This file contains a concise description accompanied by the code detailing the decision-making process.

Modelling Linguistic Variables and Fuzzy Rules for Social Media Sentiment Analysis

Comparative Analysis among Fuzzy Rules, Machine Learning models, Ensemble Models and Deep Learning Models

Sentiment analysis is a language processing task to detect the true emotions in social media statements. The main aim of this study is to explore the use of fuzzy logic to improve sentiment analysis in social media status. Mainly tweet data was used for this social media sentiment analysis by labelling them into 3 main categories: 'positive', 'negative', and 'neu- tral' based on the emotion of the tweet. The focus was on a fuzzy logic-based model, which used tools like Textblob and Vader to get sentiment scores. In the early stages, this model was trained with a balanced set of tweets, where there were equal numbers of each sentiment. During this phase, accuracy was high. But when more tweets were added, which weren't balanced, the accuracy dropped to 59.15%. This might have happened as the model was trained on a balanced dataset and therefore had difficulty working with the unbalanced data. The study also focused on a comparative analysis where six other models, namely Naive Bayes, Random Forest, Stacked RF NB, Stacked RF, GBM SVM, LSTM, and BLSTM, were eval- uated alongside the fuzzy logic model. The evaluation stage spanned multiple performance metrics, including precision, recall, F1 score, confusion matrix, ROC-AUC, and rigorous statistical analyses accentuated with confidence intervals, error sticks, and bell curves. The Naive Bayes (NB) model had difficulties identifying positive sentiments. Random Forest (RF) showed promise in decoding sentiment patterns but occasionally made errors in predictions. The stacked models displayed a harmonious balance between precision and recall. The fuzzy logic model was found to be particularly good at recognising negative sentiments. On the other hand, deep learning models like LSTM and BLSTM underperformed. The reason for the poor performance of deep learning models might be the insufficient data to train those models. Designing the fuzzy technique process, however, faced some challenges like fuzzification and rule evaluation steps. Despite these obstacles, this study tried to extract the true emotion in a tweet through a quantitative approach.

Methodology:

Sentiment analysis is indeed a detailed and intricate pro- cess. This project revealed how essential fuzzy systems are in performing sentiment analysis on social media content. Specifically, the fuzzy information is helpful in interpreting emotions expressed on social media that are often unclear or ambiguous. In this section, a detailed version of the research method has been discussed. The fuzzy system has been chosen as the main tool for this project. Fuzzy logic can work with language-based variables, which is why it can establish fuzzy rules based on specific conditions. In addition to the fuzzy logic, various other models have been applied to understand the comprehensive workflow. This included two traditional machine models, two ensemble models, and two deep learning models. These models were chosen to understand how other models perform in sentiment analysis. Detailed explanations of all models will be provided in the subsequent sections. Moreover, a roadmap outlining the sentiment analysis work- flow will be introduced later in this section. This roadmap illustrates the sequence that has been followed for sentiment analysis. As the section progresses, the specifics of each model have been elaborated in a thorough manner.

Models:

Fuzzy: The fuzzy analysis was conducted using TextBlob and Vader.

ML: Random Forest & Naive Bayes

Ensemble: Stacked RF & NB

DL: LSTM & BLSTM

Key Findings of Sentiment Analysis Evaluation

The main insights came from looking at how different models performed. These results gave the key findings.

Overall Accuracy

• The Fuzzy Logic was found to be the most accurate, with a score of 59.15%, capturing its ability to effectively capture sentiments using logic-based categorizations. • Among the ML models, RF (56.81%) performed better than NB (51.17%), suggesting that ensemble methods were more adept at capturing complex sentiment patterns. • The Ensemble Methods, designed to combine strengths from various models, achieved results around the mid-50s. This performance underscores both their potential and the complexities of integrating different algorithms. • Deep Learning models, LSTM and BLSTM, shown accuracies below 50%.

Performance Metrics Report

• Negative sentiments were effectively identified by the Fuzzy Logic suggesting its value in contexts where recognizing negativity is important. • The NB model had trouble spotting positive feelings. Meanwhile, the RF model didn't always predict correctly. • A balance between precision and recall was achieved by Ensemble Techniques. • While the LSTM model was found to be good at detecting neutral sentiments, both LSTM and BLSTM models indicated areas that might need further improvement.

Confusion Matrix Analysis

• The Fuzzy logic is noted for its accuracy in identifying negative sentiments. • Machine Learning models present their own challenges, with NB struggling between neutral and negative sentiments and RF sometimes confusing the margins between positive and neutral sentiments. • Ensemble Techniques demonstrate a effectively merging strengths from various models, yet they sometimes mix-up sentiments. • Deep Learning models lean towards specific sentiment groups. This might mean they're overfitting or they need different training data.

ROC-AUC Values

• The Fuzzy logic consistently demonstrates reliable performance across sentiments, with a commendable AUC of 0.77 for positive sentiments. • Machine Learning & Ensemble Models display consistent AUC values, but slight fluctuations among them highlight the challenges of selective overlapping sentiments. • The BLSTM model, even with its two-way design, has low AUC values. This shows it has trouble telling sentiments apart.

Statistical Analysis

• The combination of Confidence Intervals, Error Bars, and the Bell Curve gives a thorough review of how models perform in sentiment analysis. • The central dot in error bars shows the average accuracy, giving an idea of how well a model can detect sentiments. At the same time, the bell curve shows how consistent the models are in their results. • The Fuzzy Logic model stands out with an average accuracy of 59.15%. Its sharp peak on the bell curve indicates consistent results, supported by its confidence interval between 52.56% and 65.74%. This consistency in both the curve and interval underlines the model's reliability. • On the other hand, the NB model has an average accuracy of 51.17% and a wider bell curve, suggesting it might not always give consistent results. Its larger confidence interval supports this observation. • Models like RF and Ensemble have good results, but their bell curves and intervals indicate they might not be as consistent as the Fuzzy Logic model. • LSTM and BLSTM models have wider bell curves, suggesting they can work in varied sentiment situations. But their lower average accuracies and large confidence intervals hint at challenges they face in detecting sentiments.

Optimal Model Analysis

During the evaluation, the Fuzzy logic performed as the best model. Its high accuracy shows its skill in detecting small details in sentiments, especially negative ones, which other models might miss. This strong performance of the Fuzzy logic is further backed up by its ROC-AUC scores, which suggest we can trust its predictions. While the Fuzzy logic clearly performed well in this study, it was seen that other models have their own benefits. However, in this research, the top performance of the Fuzzy logic was clearly shown, reminding us that the best solution can differ depending on the situation and data.

colab environment configuration

```
!nvidia-smi
```

```
Sat Sep  9 21:39:05 2023
+-----------------------------------------------------------------------------
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0
|-------------------------------+----------------------+----------------------
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
|                               |                      |               MIG M.
|===============================+======================+======================
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0
| N/A   67C    P8    11W /  70W |      0MiB / 15360MiB |      0%      Default
|                               |                      |                  N/A
+-------------------------------+----------------------+----------------------

+-----------------------------------------------------------------------------
| Processes:
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory
|        ID   ID                                                   Usage
|=============================================================================
|  No running processes found
+-----------------------------------------------------------------------------
```

```
!cat /proc/meminfo
!cat /proc/cpuinfo
```

```
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cm
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf m
bogomips        : 4399.99
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 79
model name      : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping        : 0
microcode       : 0xffffffff
cpu MHz         : 2199.998
cache size      : 56320 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 1
apicid          : 1
initial apicid  : 1
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cm
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf m
bogomips        : 4399.99
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

```
!pip install scikit-fuzzy
!pip install textblob
!python -m textblob.download_corpora
!pip install nltk
```

```
Collecting scikit-fuzzy
  Downloading scikit-fuzzy-0.4.2.tar.gz (993 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 994.0/994.0 kB 9.8 MB/s eta 0:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.6.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: scipy>=0.9.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: networkx>=1.9.0 in /usr/local/lib/python3.10/d
Building wheels for collected packages: scikit-fuzzy
  Building wheel for scikit-fuzzy (setup.py) ... done
  Created wheel for scikit-fuzzy: filename=scikit_fuzzy-0.4.2-py3-none-any.wh
  Stored in directory: /root/.cache/pip/wheels/4f/86/1b/dfd97134a2c8313e519bc
Successfully built scikit-fuzzy
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.4.2
Requirement already satisfied: textblob in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: nltk>=3.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/d
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-package
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package conll2000 to /root/nltk_data...
[nltk_data]   Unzipping corpora/conll2000.zip.
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Unzipping corpora/movie_reviews.zip.
Finished.
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/d
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-package
```

```
!pip install numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packag
```

```
import nltk
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
True
```

```
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns
import re
from textblob import TextBlob
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('punkt')
nltk.download('stopwords')

import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
combined_file_path = '/content/drive/MyDrive/combined_file.txt'

tweets = []
with open(combined_file_path, 'r', encoding='utf-8', errors='ignore') as file:
    for line in file:
        tweets.append(line.strip())

twt = pd.DataFrame(tweets, columns=['tweet_text'])
```

```
twt.shape
```

```
(1070, 1)
```

```
twt.head()
```

	tweet_text
0	sentiment,timestamp,tweet_text
1	positive,Tue Jun 16 18:18:12 PDT 2009, AHHH I ...
2	neutral,Mon Apr 06 23:11:14 PDT 2009," cool , ...
3	neutral,Tue Jun 23 13:40:11 PDT 2009," i know ...
4	negative,Mon Jun 01 10:26:07 PDT 2009,School e...

```
## url removing

def remove_urls(text):
    return re.sub(r'http\S+', '', text)

## Applying the remove_urls function to the 'tweet_text' column and creating a ne
twt['cleaned_text'] = twt['tweet_text'].apply(remove_urls)
```

```
twt.tail()
```

	tweet_text	cleaned_text
1065	positive,Mon Apr 06 23:28:39 PDT 2009," thanks...	positive,Mon Apr 06 23:28:39 PDT 2009," thanks...
1066	positive,Mon Apr 06 23:28:41 PDT 2009, i miss...	positive,Mon Apr 06 23:28:41 PDT 2009, i miss...
1067	negative,Mon Apr 06 23:28:43 PDT 2009, ohhh. I...	negative,Mon Apr 06 23:28:43 PDT 2009, ohhh. I...
1068	neutral,Mon Apr 06 23:28:45 PDT 2009,And	neutral,Mon Apr 06 23:28:45 PDT 2009,And

```
## dropping the previous col
twt.drop('tweet_text', axis=1, inplace=True)
```

```
twt.sample(5)
```

	cleaned_text
85	neutral,Fri May 29 15:54:31 PDT 2009," seriosu...
267	neutral,Wed Jun 17 01:05:46 PDT 2009,"saw a re...
530	positive,Fri May 29 23:34:38 PDT 2009,sunny mo...
251	negative,Thu Jun 25 04:10:25 PDT 2009,I can't ...
1036	neutral,Mon Apr 06 23:27:03 PDT 2009, you fel...

