REPORT - NLP ASSIGNMENT 3

Name: Akshara B

Roll no: 211AI012

Datasets used:

1. https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews

Task 1: Pre-Processing the text

First, we download the csv file of dataset and extract the sentences from 'text' column

Downloading Text Corpus

```
!pip install opendatasets --upgrade
import opendatasets as od
dataset_url = 'https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews'
od.download(dataset_url)
```

```
Requirement already satisfied: opendatasets in /usr/local/lib/python3.10/dist-packages (0.1.22)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from opendatasets) (4.66.1)
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (from opendatasets) (1.5.16)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from opendatasets) (8.1.7)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (1.16.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (2023.7.22)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (2.31.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (8.0.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (2.0.6)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (6.0.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle->opendatasets) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle->opendatasets) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle->opendatasets) (3.3.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle->opendatasets) (3.4)
Skipping, found downloaded files in "./amazon-fine-food-reviews" (use force=True to force download)
```

```
data=pd.read_csv("/content/amazon-fine-food-reviews/Reviews.csv")
```

```
data
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Qual

preprocessing the English sentences dataset containing 568454 amazon reviews

568452	568453	B004I613EE	A3IBEVCTXKNOH	Kathy A. Welch "katwel"	1	1	5	1331596800
568453	568454	B001LR2CU2	A3LGQPJCZVL9UC	srfell17	0	0	5	1338422400

568454 rows × 10 columns

```
corpus_text = '\n'.join(data[:50000]['Text'])
lemmatizer = WordNetLemmatizer()
```

```
t_data=[]
for i in sent_tokenize(corpus_text):
  temp=[]
  tokens=word_tokenize(i)
  lemmatized_tokens=[lemmatizer.lemmatize(token) for token in tokens]
  for j in lemmatized_tokens:
    if(j.isalnum()):
      temp.append(j.lower())
  t_data.append(temp)
```

```
t_data
```

```
'most',
'other',
'flour',
'is',
'nice',
'and',
'chewy',
'the',
'same',
```

t_data is tokenized sentences data. Each item in t_data is tokenized sentence.

Task 2: Building word2vec model

WORD2VEC WORD EMBEDDINGS

```
[ ] model1=Word2Vec(t_data,min_count=1,vector_size=100,window=5,sg=0)
    model2=Word2Vec(t_data,min_count=1,vector_size=100,window=5,sg=1)
```

```
[ ] #print('similarity between two words is ',model1.wv.similarity('highly','recommend'))
    print('similarity between two words is ',model2.wv.similarity('highly','recommend'))

    similarity between two words is  0.77576715
```

```
[ ] print('similarity between two words is ',model2.wv.similarity('tea','coffee'))

    similarity between two words is  0.6400501
```

```
[ ] embedding = model2.wv['recommend']
    print(f"Embedding for '{'recommend'}':\n{embedding}")

    Embedding for 'recommend':
    [-0.41012725  0.33869237 -0.5080207  -0.17469695  0.6328387  -1.0314417
     -0.27159083  0.12155338 -0.59288603 -0.39580616  0.0960302  -0.4875321
      0.7118886  -0.03852042 -0.05909441 -0.30176657  0.67341983 -0.4019283
     -0.44032663 -1.4359605   0.2554614   0.03020154  0.22151908 -0.263304
     -0.36152905 -0.10858025 -0.03071206 -0.43985462 -0.1958771  -0.52328426
      0.52491     0.13550341 -0.01175502 -0.32755888 -0.785504    0.7268203
     -0.17600393 -0.40390185 -0.7756213   0.6444981  -0.03955012 -1.0056684
     -0.1254981   0.35939384  0.0095896   0.63096607 -1.0160226  -0.18214352
      0.946967    0.36166185 -0.27851245  0.50123096 -0.31324187 -0.21374281
      0.16541177  0.07369463 -0.4006274  -0.35190085 -0.18771462  0.2119276
     -0.07123286 -0.24851067  0.01692431  0.20866403 -0.0233308   0.14644043
      0.06863366  0.713811   -0.27466762  0.1957885   0.00458886  0.18238312
      0.78654855 -0.42929548  0.06531107  0.24262783  0.16879173 -0.37192488
      0.11346351 -0.08352185 -0.03344381  0.28778487 -0.16743922 -0.21237758
      0.17945975 -0.19905747  0.18172199  0.2777019   0.21995959 -0.22666152
     -0.7904144   0.02900792  0.37151238  0.30947277  0.4396951  -0.13326128
     -0.16913392 -0.00600272  0.3329618  -0.06904718]
```

I built the Word2Vec model using genism.models.Word2Vec

I built both CBOW and SG models with vector_size as 100 and window=5

SG model works better for my dataset

I found similarity between words like ‘highly’,’recommend’ and ‘tea’,’coffee’

Next I found sementic textual similarity between two sentences using this model

First sentences are tokenized and then sentence vector is obtained by taking mean of each individual word2vec embeddings of tokens. Since all word2vec embedding of same dimension(vector_size=100) we don’t need any padding

We then fine cosine similarity of sentence vectors

Semantic Textual Similarity

```python
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np



sentence1 = "This Product is highly recommended."
sentence2 = "I like the product."


from nltk.tokenize import word_tokenize

def tokenize_and_preprocess_text(sentence):
    tokens = word_tokenize(sentence)
    tokens=[lemmatizer.lemmatize(token) for token in tokens]
    tokens = [token.lower() for token in tokens if token.isalnum()]

    return tokens


tokens1 = tokenize_and_preprocess_text(sentence1)
tokens2 = tokenize_and_preprocess_text(sentence2)

def get_sentence_vector(tokens, model):
    # Filter out tokens that are not in the model's vocabulary
    if not tokens:
        return np.zeros(model.vector_size)
    return np.mean(model.wv[tokens], axis=0)

vector1 = get_sentence_vector(tokens1, model2)
vector2 = get_sentence_vector(tokens2, model2)

