CS 6120: Natural Language Processing - Prof. Ahmad Uzair

Assignment 2: Text Classification and Neural Network

Total Points: 100 points

In Assignment 2, you will be dealing with text classification using Multinomial Naive Bayes and Neural Networks. You will also be dealing with vector visualization.In this assignment you will be using TF-IDF Vectorization instead of Bag of Words. We recommend starting with this assignment a little early as the datasets are quite large and several parts of the assignment might take long duration to execute.

Question 1 Text Classification

In the first question you will be dealing with 20 News Group Dataset. You are required to implement TF-IDF vectorization from scratch and perform Multinomial Naive Bayes Classification on the News Group Dataset. You may use appropriate packages or modules for fitting the Multinomial Naive Bayes Model, however, the implementation of the TF-IDF Vectorization should be from the scratch.

The 20 newsgroups dataset comprises around 20000 newsgroups posts on 20 topics split in two subsets: one for training (or development) and the other one for testing (or for performance evaluation). The split between the train and test set is based upon a messages posted before and after a specific date.

Link to the original dataset: http://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups

You can also import the dataset from sklearn.datasets

In [ ]:
```
#importing the necessary libraries

import numpy as np
import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.datasets import fetch_20newsgroups
from pprint import pprint
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import preprocessing
import pandas as pd
import re
import numpy as np
from nltk.tokenize import  word_tokenize
import nltk
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
```

```
/var/folders/4p/7qv47hkj16s6wsk000sgyl0r0000gn/T/ipykernel_17763/3370255843.p
y:10: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release
of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and bette
r interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/5446
6

  import pandas as pd
```

In [ ]:
```python
# Import the 20 news group dataset utilizing sklearn library
from sklearn.datasets import fetch_20newsgroups
mydata_train = fetch_20newsgroups(subset='train')

mydata_test =  fetch_20newsgroups(subset='test')
```

In [ ]:
```python
# Printing the news groups(target) in the dataset

pprint(list(mydata_train.target_names))
```

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

In [ ]:
```python
# What is the type of 'mydata_train' and 'mydata_test'

print(type(mydata_train))
print(type(mydata_test))

# Type : Bunch : Container object exposing keys as attributes.
# They extend dictionaries by enabling values to be accessed by key, bunch["va
```

```
<class 'sklearn.utils._bunch.Bunch'>
<class 'sklearn.utils._bunch.Bunch'>
```

In [ ]:
```python
# Check the length of the data

print(len(mydata_train.data))
print(len(mydata_train.filenames))
```

```
print(len(mydata_test.data))
print(len(mydata_test.filenames))
```

```
11314
11314
7532
7532
```

Expected Output:

11314

11314

7532

7532

Extracting Features from the Dataset (20 Points)

In order to perform machine learning on text documents, we first need to turn the text content into numerical feature vectors.

TF-IDF Vectorization

Our model cannot simply read the text data so we convert it into numerical format. In order to convert the data into numerical format we create vectors from text.

For this particular purpose we could either employ Bag of Words or TF-IDF Vectorization

Bag of Words just creates a set of vectors containing the count of word occurrences in the document (reviews), while the TF-IDF model contains information on the more important words and the less important ones as well.

TF-IDF stands for Term Frequency-Inverse Document Frequency, which instead of giving more weight to words that occur more frequently, it gives a higher weight to words that occur less frequently.

Ref:https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf/#:~:text=Bag%20of%20Words%20just%20creates,less%20important%20ones%20as%20w

TF-IDF = Term Frequency (TF) * Inverse Document Frequency (IDF)

Term Frequency is the measure of the frequency of words in a document. It is the ratio of the number of times the word appears in a document compared to the total number of words in that document.

The words that occur rarely in the corpus have a high IDF score. It is the log of the ratio of the number of documents to the number of documents containing the word.

idf(t) = log(N/(df + 1))

In [ ]:
```
# Importing the test and train dataset
text = mydata_train.data
test = mydata_test.data
```

Preprocessing the Corpus

In [ ]:
```
# Preprocessing the data

lines = []
word_list = []

for line in text:
    #tokenize the text documents and update the lists word_list and lines
    words = [word.lower() for word in word_tokenize(line) if word.isalpha()]
    #print("words", words)
    lines.append(words)
    for word in words:
        if word not in word_list:
            word_list.append(word)



# Make sure the word_list contains unique tokens
word_list = set(word_list)

# Calculate the total documents present in the corpus
total_docs = len(text)

#Create a dictionary to keep track of index of each word
dict_idx = {}
for i, word in enumerate(word_list):
    dict_idx[word] = i
```

In [ ]:
```
# Printing subset
print(lines[:2])
```

```
[['from', 'lerxst', 'where', 'my', 'thing', 'subject', 'what', 'car', 'is', 't
his', 'organization', 'university', 'of', 'maryland', 'college', 'park', 'line
s', 'i', 'was', 'wondering', 'if', 'anyone', 'out', 'there', 'could', 'enlight
en', 'me', 'on', 'this', 'car', 'i', 'saw', 'the', 'other', 'day', 'it', 'wa
s', 'a', 'sports', 'car', 'looked', 'to', 'be', 'from', 'the', 'late', 'earl
y', 'it', 'was', 'called', 'a', 'bricklin', 'the', 'doors', 'were', 'really',
'small', 'in', 'addition', 'the', 'front', 'bumper', 'was', 'separate', 'fro
m', 'the', 'rest', 'of', 'the', 'body', 'this', 'is', 'all', 'i', 'know', 'i
f', 'anyone', 'can', 'tellme', 'a', 'model', 'name', 'engine', 'specs', 'year
s', 'of', 'production', 'where', 'this', 'car', 'is', 'made', 'history', 'or',
'whatever', 'info', 'you', 'have', 'on', 'this', 'funky', 'looking', 'car', 'p
lease', 'thanks', 'il', 'brought', 'to', 'you', 'by', 'your', 'neighborhood',
'lerxst'], ['from', 'guykuo', 'guy', 'kuo', 'subject', 'si', 'clock', 'poll',
'final', 'call', 'summary', 'final', 'call', 'for', 'si', 'clock', 'reports',
'keywords', 'si', 'acceleration', 'clock', 'upgrade', 'organization', 'univers
ity', 'of', 'washington', 'lines', 'a', 'fair', 'number', 'of', 'brave', 'soul
s', 'who', 'upgraded', 'their', 'si', 'clock', 'oscillator', 'have', 'shared',
'their', 'experiences', 'for', 'this', 'poll', 'please', 'send', 'a', 'brief',
'message', 'detailing', 'your', 'experiences', 'with', 'the', 'procedure', 'to
p', 'speed', 'attained', 'cpu', 'rated', 'speed', 'add', 'on', 'cards', 'and',
'adapters', 'heat', 'sinks', 'hour', 'of', 'usage', 'per', 'day', 'floppy', 'd
isk', 'functionality', 'with', 'and', 'm', 'floppies', 'are', 'especially', 'r
equested', 'i', 'will', 'be', 'summarizing', 'in', 'the', 'next', 'two', 'day
s', 'so', 'please', 'add', 'to', 'the', 'network', 'knowledge', 'base', 'if',
'you', 'have', 'done', 'the', 'clock', 'upgrade', 'and', 'have', 'answered',
'this', 'poll', 'thanks', 'guy', 'kuo', 'guykuo']]
```

