Shri Vile Parle Kelavani Mand
DWARKADAS J. SANGH
(Autonomous College Affiliated
NAAC Accredited with "A" Grade (
Department of Computer

Experiment 2

NAME:DIVYESH KHUNT SAPID:60009210116 BATCH:D12

Aim: - Implement a Sentiment Analysis on linguistic data to study different feature extraction techniques like bag of words, TF_IDF, word to vector and compare their performance.

Theory:

Sentiment analysis, also known as opinion mining, is a natural language processing task that involves determining the sentiment expressed in a piece of text, such as a review, comment, or tweet. The goal of sentiment analysis is to classify the text into different sentiment categories, such as positive, negative, neutral, or even more fine-grained sentiments.

To implement sentiment analysis and study different feature extraction techniques like bag of words, TF-IDF, and word embeddings (Word2Vec), we need to follow these general steps:

Data Collection and Preprocessing:

Collect the linguistic data (text) along with their corresponding sentiment labels (e.g., positive, negative). Preprocess the text data by removing punctuation, converting to lowercase, removing stop words, and performing lemmatization or stemming.

Different Feature Extraction Techniques:
Implement three different feature extraction techniques: bag of words, TF-IDF, and word embeddings (Word2Vec).

Bag of Words (BoW) Feature Extraction:
Represent each document (piece of text) as a fixed-length vector, where each element corresponds to the frequency of a word in the document.
Create a vocabulary of unique words from the training data and assign an index to each word.
Convert each document into a vector representation by counting the occurrences of each word from the vocabulary.

TF-IDF Feature Extraction:
TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical representation of a document's importance in a corpus of documents.
Calculate the TF-IDF value for each word in each document, which reflects its importance in the document relative to the entire corpus.
Here's how you can calculate TF-IDF for a term in a document:

1. **Term Frequency (TF):** Calculate how often a term appears in a document.
2. **TF (term, document) = (Number of times the term appears in the document) / (Total number of terms in the document)**
3. **Inverse Document Frequency (IDF):** Calculate the logarithmically scaled inverse fraction of the documents that contain the term.
4. **IDF (term, corpus) = log ((Total number of documents in the corpus) / (Number of documents containing the term))**
5. **TF-IDF Score:** Multiply the TF value by the IDF value for a specific term in a specific document.
6. **TF-IDF (term, document, corpus) = TF (term, document) * IDF (term, corpus)**

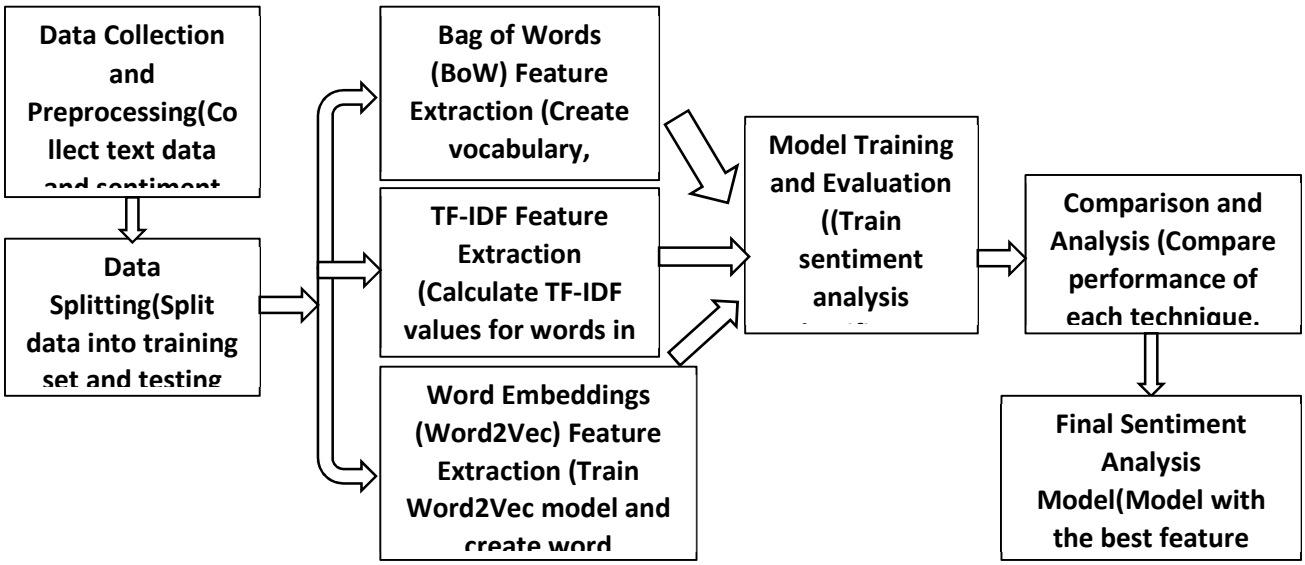
Here's a simple example with some numbers:
Let's say you have a corpus of 100 documents, and you want to calculate the TF-IDF score for the term "apple" in document #5.