# Calculate the cosine similarity between the two sentence vectors
similarity = cosine_similarity([vector1], [vector2])[0][0]

print(f"Cosine Similarity: {similarity}")
```

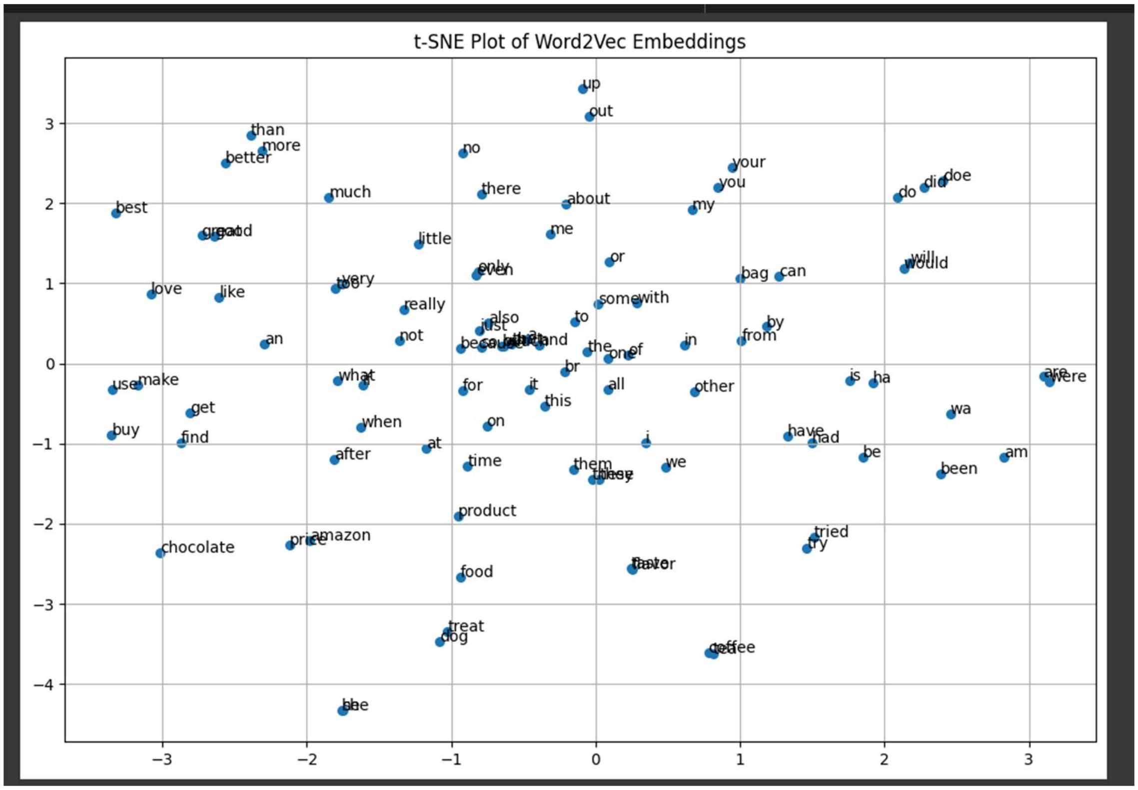
```python
print(f"Cosine Similarity: {similarity}")
```

```
Cosine Similarity: 0.7871932983398438
```

Then I plotted the T_sne plot for first 100 words in models vocabulary

T_SNE

```python
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
from gensim.models import Word2Vec


# Get word vectors and corresponding words from the model
words = list(model2.wv.index_to_key)
words=words[:100]
word_vectors = [model2.wv[word] for word in words]

# Convert word_vectors to a NumPy array
word_vectors = np.array(word_vectors)

# Perform t-SNE dimensionality reduction
tsne = TSNE(n_components=2, random_state=42)
word_vectors_2d = tsne.fit_transform(word_vectors)

# Create a scatter plot
plt.figure(figsize=(12, 8))
plt.scatter(word_vectors_2d[:, 0], word_vectors_2d[:, 1], marker='o', s=30)

# Label some points for reference (optional)
sample_words = words[:100]  # Label the first 5 words from your list
for word, (x, y) in zip(sample_words, word_vectors_2d[:100]):
    plt.annotate(word, (x, y))

# # Label some points for reference (optional)
# sample_words = ['word1', 'word2', 'word3']  # Replace with words from your model
# for word in sample_words:
#     idx = words.index(word)
#     plt.annotate(word, (word_vectors_2d[idx, 0], word_vectors_2d[idx, 1]))

# Show the plot
plt.title('t-SNE Plot of Word2Vec Embeddings')
plt.grid(True)
plt.show()
```



Here we can see that words like (great,good) and words like (tea,coffee) are close to each other which means they are more similar

Task 3: Building GloVe model

```
[ ] import os
    import urllib.request
    import matplotlib.pyplot as plt
    from scipy import spatial
    from sklearn.manifold import TSNE
    import numpy as np
```

```
[ ] from tensorflow.keras.preprocessing.text import Tokenizer
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    import numpy as np



    !wget http://nlp.stanford.edu/data/glove.6B.zip
    !unzip glove*.zip
```

```
--2023-10-11 04:40:54--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2023-10-11 04:40:54--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
```

First we download GloVe embeddings from Stanford website

```
replace glove.6B.200d.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace glove.6B.300d.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
```

```python
# create the dict.
x=[token for token in list(i for i in t_data)]
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x)

# number of unique words in dict.
print("Number of unique words in dictionary=",
  len(tokenizer.word_index))
print("Dictionary is = ", tokenizer.word_index)
```

```
Number of unique words in dictionary= 35605
Dictionary is =  {'the': 1, 'i': 2, 'a': 3, 'and': 4, 'it': 5, 'to': 6, 'of': 7, 'is': 8, 'this': 9, 'br': 10, 'for': 11, 'in': 12, 'that': 13
```