```
## looking the full text

pd.set_option('display.max_colwidth', -1)
print(twt['cleaned_text'].head(5))
```

```
0    sentiment,timestamp,tweet_text
1    positive,Tue Jun 16 18:18:12 PDT 2009, AHHH I HOPE YOUR OK!!!
2    neutral,Mon Apr 06 23:11:14 PDT 2009," cool , i have no tweet apps  for r
3    neutral,Tue Jun 23 13:40:11 PDT 2009," i know  just family drama. its lam
4    negative,Mon Jun 01 10:26:07 PDT 2009,School email won't open  and I have
Name: cleaned_text, dtype: object
<ipython-input-12-a1fa9bffe8bb>:3: FutureWarning: Passing a negative integer i
  pd.set_option('display.max_colwidth', -1)
```

```
twt.drop(0, inplace=True)
print(twt['cleaned_text'].head())
```

```
1    positive,Tue Jun 16 18:18:12 PDT 2009, AHHH I HOPE YOUR OK!!!
2    neutral,Mon Apr 06 23:11:14 PDT 2009," cool , i have no tweet apps  for r
3    neutral,Tue Jun 23 13:40:11 PDT 2009," i know  just family drama. its lam
4    negative,Mon Jun 01 10:26:07 PDT 2009,School email won't open  and I have
5    neutral,Sat Jun 20 12:56:51 PDT 2009,upper airways problem
Name: cleaned_text, dtype: object
```

```
df = twt.copy()

## Removing timestamps from the 'cleaned_text' column
df['cleaned_text'] = df['cleaned_text'].apply(lambda text: re.sub(r'\b\w{3}\s\w{3

print(df.head())
```

```
1  positive,, AHHH I HOPE YOUR OK!!!
2  neutral,," cool , i have no tweet apps  for my razr 2"
3  neutral,," i know  just family drama. its lame.hey next time u hang out wi
4  negative,,School email won't open  and I have geography stuff on there to
5  neutral,,upper airways problem
```

```
twt=df


## Split the combined text into sentiment and text columns
twt[['sentiment', 'text']] = twt['cleaned_text'].str.split(',', n=1, expand=True)

## Drop the unnecessary columns
twt.drop(['cleaned_text'], axis=1, inplace=True)

## Remove any leading spaces from the sentiment column
twt['sentiment'] = twt['sentiment'].str.strip()
twt.head()
```

	sentiment	text
1	positive	, AHHH I HOPE YOUR OK!!!
2	neutral	," cool , i have no tweet apps for my razr 2"
3	neutral	," i know just family drama. its lame.hey next time u hang out with kim n u guys like have a sleepover or whatever, ill call u"
4	negative	,School email won't open and I have geography stuff on there to revise! *Stupid School* :'(

```
!pip install emoji
```

```
Collecting emoji
  Downloading emoji-2.8.0-py2.py3-none-any.whl (358 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 358.9/358.9 kB 4.2 MB/s eta 0:00
Installing collected packages: emoji
Successfully installed emoji-2.8.0
```

lowercase conversion, tokenization, removal of punctuation and special characters, removal of stopwords, and stemming or lemmatization, emojis

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

## Initialise the lemmatizer
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()

    ## Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

    ## Remove mentions and hashtags
    text = re.sub(r'\@\w+|\#','', text)

    ## Remove emojis
    text = text.encode('ascii', 'ignore').decode('ascii')

    ## tokenization
    tokens = word_tokenize(text)

    ## remove punctuation
    filtered_tokens = [word for word in tokens if word.isalnum()]

    # #lemmatization
    filtered_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]

    ## remove stopwords
    filtered_tokens = [word for word in filtered_tokens if word not in stopwords.

    ## reassemble the preprocessed text
    preprocessed_text = ' '.join(filtered_tokens)

    return preprocessed_text

### applying the preprocessing function to the 'tweet_text' column
twt['preprocessed_text'] = twt['text'].apply(preprocess_text)

print(twt['preprocessed_text'].head())
```

```
1    ahhh hope ok
2    cool tweet apps razr 2
3    know family drama next time u hang kim n u guy like sleepover whatever i
4    school email wo open geography stuff revise stupid school
5    upper airway problem
Name: preprocessed_text, dtype: object
```

```
twt.head()
```

	sentiment	text	preprocessed_text
1	positive	, AHHH I HOPE YOUR OK!!!	ahhh hope ok
2	neutral	," cool , i have no tweet apps for my razr 2"	cool tweet apps razr 2
3	neutral	," i know just family drama. its lame.hey next time u hang out with kim n u guys like have a sleepover or whatever, ill call	know family drama next time u hang kim n u guy like sleepover whatever ill

```
twt.drop(columns=['text'], inplace=True)
twt.head()
```

	sentiment	preprocessed_text
1	positive	ahhh hope ok
2	neutral	cool tweet apps razr 2
3	neutral	know family drama next time u hang kim n u guy like sleepover whatever ill call u
4	negative	school email wo open geography stuff revise stupid school
5	neutral	upper airway problem

```
#
```

```
twt.shape
```

```
(1069, 2)
```

```
!pip install vaderSentiment
```

```
Collecting vaderSentiment
  Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl (125 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 126.0/126.0 kB 3.3 MB/s eta 0:00
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyt
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-3.3.2
```

```
sentiment_counts = twt['sentiment'].value_counts()
print(sentiment_counts)
```

```
neutral      397
negative     384
positive     262
sentiment     19
              5
posiitve      1
neative       1
Name: sentiment, dtype: int64
```

```
## replace variations of sentiment labels
twt['sentiment'] = twt['sentiment'].replace(['sentiment', 'posiitve'], 'positive'
twt['sentiment'] = twt['sentiment'].replace('neative', 'negative')

## dropping rows with invalid sentiment labels
valid_sentiments = ['positive', 'negative', 'neutral']
twt = twt[twt['sentiment'].isin(valid_sentiments)]

## total counts of sentiment category
sentiment_counts = twt['sentiment'].value_counts()
print(sentiment_counts)
```

```
neutral     397
negative    385
positive    282
Name: sentiment, dtype: int64
```

```
## creating bar plot to see the counts of each sentiment

colours = ['orange', 'darkred', 'teal']

sentiment_counts = twt['sentiment'].value_counts()

plt.figure(figsize=(6, 4))
sentiment_counts.plot(kind='bar', color=colours)
plt.title('Distribution of Sentiment Categories')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```

fuzzy technique

```
!pip install git+https://github.com/scikit-fuzzy/scikit-fuzzy.git
```

```
Collecting git+https://github.com/scikit-fuzzy/scikit-fuzzy.git
  Cloning https://github.com/scikit-fuzzy/scikit-fuzzy.git to /tmp/pip-req-bu
  Running command git clone --filter=blob:none --quiet https://github.com/sci
  Resolved https://github.com/scikit-fuzzy/scikit-fuzzy.git to commit 0545430
  Running command git submodule update --init --recursive -q
  Preparing metadata (setup.py) ... done
Requirement already satisfied: matplotlib>=3.1.0 in /usr/local/lib/python3.10
Requirement already satisfied: networkx>=1.9.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: numpy>=1.6.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: scipy>=0.9.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pac
```

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
from textblob import TextBlob
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer


## linguistic variable and terms

### for the input

# TextBlob sentiment score linguistic variable
sentiment_score_textblob = ctrl.Antecedent(np.linspace(-1, 1, 1000), 'sentiment_s

# VADER sentiment score linguistic variable
sentiment_score_vader = ctrl.Antecedent(np.linspace(-1, 1, 1000), 'sentiment_scor


## for the output

# Final sentiment linguistic variable (ranges from -1 to 1 where -1 is negative a
sentiment = ctrl.Consequent(np.linspace(-1, 1, 1000), 'sentiment')


## membership function
### for the input
sentiment_score_textblob.automf(3)  # Negative, Neutral, Positive
sentiment_score_vader.automf(3)    # Negative, Neutral, Positive
```

```
### for the output

sentiment['negative'] = fuzz.trimf(sentiment.universe, [-1, -0.5, 0])
sentiment['neutral'] = fuzz.trimf(sentiment.universe, [-0.5, 0, 0.5])
sentiment['positive'] = fuzz.trimf(sentiment.universe, [0, 0.5, 1])


## fuzzy rules

rule1 = ctrl.Rule(sentiment_score_textblob['poor'] | sentiment_score_vader['poor'
rule2 = ctrl.Rule(sentiment_score_textblob['average'] | sentiment_score_vader['av
rule3 = ctrl.Rule(sentiment_score_textblob['good'] & sentiment_score_vader['good'


## create control system

sentiment_control_system = ctrl.ControlSystem([rule1, rule2, rule3])


## control system simulation

sentiment_simulation = ctrl.ControlSystemSimulation(sentiment_control_system)


## function to predict

from textblob import TextBlob
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

def infer_sentiment_fuzzy(tweet):
    # Calculate sentiment scores using TextBlob and Vader
    tb = TextBlob(tweet)
    textblob_score = tb.sentiment.polarity

    analyzer = SentimentIntensityAnalyzer()
    vader_score = analyzer.polarity_scores(tweet)['compound']

    # Input scores into fuzzy control simulation
    sentiment_simulation.input['sentiment_score_textblob'] = textblob_score
    sentiment_simulation.input['sentiment_score_vader'] = vader_score

    # Compute the result
    sentiment_simulation.compute()

    # Get the result
    output = sentiment_simulation.output['sentiment']
    if output > 0.5:
        return "positive"
    elif output < -0.5:
        return "negative"
    else:
        return "neutral"
```

```
## predict sentiment

tweets = ["hey you are up early", "hey nick how are youu? x",
          "wow i never thought i would have 30 followers. thanks!",
          "Happy b-day Backstreets!!!!Hope you go for many many moree!!Thank you
          "do you hate us ?", "my car is dead........ what am i to do Poor bug"]

predicted_labels = []

for tweet in tweets:
    predicted_label = infer_sentiment_fuzzy(tweet)
    predicted_labels.append(predicted_label)

for tweet, label in zip(tweets, predicted_labels):
    print(f"Tweet: {tweet}, Predicted Label: {label}")
```

```
Tweet: hey you are up early, Predicted Label: neutral
Tweet: hey nick how are youu? x, Predicted Label: neutral
Tweet: wow i never thought i would have 30 followers. thanks!, Predicted Labe
Tweet: Happy b-day Backstreets!!!!Hope you go for many many moree!!Thank you
Tweet: do you hate us ?, Predicted Label: neutral
Tweet: my car is dead........ what am i to do Poor bug, Predicted Label: neut
```

fuzzy technique with 6 sample data

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
from textblob import TextBlob
from nltk.sentiment.vader import SentimentIntensityAnalyzer

## linguistic variables for sentiment scores from (TextBlob) and (VADER)
sentiment_score_textblob = ctrl.Antecedent(np.arange(-1, 1.01, 0.01), 'sentiment_
sentiment_score_vader = ctrl.Antecedent(np.arange(-1, 1.01, 0.01), 'sentiment_sco
```

Linguistic variables for sentiment scores have been defined, and these scores are sourced from TextBlob and VADER. The range of these sentiment scores, representing the spectrum from very negative (-1) to very positive (1), has been set by the array np.arange(-1, 1.01, 0.01).

```
## linguistic variable for predicted sentiment
sentiment = ctrl.Consequent(np.arange(0, 1.01, 0.01), 'sentiment')
```

A detailed array has then created to measure the full range of those sentiments, from very negative to very positive.

```
### membership functions for sentiment scores from (TextBlob) n (VADER)

sentiment_score_textblob['negative'] = fuzz.trimf(sentiment_score_textblob.univer
sentiment_score_textblob['neutral'] = fuzz.trimf(sentiment_score_textblob.univers
sentiment_score_textblob['positive'] = fuzz.trimf(sentiment_score_textblob.univer

sentiment_score_vader['negative'] = fuzz.trimf(sentiment_score_vader.universe, [-
sentiment_score_vader['neutral'] = fuzz.trimf(sentiment_score_vader.universe, [-0
sentiment_score_vader['positive'] = fuzz.trimf(sentiment_score_vader.universe, [0
```

Linguistic terms for each type of sentiment (variable) were defined using the triangular membership function method, and these definitions were based on sentiment scores provided by TextBlob and VADER.

