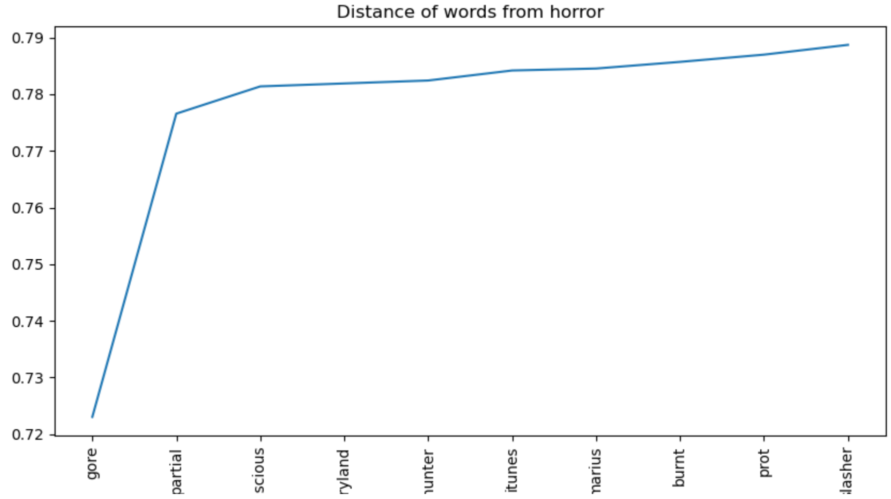


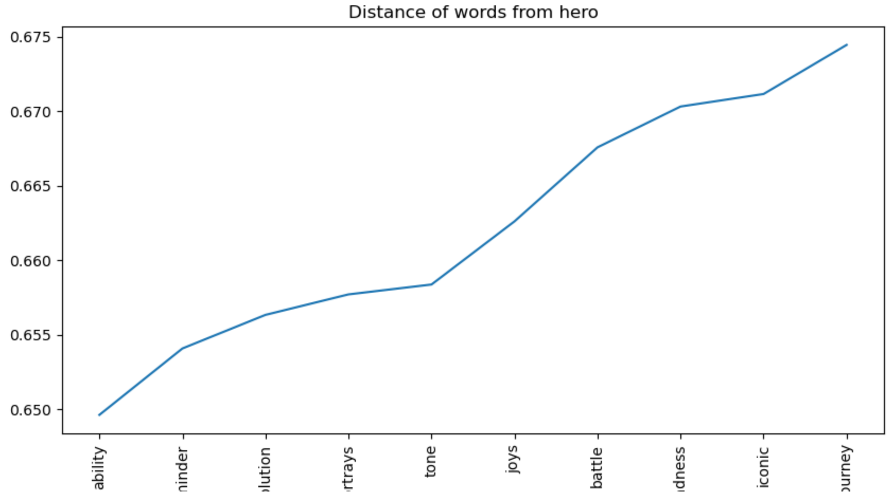


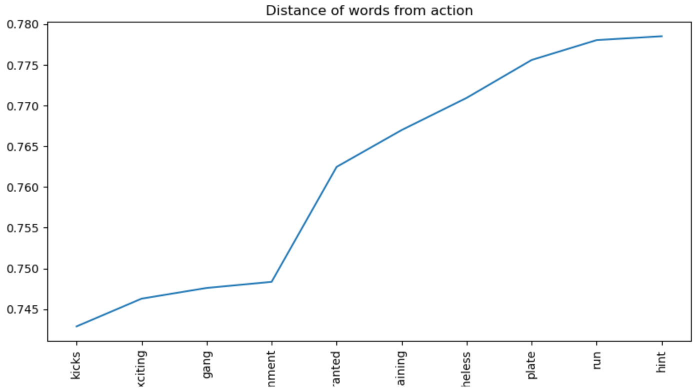
Here, the right image indicates the embeddings of the words when the model is trained on 500 reviews and the ;eft image indicates the embeddings of the words when the model is trained on 40,000 reviews. It is clearly visible that the model has improved a lot and thus the learnt embeddings are going in a right direction

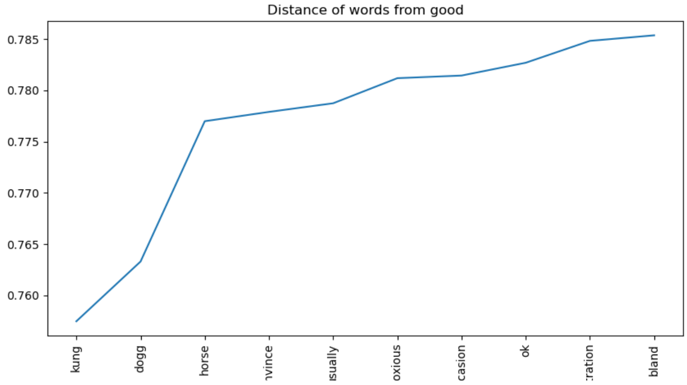
The results for the CBOW model were as follows :