```python
def embedding_for_vocab(filepath, word_index,
            embedding_dim):
  vocab_size = len(word_index) + 1

  # Adding again 1 because of reserved 0 index
  embedding_matrix_vocab = np.zeros((vocab_size,
                embedding_dim))

  with open(filepath, encoding="utf8") as f:
    for line in f:
      word, *vector = line.split()
      if word in word_index:
        idx = word_index[word]
        embedding_matrix_vocab[idx] = np.array(
          vector, dtype=np.float32)[:embedding_dim]
```

Connected to Python 3 Google Compute Engine backend

Then we create a dictionary which maps each word token in datasets vocabulary to a specific index. This dictionary will be later used to get embeddings for words

```python
def embedding_for_vocab(filepath, word_index,
            embedding_dim):
  vocab_size = len(word_index) + 1

  # Adding again 1 because of reserved 0 index
  embedding_matrix_vocab = np.zeros((vocab_size,
                embedding_dim))

  with open(filepath, encoding="utf8") as f:
    for line in f:
      word, *vector = line.split()
      if word in word_index:
        idx = word_index[word]
        embedding_matrix_vocab[idx] = np.array(
          vector, dtype=np.float32)[:embedding_dim]

  return embedding_matrix_vocab


# matrix for vocab: word_index
embedding_dim = 50
embedding_matrix_vocab = embedding_for_vocab(
  '/content/glove.6B.50d.txt', tokenizer.word_index,
embedding_dim)

print("Dense vector for first word 'the' is => ",
  embedding_matrix_vocab[1])
```

```
Dense vector for first word 'the' is =>  [ 4.18000013e-01  2.49679998e-01 -4.12420005e-01  1.21699996e-01
  3.45270008e-01 -4.44569997e-02 -4.96879995e-01 -1.78619996e-01
 -6.60229998e-04 -6.56599998e-01  2.78430015e-01 -1.47670001e-01
 -5.56770027e-01  1.46579996e-01 -9.50950012e-03  1.16579998e-02
  1.02040000e-01 -1.27920002e-01 -8.44299972e-01 -1.21809997e-01
 -1.68009996e-02 -3.32789987e-01 -1.55200005e-01 -2.31309995e-01
 -1.91809997e-01 -1.88230002e+00 -7.67459989e-01  9.90509987e-02
 -4.21249986e-01 -1.95260003e-01  4.00710011e+00 -1.85939997e-01
 -5.22870004e-01 -3.16810012e-01  5.92130003e-04  7.44489999e-03
  1.77780002e-01 -1.58969998e-01  1.20409997e-02 -5.42230010e-02
```

Connected to Python 3 Google Compute Engine backend

Then we create a embedding _vocab_matrix where each row is a word in dataset vocabulary and the columns are dimensions. Every row represents vector of given word. We get the embeddings from the Glove 50d text file and 50 is our chosen embedding dimesion.

```
    embedding_matrix_vocab[1])
```

```
Dense vector for first word 'the' is =>  [ 4.18000013e-01  2.49679998e-01 -4.12420005e-01  1.21699996e-01
  3.45270008e-01 -4.44569997e-02 -4.96879995e-01 -1.78619996e-01
 -6.60229998e-04 -6.56599998e-01  2.78430015e-01 -1.47670001e-01
 -5.56770027e-01  1.46579996e-01 -9.50950012e-03  1.16579998e-02
  1.02040000e-01 -1.27920002e-01 -8.44299972e-01 -1.21809997e-01
 -1.68009996e-02 -3.32789987e-01 -1.55200005e-01 -2.31309995e-01
 -1.91809997e-01 -1.88230002e+00 -7.67459989e-01  9.90509987e-02
 -4.21249986e-01 -1.95260003e-01  4.00710011e+00 -1.85939997e-01
 -5.22870004e-01 -3.16810012e-01  5.92130003e-04  7.44489999e-03
  1.77780002e-01 -1.58969998e-01  1.20409997e-02 -5.42230010e-02
 -2.98709989e-01 -1.57490000e-01 -3.47579986e-01 -4.56370004e-02
 -4.42510009e-01  1.87849998e-01  2.78489990e-03 -1.84110001e-01
 -1.15139998e-01 -7.85809994e-01]
```

```python
from sklearn.metrics.pairwise import cosine_similarity
```

```python
from sklearn.metrics.pairwise import cosine_similarity

word_to_find = 'good'
if word_to_find in tokenizer.word_index:
    idx = tokenizer.word_index[word_to_find]
    # Access the word embedding vector for 'good'
    embedding_of_good = embedding_matrix_vocab[idx]
    print(f"Word embedding vector for '{word_to_find}':\n{embedding_of_good}")
else:
    print(f"'{word_to_find}' not found in the vocabulary.")

# Calculate similarity between two words (e.g., 'good' and 'excellent')
word1 = 'good'
word2 = 'excellent'
if word1 in tokenizer.word_index and word2 in tokenizer.word_index:
    idx1 = tokenizer.word_index[word1]
    idx2 = tokenizer.word_index[word2]
    embedding1 = embedding_matrix_vocab[idx1]
    embedding2 = embedding_matrix_vocab[idx2]
    similarity = cosine_similarity([embedding1], [embedding2])[0][0]
    print(f"Similarity between '{word1}' and '{word2}': {similarity}")
else:
    print("One or both of the words not found in the vocabulary.")
```