```
## membership functions for predicted sentiment
sentiment['negative'] = fuzz.trimf(sentiment.universe, [0, 0, 0.5])
sentiment['neutral'] = fuzz.trimf(sentiment.universe, [0.2, 0.5, 0.8])
sentiment['positive'] = fuzz.trimf(sentiment.universe, [0.5, 1, 1])
```

```
## fuzzy logic rules
rule1 = ctrl.Rule(sentiment_score_textblob['negative'] | sentiment_score_vader['n
rule2 = ctrl.Rule(sentiment_score_textblob['neutral'] & sentiment_score_vader['ne
rule3 = ctrl.Rule(sentiment_score_textblob['positive'] | sentiment_score_vader['p
```

```
# control system

sentiment_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
sentiment_pred = ctrl.ControlSystemSimulation(sentiment_ctrl)
```

```
## defining a function to infer sentiment using TextBlob
def infer_sentiment_textblob(text):
    blob = TextBlob(text)
    sentiment = blob.sentiment.polarity
    return sentiment

## defining a function to infer sentiment using VADER
def infer_sentiment_vader(text):
    analyzer = SentimentIntensityAnalyzer()
    sentiment = analyzer.polarity_scores(text)['compound']
    return sentiment

## Defining a function to infer sentiment using fuzzy logic
def infer_sentiment_fuzzy(text):
    sentiment_score_tb = infer_sentiment_textblob(text)
    sentiment_score_vd = infer_sentiment_vader(text)

    sentiment_pred.input['sentiment_score_textblob'] = sentiment_score_tb
    sentiment_pred.input['sentiment_score_vader'] = sentiment_score_vd
    sentiment_pred.compute()
    sentiment_value = sentiment_pred.output['sentiment']

    if sentiment_value <= 0.3:
        label = "negative"
    elif sentiment_value >= 0.7:
        label = "positive"
    else:
        label = "neutral"

    return label

tweets = ["hey you are up early", "hey nick  how are youu? x", "wow i never thoug
          "happy b-day backstreets hope you go for many many moree!!thank you for
          "do you hate us ?", "my car is dead........ what am i to do  poor bug"]

predicted_labels = []

for tweet in tweets:
    predicted_label = infer_sentiment_fuzzy(tweet)
    predicted_labels.append(predicted_label)

for tweet, label in zip(tweets, predicted_labels):
    print(f"Tweet: {tweet}, Predicted Label: {label}")
```

```
Tweet: hey you are up early, Predicted Label: neutral
Tweet: hey nick  how are youu? x, Predicted Label: neutral
Tweet: wow i never thought i would have 30 followers. thanks, Predicted Label
Tweet: happy b-day backstreets hope you go for many many moree!!thank you for
Tweet: do you hate us ?, Predicted Label: negative
Tweet: my car is dead........ what am i to do  poor bug, Predicted Label: nega
```

```
twt.sample(20)
```

	sentiment	preprocessed_text
892	negative	say kal penn leaving house noooooo totally missed tonight
352	positive	kick butt chuck ha renewed third season suck 13 episode though
342	neutral	got ta work today
17	neutral	feeling quite sleepy today wish could stay bed today ok last year let go school
695	positive	baking amp packinggg cristinas staying tonight usualllll pittsburgh see giiiiirlllfriends
685	neutral	aww kiro got cut
175	negative	ugh bored today
39	neutral	time sleep long day tomorrow
92	neutral	hi seb spanish fan addicted sp think best band world please come spain early
749	negative	wa sad unexpected totally cried haha
211	neutral	much waffle amp enough drinking going le annoying version numpty amp make
170	negative	still cant get jonas bros ticket dreading facing niece later see tantrum hope put another gig dublin
746	neutral	going late one mqu today
883	negative	killed character one favorite show upset
100	negative	miss zack much day
21	negative	apparently dont time ur fan
312	negative	ca believe last survivor titanic died really sad
351	neutral	doe know today might get blonde put hair summer

```
twt.tail(20)
```

	sentiment	preprocessed_text
1050	negative	bedtime school tomorrow still book broke suck
1051	negative	seating helping baby paper well forcing seat im sleepy
1052	negative	synching contact old mobile iphone import doe work well
1053	negative	ca concentrate
1054	negative	spent 1 hour enter bureaucratic nonsense march waste time
1055	neutral	nw confused ever
1056	negative	feeling well stupid migraine making tummy upset whole body ache shoot
1057	neutral	reading buyology bedtime great premise turning quot ok quot book lot info already knew
1058	neutral	home really wana sleep due wasting free line town assignment finish

working with 40 data

```
## defining a function to infer sentiment using TextBlob
def infer_sentiment_textblob(text):
    blob = TextBlob(text)
    sentiment = blob.sentiment.polarity
    return sentiment

## defining a function to infer sentiment using VADER
def infer_sentiment_vader(text):
    analyzer = SentimentIntensityAnalyzer()
    sentiment = analyzer.polarity_scores(text)['compound']
    return sentiment

## Defining a function to infer sentiment using fuzzy logic
def infer_sentiment_fuzzy(text):
    sentiment_score_tb = infer_sentiment_textblob(text)
    sentiment_score_vd = infer_sentiment_vader(text)

    sentiment_pred.input['sentiment_score_textblob'] = sentiment_score_tb
    sentiment_pred.input['sentiment_score_vader'] = sentiment_score_vd
    sentiment_pred.compute()
    sentiment_value = sentiment_pred.output['sentiment']

    if sentiment_value <= 0.3:
        label = "negative"
    elif sentiment_value >= 0.7:
        label = "positive"
    else:
        label = "neutral"

    return label

tweets = ["ahhh hope ok", "bit tongue", "ugh morning rough start","oh really grea
"poorly bed","great mind think alike","sum day one word kackered","lunch dj come
          "zach make pee sitting grown gay man","thank glad like product review b
          "gahh noo peyton need live horrible","oh feeling like","going miss past
          "yeah mathieu totally choked 3rd set let rog win well djokovic played t
          "feeling quite sleepy today wish could stay bed today ok last year let
          "school email wo open geography stuff revise stupid school","cool tweet
          "know family drama next time u hang kim n u guy like sleepover whatever
          "apparently dont time ur fan","feel like shit way want spend birthday e
          "ugh bored today","got ta work today","aww kiro got cut","time sleep lo
          "wa sad unexpected totally cried haha","killed character one favorite s
          "much waffle amp enough drinking going le annoying version numpty amp m
          "killed character one favorite show upset","oh sad poor","happy camper"
          "ca concentrate","bedtime school tomorrow still book broke suck","tried
          "home really wana sleep due wasting free line town assignment finish"]

predicted_labels = []

for tweet in tweets:
    predicted_label = infer_sentiment_fuzzy(tweet)
    predicted_labels.append(predicted_label)

for tweet, label in zip(tweets, predicted_labels):
    print(f"Tweet: {tweet}, Predicted Label: {label}")
```

```
Tweet: ahhh hope ok, Predicted Label: positive
Tweet: bit tongue, Predicted Label: neutral
Tweet: ugh morning rough start, Predicted Label: negative
Tweet: oh really great small blizzard also cold wind blow, Predicted Label: p
Tweet: poorly bed, Predicted Label: negative
Tweet: great mind think alike, Predicted Label: positive
Tweet: sum day one word kackered, Predicted Label: neutral
Tweet: lunch dj come eat, Predicted Label: neutral
Tweet: upper airway problemzach make pee sitting grown gay man, Predicted Lab
Tweet: thank glad like product review bit site enjoy knitting, Predicted Labe
Tweet: gahh noo peyton need live horrible, Predicted Label: negative
Tweet: oh feeling like, Predicted Label: positive
Tweet: going miss pastor sermon faithyeah mathieu totally choked 3rd set let
Tweet: feeling quite sleepy today wish could stay bed today ok last year let
Tweet: lol calm got 30day loan offer 1500, Predicted Label: positive
Tweet: school email wo open geography stuff revise stupid school, Predicted L
Tweet: cool tweet apps razr 2, Predicted Label: positive
Tweet: know family drama next time u hang kim n u guy like sleepover whatever
Tweet: miss zack much day, Predicted Label: neutral
Tweet: apparently dont time ur fan, Predicted Label: neutral
Tweet: feel like shit way want spend birthday eve, Predicted Label: negative
Tweet: ca believe last survivor titanic died really sad, Predicted Label: neg
Tweet: ugh bored today, Predicted Label: negative
Tweet: got ta work today, Predicted Label: neutral
Tweet: aww kiro got cut, Predicted Label: neutral
Tweet: time sleep long day tomorrow, Predicted Label: neutral
Tweet: doe know today might get blonde put hair summer, Predicted Label: neut
Tweet: wa sad unexpected totally cried haha, Predicted Label: negative
Tweet: killed character one favorite show upset, Predicted Label: neutral
Tweet: much waffle amp enough drinking going le annoying version numpty amp m
Tweet: say kal penn leaving house noooooo totally missed tonight, Predicted L
Tweet: killed character one favorite show upset, Predicted Label: neutral
Tweet: oh sad poor, Predicted Label: negative
Tweet: happy camper, Predicted Label: positive
Tweet: somehow still end place, Predicted Label: neutral
Tweet: omg quot reader quot making, Predicted Label: neutral
Tweet: ca concentrate, Predicted Label: neutral
Tweet: bedtime school tomorrow still book broke suck, Predicted Label: negati
Tweet: tried download tweetdeck wont download, Predicted Label: neutral
Tweet: home really wana sleep due wasting free line town assignment finish, P
```

this time(40 data) can not predict 100% as it can predict 100% for 6 data

working with whole data

```
twt.head(2)
```

	sentiment	preprocessed_text
1	positive	ahhh hope ok
2	neutral	cool tweet apps razr 2

```
from sklearn.model_selection import train_test_split


train_data, test_data = train_test_split(twt, test_size=0.2, random_state=42)


from sklearn.model_selection import train_test_split

X = twt['preprocessed_text']
y = twt['sentiment']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

## fuzzy system to predict sentiments on the test set
y_pred = [infer_sentiment_fuzzy(text) for text in X_test]
y_true = y_test.tolist()


## applying fuzzy logic system to training and test data

train_predicted_labels = [infer_sentiment_fuzzy(tweet) for tweet in train_data['p
test_predicted_labels = [infer_sentiment_fuzzy(tweet) for tweet in test_data['pre


### numerical transformation
 ## define the function to convert the labels

def label_to_number(label):
    mapping = {
        'negative': -1,
        'neutral': 0,
        'positive': 1
    }
    return mapping[label]

train_numeric_labels = [label_to_number(label) for label in train_predicted_label
test_numeric_labels = [label_to_number(label) for label in test_predicted_labels]



### evaluate the fuzzy system on test data

actual_test_labels = test_data['sentiment'].tolist()
correct_predictions = sum([1 for actual, predicted in zip(actual_test_labels, tes
accuracy = correct_predictions / len(test_predicted_labels)
print(f"Accuracy on test data: {accuracy:.2%}")
```

```
Accuracy on test data: 59.15%
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Split the dataset
X = twt['preprocessed_text']
y = twt['sentiment']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

# Use the fuzzy system to predict sentiments on the test set
y_pred = [infer_sentiment_fuzzy(text) for text in X_test]
y_true = y_test.tolist()

