Manish Bhardwaj

Roll no - 29

Section-3CB

NLP Embedding Techniques Assignment

Importing Libraries

```
import pandas as pd
import string
from textblob import TextBlob

import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

# Download NLTK resources
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
True
```

Loading Dataset

```
data = pd.read_csv("/content/dataset_25000.csv")
data.head()
```

	text	humor
0	Joe biden rules out 2020 bid: 'guys, i'm not r...	False
1	Watch: darvish gave hitter whiplash with slow ...	False
2	What do you call a turtle without its shell? d...	True
3	5 reasons the 2016 election feels so personal	False
4	Pasco police shot mexican migrant from behind,...	False

Data Preprocessing

```
data.shape
```

```
(25000, 2)
```

```
print("Class distribution (humor column):")
print(data['humor'].value_counts())
data['label'] = data['humor'].astype(int)
data[['text', 'label']].head(10)
```

```
Class distribution (humor column):
humor
True     12545
False    12455
Name: count, dtype: int64
```

	text	label
0	Joe biden rules out 2020 bid: 'guys, i'm not r...	0
1	Watch: darvish gave hitter whiplash with slow ...	0
2	What do you call a turtle without its shell? d...	1
3	5 reasons the 2016 election feels so personal	0
4	Pasco police shot mexican migrant from behind,...	0
5	Martha stewart tweets hideous food photo, twit...	0
6	What is a pokemon master's favorite kind of pa...	1
7	Why do native americans hate it when it rains ...	1
8	Obama's climate change legacy is impressive, i...	0
9	My family tree is a cactus, we're all pricks.	1

Remove punctuation

```
text = data["text"].copy()

removePun= []

for line in text:
    for p in string.punctuation:
        line = line.replace(p, "")
    removePun.append(line)

removePun[:5]
```

```
['Joe biden rules out 2020 bid guys im not running',
 'Watch darvish gave hitter whiplash with slow pitch',
 'What do you call a turtle without its shell dead',
 '5 reasons the 2016 election feels so personal',
 'Pasco police shot mexican migrant from behind new autopsy shows']
```

Convert to lowercase

```
lowercase = [line.lower() for line in removePun]
lowercase[:5]
```

```
['joe biden rules out 2020 bid guys im not running',
 'watch darvish gave hitter whiplash with slow pitch',
 'what do you call a turtle without its shell dead',
 '5 reasons the 2016 election feels so personal',
 'pasco police shot mexican migrant from behind new autopsy shows']
```

Tokenization

```
token = [word_tokenize(line) for line in lowercase]

token[:3]   # display first 3 token lists
```

```
[['joe',
  'biden',
  'rules',
  'out',
  '2020',
  'bid',
  'guys',
  'im',
  'not',
  'running'],
 ['watch', 'darvish', 'gave', 'hitter', 'whiplash', 'with', 'slow', 'pitch'],
 ['what',
```

```
 'do',
 'you',
 'call',
 'a',
 'turtle',
 'without',
 'its',
 'shell',
 'dead']]
```

StopWord

```
stop_words = set(stopwords.words("english"))

no_stopwords = [
    [word for word in tokens if word not in stop_words]
    for tokens in token
]

no_stopwords[:3]
```

```
[['joe', 'biden', 'rules', '2020', 'bid', 'guys', 'im', 'running'],
 ['watch', 'darvish', 'gave', 'hitter', 'whiplash', 'slow', 'pitch'],
 ['call', 'turtle', 'without', 'shell', 'dead']]
```

Lemmatization

```
lemmatizer = WordNetLemmatizer()
Lemmatize = []

for tokens_list in no_stopwords:
    lemmas = [lemmatizer.lemmatize(t) for t in tokens_list]
    Lemmatize.append(" ".join(lemmas))
```

Join Sentence

```
clean_text = [
    " ".join(tokens) for tokens in Lemmatize
]

clean_text[:5]
```

```
['joe biden rule 2020 bid guy im running',
 'watch darvish gave hitter whiplash slow pitch',
 'call turtle without shell dead',
 '5 reason 2016 election feel personal',
 'pasco police shot mexican migrant behind new autopsy show']
```

```
clean_text= pd.DataFrame(Lemmatize, columns=["clean_text"])
clean_text.head()
```

	clean_text
0	joe biden rule 2020 bid guy im running
1	watch darvish gave hitter whiplash slow pitch
2	call turtle without shell dead
3	5 reason 2016 election feel personal
4	pasco police shot mexican migrant behind new a...

```
clean_text.shape
```

```
(25000, 1)
```

Vocabulary

```
from collections import Counter
```

```
all_words = []

for text in clean_text['clean_text']:
    all_words.extend(text.split())

vocab = sorted(set(all_words))
print("Vocabulary size:", len(vocab))
```

```
Vocabulary size: 22659
```

```
word2idx = {word: idx for idx, word in enumerate(vocab)}
idx2word = {idx: word for word, idx in word2idx.items()}
```

```
index_to_word = {idx: word for idx, word in enumerate(vocab)}

list(index_to_word.items())[:20]
```

```
[(0, '0'),
 (1, '000'),
 (2, '00000'),
 (3, '000000'),
 (4, '007s'),
 (5, '010'),
 (6, '07'),
 (7, '0800'),
 (8, '0k'),
 (9, '0x0000ff'),
 (10, '0x100'),
 (11, '0xff0000'),
 (12, '1'),
 (13, '10'),
 (14, '100'),
 (15, '1000'),
 (16, '10000'),
 (17, '100000'),
 (18, '100footlong'),
 (19, '100inch')]
```