```
Word embedding vector for 'good':
[-3.55859995e-01  5.21300018e-01 -6.10700011e-01 -3.01310003e-01
  9.48620021e-01 -3.15389991e-01 -5.98309994e-01  1.21880002e-01
 -3.19430009e-02  5.56949973e-01 -1.06210001e-01  6.33989990e-01
 -4.73399997e-01 -7.58949965e-02  3.82470012e-01  8.15690011e-02
  8.22139978e-01  2.22200006e-01 -8.37639999e-03 -7.66200006e-01
 -5.62529981e-01  6.17590010e-01  2.02920005e-01 -4.85979989e-02
  8.78149986e-01 -1.65489995e+00 -7.74179995e-01  1.54349998e-01
  9.48230028e-01 -3.95200014e-01  3.73020005e+00  8.28549981e-01
 -1.41039997e-01  1.63950007e-02  2.11150005e-01 -3.60849984e-02
 -1.55870005e-01  8.65830004e-01  2.63090014e-01 -7.10150003e-01
 -3.67700011e-02  1.82819995e-03 -1.77039996e-01  2.70319998e-01
  1.10260002e-01  1.41330004e-01 -5.73219992e-02  2.72069991e-01
  3.13050002e-01  9.27709997e-01]
Similarity between 'good' and 'excellent': 0.8061552559026371
```

Next we fine cosine similarity of two words ‘good’ and ‘excellent’ we see that it performs better than word2Vec

After that we find semntic similarity of sentences by converting sentnces to sentence vector and finding cosine similiarity between them.

We see that it performed better than word2vec for same two sentences

After that we plot the t_sne plot for 100 words in vocabulary

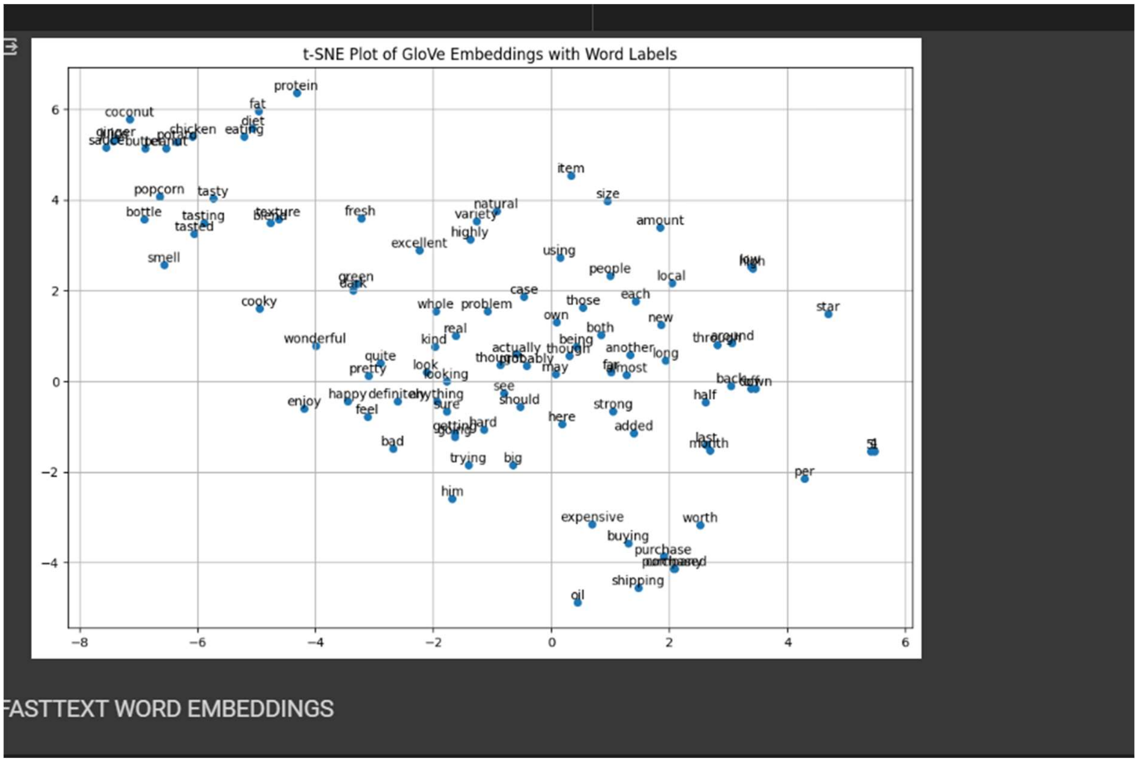
STS

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np



sentence1 = "This Product is highly recommended."
sentence2 = "I like the product."


from nltk.tokenize import word_tokenize

def tokenize_and_preprocess_text(sentence):
    tokens = word_tokenize(sentence)
    tokens=[lemmatizer.lemmatize(token) for token in tokens]
    tokens = [token.lower() for token in tokens if token.isalnum()]

    return tokens


tokens1 = tokenize_and_preprocess_text(sentence1)
tokens2 = tokenize_and_preprocess_text(sentence2)

def get_sentence_vector(tokens, model):
    # Filter out tokens that are not in the model's vocabulary
    if not tokens:
        return np.zeros(model.vector_size)
    return np.mean(list(embedding_matrix_vocab[tokenizer.word_index[word]] for word in tokens), axis=0)

vector1 = get_sentence_vector(tokens1, model2)
vector2 = get_sentence_vector(tokens2, model2)

# Calculate the cosine similarity between the two sentence vectors
similarity = cosine_similarity([vector1], [vector2])[0][0]

print(f"Cosine Similarity: {similarity}")
```

```
Cosine Similarity: 0.8792564659981774
```

```
# Load pre-trained GloVe embeddings
embedding_path = '/content/glove.6B.50d.txt'
word_embeddings = {}
with open(embedding_path, 'r', encoding='utf8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype='float32')
        word_embeddings[word] = vector

# Get word vectors and corresponding words from the GloVe model
words = list(tokenizer.word_index.keys())
words = words[200:300]  # Limit to the 100 words for the example
word_vectors = [embedding_matrix_vocab[tokenizer.word_index[word]] for word in words]

# Convert word_vectors to a NumPy array
word_vectors = np.array(word_vectors)

# Perform t-SNE dimensionality reduction
tsne = TSNE(n_components=2, random_state=42)
word_vectors_2d = tsne.fit_transform(word_vectors)

# Create a scatter plot
plt.figure(figsize=(12, 8))
plt.scatter(word_vectors_2d[:, 0], word_vectors_2d[:, 1], marker='o', s=30)

# Label points with word labels
for word, (x, y) in zip(words, word_vectors_2d):
    plt.text(x, y, word, fontsize=10, ha='center', va='bottom')

# Show the plot
plt.title('t-SNE Plot of GloVe Embeddings with Word Labels')
plt.grid(True)
plt.show()
```