In [ ]:

```python
# Create a frequency dictionary

def frequency_dict(lines):
    '''
    lines: list containing all the tokens
    ---
    freq_word: returns a dictionary which keeps the count of the number of docu
    '''
    freq_word =  {}
    for word in word_list:
        freq_word[word] = 0
    for line in lines:
        for word in line:
            freq_word[word]+= 1
    return freq_word
```

In [ ]:

```python
# Create a dictionary containing the frequency of words utilizing the 'frequenc

# Expect this chunk to take a comparatively longer time to execute since our da

freq_word = frequency_dict(lines)
```

In [ ]:

```python
# Create a function to calculate the Term Frequency

# Term frequency measures the frequently of a term that appears in a document.
def term_frequency(document, word):
    '''
    document: list containing the entire corpus
    word: word whose term frequency is to be calculated
    ---
    tf: returns term frequency value
    '''
```

```
    # Calculating the length of the list containing the entire corpus
    n = len(document)
    # Calculating the number of time the word occurs in the document
    occur = len([token for token in document if token == word ])
    # Calculating the term frequency
    tf = occur/n
    return tf
```

In [ ]:
```
# Create a function to calculate the Inverse Document Frequency

def inverse_df(word):
    '''
    word: word whose inverse document frequency is to be calculated
    ---
    idf: return inverse document frequency value
    '''
    try:
        word_occur = freq_word[word] + 1
    except:
        word_occur = 1
    # Calculating the inverse document frequency for each word present in the 
    idf = np.log(total_docs / word_occur)

    return idf
```

In [ ]:
```
#Create a function to combine the term frequencies (TF) and inverse document (I

def tfidf(lines):
    '''
    sentence: list containing the entire corpus
    dict: dictionary keeping track of index of each word
    ---
    tf_idf_vec: returns computed tf-idf
    '''
    tf_idf_vec = np.zeros((len(word_list),))
    for word in lines:
        tf = term_frequency(lines, word)
        idf = inverse_df(word)
        tf_idf_vec[dict_idx[word]] = tf * idf

    return tf_idf_vec
```

In [ ]:
```
#Compute the vectors utilizing the 'tfidf' function created above to obtain a 

computed_vectors = []
for line in lines:
    computed_vectors.append(tfidf(line ))
```

Multinomial Naive Bayes (10 Points)

In [ ]:
```
#Fit a Multinomial Naive Bayes Model on our dataset
from sklearn.naive_bayes import MultinomialNB

# Importing the data and target variables for training set
X_train = mydata_train.data
y_train = mydata_train.target
```

```
# Building the TFIDF vectors for training set
tfidf = TfidfVectorizer()
X_train_tfidf = tfidf.fit_transform(X_train)

# Initialzing the Multinomial NB function
model = MultinomialNB()
model.fit(X_train_tfidf, y_train)
```

Out[ ]:

```
▾ MultinomialNB
MultinomialNB()
```

In [ ]:

```
#Perform testing on the testing dataset

# Importing the data and target variables for testing set
X_test = mydata_test.data
y_test = mydata_test.target

# Building the TFIDF vectors
X_test_tfidf = tfidf.transform(X_test)

# Predicting
pred = model.predict(X_test_tfidf)
```

In [ ]:

```
# Importing necessary libarraies
from sklearn.metrics import f1_score, accuracy_score

# Calculating the metrics - Accuracy and F1 score
F1_score = f1_score(y_test, pred, average='weighted')
Accuracy = accuracy_score(y_test, pred)
print(" F1 Score: ", F1_score)
print(" Accuracy: ", Accuracy)
```

```
 F1 Score:  0.7684457156894656
 Accuracy:  0.7738980350504514
```

Question 2 Vector Visualization

In this unsupervised learning task we are going to cluster wikipedia articles into groups using T-SNE visualization after vectorization.

Collect articles from Wikipedia (10 points)

In this section we will download articles from wikipedia and then vectorize them in the next step. You can select somewhat related topics or fetch the articles randomly. (Use dir() and help() functions or refer wikipedia documentation) You may also pick any other data source of your choice instead of wikipedia.