One Hot Encoding

```
from sklearn.preprocessing import OneHotEncoder

sentences = clean_text["clean_text"].tolist()
sentences = [[word] for word in sentences]

ohe = OneHotEncoder(sparse_output=False)
one_hot_vectors = ohe.fit_transform(sentences)
print(one_hot_vectors.shape)
```

```
(25000, 24887)
```

Bag Of Word

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer()
bow_vectors = cv.fit_transform(clean_text["clean_text"])

print(bow_vectors.shape)
```

```
(25000, 22615)
```

N-gram

```
cv_ngram = CountVectorizer(ngram_range=(2,3))
bon_vectors = cv_ngram.fit_transform(clean_text["clean_text"])

print(bon_vectors.shape)
```

```
(25000, 251996)
```

TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()
tfidf_vectors = tfidf.fit_transform(clean_text["clean_text"])

print(tfidf_vectors.shape)
```

```
(25000, 22615)
```

Word2Vec(CBOW)

```
!pip install gensim
from gensim.models import Word2Vec

tokenized = [text.split() for text in clean_text["clean_text"]]

w2v_cbow = Word2Vec(sentences=tokenized, vector_size=100, window=5, min_count=1, sg=0)
print(w2v_cbow.wv["election"])
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.12/dist-packages (4.4.0)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.12/dist-packages (from gensim) (2.0.2)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from gensim) (1.16.3)
Requirement already satisfied: smart_open>=1.8.1 in /usr/local/lib/python3.12/dist-packages (from gensim) (7.5.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart_open>=1.8.1->gensim) (2.0.1)
[-0.12825358  0.51463264  0.21278651 -0.03880647  0.22125065 -0.66270876
  0.0962626   0.91714424 -0.2054072  -0.2741622  -0.26868472 -0.62724835
 -0.08221783  0.2916427   0.17115375 -0.22299248  0.08616655 -0.45500776
  0.02941257 -0.8849852   0.16126047  0.03750186  0.35353732 -0.16792384
  0.04684714 -0.00822355 -0.25039113 -0.12225629 -0.40353385 -0.01053582
  0.36215022  0.11541969  0.00442992 -0.28516203 -0.04745101  0.36862907
  0.05743418 -0.3247773  -0.23296766 -0.7048844   0.01316565 -0.45940265
 -0.22311229  0.0126473   0.2992611  -0.30117178 -0.2844626  -0.02008404
 -0.00872798  0.31499285  0.17058618 -0.41219187 -0.26770946 -0.09955876
 -0.3914253   0.07065107  0.23886521 -0.09104345 -0.43575022  0.08119639
  0.1784752   0.12688573 -0.07769826  0.10311637 -0.455057    0.38450533
  0.12406837  0.39157045 -0.4280242   0.55889046 -0.1697191   0.23845452
  0.4999008  -0.209054    0.46299723  0.2165158  -0.05852834  0.10503776
 -0.4307881   0.17956083 -0.30613282 -0.1297294  -0.35453475  0.47543323
 -0.15737554 -0.03117283 -0.1008523   0.46513835  0.3473663   0.13323662
  0.5180565   0.15537676  0.14771254 -0.03350915  0.60009503  0.31345508
  0.44193435 -0.4991281   0.09535573 -0.01499703]
```

```
import numpy as np

def get_average_word2vec(tokens_list, model, vector_size):
    vectors = []
    for tokens in tokens_list:
        word_vectors = [model.wv[word] for word in tokens if word in model.wv]
        if word_vectors:
            vectors.append(np.mean(word_vectors, axis=0))
        else:
            vectors.append(np.zeros(vector_size))
    return np.array(vectors)

X_w2v = get_average_word2vec(tokenized, w2v_cbow, 100)
```