t-SNE Plot of GloVe Embeddings with Word Labels



Task 4: Building FastText model

FASTTEXT WORD EMBEDDINGS

```
from gensim.models import FastText

model3=FastText(t_data,min_count=1,vector_size=100,window=5,sg=1, workers = 4, min_n = 1, max_n = 4)
```

```
print('similarity between two words is ',model3.wv.similarity('highly','recommend'))
print('similarity between two words is ',model3.wv.similarity('good','excellent'))
print('similarity between two words is ',model3.wv.similarity('tea','coffee'))
```

```
similarity between two words is  0.810702
similarity between two words is  0.74474615
similarity between two words is  0.73450196
```

Next I trained fastext model on 50k sentences using vector_dimensions as 100 context window as 5 and cpu cores as 4 and skipgram and min_ngram=1 and max_ngram=4

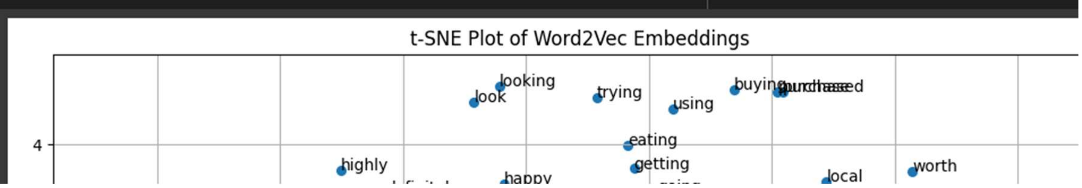
We see the similarity between two same words was better than what we got from word2vec but lesser compared to GloVe

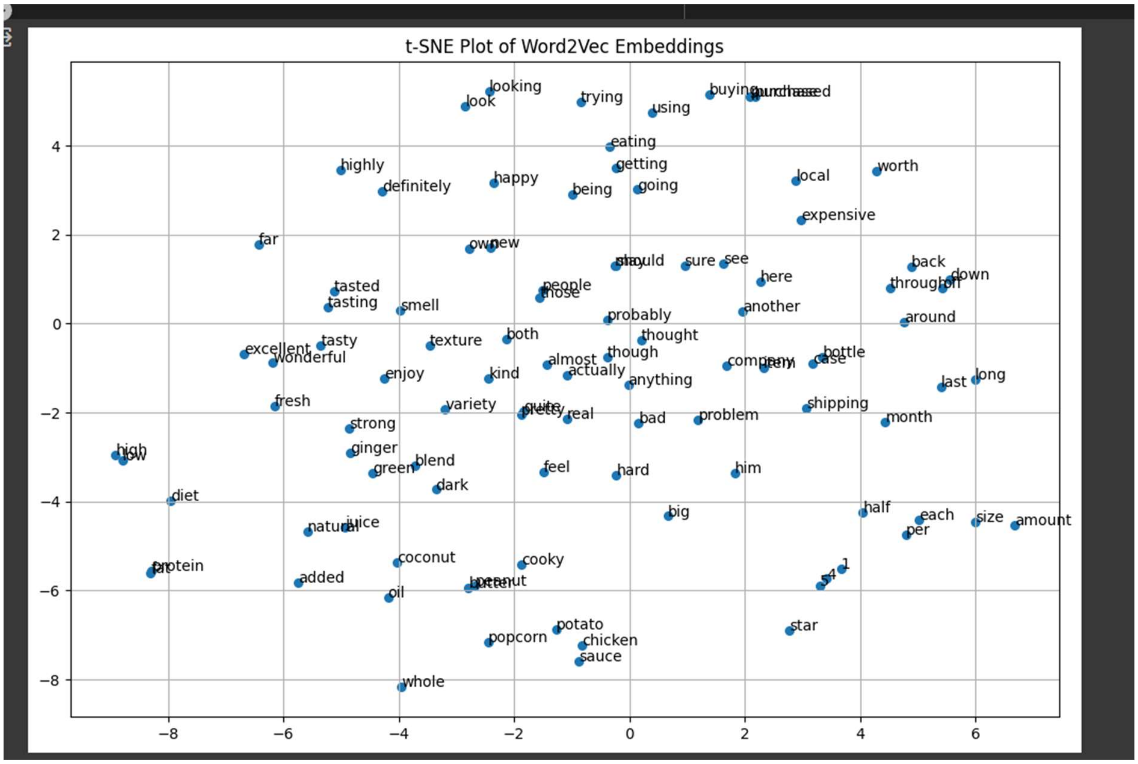
```python
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np


sentence1 = "This Product is highly recommended."
sentence2 = "I like the product."


from nltk.tokenize import word_tokenize

def tokenize_and_preprocess_text(sentence):
    tokens = word_tokenize(sentence)
    tokens=[lemmatizer.lemmatize(token) for token in tokens]
    tokens = [token.lower() for token in tokens if token.isalnum()]

    return tokens


tokens1 = tokenize_and_preprocess_text(sentence1)
tokens2 = tokenize_and_preprocess_text(sentence2)

def get_sentence_vector(tokens, model):
    # Filter out tokens that are not in the model's vocabulary
    if not tokens:
        return np.zeros(model.vector_size)
    return np.mean(model.wv[tokens], axis=0)

vector1 = get_sentence_vector(tokens1, model3)
vector2 = get_sentence_vector(tokens2, model3)

# Calculate the cosine similarity between the two sentence vectors
similarity = cosine_similarity([vector1], [vector2])[0][0]

print(f"Cosine Similarity: {similarity}")
```

```
Cosine Similarity: 0.8175958395004272
```