In [ ]:

```
# # install libraries
# ! pip install wikipedia

# Import wikipedia library
```

```
import wikipedia
from wikipedia.exceptions import WikipediaException
```

In [ ]:

```
'''
 Generate a list of wikipedia article to cluster
 You can maintain a static list of titles or generate them randomly using wiki
 Some topics include:
 ["Northeastern Unversity", "Natural language processing", "Machine learning",
 "Bank of America", "Visa Inc.", "European Central Bank", "Bank", "Financial te
 "Basketball", "Swimming", "Tennis", "Football", "College Football", "Associat

 You can add more topics from different categories so that we have a diverse da
 Ex- About 3+ categories(groups), 3+ topics in each category, 3+ articles in ea
'''

# selected topics
topics = ["Harvard University", "Boston University", "Natural language process
 "Bank of America", "Visa Inc.", "European Central Bank", "Basketball", "Tennis

# list of articles to be downloaded
articles = []
for topic in topics:
    topic_results = wikipedia.search(topic)
    selected_articles = topic_results[:3]
    articles.extend(selected_articles)
#print(articles)

# download and store articles (summaries) in this variable
data = []
newTopics = []
for article in articles:
    try:
        summary = wikipedia.summary(article)
        data.append(summary)
        newTopics.append(article)
    except Exception as e:
        pass

# for i in range(len(data)):
#     print(f"{newTopics[i]}: {data[i][:500]}...")
```

```
/Users/paarthvisharma/Documents/Spring 2024/NLP/NLP/Assignment 2/nlp2/lib/pyth
on3.10/site-packages/wikipedia/wikipedia.py:389: GuessedAtParserWarning: No pa
rser was explicitly specified, so I'm using the best available HTML parser for
this system ("html.parser"). This usually isn't a problem, but if you run this
code on another system, or in a different virtual environment, it may use a di
fferent parser and behave differently.

The code that caused this warning is on line 389 of the file /Users/paarthvish
arma/Documents/Spring 2024/NLP/NLP/Assignment 2/nlp2/lib/python3.10/site-packa
ges/wikipedia/wikipedia.py. To get rid of this warning, pass the additional ar
gument 'features="html.parser"' to the BeautifulSoup constructor.

  lis = BeautifulSoup(html).find_all('li')
```

Cleaning the Data (5 points)

In this step you will decide whether to clean the data or not. If you choose to clean, you may utilize the clean function from assignment 1.

Question: Why are you (not) choosing to clean the data? Think in terms of whether cleaning data will help in the clustering or not.

Answer(1-3 sentences):

In [ ]:
```
import re
import nltk
import string
nltk.download('punkt')
nltk.download('stopwords')

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
wn = nltk.WordNetLemmatizer()

stopwords = stopwords.words('english')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/paarthvisharma/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/paarthvisharma/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [ ]:
```
# You can use Assignment 1's clean message function

def clean_text(text):
  # From the last assignment
    text = text.lower()
    text = re.sub(r"http\S+", "", text)
    text = re.sub(r"www.\S+", "", text)
    text_links_removed = "".join([char for char in text if char not in string.p
    text_cleaned = " ".join([word for word in re.split('\W+', text_links_remov
        if word not in stopwords])
    text = " ".join([wn.lemmatize(word) for word in re.split('\W+', text_clean
    return text
```

In [ ]:
```
# Cleaning the data and appending it into a new string
toReturn = ""
for i in range(len(data)):
    data[i] = clean_text(data[i])
    toReturn += (data[i] + "\n")
# print(toReturn)
```

Vectorize the articles (5 points)

In this step, we will vectorize the text data. You can use TfidfVectorizer() or countVectorizer() from sklearn library.

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk import word_tokenize
```

```
# Creating vectors for the data
tfidf = TfidfVectorizer(tokenizer=word_tokenize)
X = tfidf.fit_transform(data)
```

```
/Users/paarthvisharma/Documents/Spring 2024/NLP/NLP/Assignment 2/nlp2/lib/pyth
on3.10/site-packages/sklearn/feature_extraction/text.py:525: UserWarning: The
parameter 'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
```

In [ ]:
```
print(X.shape)
```

```
(36, 2474)
```

Sample Output:

(36, 1552)

Plot Articles (10 points)

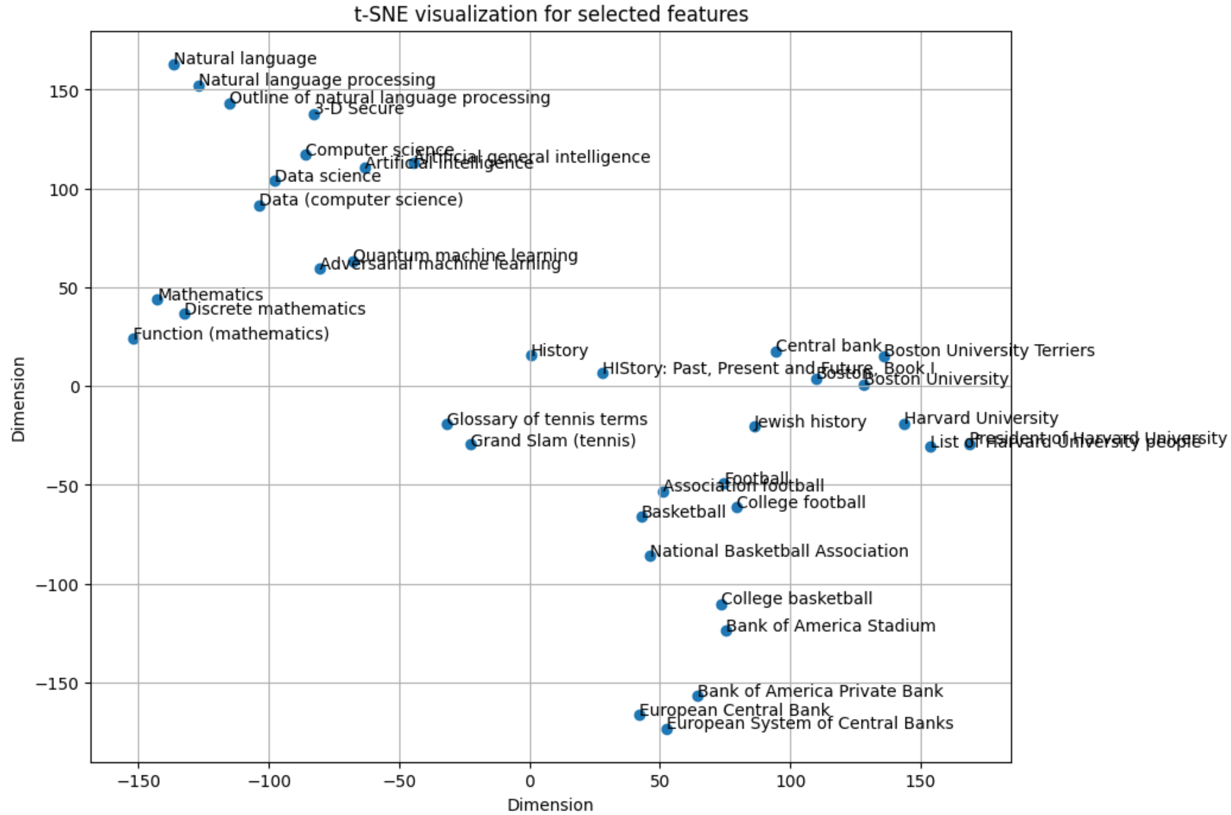
Now we will try to verify the groups of articles using T-SNE from sklearn library.