- The term "apple" appears 10 times in document #5.
- Document #5 contains a total of 500 terms.
- The term "apple" appears in 30 out of the 100 documents in the corpus.

The calculations would be:

7. **TF (apple, document #5) = 10 / 500 = 0.02**
8. **IDF (apple, corpus) = log (100 / 30) ≈ 0.52**
9. **TF-IDF (apple, document #5, corpus) = 0.02 * 0.52 ≈ 0.01**

Word Embeddings (Word2Vec) Feature Extraction:
Word embeddings are dense vector representations of words that capture semantic meaning and relationships between words.
Train a Word2Vec model on the training data to create word embeddings for each word in the vocabulary.



Lab Experiments to be Performed in This Session: -

Exercise 1: Perform Sentiment Analysis to compare the feature extraction methods like bag of words, TF-IDF, Word to Vector

Data Set: **Amazon Product Reviews Dataset**

1. **Dataset Selection:** Start by selecting an appropriate sentiment analysis dataset that contains text data labeled with positive or negative sentiments. Popular datasets include IMDB Movie Reviews, Twitter Sentiment Analysis Dataset, or Amazon Product Reviews.

2. Preprocessing: Clean and preprocess the text data by removing punctuation, converting text to lowercase, removing stop words, and performing tokenization. Ensure that the data is ready for analysis.

3. Feature Extraction:

a. Bag of Words: Convert the preprocessed text into a numerical vector representation using the Bag of Words model. Each document will be represented as a vector of word frequencies.

b. TF-IDF: Compute the Term Frequency-Inverse Document Frequency (TF-IDF) representation for the text data. TF-IDF accounts for the importance of words by considering both their frequency in a document and their rarity in the entire dataset.

c. Word2Vec: Train a Word2Vec model using the preprocessed text data to convert words into dense word embeddings.

4. Sentiment Analysis Model: Choose a suitable sentiment analysis model (e.g., Naive Bayes, Support Vector Machine, or a deep learning-based model) to classify the sentiment of the text data.

5. Experiment Setup: Split the dataset into training and testing sets. Ensure a consistent and fair evaluation by using the same train-test split across all three feature representations (Bag of Words, TF-IDF, and Word2Vec).

6. Model Training and Evaluation: Train the sentiment analysis model on each of the three feature representations separately and evaluate its performance on the test set using appropriate evaluation metrics such as accuracy, precision, recall, and F1-score.

Exercise 2: Analyze and compare the performance of each feature representation on the sentiment analysis task. Consider factors such as accuracy, computational efficiency, and interpretability.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data = pd.read_csv("IMDB Dataset.csv")
data.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
```

```
#!pip install num2words
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, LancasterStemmer
from nltk.stem import WordNetLemmatizer
```

```
nltk.download('punkt')
nltk.download('stopwords')
```

```
import string
def preprocess_text(text):

    text = text.translate(str.maketrans('', '', string.punctuation))
    tokens = word_tokenize(text)
    tokens = [token.lower() for token in tokens]
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(token) for token in tokens]
    return ' '.join(tokens)
```

```
processed_review = []
for i in range(0, len(data)):
    message = preprocess_text(data.review[i])
    processed_review.append(message)
```

