Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)
Department of Computer Science and Engineering (Data Science)

Advanced Computational Linguistics

Experiment 2

Name: Umang Kirit Lodaya **SAP ID: 60009200032**

Batch: K – 1 / D11

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.

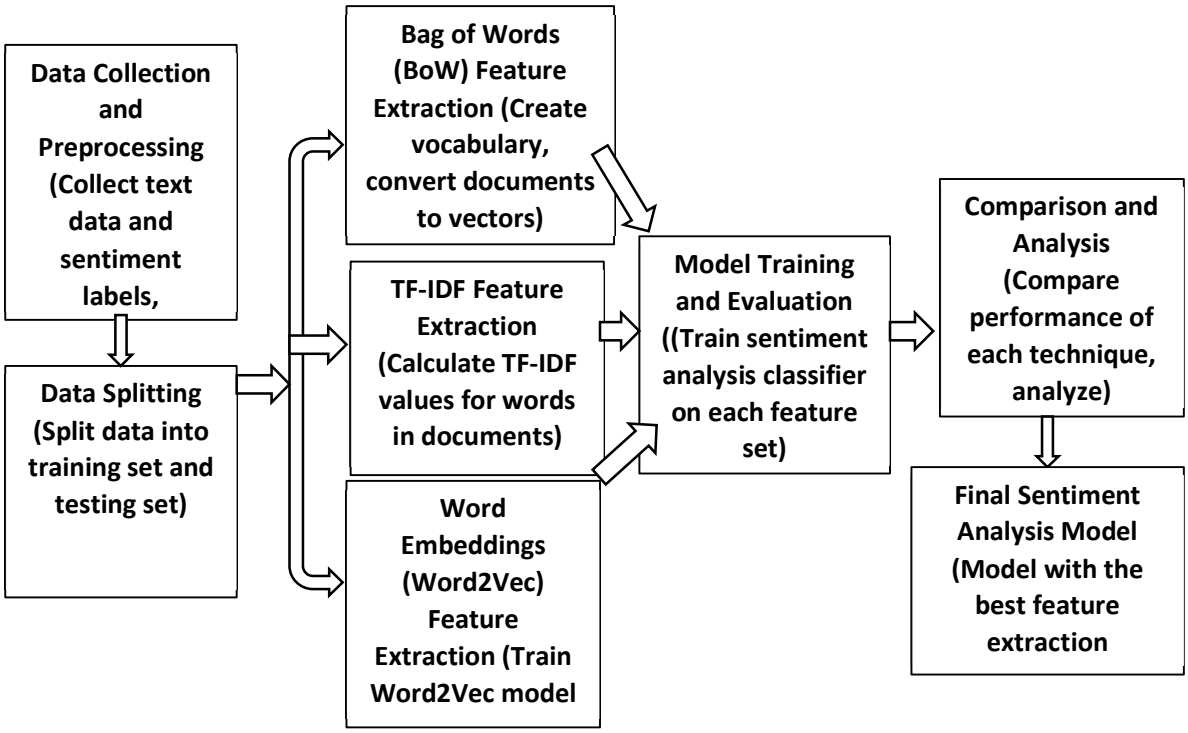
The calculations would be:

1. **TF (apple, document #5) = 10 / 500 = 0.02**
2. **IDF (apple, corpus) = log (100 / 30) ≈ 0.52**
3. **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.

NAME: UMANG KIRIT LODAYA

SAP ID: 60009200032

BATCH: K - K1 / D11

IMPORTING LIBRARIES

In [1]:

```
import numpy as np
import pandas as pd

from tqdm import tqdm

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/imdb-dataset-of-50k-movie-reviews/IMDB Dataset.csv
/kaggle/input/amazon-kindle-book-review-for-sentiment-analysis/all_k
indle_review .csv
/kaggle/input/amazon-kindle-book-review-for-sentiment-analysis/prepr
ocessed_kindle_review .csv
```

In [2]:

```
import re
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data.
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /usr/share/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[2]:

```
True
```

In [3]:

```
from nltk.corpus import stopwords
from nltk import sent_tokenize

from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer

ps = PorterStemmer()
```

In [4]:

```
from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, classification_report
```

IMPORTING DATA

In [5]:

```
df = pd.read_csv('/kaggle/input/imdb-dataset-of-50k-movie-reviews/IMDB Dataset
sv')
df = df.iloc[:2*df.shape[0]//5, :]

df['rating'] = df['sentiment'].apply(lambda x: 1 if x == 'positive' else 0)
df.head()
```

Out[5]:

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

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   review     20000 non-null  object
 1   sentiment  20000 non-null  object
 2   rating     20000 non-null  int64 
dtypes: int64(1), object(2)
memory usage: 468.9+ KB
```