In [ ]:
```
from sklearn.manifold import TSNE

# Calling the TSNE() to fit and transform the data

tsne = TSNE(n_components=2, perplexity=3)
tsne_vec = tsne.fit_transform(X.toarray())
print(tsne_vec)
```

```
[[  35.755505   134.28542  ]
 [  49.055656   155.00964  ]
 [  45.35424    142.61377  ]
 [  20.458288   115.94647  ]
 [   4.156275   114.01497  ]
 [  26.859695   103.49555  ]
 [  -2.152786  -124.083855 ]
 [  -8.880749  -134.60587  ]
 [   4.3234468 -112.950294 ]
 [  94.225044  -112.653885 ]
 [  86.483765  -120.48485  ]
 [  51.69689    -69.49294  ]
 [  60.01469    -55.432274 ]
 [  44.889526   -99.148026 ]
 [  51.35628   -108.54946  ]
 [ -93.81609     59.54636  ]
 [-105.7229      98.4351   ]
 [  21.548717   -77.94378  ]
 [-102.763695   119.00361  ]
 [ -12.028529   121.85576  ]
 [ -94.6759     111.76052  ]
 [ -35.384228    31.777021 ]
 [ -52.686607    31.464855 ]
 [ -87.30568     50.199562 ]
 [-504.8213    -414.9873   ]
 [ 186.044      446.32864  ]
 [ -26.826258    62.28375  ]
 [ -25.422009    40.642246 ]
 [ -37.9684      63.13024  ]
 [ -31.220434   -58.020287 ]
 [ -49.08662    -54.66867  ]
 [ -34.76212    -68.29659  ]
 [  38.609283   -84.899895 ]
 [  60.292683    12.892051 ]
 [  65.50948     22.42746  ]
 [  -9.919654    84.67454  ]]
```

Plot and annotate the points with different markers for different expected groups.

In [ ]:
```python
# Plotting the graph
import matplotlib.pyplot as plt

# Figure
fig, ax = plt.subplots(figsize=(10, 8))
scatter = ax.scatter([x[0] for x in tsne_vec],[x[1] for x in tsne_vec])

# Enumerating through the vector list and mapping the topics to plot thr graph
for i, pos in enumerate(tsne_vec):
    ax.annotate(newTopics[i], (pos[0], pos[1]))

# Adding the titles grid
ax.set_title('t-SNE visualization for selected features')
ax.set_xlabel('Dimension')
ax.set_ylabel('Dimension')
plt.grid()
plt.show()
```



Question: Comment about the categorizion done by T-SNE. Do the articles of related topics cluster together? (5 points)

Answer(1-3 sentences):

Question 3 Building Neural Networks

We are gonna use Emotions Dataset for this task. We need to classify the given text into different kind of emotions like happy,sad,anger etc..,

We are providing train.txt and val.txt files along with this notebook.

Library Imports and Utility functions

In [ ]:
```
# Importing the required libraries

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
import string
import pandas as pd
import re
```

```
#string.punctuation
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('words')

stopword = nltk.corpus.stopwords.words('english')
wn = nltk.WordNetLemmatizer()
ps = nltk.PorterStemmer()
words = set(nltk.corpus.words.words())

# Clean text function from last assignment

def clean_text(text):
  # From the last assignment
    text = text.lower()
    text = re.sub(r"http\S+", "", text)
    text = re.sub(r"www.\S+", "", text)
    text_links_removed = "".join([char for char in text if char not in string.p
    text_cleaned = " ".join([word for word in re.split('\W+', text_links_remove
        if word not in stopword])
    text = " ".join([wn.lemmatize(word) for word in re.split('\W+', text_clean
    return text
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/paarthvisharma/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/paarthvisharma/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data]     /Users/paarthvisharma/nltk_data...
[nltk_data]   Package words is already up-to-date!
```

Q) Importing the datasets and do the necessary cleaning and convert the text into the vectors which are mentioned in the below code blocks. (10 points)

In [ ]:

```
# Import the train.txt and val.txt file into pandas dataframe format
import pandas as pd

# Importing the training and validation dataset using the below function:
def import_data(filename):
    lines = []
    labels = []
    with open(filename, "r", encoding='utf-8') as file:
        # For each line, strippng the space and spliting it by semicolon to ge
        for i in file:
            # print(i)
            line, label = i.strip().split(';')
            lines.append(line)
            labels.append(label)
    # Crating and return the dataframe with sentences and labels as columns
    return pd.DataFrame({'Sentence': lines, 'Label': labels})

# Importing training dataset
df_train = import_data("train-1.txt")
```

```
# Importing validation dataset
df_val = import_data("val-1.txt")

# and printout the train.shape and validation.shape
print("Train shape" , df_train.shape)
print("Validation shape" , df_val.shape)

# expected shape of train dataset is (16000,2) and validation dataset is (2000,
```

```
Train shape (16000, 2)
Validation shape (2000, 2)
```