Skip-Gram

```
w2v_skipgram = Word2Vec(sentences=tokenized, vector_size=100, window=5, min_count=1, sg=1)
print(w2v_skipgram.wv["election"])
```

```
[-3.2134384e-01  2.7320775e-01  2.1013021e-01  6.0424272e-02
  6.2763259e-02 -4.2179233e-01 -1.3101479e-04  3.9136183e-01
 -1.6654891e-01  6.7965955e-02 -2.0026231e-01 -2.6787412e-01
 -3.2579765e-02  2.5303084e-01  1.9599886e-01 -2.3100521e-01
 -6.5866113e-03 -3.8756615e-01 -2.1665523e-02 -5.4612148e-01
  5.8197450e-02 -5.6684960e-02  3.4762689e-01  7.1810834e-02
  1.9447412e-01 -5.5114560e-02 -2.1653685e-01  4.6870656e-02
 -3.0187169e-01 -3.7750456e-02  3.1100059e-01  4.0698532e-02
 -1.8882291e-01 -1.8512727e-01  2.8044047e-02  5.1474482e-02
  3.5751404e-04 -7.3400073e-02 -2.1510844e-01 -2.3200862e-01
```

```
  8.0282260e-03 -4.7678638e-01 -2.3464270e-01  5.5041011e-02
  2.3037355e-01 -2.6801533e-01 -2.9086670e-01  7.5099975e-02
 -5.9899658e-02  4.8958573e-02  9.8165624e-02 -1.7778204e-01
 -9.1755360e-02 -7.7120386e-02 -2.1737693e-01 -1.7884608e-01
  2.2732344e-01 -6.2020663e-02 -2.9531276e-01 -4.5544197e-04
  1.4508431e-01  1.1538326e-02  8.0200844e-02  4.9972713e-02
 -4.0030664e-01  1.2741594e-01  2.3034514e-01  4.9377072e-01
 -2.4961878e-01  4.6129039e-01 -8.9215800e-02  1.6485965e-01
  2.2828959e-01  9.4269909e-02  2.4495384e-01  1.3167284e-01
 -1.4399573e-01  1.6186585e-01 -2.3296778e-01  6.4771570e-02
 -4.6625781e-01 -9.1834225e-02 -3.9893436e-01  2.5989795e-01
 -3.6126557e-01 -2.7689096e-04 -1.6922979e-02  3.7810388e-01
  2.5459960e-01 -1.3884269e-02  3.3605155e-01  1.6243689e-01
  1.2961319e-01  1.3181645e-01  4.7172642e-01  2.3826042e-01
  3.5205314e-01 -3.6116305e-01 -3.1278353e-02 -1.3796583e-02]
```

Fast-Text

```
from gensim.models import FastText

fasttext_model = FastText(sentences=tokenized, vector_size=100, window=5, min_count=1)
print(fasttext_model.wv["election"])
X_fasttext = get_average_word2vec(tokenized, fasttext_model, 100)
```

```
[-0.37975177  0.5340854  -0.5208305   0.3784951   0.43396923  0.34099123
 -0.10722329  0.2698129   0.98675305 -0.8521973  -0.55237854 -0.17058457
 -0.5174411   1.0932716   0.1430338  -0.12254243 -0.2736059  -0.5353984
 -0.7308128  -1.2040753  -0.9745009   0.17180283 -0.12800951 -0.5336403
 -0.3037295  -0.46747583 -0.61694133  0.2945218   1.2483519  -0.14350547
 -0.20512435  0.44668528  0.6703505  -0.4246733   0.067508    0.65399396
  0.22678386  1.2084732  -0.8692302   0.36736834  0.7400592  -0.92630696
  0.00530838 -0.9438131  -1.0631629  -0.46267325 -0.37536034 -0.56776226
 -0.17379971  0.25656694  0.17712381 -0.07383581  1.0965534   0.6411814
  0.20268175 -0.37606376 -0.5227535   0.07074805 -0.25795156  0.0864427
  0.94802016 -0.4529999  -1.0477327   1.1541911   0.19073656  0.9293396
  0.24771918 -0.28617692  0.5593254   1.0134794   0.16830942 -0.02498368
  0.42789403 -0.78894275  0.8332746  -0.0566753   0.5170958   0.12092523
 -0.29941234  0.6665942   0.1786285  -0.2467262  -0.22005473  0.10215803
 -0.80532277 -0.3797113  -0.45083198 -0.41509113 -0.3419132  -0.17005883
 -1.7502962  -0.39023173 -0.47864497  0.19085647 -0.9032051   0.9632988
  0.17806455 -0.36551723  0.0692147   0.95934284]
```

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

```
--2025-11-20 05:17:27--  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]
--2025-11-20 05:17:27--  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
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2025-11-20 05:17:27--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.01MB/s    in 2m 39s

2025-11-20 05:20:07 (5.16 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]
```

```
!unzip glove.6B.zip
```

```
Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

Glove

```
import numpy as np
```

```
glove_file = "/content/glove.6B.100d.txt"   # REAL pretrained glove file

embeddings_index = {}
with open(glove_file, encoding="utf8") as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = vector

print("Found words:", len(embeddings_index))
print("Vector example (election):", embeddings_index["election"])

def get_glove_average_vector(tokens_list, glove_embeddings, vector_size):
    vectors = []
    for tokens in tokens_list:
        word_vectors = [glove_embeddings[word] for word in tokens if word in glove_embeddings]
        if word_vectors:
            vectors.append(np.mean(word_vectors, axis=0))
        else:
            vectors.append(np.zeros(vector_size))
    return np.array(vectors)