### Calculate the classification report
report = classification_report(y_true, y_pred, target_names=['negative', 'neutral
print(report)
```

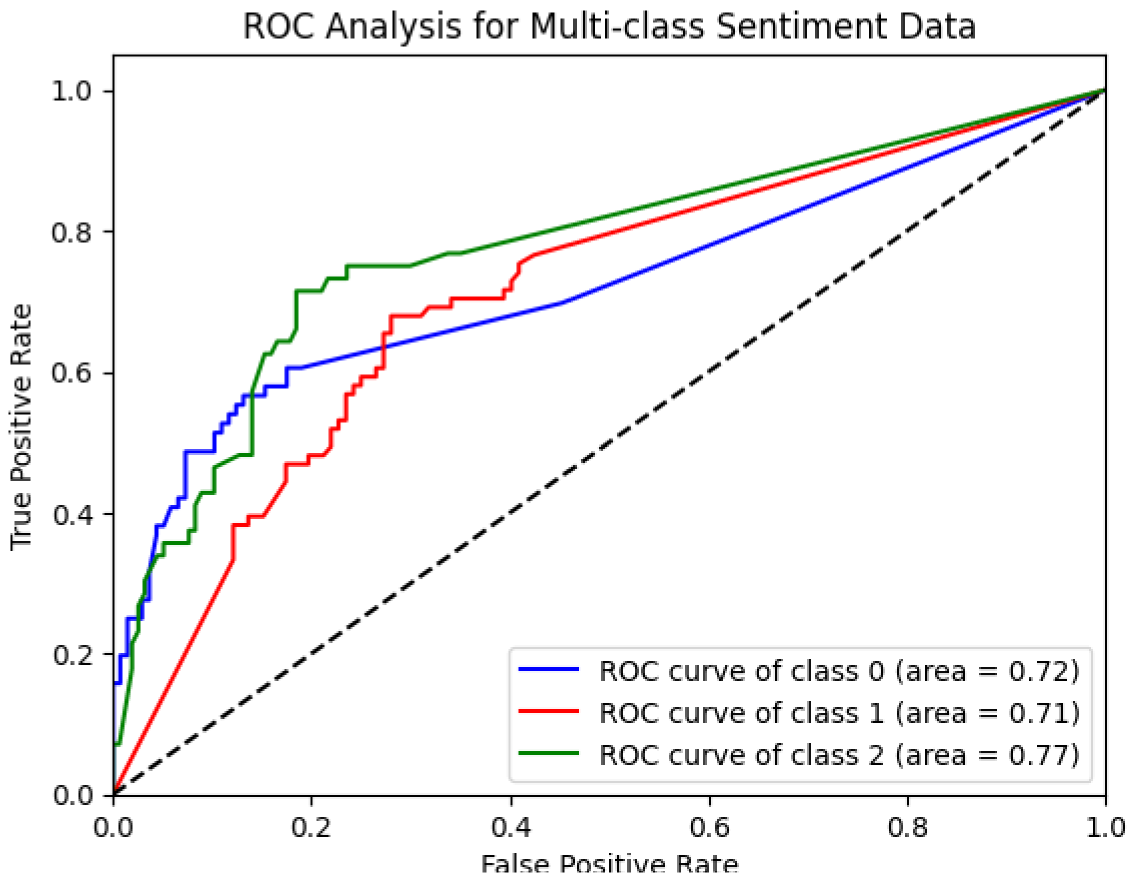
```
              precision    recall  f1-score   support

    negative       0.71      0.54      0.61        76
     neutral       0.58      0.53      0.55        81
    positive       0.52      0.75      0.61        56

    accuracy                           0.59       213
   macro avg       0.60      0.61      0.59       213
weighted avg       0.61      0.59      0.59       213
```

preparing for ROC curve

membership values for each sentiment class, derived from the fuzzy output, are utilized to plot the ROC curve, serving as a substitute for traditional probabilities.

membership value for each sentiment class (negative, neutral, positive) based on the fuzzy output

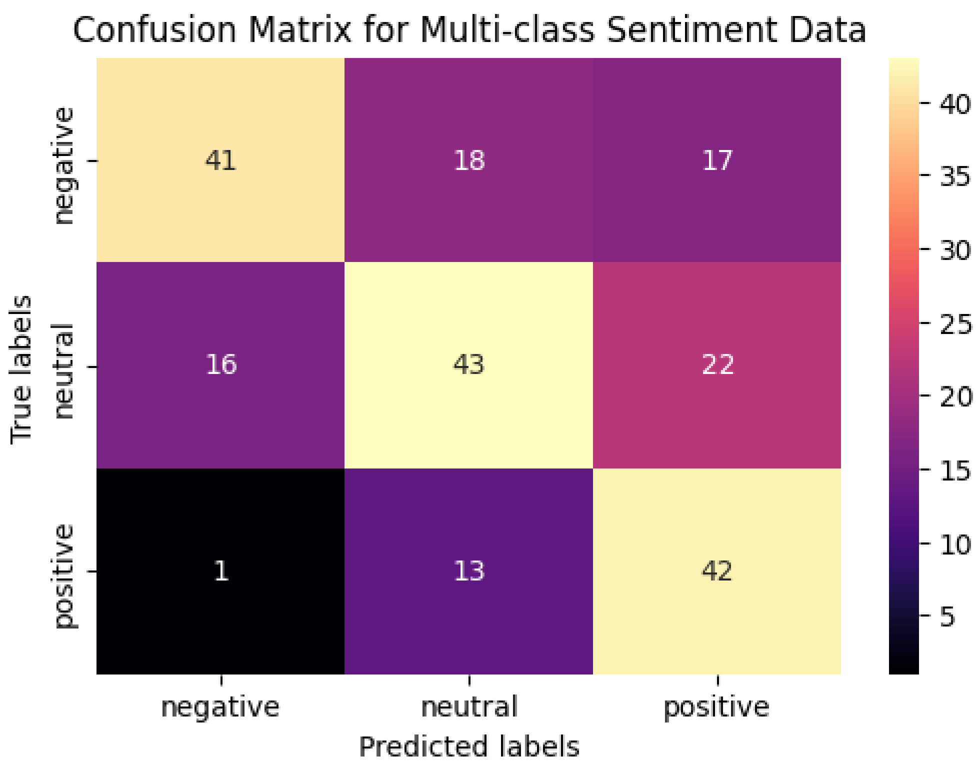
```
def infer_sentiment_fuzzy_probabilities(text):
    sentiment_score_textblob = infer_sentiment_textblob(text)
    sentiment_score_vader = infer_sentiment_vader(text)

    sentiment_pred.input['sentiment_score_textblob'] = sentiment_score_textblob
    sentiment_pred.input['sentiment_score_vader'] = sentiment_score_vader
    sentiment_pred.compute()

    # ## Get the membership values for each class
    memberships = {
        'negative': fuzz.interp_membership(sentiment.universe, sentiment['negativ
        'neutral': fuzz.interp_membership(sentiment.universe, sentiment['neutral'
        'positive': fuzz.interp_membership(sentiment.universe, sentiment['positiv
    }

    return memberships


from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

## binarise the labels
y_test_bin = label_binarize(y_test, classes=['negative', 'neutral', 'positive'])
n_classes = y_test_bin.shape[1]

## compute the predicted scores from the fuzzy system
y_score = np.array([list(infer_sentiment_fuzzy_probabilities(text).values()) for

### compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot the ROC curves
plt.figure()
colors = ['blue', 'red', 'green']
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color,
             label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc[i

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Analysis for Multi-class Sentiment Data')
plt.legend(loc="lower right")
plt.show()
```


ROC Analysis for Multi-class Sentiment Data
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.72)
ROC curve of class 1 (area = 0.71)
ROC curve of class 2 (area = 0.77)

```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

## binarise the labels
y_test_bin = label_binarize(y_test, classes=['negative', 'neutral', 'positive'])
n_classes = y_test_bin.shape[1]

## compute the predicted scores from the fuzzy system
y_score = np.array([list(infer_sentiment_fuzzy_probabilities(text).values()) for

### compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot the ROC curves
plt.figure()
colours = ['blue', 'red', 'teal']
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color,
             label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc[i
    plt.fill_between(fpr[i], tpr[i], step='post', alpha=0.2, color=colours)

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('AUC-ROC Analysis for Multi-class Sentiment Data using Fuzzy')
plt.legend(loc="lower right")
plt.show()
```

confusion matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sns


y_pred = y_score.argmax(axis=1)
y_true = y_test_bin.argmax(axis=1)

### Compute the confusion matrix
cm = confusion_matrix(y_true, y_pred)

### confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="magma",
            xticklabels=['negative', 'neutral', 'positive'],
            yticklabels=['negative', 'neutral', 'positive'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for Multi-class Sentiment Data')
plt.show()
```



Fuzzy

```
      negative  neutral  positive
```

negative 41 18 17

neutral 16 43 22

positive 1 13 42

```
 ## Fuzzy
          negative  neutral  positive
negative  41        18       17
neutral   16        43       22
positive  1         13       42
```

ML

Naive Bayes by TF-IDF

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report


### Split
X = twt['preprocessed_text']
y = twt['sentiment']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

## convert text data into TF-IDF features
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

## train the multinomial Naive Bayes classifier
naive_bayes_classifier = MultinomialNB()
naive_bayes_classifier.fit(X_train_tfidf, y_train)

## predictions
y_pred = naive_bayes_classifier.predict(X_test_tfidf)

## classification report
report = classification_report(y_test, y_pred, target_names=['negative', 'neutral
print(report)
```

```
              precision    recall  f1-score   support

    negative       0.51      0.63      0.56        76
     neutral       0.45      0.58      0.51        81
    positive       0.73      0.20      0.31        56

    accuracy                           0.50       213
   macro avg       0.57      0.47      0.46       213
weighted avg       0.55      0.50      0.48       213
```

```
from sklearn.metrics import accuracy_score

## accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Naive Bayes Classifier: {accuracy:.2%}")
```

```
Accuracy of Naive Bayes Classifier: 49.77%
```

Naive Bayes by Count Vectorization (Bag of Words)

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report

### Split
X = twt['preprocessed_text']
y = twt['sentiment']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

## Convert text data into count features (Bag of Words)
count_vectorizer = CountVectorizer(max_features=5000)
X_train_counts = count_vectorizer.fit_transform(X_train)
X_test_counts = count_vectorizer.transform(X_test)

## Train the multinomial Naive Bayes classifier
naive_bayes_classifier = MultinomialNB()
naive_bayes_classifier.fit(X_train_counts, y_train)

## predictions
y_pred = naive_bayes_classifier.predict(X_test_counts)

## classification report
report = classification_report(y_test, y_pred, target_names=['negative', 'neutral
print(report)
```

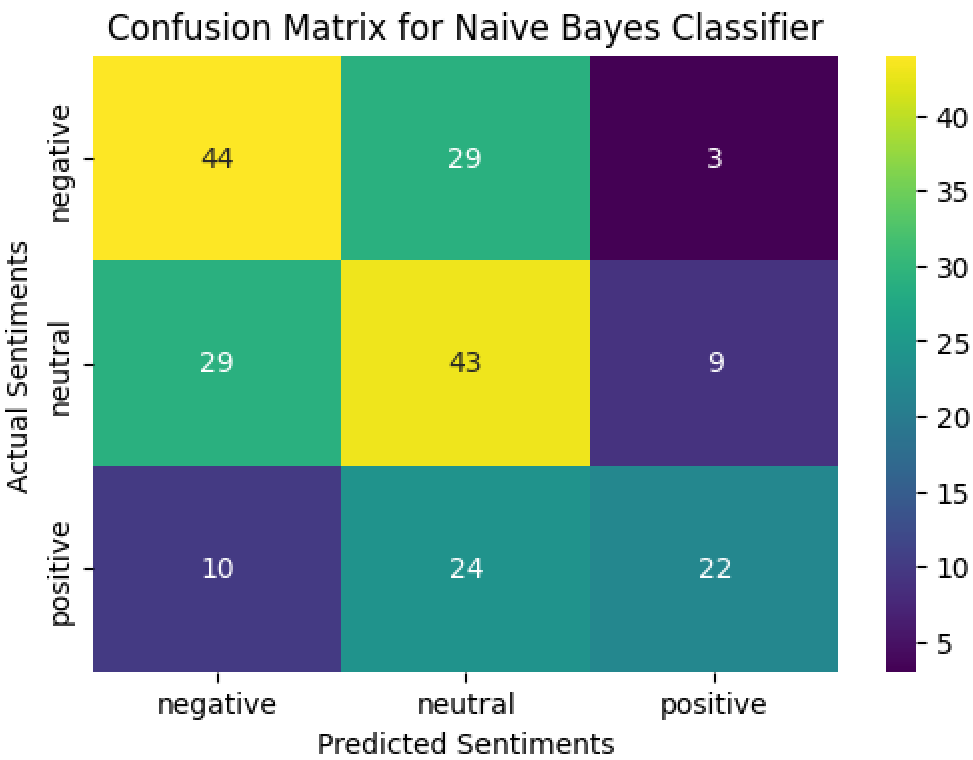
```
              precision    recall  f1-score   support

    negative       0.53      0.58      0.55        76
     neutral       0.45      0.53      0.49        81
    positive       0.65      0.39      0.49        56
```

```
    accuracy                          0.51       213
   macro avg       0.54      0.50      0.51       213
weighted avg       0.53      0.51      0.51       213
```

```
from sklearn.metrics import accuracy_score

## accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Naive Bayes Classifier: {accuracy:.2%}")
```

```
Accuracy of Naive Bayes Classifier: 51.17%
```

```
## confusion metrics

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)


# Plotting the confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='viridis',
            xticklabels=['negative', 'neutral', 'positive'],
            yticklabels=['negative', 'neutral', 'positive'])
plt.xlabel('Predicted Sentiments')
plt.ylabel('Actual Sentiments')
plt.title('Confusion Matrix for Naive Bayes Classifier')
plt.show()
```