In [ ]:
```
# clean the text in the train and validation dataframes using the clean_text fu
df_train['Sentence'] = df_train['Sentence'].apply(clean_text)
df_val['Sentence'] = df_val['Sentence'].apply(clean_text)
print(df_train.head())
```

```
                                            Sentence    Label
0                             didnt feel humiliated  sadness
1  go feeling hopeless damned hopeful around some...  sadness
2          im grabbing minute post feel greedy wrong    anger
3  ever feeling nostalgic fireplace know still pr...     love
4                                    feeling grouchy    anger
```

In [ ]:
```
# Creating the data and target variables
X_train = df_train['Sentence']
y_train = df_train['Label']

X_val = df_val['Sentence']
y_val = df_val['Label']
```

In [ ]:
```
# initialise count vectorizer from sklearn module with default parameter

# fit on train dataset and transform both train and validation dataset
tf = CountVectorizer()
X_train_tf = tf.fit_transform(X_train)
X_val_tf = tf.transform(X_val)
```

In [ ]:
```
# initialise tfidf vectorizer from sklearn module with default parameter

# fit on train dataset and transform both train and validation dataset
tfidf = TfidfVectorizer()
X_train_tfidf = tfidf.fit_transform(X_train)
X_val_tfidf = tfidf.transform(X_val)
```

In [ ]:
```
# initialise label encoder from sklearn module

# fit on train labels and transform both train and validation labels
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()

y_train_encoded = labelEncoder.fit_transform(y_train)
y_val_encoded = labelEncoder.transform(y_val)
```

In [ ]:
```
# convert the labels into one hot encoding form
from sklearn.preprocessing import OneHotEncoder
import numpy as np

# Initializing the OneHotEncoder instance
```

```
onehot_encoder = OneHotEncoder()

# Creating the input array as 2D for OneHotEncoder
y_train_reshaped = y_train_encoded.reshape(-1, 1)
y_val_reshaped = y_val_encoded.reshape(-1, 1)

# Fitting the encoder on the training labels and also trasforming them
y_train_onehot = onehot_encoder.fit_transform(y_train_reshaped).toarray()

# Transforming the validation labels
y_val_onehot = onehot_encoder.transform(y_val_reshaped).toarray()
```

Q) Build the neural networks using tensorflow keras by following the below instructions. Evaluate the model on different metrics and comment your observations. (20 points)

In [ ]:
```
# Checking for all the unique categories in the data to count the number of cla
y_train_unique = set(y_train)
print(y_train_unique)
```

```
{'anger', 'sadness', 'joy', 'surprise', 'love', 'fear'}
```

In [ ]:
```
# Checking for all the unique categories in the data to count the number of cla
y_val_unique = set(y_val)
print(y_val_unique)
```

```
{'anger', 'surprise', 'joy', 'sadness', 'love', 'fear'}
```

In [ ]:
```
import tensorflow as tf
from tensorflow.keras import layers, models, regularizers

tf.random.set_seed(42)


# complete this linear model in tensorflow
def build_model(X):
  classes = 6

  # layer 1 : input layer
  inp = tf.keras.Input((X.shape[1],))

  # layer 2 : add the dense layer with 2048 units and relu activation
  x = tf.keras.layers.Dense(2048, activation = 'relu')(inp)
  # layer 3 : add the dropout layer with dropout rate of 0.5
  x = tf.keras.layers.Dropout(0.5)(x)
  # layer 4 : add the dense layer with 1024 units with tanh activation and wit
  x = tf.keras.layers.Dense(1024, activation = 'tanh', kernel_regularizer=regu
  # layer 5 : add the dropout layer with dropout rate of 0.5
  x = tf.keras.layers.Dropout(0.5)(x)
  # layer 6 : add the dense layer with 512 units with tanh activation and with
  x = tf.keras.layers.Dense(512, activation = 'tanh', kernel_regularizer=regula
  # layer 7 : add the dropout layer with dropout rate of 0.5
  x = tf.keras.layers.Dropout(0.5)(x)
  # layer 8 : add the dense layer with 256 units with tanh activation and with
  x = tf.keras.layers.Dense(256, activation = 'tanh', kernel_regularizer=regula
  # layer 9 : add the dropout layer with dropout rate of 0.5
  x = tf.keras.layers.Dropout(0.5)(x)
  # layer 10 : add the dense layer with 128 units with tanh activation and wit
  x = tf.keras.layers.Dense(128, activation = 'tanh', kernel_regularizer=regula
```

```
    # layer 11 : add the dropout layer with dropout rate of 0.5
    x = tf.keras.layers.Dropout(0.5)(x)
    # layer 12 : output layer with units equal to number of classes and activati
    output = tf.keras.layers.Dense(classes, activation='softmax')(x)
    # use loss as categorical crossentropy, optimizer as rmsprop and evaluate mod
    model = models.Model(inputs=inp, outputs=output)
    model.compile(loss='categorical_crossentropy',
                  optimizer='rmsprop',
                  metrics=['AUC', 'Precision', 'Recall', 'Accuracy'])

    return model
```

In [ ]:

```
# call the build_model function and initialize the model
# X_train_dense = X_train_tf.toarray()
# X_val_dense = X_val_tf.toarray()

model = build_model(X_train_tf)
```

In [ ]:

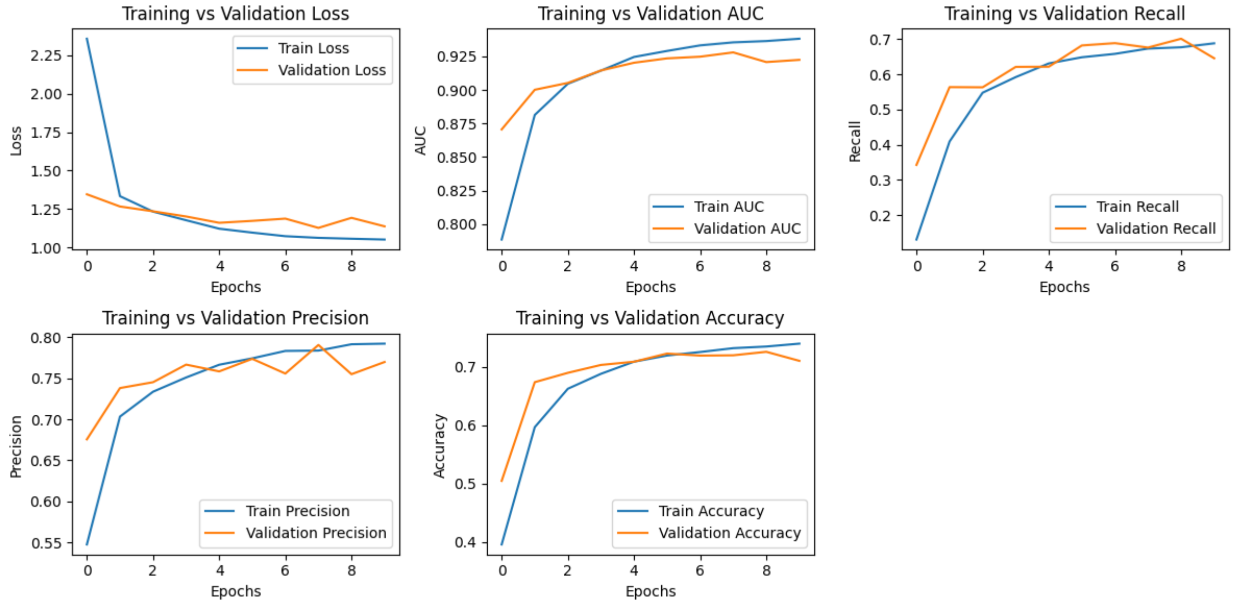
```
# train and validate the model on the count vectors of text which we have crea
# adjust batch size according to your computation power (suggestion use : 8)

# Train the model
history = model.fit(X_train_tf.toarray(), y_train_onehot,
                    epochs=10,
                    batch_size=8,
                    validation_data=(X_val_tf.toarray(), y_val_onehot))
```

```
Epoch 1/10
2000/2000 [==============================] - 188s 94ms/step - loss: 2.3577 - a
uc: 0.7884 - precision: 0.5475 - recall: 0.1304 - Accuracy: 0.3961 - val_loss:
1.3459 - val_auc: 0.8705 - val_precision: 0.6755 - val_recall: 0.3425 - val_Ac
curacy: 0.5050
Epoch 2/10
2000/2000 [==============================] - 189s 94ms/step - loss: 1.3336 - a
uc: 0.8813 - precision: 0.7033 - recall: 0.4090 - Accuracy: 0.5969 - val_loss:
1.2664 - val_auc: 0.9001 - val_precision: 0.7380 - val_recall: 0.5635 - val_Ac
curacy: 0.6740
Epoch 3/10
2000/2000 [==============================] - 190s 95ms/step - loss: 1.2332 - a
uc: 0.9046 - precision: 0.7336 - recall: 0.5477 - Accuracy: 0.6627 - val_loss:
1.2351 - val_auc: 0.9052 - val_precision: 0.7452 - val_recall: 0.5630 - val_Ac
curacy: 0.6900
Epoch 4/10
2000/2000 [==============================] - 229s 114ms/step - loss: 1.1772 -
auc: 0.9144 - precision: 0.7510 - recall: 0.5920 - Accuracy: 0.6883 - val_los
s: 1.2015 - val_auc: 0.9145 - val_precision: 0.7667 - val_recall: 0.6210 - val
_Accuracy: 0.7035
Epoch 5/10
2000/2000 [==============================] - 292s 146ms/step - loss: 1.1220 -
auc: 0.9246 - precision: 0.7665 - recall: 0.6309 - Accuracy: 0.7091 - val_los
s: 1.1605 - val_auc: 0.9203 - val_precision: 0.7584 - val_recall: 0.6215 - val
_Accuracy: 0.7090
Epoch 6/10
2000/2000 [==============================] - 211s 105ms/step - loss: 1.0965 -
auc: 0.9291 - precision: 0.7743 - recall: 0.6484 - Accuracy: 0.7196 - val_los
s: 1.1730 - val_auc: 0.9235 - val_precision: 0.7737 - val_recall: 0.6820 - val
_Accuracy: 0.7230
Epoch 7/10
2000/2000 [==============================] - 216s 108ms/step - loss: 1.0732 -
auc: 0.9332 - precision: 0.7833 - recall: 0.6581 - Accuracy: 0.7256 - val_los
s: 1.1877 - val_auc: 0.9248 - val_precision: 0.7558 - val_recall: 0.6885 - val
_Accuracy: 0.7195
Epoch 8/10
2000/2000 [==============================] - 305s 152ms/step - loss: 1.0625 -
auc: 0.9355 - precision: 0.7838 - recall: 0.6730 - Accuracy: 0.7322 - val_los
s: 1.1272 - val_auc: 0.9280 - val_precision: 0.7906 - val_recall: 0.6760 - val
_Accuracy: 0.7200
Epoch 9/10
2000/2000 [==============================] - 270s 135ms/step - loss: 1.0567 -
auc: 0.9365 - precision: 0.7915 - recall: 0.6768 - Accuracy: 0.7351 - val_los
s: 1.1928 - val_auc: 0.9208 - val_precision: 0.7550 - val_recall: 0.7010 - val
_Accuracy: 0.7260
Epoch 10/10
2000/2000 [==============================] - 200s 100ms/step - loss: 1.0515 -
auc: 0.9382 - precision: 0.7922 - recall: 0.6880 - Accuracy: 0.7400 - val_los
s: 1.1373 - val_auc: 0.9224 - val_precision: 0.7698 - val_recall: 0.6455 - val
_Accuracy: 0.7105
```