X_glove = get_glove_average_vector(tokenized, embeddings_index, 100)
```

```
Found words: 400000
Vector example (election): [-1.0099    -0.081107   0.40596   -0.67365    0.3072     0.1757
  0.17734    0.46017   -0.017079  -0.081665  -0.58082   -0.0078334
  0.45448   -0.54727   -0.028427  -0.25477    0.075155   0.24047
  0.19395   -0.11413    0.35495   -0.32314    1.3       -0.45084
 -0.92584   -0.83248    0.0070823 -0.055165   0.67146   -0.41138
  0.92317    0.16679    0.63774   -0.28868   -0.42036   -0.3118
  0.10564   -0.40331   -0.71603    0.1532    -0.98441   -0.84357
  1.0837    -0.73638   -1.0181    -1.0255     0.33402   -1.3106
 -0.59665   -0.64874    0.026353  -0.45967    0.87694    1.3043
 -0.16245   -2.6004     0.41372    0.58397    1.1991     0.0070176
 -1.0968    -0.30687   -0.34511    0.34693    1.408      0.32611
 -0.58305    0.076722   0.16697    0.14879    0.046754   0.30531
 -0.81294   -0.07248   -0.5011     0.17783   -0.46585    0.41289
 -1.518     -0.43424    0.42262   -0.12615    0.41616   -0.15648
 -0.98673   -0.73666    0.07661    0.53163    0.2517    -0.99605
  0.26203    0.032642  -0.46742    1.2543    -0.31919    0.28921
  0.68156    0.70478   -0.11105   -0.16128  ]
```

Doc2Vec(DM & DBOW)

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

tagged = [TaggedDocument(words=text.split(), tags=[str(i)])
          for i, text in enumerate(clean_text["clean_text"])]
```

```
X_skipgram = get_average_word2vec(tokenized, w2v_skipgram, 100)
```

```
model_dm = Doc2Vec(tagged, vector_size=100, window=5, min_count=1, dm=1)
print(model_dm.dv["0"])
X_doc2vec_dm = np.array([model_dm.infer_vector(doc.words) for doc in tagged])
```

```
[-0.0146557   0.01300539  0.00177199  0.00536179  0.01212659 -0.03374682
 -0.00702887  0.04234776 -0.02039402 -0.0119209  -0.00765407 -0.02389105
 -0.01038312  0.00882115  0.00755704 -0.02133937  0.0093443  -0.01302897
 -0.01063657 -0.04702391  0.00698877  0.00291501  0.0123486  -0.00794388
  0.00800272 -0.00917783 -0.01939723 -0.0191536  -0.01272282 -0.00314283
  0.02320271  0.01639876 -0.00731431 -0.0198435  -0.00573804  0.02486849
 -0.00106205 -0.02556664 -0.01533473 -0.02845724 -0.00046187 -0.02982192
 -0.0009852  -0.00535067  0.00936939 -0.02212447 -0.01303509 -0.0062977
  0.00786456  0.01292068  0.0111797  -0.01873186 -0.01382892 -0.01140728
 -0.02024563  0.01454304  0.00490358  0.00171192 -0.02740284  0.0125565
  0.00607557  0.01884883 -0.00291017 -0.00482752 -0.017457    0.01695591
  0.01181475  0.01652956 -0.02875134  0.01856536 -0.01674959  0.01772283
  0.01512185 -0.02066119  0.01335256  0.01415868 -0.01148529 -0.00808215
 -0.01486704  0.00866212 -0.00592224 -0.00176225 -0.01913121  0.01946841
 -0.00163294 -0.00082897 -0.00489829  0.01390162  0.02692566  0.01212086
  0.02046981  0.01816417 -0.0084584  -0.01175505  0.02346743  0.02042761
  0.02029514 -0.02076589  0.01739573  0.00367253]
```

```
model_dbow = Doc2Vec(tagged, vector_size=100, window=5, min_count=1, dm=0)
print(model_dbow.dv["0"])
```

```
X_doc2vec_dbow = np.array([model_dbow.infer_vector(doc.words) for doc in tagged])
```

```
[-4.79906984e-03 -8.84843152e-03 -1.04166567e-02  5.64288022e-03
  3.16875149e-03 -2.95457412e-05 -1.26489718e-02 -5.31906309e-03
 -9.88228992e-03 -8.09515186e-05  3.69514851e-03  4.59963130e-03
 -4.92556812e-03 -3.29893315e-03 -3.90193262e-03 -9.09066014e-03
  2.56925239e-03  7.60814454e-03 -8.60855822e-03 -1.88499212e-03
 -2.80338130e-03  4.23433073e-03 -5.53978002e-03  1.24073576e-03
  6.53795991e-03 -7.35883275e-03 -7.61245005e-03 -1.08230906e-02
  3.93443881e-03 -1.00491345e-02  6.49232417e-03  4.88792825e-03
 -7.74311554e-03 -4.77831811e-03 -3.24132619e-03  4.44802310e-04
 -2.35070498e-03 -5.90324402e-03 -3.87597270e-03  2.65196874e-03
 -2.81545427e-03 -6.70051016e-03  6.96028955e-03 -8.32802802e-03
  9.27843852e-04 -7.25349179e-03  2.95651844e-03 -1.61785213e-03
  5.10711875e-03 -6.32112706e-03 -7.46493228e-04 -4.41384560e-04
 -3.85364657e-03 -6.64411671e-03 -3.91688943e-03  6.01162482e-03
 -3.56618920e-03  2.84316996e-03 -7.68721569e-03  9.18895379e-03
  3.20920767e-03  8.82041082e-03  3.93536873e-03 -6.54441072e-03
  2.23059533e-03 -2.85348739e-03  6.90287072e-03  1.85355579e-03
 -2.99045839e-03 -6.87562581e-03 -9.47098620e-03 -1.58811035e-03
 -7.47998385e-03 -8.67020246e-03 -8.06926191e-03  1.11840270e-03
 -6.53443346e-03 -7.48141622e-03  3.23410123e-03  1.70652126e-03
  5.71531570e-03  6.52515097e-03 -1.06627317e-02 -2.14390247e-03
  7.18099065e-03  4.61092917e-03  1.84229400e-03 -6.56468701e-03
  7.81045109e-03  2.21336866e-03 -7.29998527e-03  8.16690084e-03
 -1.11548593e-02 -9.72310174e-03 -4.15834365e-03  9.24248435e-03
  2.64211046e-03 -2.50298786e-03  9.67951585e-03  4.27690195e-03]
```