NB

```
negative  neutral  positive
```

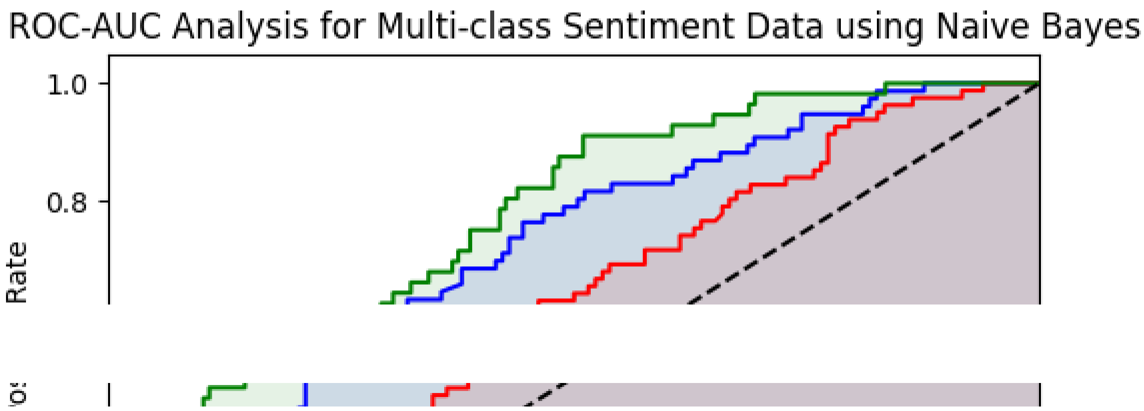
negative 44 29 3

neutral 29 43 9

positive 10 24 22

```
 ## NB
          negative  neutral  positive
negative  44        29       3
neutral   29        43       9
positive  10        24       22
```

```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Binarize the labels
y_test_bin = label_binarize(y_test, classes=['negative', 'neutral', 'positive'])
y_train_bin = label_binarize(y_train, classes=['negative', 'neutral', 'positive']
n_classes = y_test_bin.shape[1]

# Train the multinomial Naive Bayes classifier
naive_bayes_classifier = MultinomialNB()
naive_bayes_classifier.fit(X_train_counts, y_train)

# Compute the predicted probabilities
y_score = naive_bayes_classifier.predict_proba(X_test_counts)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot the ROC curves with filled colors
plt.figure(figsize=(6,4))
colors = ['blue', 'red', 'green']
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color,
             label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc[i
    plt.fill_between(fpr[i], tpr[i], color=color, alpha=0.1)  # this line fills t

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Analysis for Multi-class Sentiment Data using Naive Bayes')
plt.legend(loc="lower right")
plt.show()
```



random forest


```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

### Split
X = twt['preprocessed_text']
y = twt['sentiment']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

## convert text data into count features (Bag of Words)
count_vectorizer = CountVectorizer(max_features=5000)
X_train_counts = count_vectorizer.fit_transform(X_train)
X_test_counts = count_vectorizer.transform(X_test)

## train the Random Forest classifier
random_forest_classifier = RandomForestClassifier(n_estimators=100, random_state=
random_forest_classifier.fit(X_train_counts, y_train)  # Corrected this line

## predictions
y_pred = random_forest_classifier.predict(X_test_counts)

## classification report
report = classification_report(y_test, y_pred, target_names=['negative', 'neutral
print(report)
```

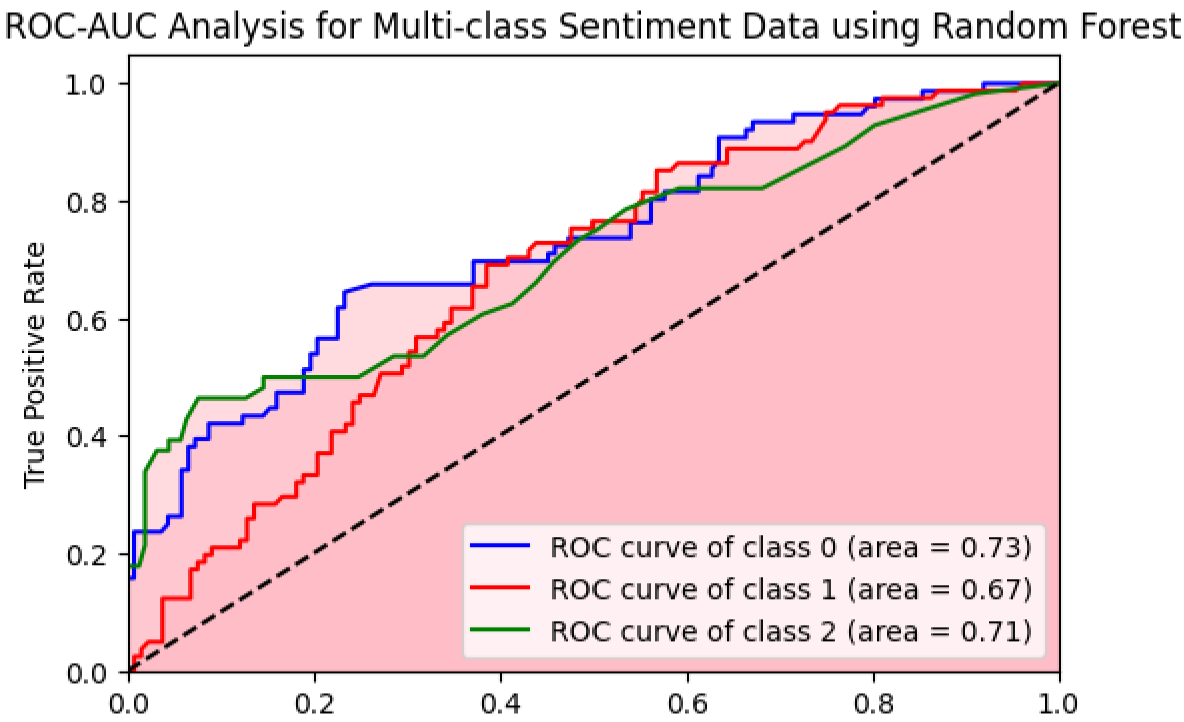
```
              precision    recall  f1-score   support

    negative       0.59      0.58      0.59        76
     neutral       0.50      0.69      0.58        81
    positive       0.78      0.38      0.51        56

    accuracy                           0.57       213
   macro avg       0.62      0.55      0.56       213
weighted avg       0.61      0.57      0.56       213
```

The Random Forest model achieved an overall accuracy of 57% on the test data. The model performed best in distinguishing positive sentiments in terms of precision (78%), but struggled in recall for the same class (38%). The neutral class showed the highest recall (69%), indicating the model's capability to capture most of the true neutral sentiments, but its precision is just at 50%. Overall, while there are areas of strength, there's room for improvement in the model's predictive capabilities, especially in balancing precision and recall for each sentiment class.

```
from sklearn.metrics import accuracy_score

### accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on test data: {accuracy:.2%}")
```

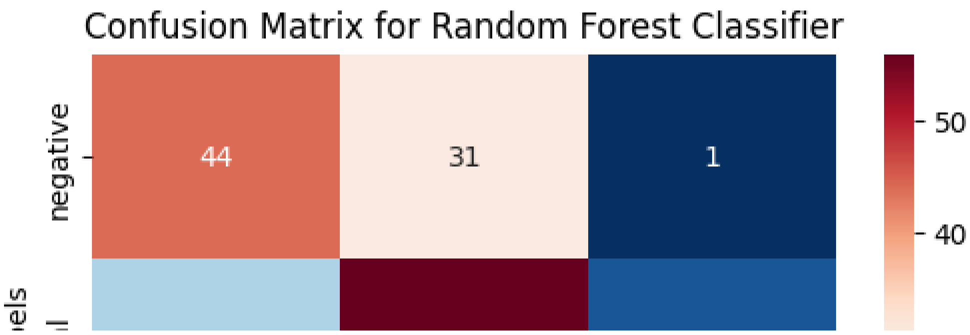
```
    Accuracy on test data: 56.81%
```

```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Binarize the labels for ROC analysis
y_test_bin = label_binarize(y_test, classes=['negative', 'neutral', 'positive'])
n_classes = y_test_bin.shape[1]

# Compute the predicted probabilities from the Random Forest
y_score = random_forest_classifier.predict_proba(X_test_counts)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot the ROC curves with filled colors
plt.figure(figsize=(6,4))
colors = ['blue', 'red', 'green']
fill_colors = ['lightpink', 'lightpink', 'lightpink']  # specify fill colors here

for i, color, fill_color in zip(range(n_classes), colors, fill_colors):
    plt.plot(fpr[i], tpr[i], color=color, label='ROC curve of class {0} (area = {
    plt.fill_between(fpr[i], tpr[i], color=fill_color, alpha=0.5)  # this line fi


plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Analysis for Multi-class Sentiment Data using Random Forest')
plt.legend(loc="lower right")
plt.show()
```



confusion matrix

```
import seaborn as sns
from sklearn.metrics import confusion_matrix

## Predictions for Random Forest
y_pred = random_forest_classifier.predict(X_test_counts)

##confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=['negative', 'neutral', 'positive'])

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='g', cmap='RdBu_r',
            xticklabels=['negative', 'neutral', 'positive'],
            yticklabels=['negative', 'neutral', 'positive'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for Random Forest Classifier')
plt.show()
```



RF

```
negative  neutral  positive
```

negative 44 31 1

neutral 20 56 5

positive 10 25 21

```
## RF
          negative  neutral  positive
negative  44        31        1
neutral   20        56        5
positive  10        25       21
```

ensemble:

Three models are employed

RandomForestClassifier: Used as a base learner and labeled 'rf'. MultinomialNB: Another base learner, labeled 'nb'. LogisticRegression: Serves as the final estimator in the StackingClassifier, making predictions based on the base learners' outputs.

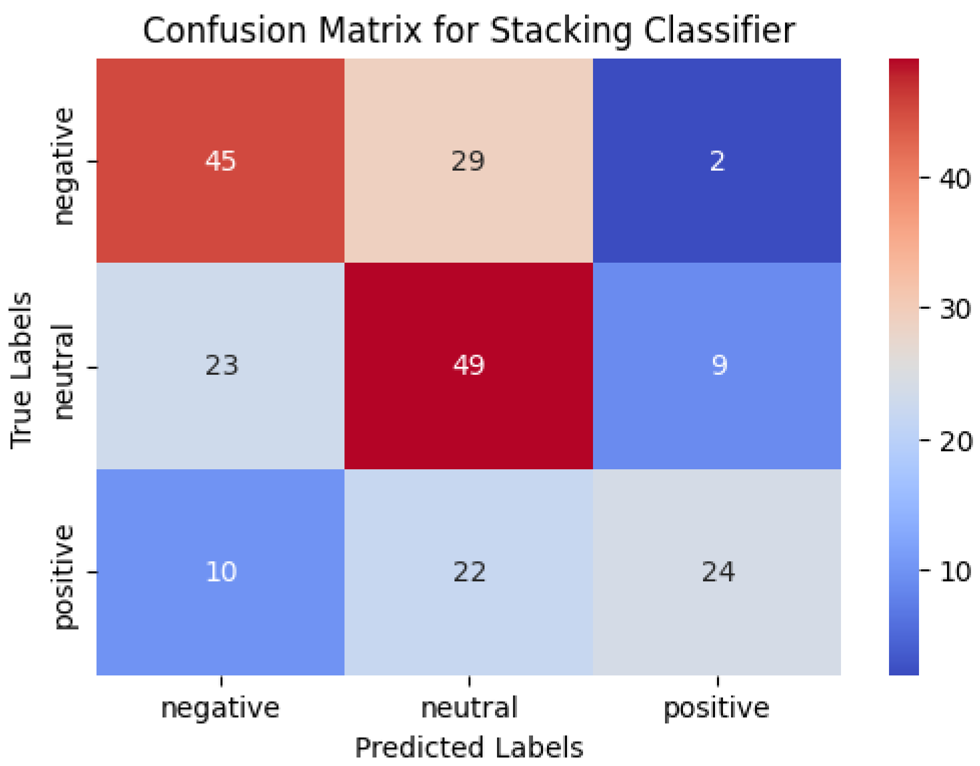
```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from sklearn.metrics import accuracy_score

# Define the base models
base_learners = [
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('nb', MultinomialNB())
]

# Initialize the Stacking Classifier with the base learners and a logistic regres
stacked_model = StackingClassifier(estimators=base_learners, final_estimator=Logi

# Fit the model
stacked_model.fit(X_train_counts, y_train)

# Predict
y_pred = stacked_model.predict(X_test_counts)

# Measure accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2%}")
```

```
    Accuracy: 55.40%
```

```
from sklearn.metrics import classification_report

### classification report
report = classification_report(y_test, y_pred, target_names=['negative', 'neutral
print(report)
```

```
                  precision    recall  f1-score   support

        negative       0.58      0.59      0.58        76
         neutral       0.49      0.60      0.54        81
        positive       0.69      0.43      0.53        56

        accuracy                           0.55       213
       macro avg       0.58      0.54      0.55       213
    weighted avg       0.57      0.55      0.55       213
```