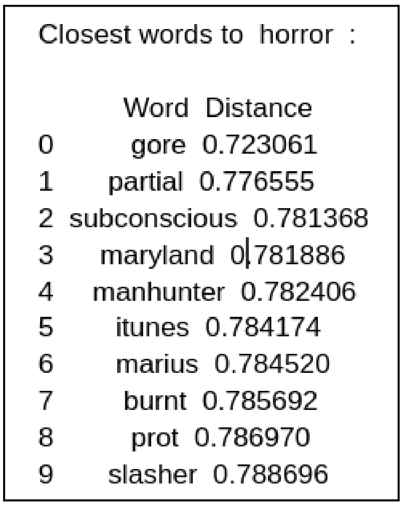






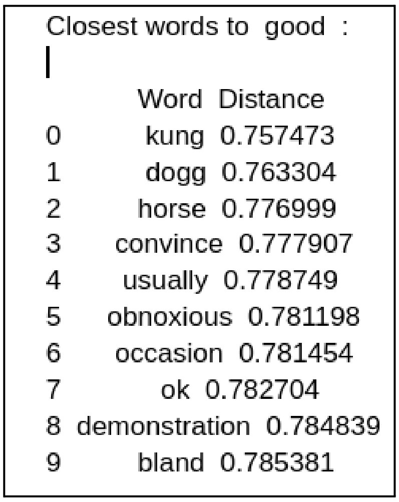






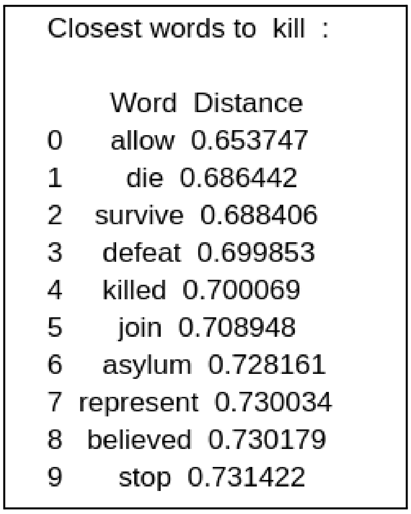
```
Closest words to  horror  :

          Word  Distance
0         gore  0.723061
1      partial  0.776555
2  subconscious  0.781368
3     maryland  0.781886
4    manhunter  0.782406
5       itunes  0.784174
6       marius  0.784520
7        burnt  0.785692
8         prot  0.786970
9      slasher  0.788696
```



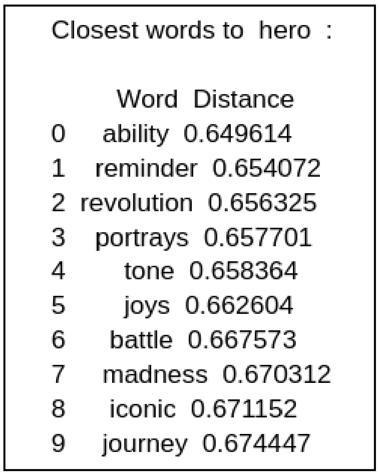
```
Closest words to  good  :

            Word  Distance
0           kung  0.757473
1           dogg  0.763304
2          horse  0.776999
3       convince  0.777907
4        usually  0.778749
5      obnoxious  0.781198
6       occasion  0.781454
7             ok  0.782704
8  demonstration  0.784839
9          bland  0.785381
```



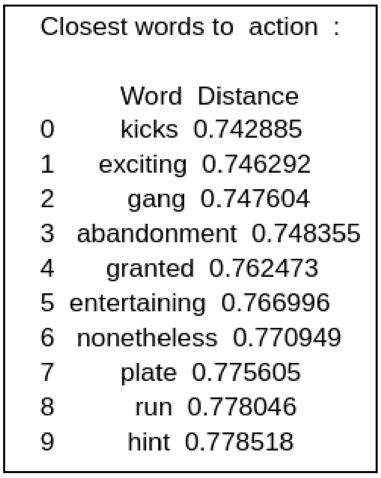
```
Closest words to  kill  :

         Word  Distance
0       allow  0.653747
1         die  0.686442
2     survive  0.688406
3      defeat  0.699853
4      killed  0.700069
5        join  0.708948
6      asylum  0.728161
7   represent  0.730034
8    believed  0.730179
9        stop  0.731422
```



```
Closest words to  hero  :

         Word  Distance
0     ability  0.649614
1    reminder  0.654072
2  revolution  0.656325
3    portrays  0.657701
4        tone  0.658364
5        joys  0.662604
6      battle  0.667573
7     madness  0.670312
8      iconic  0.671152
9     journey  0.674447
```