In [7]:

```
df['rating'].value_counts()
```

Out[7]:

```
rating
0    10097
1     9903
Name: count, dtype: int64
```

In [8]:

```
corpus = []
for i in tqdm(range(0, len(df))):
#   REPLACING NON-ALPHABETICAL WORDS WITH SPACES
    sentence = re.sub('[^a-zA-Z]', ' ', df['review'][i])
    sentence = sentence.lower()
    sentence = sentence.split()
#   STEMMING NON STOP-WORDS
    sentence = [ps.stem(word) for word in sentence if not word in stopwords.word
s('english')]
    sentence = ' '.join(sentence)

    corpus.append(sentence)
```

```
100%|██████████| 20000/20000 [10:56<00:00, 30.48it/s]
```

In [9]:

```
Y = pd.get_dummies(df['rating'])
Y = Y.iloc[:, 1].values
```

BAG OF WORDS MODEL

In [10]:

```
cv = CountVectorizer(max_features = 2500)
```

In [11]:

```
X = cv.fit_transform(corpus).toarray()
```

In [12]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.30, ra
om_state=0)
```

In [13]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(14000, 2500)
(14000,)
```

In [14]:

```
model = LogisticRegression(solver='liblinear').fit(X_train, Y_train)
Y_pred = model.predict(X_test)
```

In [15]:

```
print('ACCURACY:', round(accuracy_score(Y_test, Y_pred), 3)*100)
```

```
ACCURACY: 85.8
```

In [16]:

```
print(classification_report(Y_pred, Y_test))
```

```
              precision    recall  f1-score   support

       False       0.85      0.87      0.86      2940
        True       0.87      0.85      0.86      3060

    accuracy                           0.86      6000
   macro avg       0.86      0.86      0.86      6000
weighted avg       0.86      0.86      0.86      6000
```

TF-IDF MODEL

In [17]:

```
tv = TfidfVectorizer(max_features = 2500)
X = tv.fit_transform(corpus).toarray()
```

In [18]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.30, ra
om_state = 0)
```

In [19]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(14000, 2500)
(14000,)
```

In [20]:

```
model = LogisticRegression(solver='liblinear').fit(X_train, Y_train)
Y_pred = model.predict(X_test)
```

In [21]:

```
print('ACCURACY:', round(accuracy_score(Y_test, Y_pred), 3)*100)
```

```
ACCURACY: 87.9
```

In [22]:

```
print(classification_report(Y_pred, Y_test))
```

```
              precision    recall  f1-score   support

       False       0.86      0.89      0.88      2922
        True       0.89      0.87      0.88      3078

    accuracy                           0.88      6000
   macro avg       0.88      0.88      0.88      6000
weighted avg       0.88      0.88      0.88      6000
```

Word2Vec MODEL

In [23]:

```
import gensim
from gensim.utils import simple_preprocess
```

In [24]:

```
words = []
for sent in corpus:
    sent_token = sent_tokenize(sent)
    for sent in sent_token:
        words.append(simple_preprocess(sent))
```

In [25]:

```
word2vec = gensim.models.Word2Vec(words, window = 5, min_count = 2)
```

In [26]:

```
print(word2vec.corpus_count)
print(word2vec.epochs)
```

```
20000
5
```

In [27]:

```
def avg_word2vec(doc):
    return np.mean([word2vec.wv[word] for word in doc if word in word2vec.wv.ind
ex_to_key], axis=0)
```

In [28]:

```
X = []
for i in tqdm(range(len(words))):
    X.append(avg_word2vec(words[i]))

X_new = np.array(X)
```

```
100%|██████████| 20000/20000 [02:24<00:00, 138.04it/s]
```

In [29]:

```
X_new.shape
```

Out[29]:

```
(20000, 100)
```

In [30]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X_new, Y, test_size=0.30,
ndom_state=0)
```

In [31]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(14000, 100)
(14000,)
```

In [32]:

```
model = LogisticRegression(solver='liblinear').fit(X_train, Y_train)
Y_pred = model.predict(X_test)
```

In [33]:

```
print('ACCURACY:', round(accuracy_score(Y_test, Y_pred), 3)*100)
```

```
ACCURACY: 84.2
```

In [34]:

```
print(classification_report(Y_pred,Y_test))
```

```
              precision    recall  f1-score   support

       False       0.84      0.85      0.84      2970
        True       0.85      0.84      0.84      3030

    accuracy                           0.84      6000
   macro avg       0.84      0.84      0.84      6000
weighted avg       0.84      0.84      0.84      6000
```