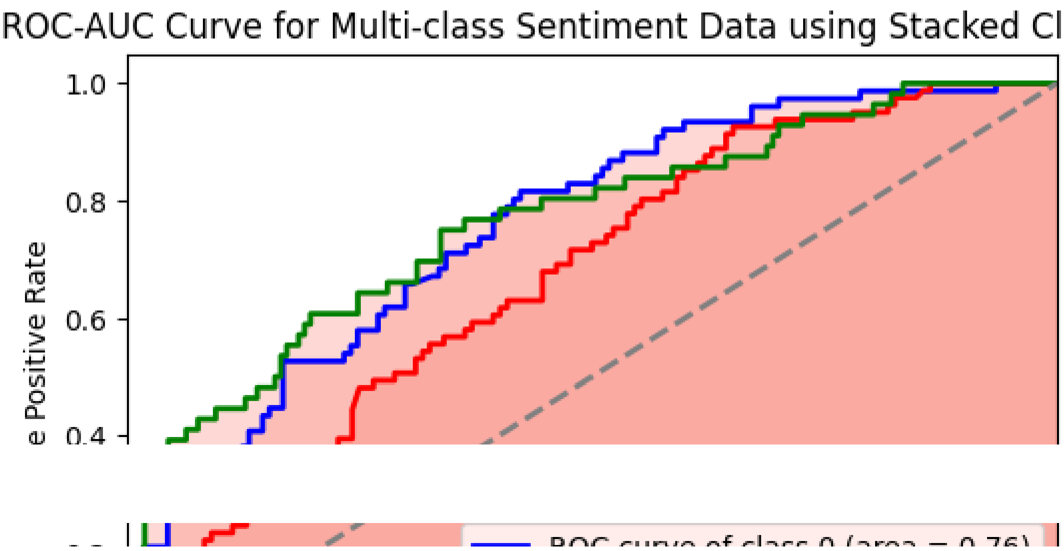
```
## confusion matrix

import seaborn as sns
from sklearn.metrics import confusion_matrix

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=['negative', 'neutral', 'positive'])

# Plot the confusion matrix using seaborn
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='g', cmap='coolwarm',
            xticklabels=['negative', 'neutral', 'positive'],
            yticklabels=['negative', 'neutral', 'positive'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for Stacking Classifier')
plt.show()
```



stacked classifier

	negative	neutral	positive
negative	45	29	2
neutral	23	49	9
positive	10	22	24

```
 ## stacked classifier
          negative  neutral  positive
negative  45        29       2
neutral   23        49       9
positive  10         22      24


## roc-auc

from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# 1. Binarize the labels
y_test_bin = label_binarize(y_test, classes=['negative', 'neutral', 'positive'])
n_classes = y_test_bin.shape[1]

# 2. Predict the probabilities
y_score = stacked_model.predict_proba(X_test_counts)

# 3. Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

## ROC-AUC curve
plt.figure(figsize=(6,4))
colors = ['blue', 'red', 'green']
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2, label='ROC curve of class {0} (ar
    plt.fill_between(fpr[i], tpr[i], color='salmon', alpha=0.3)  # Fill the area

plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve for Multi-class Sentiment Data using Stacked Classifier'
plt.legend(loc="lower right")
plt.show()
```



GBM (using GradientBoostingClassifier from scikit-learn), SVM, and RandomForest as base learners in a stacking ensemble:

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Define the base models
base_learners = [
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('gbm', GradientBoostingClassifier(n_estimators=100, random_state=42)),
    ('svm', SVC(kernel='linear', probability=True))
]

# Initialize the Stacking Classifier with the base learners and a logistic regres
stacked_model = StackingClassifier(estimators=base_learners, final_estimator=Logi

# Fit the model
stacked_model.fit(X_train_counts, y_train)

# Predict
y_pred = stacked_model.predict(X_test_counts)

# Measure accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2%}")
```

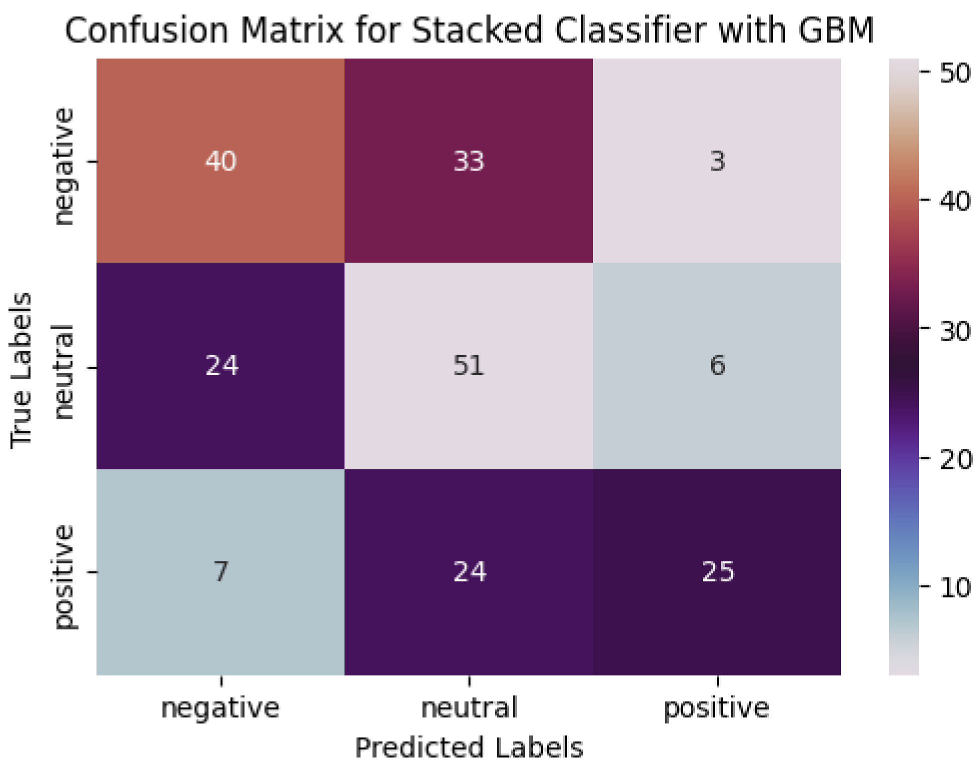
```
Accuracy: 54.46%
```

```
from sklearn.metrics import classification_report

### classification report

report = classification_report(y_test, y_pred, target_names=['negative', 'neutral
print(report)
```

```
              precision    recall  f1-score   support

    negative       0.56      0.53      0.54        76
     neutral       0.47      0.63      0.54        81
    positive       0.74      0.45      0.56        56

    accuracy                           0.54       213
   macro avg       0.59      0.53      0.55       213
weighted avg       0.57      0.54      0.55       213
```

```
## ROC-AUC

from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# 1. Binarize the labels
y_test_bin = label_binarize(y_test, classes=['negative', 'neutral', 'positive'])
n_classes = y_test_bin.shape[1]

# 2. Predict the probabilities
y_score = stacked_model.predict_proba(X_test_counts)

# 3. Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

## ROC curve
plt.figure(figsize=(6,4))
colors = ['blue', 'red', 'green']
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2, label='ROC curve of class {0} (ar
    plt.fill_between(fpr[i], tpr[i], color='lightgreen', alpha=0.3)  # Fill the a

plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve for Multi-class Sentiment Data using Stacked Classifier
plt.legend(loc="lower right")
plt.show()
```


```
## confusion matrix

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=['negative', 'neutral', 'positive'])

##confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='g', cmap='twilight',
            xticklabels=['negative', 'neutral', 'positive'],
            yticklabels=['negative', 'neutral', 'positive'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for Stacked Classifier with GBM')
plt.show()
```



GBM, SVM

```
negative   neutral   positive
```

negative 40 33 3

neutral 24 51 6

positive 7 24 25

```
 ## GBM, SVM
          negative  neutral  positive
negative  40        33       3
neutral   24        51       6
positive  7         24       25
```

```
  File "<ipython-input-1-46707a27f19c>", line 2
    negative  neutral  positive
    ^
IndentationError: unexpected indent
```

SEARCH STACK OVERFLOW

DL

LSTM

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Define maximum number of words to consider as features
max_features = 10000
# Define sequence length (number of words) for each document
maxlen = 300

tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(X_train)

X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

X_train_pad = pad_sequences(X_train_seq, maxlen=maxlen)
X_test_pad = pad_sequences(X_test_seq, maxlen=maxlen)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

model = Sequential()
model.add(Embedding(max_features, 128, input_length=maxlen))
model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(3, activation='softmax'))  # three classes: negative, neutral, po

model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.summary()
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't mee
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 300, 128)          1280000

 lstm (LSTM)                 (None, 64)                49408

 dense (Dense)               (None, 3)                 195

=================================================================
Total params: 1329603 (5.07 MB)
Trainable params: 1329603 (5.07 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectiona

model = Sequential()

# Embedding Layer (Optionally initialize with pre-trained embeddings)
model.add(Embedding(max_features, 128, input_length=maxlen))

# Bidirectional LSTM
model.add(Bidirectional(LSTM(128, return_sequences=True, dropout=0.3, recurrent_d
model.add(BatchNormalization())  # Batch normalization layer

# Optional: Another LSTM layer
model.add(LSTM(64, dropout=0.3, recurrent_dropout=0.3))
model.add(BatchNormalization())  # Batch normalization layer

# Dense layer
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))  # Dropout layer after the dense layer

# Output Layer
model.add(Dense(3, activation='softmax'))

# Compile the model
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 300, 128)          1280000

 bidirectional (Bidirection  (None, 300, 256)          263168
 al)

 batch_normalization (Batch  (None, 300, 256)          1024
 Normalization)

 lstm_1 (LSTM)               (None, 64)                82176

 batch_normalization_1 (Bat  (None, 64)                256
 chNormalization)

 dense (Dense)               (None, 64)                4160

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 3)                 195

=================================================================
Total params: 1630979 (6.22 MB)
Trainable params: 1630339 (6.22 MB)
```

```
Non-trainable params: 640 (2.50 KB)
_________________________________________________________________
```

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROn

### Converting labels to integers
y_train_int = y_train.map({'negative': 0, 'neutral': 1, 'positive': 2}).values
y_test_int = y_test.map({'negative': 0, 'neutral': 1, 'positive': 2}).values

# Define callbacks
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=T

## saving the model
checkpoint = ModelCheckpoint('best_weights.h5', monitor='val_loss', save_best_onl

## will reduce the learning rate if no improvement in val_loss after 3 epochs
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=

## model  training with 50 epochs and early stopping
model.fit(X_train_pad, y_train_int,
          batch_size=32,
          epochs=50,
          validation_split=0.2,
          callbacks=[early_stop, checkpoint, reduce_lr])
```