In [ ]:

```python
# plot train loss vs val loss, train auc vs val auc, train recall vs val recal

# Importing necessary libararies
import matplotlib.pyplot as plt

# Plotting training and validation loss as required
plt.figure(figsize=(12, 6))

# Loss Plot
```

```
plt.subplot(2, 3, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training vs Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# AUC Plot
plt.subplot(2, 3, 2)
plt.plot(history.history['auc'], label='Train AUC')
plt.plot(history.history['val_auc'], label='Validation AUC')
plt.title('Training vs Validation AUC')
plt.xlabel('Epochs')
plt.ylabel('AUC')
plt.legend()

# Recall Plot
plt.subplot(2, 3, 3)
plt.plot(history.history['recall'], label='Train Recall')
plt.plot(history.history['val_recall'], label='Validation Recall')
plt.title('Training vs Validation Recall')
plt.xlabel('Epochs')
plt.ylabel('Recall')
plt.legend()

# Precision Plot
plt.subplot(2, 3, 4)
plt.plot(history.history['precision'], label='Train Precision')
plt.plot(history.history['val_precision'], label='Validation Precision')
plt.title('Training vs Validation Precision')
plt.xlabel('Epochs')
plt.ylabel('Precision')
plt.legend()

# Accuracy Plot
plt.subplot(2, 3, 5)
plt.plot(history.history['Accuracy'], label='Train Accuracy')
plt.plot(history.history['val_Accuracy'], label='Validation Accuracy')
plt.title('Training vs Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```



In [ ]:
```
# again call the build_model function and initialize the model

model_tfidf = build_model(X_train_tfidf)
```

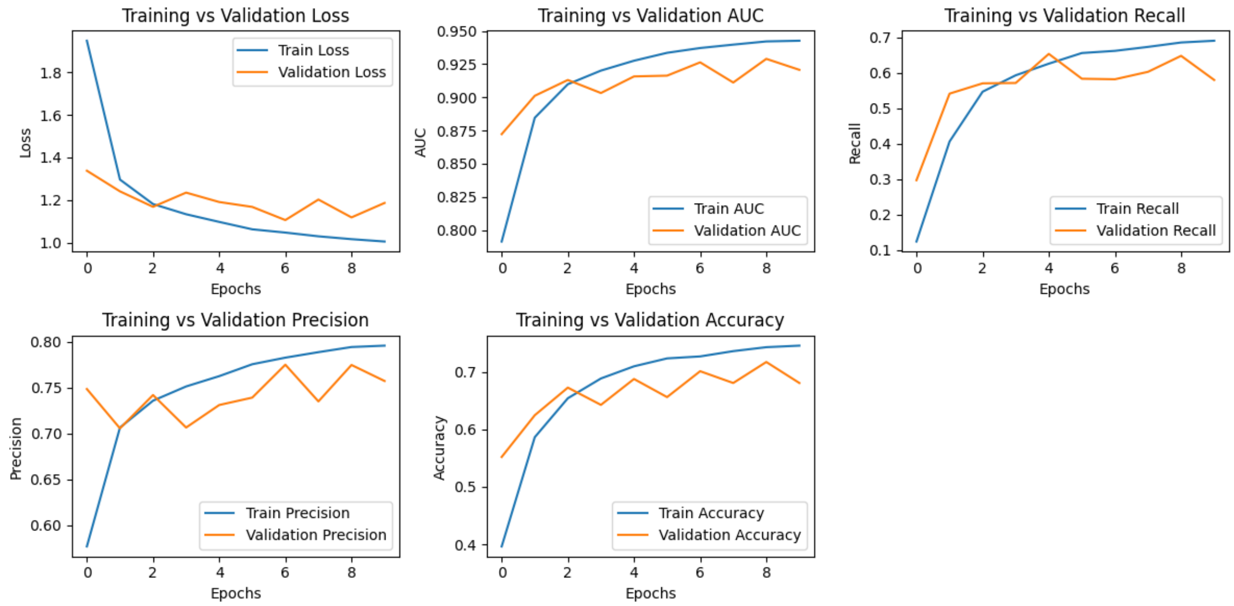
In [ ]:
```
# train and validate the model on the tfidf vectors of text which we have crea
# adjust batch size according to your computation power (suggestion use : 8)


def convert_to_sparse_tensor(sparse_matrix):
    # Converting the matrix to COO format where a matrix is represented by thre
    sparse_matrix = sparse_matrix.tocoo()
    # Create a 2D numpy matrix
    indices = np.mat([sparse_matrix.row, sparse_matrix.col]).transpose()
    # Creating and returning a TensorFlow SparseTensor using the indices, data,
    return tf.SparseTensor(indices, sparse_matrix.data, sparse_matrix.shape)

# Converting to TensorFlow SparseTensors by using the function created above.
X_train_tfidf_sparse = convert_to_sparse_tensor(X_train_tfidf)
X_val_tfidf_sparse = convert_to_sparse_tensor(X_val_tfidf)

# Reordering the indices of the SparseTensors
X_train_tfidf_reordered = tf.sparse.reorder(X_train_tfidf_sparse)
X_val_tfidf_reordered = tf.sparse.reorder(X_val_tfidf_sparse)

# Training the model
history = model_tfidf.fit(
    X_train_tfidf_reordered, y_train_onehot,
    epochs=10,
    batch_size=8,
    validation_data=(X_val_tfidf_reordered, y_val_onehot)
)
```