ACL PRACTICAL 2 – TASK

Report on Research Papers

Name: Umang Kirit Lodaya **SAP ID: 60009200032**

Batch: K – 1 / D11

" Performance Comparison of TF-IDF and Word2Vec Models for Emotion Text Classification"

Introduction:

This research paper explores the classification of human emotions in text, emphasizing the importance of feature extraction techniques. Emotions are expressed in text on social media, and the study aims to classify them into six categories: happy, sad, angry, scared, surprised, and disgusted. The paper highlights the significance of emotion classification in various fields and introduces the TF-IDF and Word2Vec models as feature extraction techniques.

Feature Extraction:

The paper discusses the feature extraction process, which involves converting unstructured textual data into structured data for machine learning algorithms. Two common techniques, TF-IDF and Word2Vec, are introduced. TF-IDF generates high-dimensional data, while Word2Vec produces low-dimensional data. The choice of feature extraction method can significantly impact classification performance.

Research Goal:

The primary goal of the research is to compare the performance of TF-IDF and Word2Vec models for text classification. The objective is to identify the best feature extraction model that ensures faster computation and improved classification performance.

Methodology:

The paper outlines the research methodology, which includes preprocessing steps such as case folding, filtering, normalization, stop words removal, and stemming. TF-IDF and Word2Vec models are explained in detail. TF-IDF involves term frequency-inverse document frequency calculations, while Word2Vec represents words in vector form using neural networks.

Emotion Classification using SVM:

The study utilizes the Support Vector Machine (SVM) classification method and applies 10-fold cross-validation. SVM is chosen for its superior text classification performance. The paper briefly mentions kernel functions and emphasizes the use of a linear kernel function.

Evaluation and Analysis:

The evaluation stage involves calculating accuracy, precision, recall, and F1-measure for classification performance. The paper presents the experimental dataset, emphasizing the use of Indonesian language data from tweets of Transjakarta and Commuterline users. The results are compared with previous research that used Multinomial Naïve Bayes (MNB) with TF-IDF.

Results and Discussion:

The research presents the results and discusses them comprehensively. The SVM with TF-IDF method outperforms other methods in terms of accuracy, precision, recall, and F1-measure, both in the first and second steps of classification. The study highlights that TF-IDF modeling is superior to Word2Vec modeling due to the unbalanced data distribution in emotional classes.

Conclusion:

The paper concludes that the SVM with TF-IDF method is the best approach for emotion text classification, achieving high accuracy and recognition of all emotion categories. It also emphasizes the importance of using balanced and large datasets in future research. Additionally, the possibility of combining TF-IDF and Word2Vec for feature extraction in text classification is mentioned as a potential avenue for future work.

"Comparing Feature Extraction Methods and Effects of Pre-Processing Methods for Multi-Label Classification of Textual Data"

Introduction

The introduction section sets the context for the research. It explains the significance of multi-label text classification and the challenges associated with it. The authors emphasize the need for effective feature extraction methods and pre-processing techniques. They outline the research objectives, highlighting the importance of finding the optimal combination of these methods for improved classification performance.

Objective of the Study

The primary objective of the research is to provide insights into how different combinations of feature extraction and pre-processing techniques can improve the accuracy of classifying textual data into multiple labels, thereby contributing to the field of natural language processing and text classification.

Methodology:

The methodology section outlines the research design and procedures. The authors detail the datasets used for experimentation and describe the feature extraction methods employed, including Term Frequency-Inverse Document Frequency (TF-IDF), Word Embeddings (Word2Vec), and others. They also explain the pre-processing techniques applied, such as stop-word removal, stemming, and lowercasing.

Experiments and Results:

This section presents the experiments conducted and the results obtained. The authors describe how they evaluated the performance of different feature extraction and pre-processing combinations using various evaluation metrics. They provide tables and graphs to showcase the experimental results, highlighting the effectiveness of specific methods and combinations in multi-label text classification.

Discussion:

In the discussion section, the authors analyze and interpret the results obtained from the experiments. They discuss the impact of feature extraction and pre-processing methods on multi-label text classification accuracy. The authors also address any unexpected findings, limitations of the study, and potential areas for further research.

Conclusion

The conclusion summarizes the key findings of the research and their implications. The authors reiterate the importance of selecting appropriate feature extraction and pre-processing methods for multi-label text classification tasks. They provide insights into the most effective combinations identified in the study and suggest directions for future research in this field.