Next I found out STS similarity between two sentneces. We see that the similarity score was much better than word2vec but not better than GLoVe

```python
# Get word vectors and corresponding words from the model
words = list(model3.wv.index_to_key)
words=words[200:300]
word_vectors = [model3.wv[word] for word in words]

# Convert word_vectors to a NumPy array
word_vectors = np.array(word_vectors)

# Perform t-SNE dimensionality reduction
tsne = TSNE(n_components=2, random_state=42)
word_vectors_2d = tsne.fit_transform(word_vectors)

# Create a scatter plot
plt.figure(figsize=(12, 8))
plt.scatter(word_vectors_2d[:, 0], word_vectors_2d[:, 1], marker='o', s=30)

# Label some points for reference (optional)
sample_words = words[:100]  # Label the first 5 words from your list
for word, (x, y) in zip(sample_words, word_vectors_2d[:100]):
    plt.annotate(word, (x, y))



# Show the plot
plt.title('t-SNE Plot of Word2Vec Embeddings')
plt.grid(True)
plt.show()
```





Next I plotted t_sne plot for visulaisation of word embeddings

Task 5: Building word2vec for Hindi

Dataset used : https://www.kaggle.com/datasets/disisbig/hindi-wikipedia-articles-172k

It has 172k sentences.

We used 50k for initial training then another 50k for finetuning

```
import opendatasets as od
dataset_url = 'https://www.kaggle.com/datasets/disisbig/hindi-wikipedia-articles-172k'
od.download(dataset_url)

Skipping, found downloaded files in "./hindi-wikipedia-articles-172k" (use force=True to force download)
```

```
def process_text_file(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        text = file.read()
        sentences = sent_tokenize(text)
        tokenized_sentences = [word_tokenize(sentence) for sentence in sentences]
    return tokenized_sentences
hin_data = []

directory_path = '/content/hindi-wikipedia-articles-172k/train/train'
num_sent=0
for filename in os.listdir(directory_path):
    if filename.endswith('.txt'):
        num_sent+=1
        file_path = os.path.join(directory_path, filename)
        tokenized_sentences = process_text_file(file_path)
        hin_data.extend(tokenized_sentences)
```

```
[11]
print(num_sent)
hin_data

137823
```

First we tokenize the words

WORD2VEC

```
[16] import numpy as np
     from sklearn.manifold import TSNE
     import matplotlib.pyplot as plt
     from gensim.models import Word2Vec
     model4=Word2Vec(hin_data,min_count=1,vector_size=100,window=5,sg=0)
```

```
[17] model4.save('word2vec_hin')
```

```
[18] print('similarity between two words is ',model4.wv.similarity('जीत','हार'))

     similarity between two words is  0.714858
```

```
[19] print('similarity between two words is ',model4.wv.similarity('जॉन','सीना' ))

     similarity between two words is  0.4836632
```

```
[20] embedding=model4.wv['भारत']
     print('embedding for word भारत')
     print(embedding)

     embedding for word भारत
     [-0.17928241 -2.5090675   1.7377151  -3.5889845  -1.7878616  -3.3994863
       1.7914537   0.5230177   2.1642044   3.5285363   3.4463224   1.422038
       1.0219145  -2.0556073  -2.2918007  -0.41904506 -0.04840325  0.9259278
      -1.5597935   0.07231115  3.7116477   1.5095265  -1.4028176  -3.0710363
       2.004685    2.85919     0.62028515 -0.36568388 -0.333528   -3.3605285
      -2.0619867   2.6690702   1.0072361   2.357408   -1.8030025   0.11208253
      -0.4933012  -3.1489487  -7.205213    1.6737251   1.837642    0.9929223
      -1.4735844   4.885152    2.337524   -1.5737438   2.3144414   0.67766964
       0.01886922 -1.5409786  -2.4652865   3.9834957  -0.33795753  0.98963284
      -1.1118922   0.79530317 -1.6574788   2.5070314  -0.4662811   1.3685678
      -0.01616253 -0.46976048 -0.09786331  2.3356435  -1.3247796  -1.6336318
       1.6434901   0.67346525 -1.818706   -2.314964   -3.3890915   3.9935675
       0.5074495  -6.410521    1.56156    -1.934775   -1.3586838  -0.23483525
      -3.0212235   2.4127526   0.72237504  1.251552   -2.1136      0.5970809
```

Then we build word2vec model with vector size=100 and context size=5

We get the similarity score between जीत, हार as 0.714

```
[20] embedding=model4.wv['भारत']
     print('embedding for word भारत')
     print(embedding)

     embedding for word भारत
     [-0.17928241 -2.5090675   1.7377151  -3.5889845  -1.7878616  -3.3994863
       1.7914537   0.5230177   2.1642044   3.5285363   3.4463224   1.422038
       1.0219145  -2.0556073  -2.2918007  -0.41904506 -0.04840325  0.9259278
      -1.5597935   0.07231115  3.7116477   1.5095265  -1.4028176  -3.0710363
       2.004685    2.85919     0.62028515 -0.36568388 -0.333528   -3.3605285
      -2.0619867   2.6690702   1.0072361   2.357408   -1.8030025   0.11208253
      -0.4933012  -3.1489487  -7.205213    1.6737251   1.837642    0.9929223
      -1.4735844   4.885152    2.337524   -1.5737438   2.3144414   0.67766964
       0.01886922 -1.5409786  -2.4652865   3.9834957  -0.33795753  0.98963284
      -1.1118922   0.79530317 -1.6574788   2.5070314  -0.4662811   1.3685678
      -0.01616253 -0.46976048 -0.09786331  2.3356435  -1.3247796  -1.6336318
       1.6434901   0.67346525 -1.818706   -2.314964   -3.3890915   3.9935675
       0.5074495  -6.410521    1.56156    -1.934775   -1.3586838  -0.23483525
      -3.0212235   2.4127526   0.72237504  1.251552   -2.1136      0.5970809
      -0.54009926 -0.3519666   2.5493937  -2.0591304  -0.3024065  -1.6132652
      -2.4392366  -1.6698576  -0.43925938 -2.7044125  -0.7876917   4.494662
      -0.31098637  2.2215815  -0.31238714  1.6942233 ]
```