Train Test Split(OHE , BOW ,N-gram ,TF-IDF ,W2V(Cbow,SkipG) , Doc2vec(DM,Dbow),Fast-text,Glove)

```
from sklearn.model_selection import train_test_split

y = data['humor'] # Define the target variable
```

```
# 1. Bag of Words
X_train_bow, X_test_bow, y_train_bow, y_test_bow = train_test_split(
    bow_vectors, y, test_size=0.2, random_state=42)
```

```
# 2. N-gram (bigram/trigram)
X_train_ng, X_test_ng, y_train_ng, y_test_ng = train_test_split(
    bon_vectors, y, test_size=0.2, random_state=42)
```

```
# 3. TF-IDF (unigram)
X_train_tfidf, X_test_tfidf, y_train_tfidf, y_test_tfidf = train_test_split(
    tfidf_vectors, y, test_size=0.2, random_state=42)
```

```
# 4. Word2Vec CBOW
X_train_w2v, X_test_w2v, y_train_w2v, y_test_w2v = train_test_split(
    X_w2v, y, test_size=0.2, random_state=42)
```

```
# 5. Word2Vec SkipGram
X_train_sg, X_test_sg, y_train_sg, y_test_sg = train_test_split(
    X_skipgram, y, test_size=0.2, random_state=42)
```

```
# 6. fastText
X_train_fast, X_test_fast, y_train_fast, y_test_fast = train_test_split(
    X_fasttext, y, test_size=0.2, random_state=42)
```

```
# 7. GloVe
X_train_glove, X_test_glove, y_train_glove, y_test_glove = train_test_split(
    X_glove, y, test_size=0.2, random_state=42)
```

```
X_train_dm, X_test_dm, y_train_dm, y_test_dm = train_test_split(
    X_doc2vec_dm, y, test_size=0.2, random_state=42)
```

```
# 9. Doc2Vec DBOW
X_train_dbow, X_test_dbow, y_train_dbow, y_test_dbow = train_test_split(
    X_doc2vec_dbow, y, test_size=0.2, random_state=42)
```

```
```

```
X_train_glove, X_test_glove, y_train_glove, y_test_glove = train_test_split(X_glove, y, test_size=0.2, random_state=42)
```

Logistic Regression for All Embedding tech

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

```
def evaluate_lr(X_train, X_test, y_train, y_test):
    model = LogisticRegression(max_iter=2000)
    model.fit(X_train, y_train)
    preds = model.predict(X_test)

    return {
        "Accuracy": accuracy_score(y_test, preds),
        "Precision": precision_score(y_test, preds),
        "Recall": recall_score(y_test, preds),
        "F1-Score": f1_score(y_test, preds)
    }
```

```
lr_datasets = {
    "BoW": (X_train_bow, X_test_bow, y_train_bow, y_test_bow),
    "N-gram": (X_train_ng, X_test_ng, y_train_ng, y_test_ng),
    "TF-IDF": (X_train_tfidf, X_test_tfidf, y_train_tfidf, y_test_tfidf),
    "Word2Vec_CBOW": (X_train_w2v, X_test_w2v, y_train_w2v, y_test_w2v),
    "Word2Vec_SkipGram": (X_train_sg, X_test_sg, y_train_sg, y_test_sg),
    "fastText": (X_train_fast, X_test_fast, y_train_fast, y_test_fast),
    "GloVe": (X_train_glove, X_test_glove, y_train_glove, y_test_glove),
    "Doc2Vec_DM": (X_train_dm, X_test_dm, y_train_dm, y_test_dm),
    "Doc2Vec_DBOW": (X_train_dbow, X_test_dbow, y_train_dbow, y_test_dbow)
}
```

```
lr_results = {}

for name, (Xtr, Xte, ytr, yte) in lr_datasets.items():
    print(f"Running LR on {name}...")
    lr_results[name] = evaluate_lr(Xtr, Xte, ytr, yte)
```

```
Running LR on BoW...
Running LR on N-gram...
Running LR on TF-IDF...
Running LR on Word2Vec_CBOW...
Running LR on Word2Vec_SkipGram...
Running LR on fastText...
Running LR on GloVe...
Running LR on Doc2Vec_DM...
Running LR on Doc2Vec_DBOW...
```

```
import pandas as pd
lr_results_df = pd.DataFrame(lr_results).T
lr_results_df
```