```
22/22 [==============================] - ETA: 0s - loss: 1.2383 - accuracy: 0
Epoch 3: val_loss improved from 1.09404 to 1.09363, saving model to best_weig
22/22 [==============================] - 69s 3s/step - loss: 1.2383 - accurac
Epoch 4/50
22/22 [==============================] - ETA: 0s - loss: 1.1588 - accuracy: 0
Epoch 4: val_loss improved from 1.09363 to 1.09213, saving model to best_weig
22/22 [==============================] - 64s 3s/step - loss: 1.1588 - accurac
Epoch 5/50
22/22 [==============================] - ETA: 0s - loss: 0.9579 - accuracy: 0
Epoch 5: val_loss improved from 1.09213 to 1.08590, saving model to best_weig
22/22 [==============================] - 70s 3s/step - loss: 0.9579 - accurac
Epoch 6/50
22/22 [==============================] - ETA: 0s - loss: 0.5566 - accuracy: 0
Epoch 6: val_loss did not improve from 1.08590
22/22 [==============================] - 66s 3s/step - loss: 0.5566 - accurac
Epoch 7/50
22/22 [==============================] - ETA: 0s - loss: 0.2991 - accuracy: 0
Epoch 7: val_loss improved from 1.08590 to 1.08528, saving model to best_weig
22/22 [==============================] - 66s 3s/step - loss: 0.2991 - accurac
Epoch 8/50
22/22 [==============================] - ETA: 0s - loss: 0.1924 - accuracy: 0
```

```
22/22 [==============================] - ETA: 0s - loss: 0.0585 - accuracy: 0
Epoch 11: val_loss improved from 1.02343 to 1.00311, saving model to best_wei
22/22 [==============================] - 68s 3s/step - loss: 0.0585 - accurac
Epoch 12/50
22/22 [==============================] - ETA: 0s - loss: 0.0631 - accuracy: 0
Epoch 12: val_loss did not improve from 1.00311
22/22 [==============================] - 64s 3s/step - loss: 0.0631 - accurac
Epoch 13/50
22/22 [==============================] - ETA: 0s - loss: 0.0326 - accuracy: 0
Epoch 13: val_loss did not improve from 1.00311
22/22 [==============================] - 62s 3s/step - loss: 0.0326 - accurac
Epoch 14/50
22/22 [==============================] - ETA: 0s - loss: 0.0290 - accuracy: 0
Epoch 14: val_loss did not improve from 1.00311

Epoch 14: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
22/22 [==============================] - 64s 3s/step - loss: 0.0290 - accurac
Epoch 15/50
22/22 [==============================] - ETA: 0s - loss: 0.0409 - accuracy: 0
Epoch 15: val_loss did not improve from 1.00311
22/22 [==============================] - 63s 3s/step - loss: 0.0409 - accurac
Epoch 16/50
22/22 [==============================] - ETA: 0s - loss: 0.0175 - accuracy: 0
Epoch 16: val_loss did not improve from 1.00311
22/22 [==============================] - 64s 3s/step - loss: 0.0175 - accurac
<keras.src.callbacks.History at 0x7811285ac4c0>
```

```
# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test_pad, y_test_int)

print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

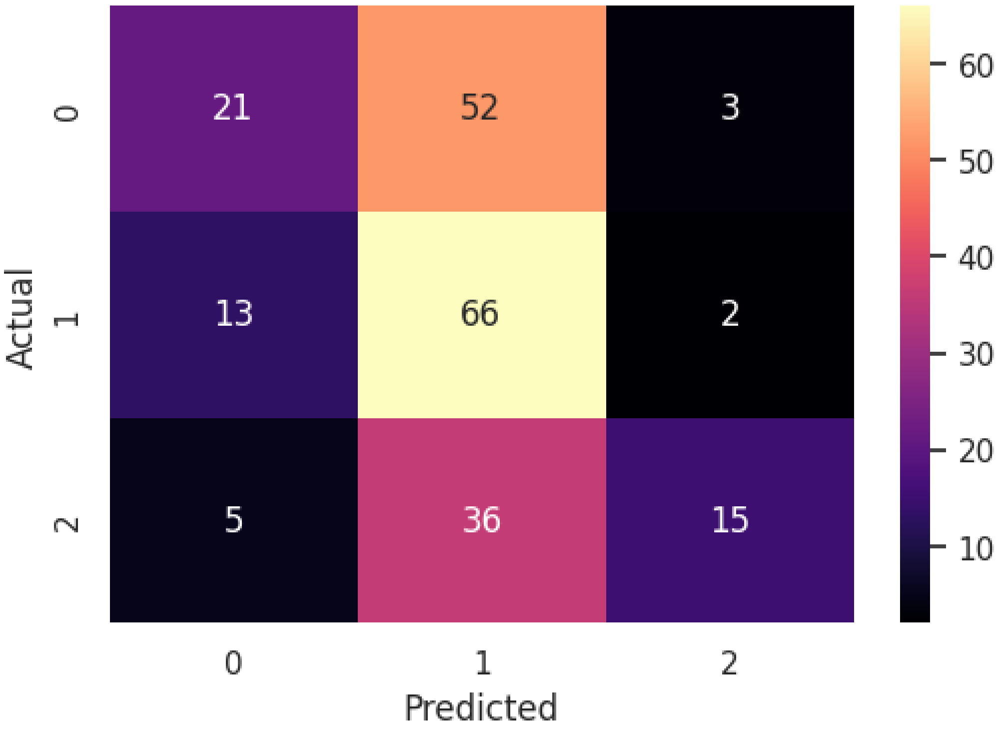
```
7/7 [==============================] - 3s 489ms/step - loss: 1.0181 - accurac
Test Accuracy: 47.89%
```

```
from sklearn.metrics import classification_report

y_pred = model.predict(X_test_pad).argmax(axis=1)
report = classification_report(y_test_int, y_pred, target_names=['negative', 'neu
print(report)
```

```
7/7 [==============================] - 5s 488ms/step
              precision    recall  f1-score   support

    negative       0.54      0.28      0.37        76
     neutral       0.43      0.81      0.56        81
    positive       0.75      0.27      0.39        56

    accuracy                           0.48       213
   macro avg       0.57      0.45      0.44       213
weighted avg       0.55      0.48      0.45       213
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

## classes prediction
y_pred = model.predict(X_test_pad).argmax(axis=1)

## confusion matrix generate
cm = confusion_matrix(y_test_int, y_pred)

## confusion matrix
plt.figure(figsize=(6, 4))
sns.set(font_scale=1)  # for label size
sns.heatmap(cm, annot=True, annot_kws={"size": 12}, cmap="magma", fmt='g')

plt.xlabel('Predicted')
plt.ylabel('Actual')

plt.show()
```

```
7/7 [==============================] - 3s 465ms/step
```




```
import pandas as pd
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test_int, y_pred)

## Convert the confusion matrix to a DataFrame
cm_df = pd.DataFrame(cm, index=['negative', 'neutral', 'positive'], columns=['neg

print(cm_df)
```

```
          negative  neutral  positive
negative  21        52       3
neutral   13        66       2
positive  5         36       15
```

LSTM

```
          negative  neutral  positive
```

negative 21 52 3

neutral 13 66 2

positive 5 36 15

roc-auc

```
y_score = model.predict(X_test_pad)
```

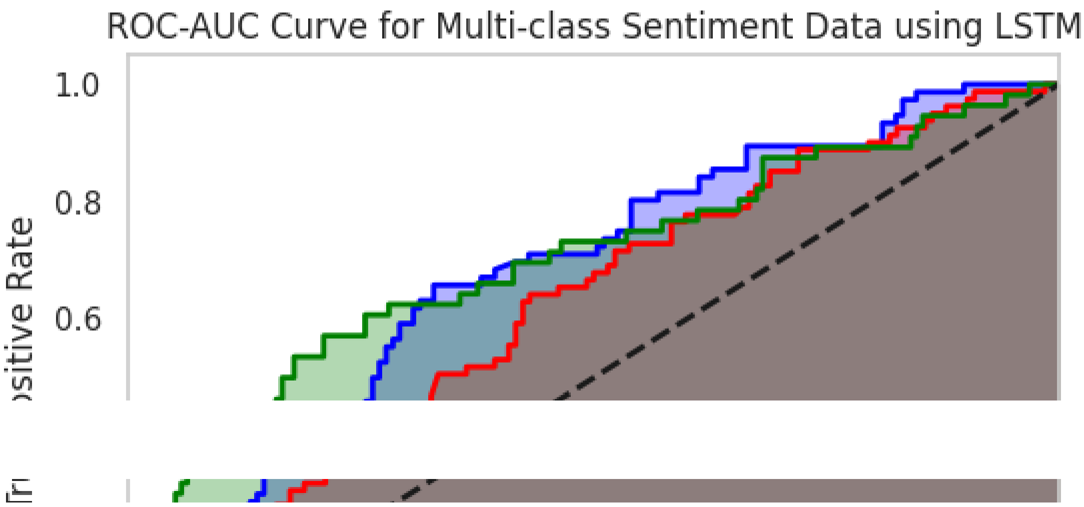
```
7/7 [==============================] - 0s 2ms/step
```

```
print(y_score.shape)
```

```
(213, 300, 128)
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize
import seaborn as sns

# Set style for white plain background
sns.set_style("whitegrid", {'axes.grid' : False})

# Predict the probabilities for each class
y_score = model.predict(X_test_pad)

# Convert y_test_int into a one-hot encoded format
y_test_bin = label_binarize(y_test_int, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot all ROC curves
plt.figure(figsize=(6,4))

colors = ['blue', 'red', 'green']
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))
    # Fill the area under the ROC curve
    plt.fill_between(fpr[i], tpr[i], color=color, alpha=0.3)

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve for Multi-class Sentiment Data using LSTM')
plt.legend(loc="lower right")
plt.show()
```

```
7/7 [==============================] - 5s 623ms/step
```



BLSTM

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense

model = Sequential()
model.add(Embedding(max_features, 128, input_length=maxlen))
model.add(Bidirectional(LSTM(64, dropout=0.2, recurrent_dropout=0.2)))
model.add(Dense(3, activation='softmax'))  # three classes: negative, neutral, po

model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 300, 128)          1280000

 bidirectional_1 (Bidirecti  (None, 128)               98816
 onal)

 dense_2 (Dense)             (None, 3)                 387

=================================================================
Total params: 1379203 (5.26 MB)
Trainable params: 1379203 (5.26 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
from tensorflow.keras.callbacks import EarlyStopping

# Define the early stopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=T

# Train the model
model.fit(X_train_pad, y_train_int,
          batch_size=32,
          epochs=50,
          validation_split=0.2,
          callbacks=[early_stop])
```

```
Epoch 1/50
22/22 [==============================] - 40s 1s/step - loss: 1.0926 - accurac
Epoch 2/50
22/22 [==============================] - 26s 1s/step - loss: 1.0621 - accurac
Epoch 3/50
22/22 [==============================] - 24s 1s/step - loss: 0.9294 - accurac
Epoch 4/50
22/22 [==============================] - 28s 1s/step - loss: 0.6161 - accurac
Epoch 5/50
22/22 [==============================] - 26s 1s/step - loss: 0.3355 - accurac
Epoch 6/50
22/22 [==============================] - 26s 1s/step - loss: 0.4288 - accurac
Epoch 7/50
22/22 [==============================] - 24s 1s/step - loss: 0.2873 - accurac
<keras.src.callbacks.History at 0x781122b4f550>
```

```
# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test_pad, y_test_int, verbose=1)

print(f"Test accuracy: {accuracy * 100:.2f}%")
```

```
7/7 [==============================] - 1s 120ms/step - loss: 1.0713 - accurac
Test accuracy: 39.91%
```