```
data.review[0]
```

```
"One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is ex
actly what happened with me.<br /><br />The first thing that struck me about Oz was its brutality and unflinching scenes of vio
lence, which set in right from the word GO. Trust me, this is not a show for the faint hearted or timid. This show pulls no pun
ches with regards to drugs, sex or violence. Its is hardcore, in the classic use of the word.<br /><br />It is called OZ as tha
t is the nickname given to the Oswald Maximum Security State Penitentary. It focuses mainly on Emerald City, an experimental se
ction of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda. Em City is ho
me to many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, Irish and more....so scuffles, death stares, dodgy dealin
gs and shady agreements are never far away.<br /><br />I would say the main appeal of the show is due to the fact that it goes
where other shows wouldn't dare. Forget pretty pictures painted for mainstream audiences, forget charm, forget romance...OZ doe
sn't mess around. The first episode I ever saw struck me as so nasty it was surreal, I couldn't say I was ready for it, but as
I watched more, I developed a taste for Oz, and got accustomed to the high levels of graphic violence. Not just violence, but i
njustice (crooked guards who'll be sold out for a nickel, inmates who'll kill on order and get away with it, well mannered, mid
dle class inmates being turned into prison bitches due to their lack of street skills or prison experience) Watching Oz, you ma
y become comfortable with what is uncomfortable viewing....thats if you can get in touch with your darker side."
```

In [22]:
```
processed_review[0]
```

Out[22]:
```
['one review mention watch 1 oz episod youll hook right exactli happen mebr br first thing struck oz brutal unflinch scene vi
olenc set right word go trust show faint heart timid show pull punch regard drug sex violenc hardcor classic use wordbr br ca
ll oz nicknam given oswald maximum secur state penitentari focus mainli emerald citi experiment section prison cell glass fro
nt face inward privaci high agenda em citi home manyaryan muslim gangsta latino christian italian irish moreso scuffl death s
tare dodgi deal shadi agreement never far awaybr br would say main appeal show due fact goe show wouldnt dare forget pretti p
ictur paint mainstream audienc forget charm forget romanceoz doesnt mess around first episod ever saw struck nasti surreal co
uldnt say readi watch develop tast oz got accustom high level graphic violenc violenc injustic crook guard wholl sold nickel
inmat wholl kill order get away well manner middl class inmat turn prison bitch due lack street skill prison experi watch oz
may becom comfort uncomfort viewingthat get touch darker side',
 'wonder littl product br br film techniqu unassum oldtimebbc fashion give comfort sometim discomfort sens realism entir piec
br br actor extrem well chosen michael sheen got polari voic pat truli see seamless edit guid refer william diari entri well
worth watch terrificli written perform piec master product one great master comedi life br br realism realli come home littl
thing fantasi guard rather use tradit dream techniqu remain solid disappear play knowledg sens particularli scene concern ort
on halliwel set particularli flat halliwel mural decor everi surfac terribl well done',
 'thought wonder way spend time hot summer weekend sit air condit theater watch lightheart comedi plot simplist dialogu witti
charact likabl even well bread suspect serial killer may disappoint realiz match point 2 risk addict thought proof woodi alle
n still fulli control style mani us grown lovebr br id laugh one woodi comedi year dare say decad ive never impress scarlet j
ohanson manag tone sexi imag jump right averag spirit young womanbr br may crown jewel career wittier devil wear prada intere
st superman great comedi go see friend',
```

In [24]:
```
clean_data = data.drop(["review"],axis=1)
clean_data['processed_review'] = processed_review
clean_data.head()
```

Out[24]:

	sentiment	processed_review
0	positive	one review mention watch 1 oz episod youll hoo...
1	positive	wonder littl product br br film techniqu unass...
2	positive	thought wonder way spend time hot summer weeke...
3	negative	basic there famili littl boy jake think there ...
4	positive	petter mattei love time money visual stun film...

```
]: ##bag of words
```

```
]: y = clean_data["sentiment"].values
x = clean_data["processed_review"].values

from sklearn.model_selection import train_test_split
xtrain , xtest , ytrain , ytest = train_test_split(x,y, test_size = 0.2, random_state = 1)

print(xtrain.shape, ytrain.shape, xtest.shape, ytest.shape)
```

```
(40000,) (40000,) (10000,) (10000,)
```

```
]: from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()

xtrain_bow = vectorizer.fit_transform(xtrain)

xtest_bow = vectorizer.transform(xtest)

print("Shape of xtrain_bow:", xtrain_bow.shape)
print("Shape of xtest_bow:", xtest_bow.shape)
```

```
Shape of xtrain_bow: (40000, 124577)
Shape of xtest_bow: (10000, 124577)
```

```
##TI-IDF
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer_tfidf = TfidfVectorizer()

xtrain_tfidf = vectorizer_tfidf.fit_transform(xtrain)

# Transform the test data
xtest_tfidf = vectorizer_tfidf.transform(xtest)

print("Shape of xtrain_bow:", xtrain_tfidf.shape)
print("Shape of xtest_bow:", xtest_tfidf.shape)
```

```
Shape of xtrain_bow: (40000, 124577)
Shape of xtest_bow: (10000, 124577)
```

```
pip install gensim
```

```
Collecting gensim
  Using cached gensim-4.3.3-cp38-cp38-win_amd64.whl (24.0 MB)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in c:\users\djsce.student\anaconda3\lib\site-p
1)
Requirement already satisfied: smart-open>=1.8.1 in c:\users\djsce.student\anaconda3\lib\site-pack
Requirement already satisfied: numpy<2.0,>=1.18.5 in c:\users\djsce.student\anaconda3\lib\site-pac
Requirement already satisfied: wrapt in c:\users\djsce.student\anaconda3\lib\site-packages (from s
12.1)
Installing collected packages: gensim
Successfully installed gensim-4.3.3
Note: you may need to restart the kernel to use updated packages.
```