	Accuracy	Precision	Recall	F1-Score
BoW	0.8816	0.882446	0.882446	0.882446
N-gram	0.7632	0.848485	0.644956	0.732852
TF-IDF	0.8814	0.891738	0.870135	0.880804
Word2Vec_CBOW	0.7636	0.799552	0.708102	0.751053
Word2Vec_SkipGram	0.8396	0.852217	0.824464	0.838111
fastText	0.7442	0.769700	0.702145	0.734372
GloVe	0.8162	0.810968	0.828038	0.819414
Doc2Vec_DM	0.8028	0.808126	0.797855	0.802958
Doc2Vec_DBOW	0.6244	0.627085	0.627085	0.627085

```
def evaluate_rf(X_train, X_test, y_train, y_test):
    model = RandomForestClassifier(
```

```
        n_estimators=50,
        max_depth=None,
        random_state=42,
        n_jobs=-1
    )
    model.fit(X_train, y_train)
    preds = model.predict(X_test)

    return {
        "Accuracy": accuracy_score(y_test, preds),
        "Precision": precision_score(y_test, preds),
        "Recall": recall_score(y_test, preds),
        "F1-Score": f1_score(y_test, preds)
    }
```

```
rf_datasets = {
    "Word2Vec_CBOW": (X_train_w2v, X_test_w2v, y_train_w2v, y_test_w2v),
    "Word2Vec_SkipGram": (X_train_sg, X_test_sg, y_train_sg, y_test_sg),
    "fastText": (X_train_fast, X_test_fast, y_train_fast, y_test_fast),
    "GloVe": (X_train_glove, X_test_glove, y_train_glove, y_test_glove),
    "Doc2Vec_DM": (X_train_dm, X_test_dm, y_train_dm, y_test_dm),
    "Doc2Vec_DBOW": (X_train_dbow, X_test_dbow, y_train_dbow, y_test_dbow),
}
```

```
rf_results = {}

for name, (Xtr, Xte, ytr, yte) in rf_datasets.items():
    print(f"Training Random Forest on {name}...")
    rf_results[name] = evaluate_rf(Xtr, Xte, ytr, yte)
```

```
Training Random Forest on Word2Vec_CBOW...
Training Random Forest on Word2Vec_SkipGram...
Training Random Forest on fastText...
Training Random Forest on GloVe...
Training Random Forest on Doc2Vec_DM...
Training Random Forest on Doc2Vec_DBOW...
```

```
import pandas as pd

rf_results_df = pd.DataFrame(rf_results).T
rf_results_df
```

	Accuracy	Precision	Recall	F1-Score
Word2Vec_CBOW	0.7850	0.796791	0.769261	0.782784
Word2Vec_SkipGram	0.8480	0.854150	0.841938	0.848000
fastText	0.7542	0.758110	0.751787	0.754935
GloVe	0.7954	0.804481	0.784353	0.794289
Doc2Vec_DM	0.8108	0.813397	0.810167	0.811779
Doc2Vec_DBOW	0.6358	0.647607	0.607228	0.626768

ANN for Advance Embedding

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
def build_ann(input_dim):
    model = Sequential([
        Dense(128, activation='relu', input_dim=input_dim),
        Dense(64, activation='relu'),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

```
```

```
def evaluate_ann(X_train, X_test, y_train, y_test, epochs=10, batch_size=32):
    input_dim = X_train.shape[1]
    model = build_ann(input_dim)
    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=0)
    _, accuracy_ann = model.evaluate(X_test, y_test, verbose=0)
    y_pred_proba = model.predict(X_test, verbose=0)
    y_pred = (y_pred_proba > 0.5).astype(int)

    return {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1-Score": f1_score(y_test, y_pred)
    }

ann_sets = {
    "Word2Vec_CBOW": (X_train_w2v, X_test_w2v, y_train_w2v, y_test_w2v),
    "Word2Vec_SkipGram": (X_train_sg, X_test_sg, y_train_sg, y_test_sg),
    "fastText": (X_train_fast, X_test_fast, y_train_fast, y_test_fast),
    "GloVe": (X_train_glove, X_test_glove, y_train_glove, y_test_glove),
    "Doc2Vec_DM": (X_train_dm, X_test_dm, y_train_dm, y_test_dm),
    "Doc2Vec_DBOW": (X_train_dbow, X_test_dbow, y_train_dbow, y_test_dbow)
}
```

```
ann_results = {}

for name, (Xtr, Xte, ytr, yte) in ann_sets.items():
    print(f"Training ANN on {name}...")
    ann_results[name] = evaluate_ann(Xtr, Xte, ytr, yte)
```

```
Training ANN on Word2Vec_CBOW...
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Training ANN on Word2Vec_SkipGram...
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Training ANN on fastText...
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Training ANN on GloVe...
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Training ANN on Doc2Vec_DM...
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Training ANN on Doc2Vec_DBOW...
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
ann_results_df = pd.DataFrame(ann_results).T
ann_results_df
```

	Accuracy	Precision	Recall	F1-Score
Word2Vec_CBOW	0.7868	0.785827	0.792693	0.789245
Word2Vec_SkipGram	0.8268	0.776995	0.920175	0.842545
fastText	0.7406	0.714135	0.808578	0.758428
GloVe	0.8336	0.811530	0.872121	0.840735
Doc2Vec_DM	0.8160	0.827728	0.801430	0.814366
Doc2Vec_DBOW	0.6382	0.640396	0.642176	0.641285