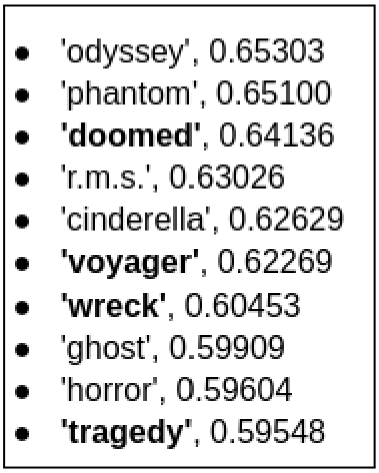


```
Closest words to  action  :

           Word  Distance
0         kicks  0.742885
1      exciting  0.746292
2          gang  0.747604
3   abandonment  0.748355
4       granted  0.762473
5  entertaining  0.766996
6   nonetheless  0.770949
7         plate  0.775605
8           run  0.778046
9          hint  0.778518
```

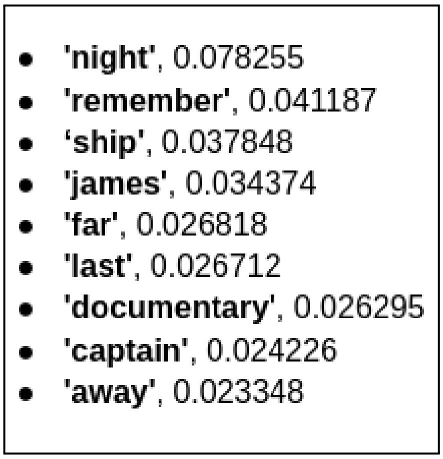
2.3.2) . What are the top 10 closest words for the word ‘titanic’ in the embeddings generated by your program? Compare them against the pre-trained word2vec embeddings that you can download off-the-shelf.

Glove Pretrained Model



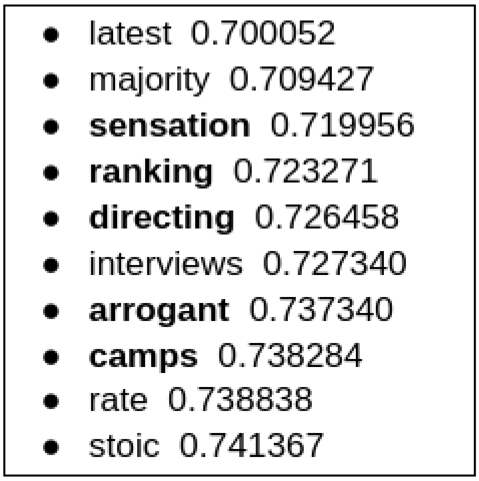
- 'odyssey', 0.65303
- 'phantom', 0.65100
- **'doomed'**, 0.64136
- 'r.m.s.', 0.63026
- 'cinderella', 0.62629
- **'voyager'**, 0.62269
- **'wreck'**, 0.60453
- 'ghost', 0.59909
- 'horror', 0.59604
- **'tragedy'**, 0.59548

SVD



- **'night'**, 0.078255
- **'remember'**, 0.041187
- **‘ship'**, 0.037848
- **'james'**, 0.034374
- **'far'**, 0.026818
- **'last'**, 0.026712
- **'documentary'**, 0.026295
- **'captain'**, 0.024226
- **'away'**, 0.023348

CBOW



- latest 0.700052
- majority 0.709427
- **sensation** 0.719956
- **ranking** 0.723271
- **directing** 0.726458
- interviews 0.727340
- **arrogant** 0.737340
- **camps** 0.738284
- rate 0.738838
- stoic 0.741367

As we can see all the results given by the SVD model, still make sense on with respect to the movie ‘Titanic’, where only 4 out of 10 words from gensim’s pretrained model make sense with the original movie

Although, the CBOW model did not perform as well as expected, especially as compared to the SVD model.

In case the results are not uploaded due to lack computational resources, the final report will be available on the following github link : https://github.com/mhardik003/NLP_Assignment3.git