Then we get word embedding for word भारत

next we finetune the model for larger dataset and test with different hyperparameters and choose best from them

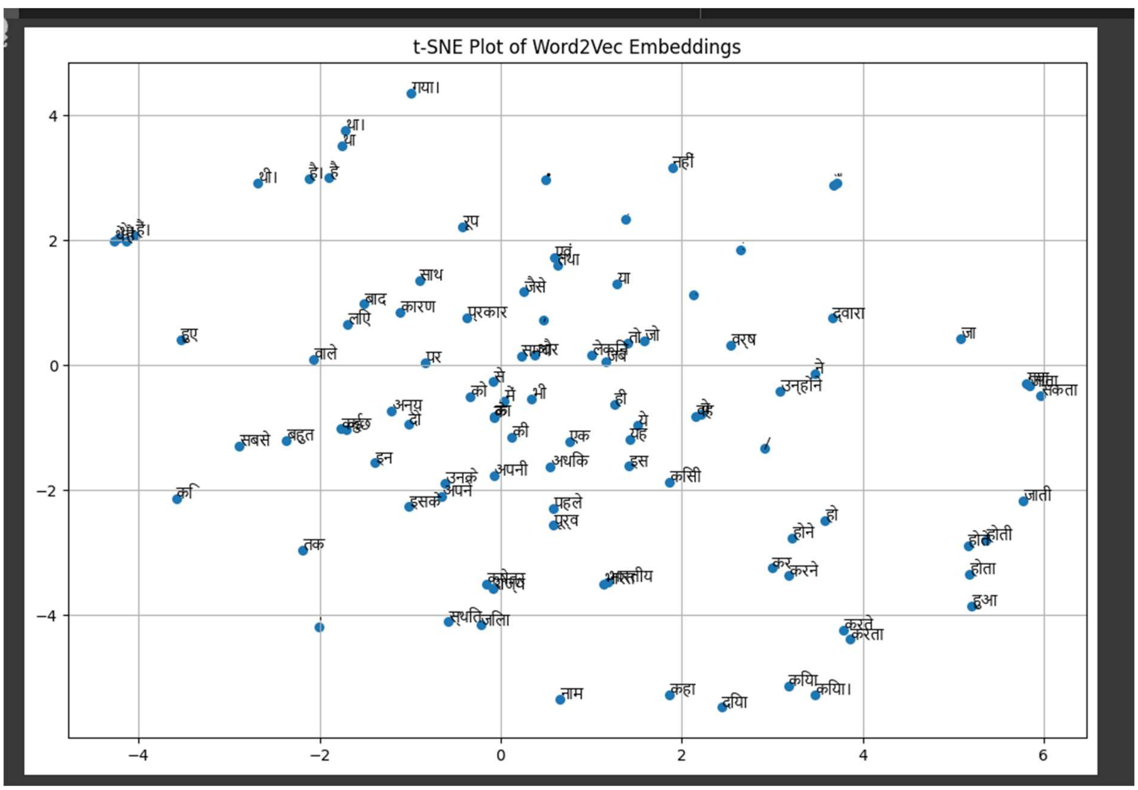
Finetuning

```
[32] from gensim.models import Word2Vec

     modelf = Word2Vec.load("/content/word2vec_hin")

     # Load a larger Hindi text corpus
     larger_corpus = hino_data[50000:150000]

     # Update the model with the larger corpus
     modelf.build_vocab(larger_corpus, update=True)
     modelf.train(larger_corpus, total_examples=modelf.corpus_count, epochs=10)

     # Hyperparameter tuning
     modelf.vector_size = 200
     modelf.window = 10
     modelf.sg = 1

     # Save the fine-tuned model
     modelf.save("fine_tuned_model_hindi")

     WARNING:gensim.models.word2vec:Effective 'alpha' higher than previous training cycles
```

```
print('similarity between two words is ',modelf.wv.similarity('जीत','हार'))

similarity between two words is  0.7192382
```

We see that similarity score of जीत, हार has improved

After that we plot t_sne plot for 100 words in hindi dataset vocabulary



Next I built fasttext model. We see that fasttext get better similarity score for ‘jeeth’ and ‘haar’ than word2vec

FASTTEXT

```
[ ] from gensim.models import FastText

    model5=FastText(hin_data,min_count=1,vector_size=100,window=5,sg=1, workers = 4, min_n = 1, max_n = 4)
```

```
[ ] print('similarity between two words is ',model5.wv.similarity('जीत','हार'))

    similarity between two words is  0.7516026
```

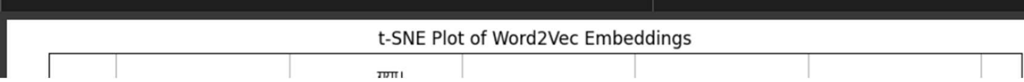
```
[ ] print('similarity between two words is ',model5.wv.similarity('जॉन','सीना' ))

    similarity between two words is  0.39044362
```