Embedding matrix using the w2v_cbow model

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np

# 1. Convert tokenized sentences to sequences of word indices
indexed_sentences = []
for sentence_tokens in tokenized:
    indexed_sentence = [word2idx[word] for word in sentence_tokens if word in word2idx]
    indexed_sentences.append(indexed_sentence)
```

```
# 2. Determine the maximum sequence length
max_len = max(len(s) for s in indexed_sentences)
print(f"Maximum sequence length: {max_len}")

# 3. Pad the indexed_sentences to max_len
X_padded_sequences = pad_sequences(indexed_sentences, maxlen=max_len, padding='post')
print(f"Shape of padded sequences: {X_padded_sequences.shape}")

# 4. Create an embedding matrix using the w2v_cbow model
vector_size = 100  # Word2Vec vector size
embedding_matrix = np.zeros((len(vocab), vector_size))

for word, i in word2idx.items():
    if word in w2v_cbow.wv:
        embedding_matrix[i] = w2v_cbow.wv[word]

print(f"Shape of embedding matrix: {embedding_matrix.shape}")
```

```
Maximum sequence length: 16
Shape of padded sequences: (25000, 16)
Shape of embedding matrix: (22659, 100)
```

RNN

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

X_train_rnn, X_test_rnn, y_train_rnn, y_test_rnn = train_test_split(X_padded_sequences, y, test_size=0.2, random_state=42)

def create_and_evaluate_sequential_model(model_type, X_train, X_test, y_train, y_test, embedding_matrix, vocab_size, vector_siz
    model = Sequential()
    model.add(Embedding(input_dim=vocab_size,
                        output_dim=vector_size,
                        weights=[embedding_matrix],
                        trainable=False))

    if model_type == 'RNN':
        model.add(SimpleRNN(units=100))
    elif model_type == 'LSTM':
        model.add(LSTM(units=100))
    elif model_type == 'GRU':
        model.add(GRU(units=100))

    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    print(f"Training {model_type}...")
    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=0)

    y_pred_proba = model.predict(X_test, verbose=0)
    y_pred = (y_pred_proba > 0.5).astype(int)

    return {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1-Score": f1_score(y_test, y_pred)
    }

results_rnn = create_and_evaluate_sequential_model('RNN', X_train_rnn, X_test_rnn, y_train_rnn, y_test_rnn, embedding_matrix, l
print(f"\nModel Performance with RNN:")
for key, value in results_rnn.items():
    print(f"{key}: {value:.4f}")

accuracy_rnn = results_rnn['Accuracy']
precision_rnn = results_rnn['Precision']
recall_rnn = results_rnn['Recall']
f1_rnn = results_rnn['F1-Score']
```

```
Training RNN...

Model Performance with RNN:
Accuracy: 0.7280
Precision: 0.6592
```

```
Recall: 0.9523
F1-Score: 0.7791
```

LSTM

```
from tensorflow.keras.layers import LSTM

results_lstm = create_and_evaluate_sequential_model('LSTM', X_train_rnn, X_test_rnn, y_train_rnn, y_test_rnn, embedding_matrix,

print(f"\nModel Performance with LSTM:")
for key, value in results_lstm.items():
    print(f"{key}: {value:.4f}")

accuracy_lstm = results_lstm['Accuracy']
precision_lstm = results_lstm['Precision']
recall_lstm = results_lstm['Recall']
f1_lstm = results_lstm['F1-Score']
```

```
Training LSTM...

Model Performance with LSTM:
Accuracy: 0.7984
Precision: 0.8473
Recall: 0.7315
F1-Score: 0.7852
```

GRU

```
from tensorflow.keras.layers import GRU

results_gru = create_and_evaluate_sequential_model('GRU', X_train_rnn, X_test_rnn, y_train_rnn, y_test_rnn, embedding_matrix, l

print(f"\nModel Performance with GRU:")
for key, value in results_gru.items():
    print(f"{key}: {value:.4f}")

accuracy_gru = results_gru['Accuracy']
precision_gru = results_gru['Precision']
recall_gru = results_gru['Recall']
f1_gru = results_gru['F1-Score']
```

```
Training GRU...

Model Performance with GRU:
Accuracy: 0.8076
Precision: 0.8559
Recall: 0.7431
F1-Score: 0.7955
```