```
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess

xtrain_tokens = [simple_preprocess(review) for review in xtrain]
xtest_tokens = [simple_preprocess(review) for review in xtest]

word2vec_model = Word2Vec(sentences=xtrain_tokens, vector_size=100, window=5, min_count=1, workers

def get_average_vector(tokens, model):
    vectors = [model.wv[token] for token in tokens if token in model.wv]
    return np.mean(vectors, axis=0) if vectors else np.zeros(model.vector_size)

xtrain_w2v = np.array([get_average_vector(tokens, word2vec_model) for tokens in xtrain_tokens])
xtest_w2v = np.array([get_average_vector(tokens, word2vec_model) for tokens in xtest_tokens])

print("Shape of xtrain_bow:", xtrain_w2v.shape)
print("Shape of xtest_bow:", xtest_w2v.shape)
```

```
Shape of xtrain_bow: (40000, 100)
Shape of xtest_bow: (10000, 100)
```

```
: from sklearn.metrics import classification_report
```

```
: nb_classifier.fit(xtrain_bow, ytrain)
  y_pred_bow = nb_classifier.predict(xtest_bow)
  print("Bag of Words Model Evaluation:")
  print(classification_report(ytest, y_pred_bow, target_names=['Positive', 'Negative']))
```

```
Bag of Words Model Evaluation:
              precision    recall  f1-score   support

    Positive       0.85      0.88      0.86      5044
    Negative       0.87      0.84      0.85      4956

    accuracy                           0.86     10000
   macro avg       0.86      0.86      0.86     10000
weighted avg       0.86      0.86      0.86     10000
```

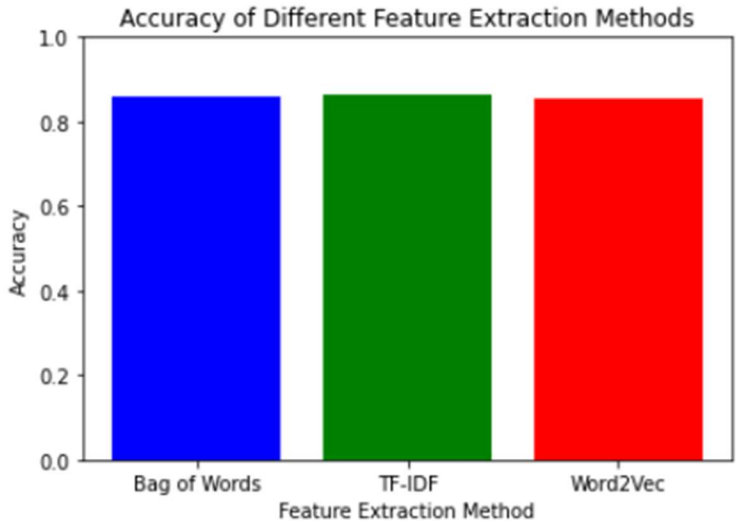
```
: nb_classifier.fit(xtrain_tfidf, ytrain)
  y_pred_tfidf = nb_classifier.predict(xtest_tfidf)
  print("TF-IDF Model Evaluation:")
  print(classification_report(ytest, y_pred_tfidf, target_names=['Positive', 'Negative']))
```

```
TF-IDF Model Evaluation:
              precision    recall  f1-score   support

    Positive       0.86      0.88      0.87      5044
    Negative       0.87      0.85      0.86      4956

    accuracy                           0.86     10000
   macro avg       0.86      0.86      0.86     10000
weighted avg       0.86      0.86      0.86     10000
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report

gnb_classifier = GaussianNB()

gnb_classifier.fit(xtrain_w2v, ytrain)
y_pred_w2v = gnb_classifier.predict(xtest_w2v)

print("Word2Vec Model Evaluation:")
print(classification_report(ytest, y_pred_w2v, target_names=['Positive', 'Negative']))
```

```
Word2Vec Model Evaluation:
              precision    recall  f1-score   support

    Positive       0.77      0.77      0.77      5044
    Negative       0.76      0.76      0.76      4956

    accuracy                           0.76     10000
   macro avg       0.76      0.76      0.76     10000
weighted avg       0.76      0.76      0.76     10000
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

accuracies = {
    'Bag of Words': accuracy_score(ytest, y_pred_bow),
    'TF-IDF': accuracy_score(ytest, y_pred_tfidf),
    'Word2Vec': accuracy_score(ytest, y_pred_w2v)
}

fig, ax = plt.subplots()

ax.bar(accuracies.keys(), accuracies.values(), color=['blue', 'green', 'red'])

ax.set_xlabel('Feature Extraction Method')
ax.set_ylabel('Accuracy')
ax.set_title('Accuracy of Different Feature Extraction Methods')
ax.set_ylim(0, 1)

plt.show()
```



Conclusion:

- **Best Performing Model:** The TF-IDF model performed slightly better than the Bag of Words model in terms of F1-score and recall, especially for the positive class. Both the Bag of Words and TF-IDF models achieved the same accuracy (0.86).
- **Worst Performing Model:** The Word2Vec model had the lowest performance with an accuracy of 0.77, and its precision, recall, and F1-score were also significantly lower than those of the other two models.