```
Epoch 1/10
2000/2000 [==============================] - 90s 45ms/step - loss: 1.9476 - au
c: 0.7913 - precision: 0.5768 - recall: 0.1247 - Accuracy: 0.3966 - val_loss:
1.3379 - val_auc: 0.8723 - val_precision: 0.7484 - val_recall: 0.2975 - val_Ac
curacy: 0.5520
Epoch 2/10
2000/2000 [==============================] - 90s 45ms/step - loss: 1.2964 - au
c: 0.8846 - precision: 0.7064 - recall: 0.4066 - Accuracy: 0.5863 - val_loss:
1.2411 - val_auc: 0.9012 - val_precision: 0.7055 - val_recall: 0.5415 - val_Ac
curacy: 0.6245
Epoch 3/10
2000/2000 [==============================] - 89s 44ms/step - loss: 1.1816 - au
c: 0.9099 - precision: 0.7357 - recall: 0.5470 - Accuracy: 0.6540 - val_loss:
1.1686 - val_auc: 0.9131 - val_precision: 0.7419 - val_recall: 0.5705 - val_Ac
curacy: 0.6725
Epoch 4/10
2000/2000 [==============================] - 91s 45ms/step - loss: 1.1337 - au
c: 0.9202 - precision: 0.7512 - recall: 0.5932 - Accuracy: 0.6884 - val_loss:
1.2353 - val_auc: 0.9033 - val_precision: 0.7064 - val_recall: 0.5715 - val_Ac
curacy: 0.6425
Epoch 5/10
2000/2000 [==============================] - 89s 44ms/step - loss: 1.0978 - au
c: 0.9276 - precision: 0.7625 - recall: 0.6259 - Accuracy: 0.7095 - val_loss:
1.1911 - val_auc: 0.9158 - val_precision: 0.7310 - val_recall: 0.6535 - val_Ac
curacy: 0.6875
Epoch 6/10
2000/2000 [==============================] - 88s 44ms/step - loss: 1.0631 - au
c: 0.9336 - precision: 0.7755 - recall: 0.6559 - Accuracy: 0.7232 - val_loss:
1.1681 - val_auc: 0.9163 - val_precision: 0.7391 - val_recall: 0.5835 - val_Ac
curacy: 0.6560
Epoch 7/10
2000/2000 [==============================] - 85s 42ms/step - loss: 1.0477 - au
c: 0.9372 - precision: 0.7826 - recall: 0.6621 - Accuracy: 0.7267 - val_loss:
1.1064 - val_auc: 0.9264 - val_precision: 0.7750 - val_recall: 0.5820 - val_Ac
curacy: 0.7010
Epoch 8/10
2000/2000 [==============================] - 82s 41ms/step - loss: 1.0302 - au
c: 0.9397 - precision: 0.7886 - recall: 0.6733 - Accuracy: 0.7359 - val_loss:
1.2027 - val_auc: 0.9112 - val_precision: 0.7349 - val_recall: 0.6030 - val_Ac
curacy: 0.6805
Epoch 9/10
2000/2000 [==============================] - 83s 41ms/step - loss: 1.0169 - au
c: 0.9421 - precision: 0.7942 - recall: 0.6856 - Accuracy: 0.7427 - val_loss:
1.1190 - val_auc: 0.9290 - val_precision: 0.7747 - val_recall: 0.6480 - val_Ac
curacy: 0.7170
Epoch 10/10
2000/2000 [==============================] - 84s 42ms/step - loss: 1.0058 - au
c: 0.9427 - precision: 0.7958 - recall: 0.6906 - Accuracy: 0.7454 - val_loss:
1.1868 - val_auc: 0.9207 - val_precision: 0.7572 - val_recall: 0.5800 - val_Ac
curacy: 0.6805
```

In [ ]:

```python
# plot train loss vs val loss, train auc vs val auc, train recall vs val recal

# Importing necessary libararies
import matplotlib.pyplot as plt

# Plotting training and validation loss
plt.figure(figsize=(12, 6))

# Loss Plot
```

```
plt.subplot(2, 3, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training vs Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# AUC Plot
plt.subplot(2, 3, 2)
plt.plot(history.history['auc'], label='Train AUC')
plt.plot(history.history['val_auc'], label='Validation AUC')
plt.title('Training vs Validation AUC')
plt.xlabel('Epochs')
plt.ylabel('AUC')
plt.legend()

# Recall Plot
plt.subplot(2, 3, 3)
plt.plot(history.history['recall'], label='Train Recall')
plt.plot(history.history['val_recall'], label='Validation Recall')
plt.title('Training vs Validation Recall')
plt.xlabel('Epochs')
plt.ylabel('Recall')
plt.legend()

# Precision Plot
plt.subplot(2, 3, 4)
plt.plot(history.history['precision'], label='Train Precision')
plt.plot(history.history['val_precision'], label='Validation Precision')
plt.title('Training vs Validation Precision')
plt.xlabel('Epochs')
plt.ylabel('Precision')
plt.legend()

# Accuracy Plot
plt.subplot(2, 3, 5)
plt.plot(history.history['Accuracy'], label='Train Accuracy')
plt.plot(history.history['val_Accuracy'], label='Validation Accuracy')
plt.title('Training vs Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```



Question 4 Theory Question

What is the difference between Count Vectorizer, TFIDF, Word2Vec and Glove? (5 points)

Answer

Count Vectorizer: Count Vectorizer converts documents into vectors where each vector represents the count of times a particular word appears in the document. It is a category of bag of word model and disregards the order of the words and just focuses on the frequency. It is good for simpler tasks where only knowing the frequency of words is important.

Term Frequency-Inverse Document Frequency (TF-IDF): TFIDF changes the term frequency by scaling it with the inverse document frequency. This reduces the weight of terms that appear frequently in the document and vice versa. This leads to a more balanced representation where the importance of rare but significant terms are still highlighted.

Word2Vec: Word2vec is a technique in natural language processing for creating vector representations of words. This uses neural network to learn word association from a large corpus of text. It represents words in a continuous vector space, capturing semantic relationships between words. This is good for understanding word menaings and relationships.

Glove: Glove is an unsupervised machine learning algorithm for obtaining vector representations for words. It has the capability of combining the advantages of the matrix factorization techniques used in approaches like TF-IDF with the various contextual learning capabilities of models like Word2Vec. It aims to keep related words close in vector space and is helpful in cases where we require to understand the relationship between words on a global space.

What is the significant difference between the Naive Bayes Implementation using Bag of Words and TF-IDF? (5 points)

Answer

Both Bag of words and TF-IDF basically serves the purpose of converting text to a numerical format suitable for ML models. The main difference lies in how they treat the significance of words. TF-IDF's approach to weighting words by their relative importance across the corpus provides a more meaningful and contextually relevant feature set. This added complexity leads to better results in case of classification tasks such as Naive Bayes.