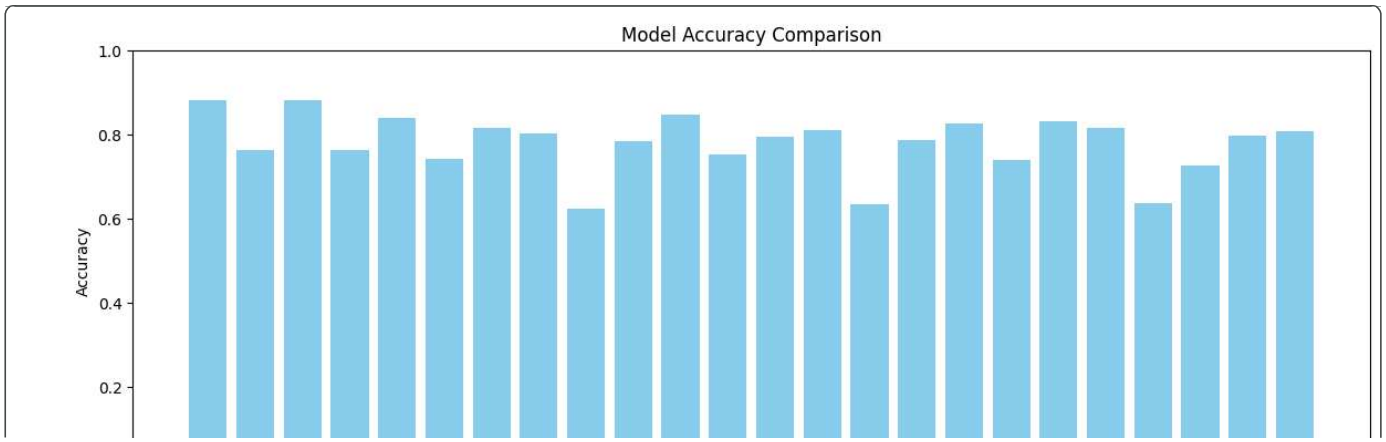
Final Result

```
new_results_rnn_lstm_gru = {
    "Model": [
        "RNN (CBOW)",
        "LSTM (CBOW)",
        "GRU (CBOW)"
    ],
    "Accuracy": [
        accuracy_rnn,
        accuracy_lstm,
        accuracy_gru
    ],
    "Precision": [
        precision_rnn,
        precision_lstm,
        precision_gru
    ],
    "Recall": [
        recall_rnn,
        recall_lstm,
        recall_gru
    ],
    "F1-Score": [
        f1_rnn,
```

```
        f1_lstm,
        f1_gru
    ]
}

df_new_results = pd.DataFrame(new_results_rnn_lstm_gru)
# Add prefixes to the new results
df_new_results['Model'] = df_new_results['Model'].apply(lambda x: x.split(' ')[0] + '_' + '_'.join(x.split(' ')[1:]))

# Initialize df_results with existing dataframes and add prefixes
df_results = pd.concat([
    lr_results_df.reset_index().rename(columns={'index': 'Model'}).assign(Model=lambda x: 'LR_' + x['Model']),
    rf_results_df.reset_index().rename(columns={'index': 'Model'}).assign(Model=lambda x: 'RF_' + x['Model']),
    ann_results_df.reset_index().rename(columns={'index': 'Model'}).assign(Model=lambda x: 'ANN_' + x['Model'])
], ignore_index=True)

df_results = pd.concat([df_results, df_new_results], ignore_index=True)
print(df_results)
```

```
                    Model  Accuracy  Precision    Recall  F1-Score
0                  LR_BoW    0.8816   0.882446  0.882446  0.882446
1              LR_N-gram     0.7632   0.848485  0.644956  0.732852
2              LR_TF-IDF     0.8814   0.891738  0.870135  0.880804
3       LR_Word2Vec_CBOW     0.7636   0.799552  0.708102  0.751053
4    LR_Word2Vec_SkipGram    0.8396   0.852217  0.824464  0.838111
5             LR_fastText    0.7442   0.769700  0.702145  0.734372
6               LR_GloVe     0.8162   0.810968  0.828038  0.819414
7          LR_Doc2Vec_DM     0.8028   0.808126  0.797855  0.802958
8         LR_Doc2Vec_DBOW    0.6244   0.627085  0.627085  0.627085
9        RF_Word2Vec_CBOW    0.7850   0.796791  0.769261  0.782784
10   RF_Word2Vec_SkipGram    0.8480   0.854150  0.841938  0.848000
11            RF_fastText    0.7542   0.758110  0.751787  0.754935
12              RF_GloVe     0.7954   0.804481  0.784353  0.794289
13         RF_Doc2Vec_DM     0.8108   0.813397  0.810167  0.811779
14        RF_Doc2Vec_DBOW    0.6358   0.647607  0.607228  0.626768
15      ANN_Word2Vec_CBOW    0.7868   0.785827  0.792693  0.789245
16  ANN_Word2Vec_SkipGram    0.8268   0.776995  0.920175  0.842545
17           ANN_fastText    0.7406   0.714135  0.808578  0.758428
18             ANN_GloVe     0.8336   0.811530  0.872121  0.840735
19         ANN_Doc2Vec_DM    0.8160   0.827728  0.801430  0.814366
20       ANN_Doc2Vec_DBOW    0.6382   0.640396  0.642176  0.641285
21             RNN_(CBOW)    0.7280   0.659153  0.952343  0.779077
22            LSTM_(CBOW)    0.7984   0.847286  0.731533  0.785166
23             GRU_(CBOW)    0.8076   0.855901  0.743050  0.795493
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.bar(df_results['Model'], df_results['Accuracy'], color='skyblue')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.xticks(rotation=45, ha='right')
plt.ylim(0, 1) # Accuracy is between 0 and 1
plt.tight_layout()
plt.show()
```


Model Accuracy Comparison
Accuracy
1.0
0.8
0.6
0.4
0.2