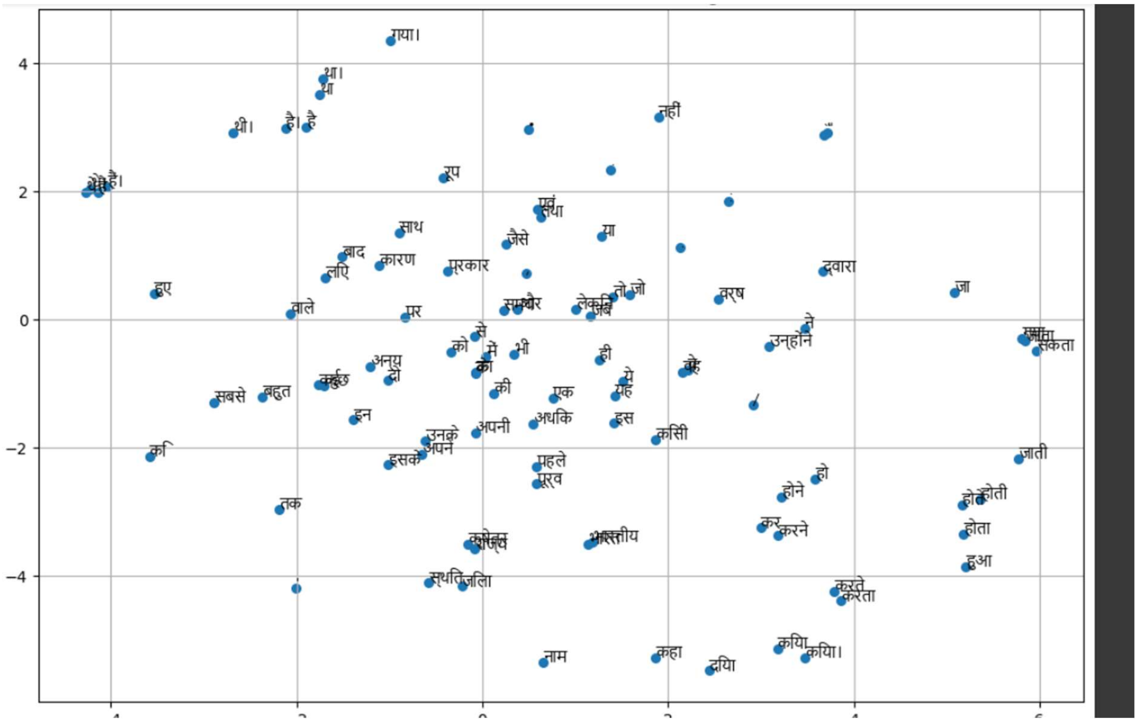
```
embedding=model5.wv['भारत']
print('embedding for word भारत')
print(embedding)
```

```
embedding for word भारत
[-0.2659481  -0.4535157  -0.3335495   0.39342457 -0.33875507 -0.3646687
  0.06733703  0.1659797   0.38835657 -0.15535024 -0.17868085 -0.47745845
 -0.4247472  -0.13319875  0.29882297  0.02743383  1.1597532   0.01525269
 -0.21422327 -0.29802677 -0.70926625  0.04692046 -0.01474076 -0.01504554
 -0.04613515  0.32577458  0.07765359 -0.378071   -0.02297987  0.22573124
  0.61969995 -0.59851867  0.02121556 -0.3220449   0.2889874  -0.11270276
  1.2412835   0.26516277 -0.91333604  0.18251646 -0.61753136 -0.05246259
 -0.21854737  0.19942863 -0.30353433  0.18025488 -0.03419732 -0.10376083
  0.2943418  -0.08525711 -0.42740798 -0.14966376 -0.5039572   0.4888843
  0.14233334 -0.40520573 -0.22769773 -1.034245   -0.07504398  0.1691202
 -0.08227174 -0.23074757  0.48883808 -0.20022564 -0.4622245   1.3370237
  0.35826278  0.40034443 -0.18523881 -0.46905673 -0.33806792 -0.5446332
  1.0005012  -0.01415118  0.5463973  -0.11245054  0.6908057  -0.16835314
  0.03815032  0.61620116 -0.14206061 -0.16658336 -0.4376223   0.8589682
  0.19783203 -0.20501687  0.25402263  0.2589313   0.18000211 -0.3182608
```

Next we plot tsne for fasttext too

```
# Get word vectors and corresponding words from the model
words = list(model5.wv.index_to_key)
words=words[:100]
word_vectors = [model5.wv[word] for word in words]

# Convert word_vectors to a NumPy array
word_vectors = np.array(word_vectors)

# Perform t-SNE dimensionality reduction
tsne = TSNE(n_components=2, random_state=42)
word_vectors_2d = tsne.fit_transform(word_vectors)

# Create a scatter plot
plt.figure(figsize=(12, 8))
plt.scatter(word_vectors_2d[:, 0], word_vectors_2d[:, 1], marker='o', s=30)

sample_words = words[:100]
for word, (x, y) in zip(sample_words, word_vectors_2d[:100]):
    plt.annotate(word, (x, y))


# Show the plot
plt.title('t-SNE Plot of Word2Vec Embeddings')
plt.grid(True)
plt.show()
```





4
2
0
−2
−4
गया।
था।
था
थी।
है।
है
नहीं
रूप
एवं
तथा
साथ
जैसे
या
बाद
कारण
लिए
प्रकार
द्वारा
जा
हुए
वाले
पर
तो
जो
वर्ष
ने
उन्होंने
से
को
में
भी
ही
की
अन्य
सबसे
बहुत
एक
यह
इन
अधिक
इस
किसी
कि
उनके
अपनी
अपने
इसके
पहले
पूर्व
जाती
हो
होने
होती
तक
कर
करने
होता
हुआ
स्थित
जिला
करते
करता
नाम
कहा
किया
किया।
दिया
सकता