```
from tensorflow.keras.callbacks import EarlyStopping


### LSTM model

model = Sequential()
model.add(Embedding(max_features, 128, input_length=maxlen))
model.add(Bidirectional(LSTM(64, dropout=0.2, recurrent_dropout=0.2)))
model.add(Dense(3, activation='softmax'))  # Assuming three classes: negative, ne

model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 300, 128)          1280000

 bidirectional_2 (Bidirecti  (None, 128)               98816
 onal)

 dense_3 (Dense)             (None, 3)                 387

=================================================================
Total params: 1379203 (5.26 MB)
Trainable params: 1379203 (5.26 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
### early stopping callback

early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=
```

```
model.fit(X_train_pad, y_train_int, batch_size=32, epochs=50, validation_split=0.
```

```
Epoch 1/50
22/22 [==============================] - 44s 1s/step - loss: 1.0907 - accuracy
Epoch 2/50
22/22 [==============================] - 29s 1s/step - loss: 1.0649 - accuracy
Epoch 3/50
22/22 [==============================] - 32s 1s/step - loss: 0.9754 - accuracy
Epoch 4/50
22/22 [==============================] - 37s 2s/step - loss: 0.7413 - accuracy
Epoch 5/50
22/22 [==============================] - 30s 1s/step - loss: 0.4996 - accuracy
Epoch 6/50
22/22 [==============================] - 29s 1s/step - loss: 0.2831 - accuracy
Epoch 7/50
22/22 [==============================] - 30s 1s/step - loss: 0.2463 - accuracy
Epoch 8/50
22/22 [==============================] - 29s 1s/step - loss: 0.1467 - accuracy
Epoch 9/50
22/22 [==============================] - 32s 1s/step - loss: 0.1011 - accuracy
Epoch 10/50
22/22 [==============================] - 29s 1s/step - loss: 0.0728 - accuracy
Epoch 11/50
22/22 [==============================] - 30s 1s/step - loss: 0.0556 - accuracy
Epoch 12/50
22/22 [==============================] - 29s 1s/step - loss: 0.0442 - accuracy
Epoch 13/50
22/22 [==============================] - 29s 1s/step - loss: 0.0386 - accuracy
Epoch 14/50
22/22 [==============================] - 30s 1s/step - loss: 0.0323 - accuracy
<keras.callbacks.History at 0x783564307eb0>
```

```
loss, accuracy = model.evaluate(X_test_pad, y_test_int)
print(f"Test accuracy: {accuracy:.2%}")
```

```
7/7 [==============================] - 1s 147ms/step - loss: 1.0075 - accuracy
Test accuracy: 45.54%
```

```
y_pred = model.predict(X_test_pad).argmax(axis=1)
report = classification_report(y_test_int, y_pred, target_names=['negative', 'neu
print(report)
```

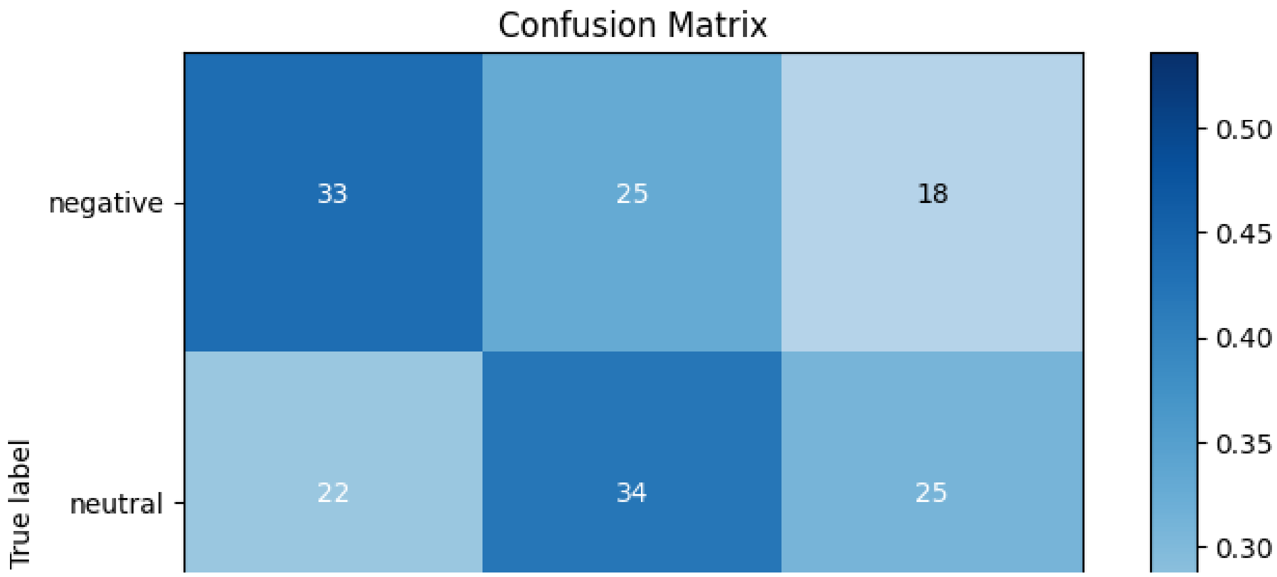
```
7/7 [==============================] - 1s 130ms/step
              precision    recall  f1-score   support

    negative       0.54      0.43      0.48        76
     neutral       0.43      0.42      0.43        81
    positive       0.41      0.54      0.47        56

    accuracy                           0.46       213
   macro avg       0.46      0.46      0.46       213
weighted avg       0.46      0.46      0.46       213
```

```
pip install seaborn
```

```
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: numpy!=1.24.0,>=1.17 in /usr/local/lib/python3
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /usr/local/lib/pytho
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pack
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

```
## confusion metrics function

def plot_confusion_matrix(y_true, y_pred, classes, cmap=plt.cm.Blues):
    """
    Plot the confusion matrix.
    """
    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.figure(figsize=(8, 6))
    plt.imshow(cm_normalized, interpolation='nearest', cmap=cmap)
    plt.title("Confusion Matrix")
    plt.colorbar()

    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    thresh = cm_normalized.max() / 2.
    for i, j in itertools.product(range(cm_normalized.shape[0]), range(cm_normali
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if cm_normalized[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.show()



## confusion metrics plot

import itertools

classes = ['negative', 'neutral', 'positive']
plot_confusion_matrix(y_test_int, y_pred, classes=classes)
```



BLSTM

```
        negative  neutral  positive
```

negative 33 25 18

neutral 22 34 25

positive 6 20 30

	negative	neutral	positive
negative	33	25	18
neutral	22	34	25
positive	6	20	30

```
pip install pandas
```

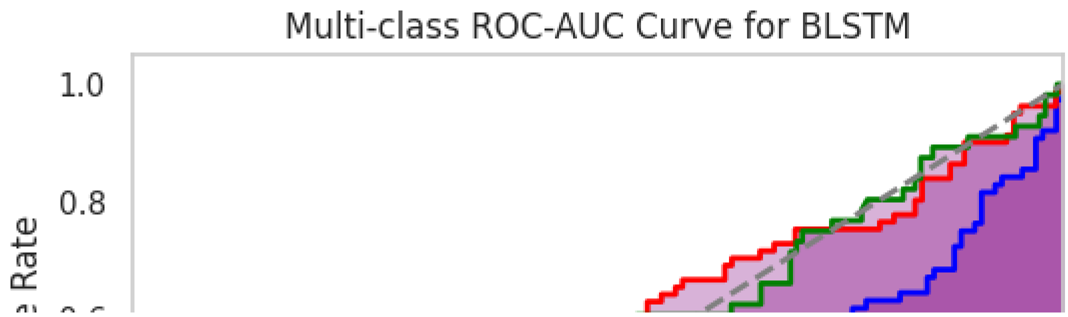
```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/pytho
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pac
```

```
import pandas as pd
from sklearn.metrics import confusion_matrix

# Assuming y_test_int contains the true labels and y_pred contains the predicted
cm = confusion_matrix(y_test_int, y_pred)

# Convert the confusion matrix to a DataFrame
cm_df = pd.DataFrame(cm, index=['negative', 'neutral', 'positive'], columns=['neg

# Display the DataFrame
print(cm_df)
```

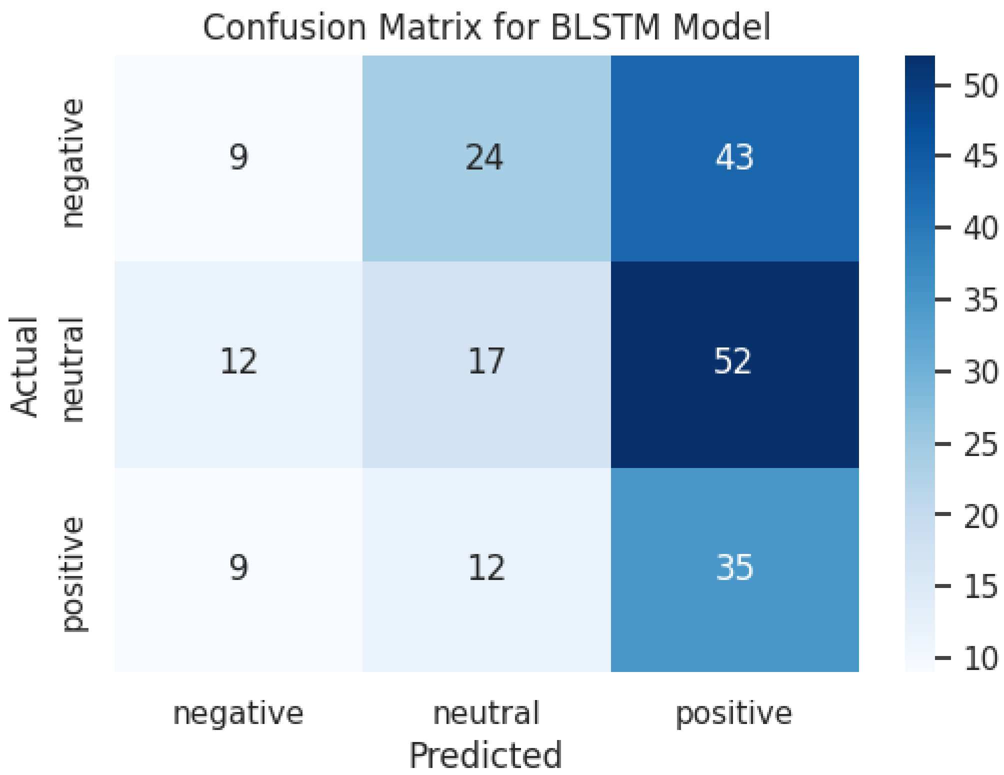
```
          negative  neutral  positive
negative  33        25       18
neutral   22        34       25
positive  6         20       30
```

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

# 1. Predict the probabilities for each class
y_prob = model.predict(X_test_pad)

# 2. Binarize the true labels
y_test_bin = label_binarize(y_test_int, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]

# 3. Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# 4. Plot the ROC curve
plt.figure(figsize=(6, 4))
for i, color in zip(range(n_classes), ['blue', 'red', 'green']):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc[i
    plt.fill_between(fpr[i], tpr[i], color='purple', alpha=0.3)

plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multi-class ROC-AUC Curve for BLSTM')
plt.legend(loc="lower right")
plt.show()
```

```
7/7 [==============================] - 1s 204ms/step
```




```
import numpy as np
import seaborn as sns
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

# 1. Get predicted class labels
y_pred = np.argmax(y_prob, axis=1)

# 2. Compute the confusion matrix
cm = confusion_matrix(y_test_int, y_pred)

# 3. Visualize the confusion matrix
plt.figure(figsize=(6, 4))
sns.set(font_scale=1)  # Adjust to fit
sns.heatmap(cm, annot=True, fmt='g', cmap="Blues",
            xticklabels=['negative', 'neutral', 'positive'],
            yticklabels=['negative', 'neutral', 'positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for BLSTM Model')
plt.show()
```



```
import pandas as pd
from sklearn.metrics import confusion_matrix

# Assuming y_test_int contains the true labels and y_pred contains the predicted
cm = confusion_matrix(y_test_int, y_pred)

# Convert the confusion matrix to a DataFrame
cm_df = pd.DataFrame(cm, index=['negative', 'neutral', 'positive'], columns=['neg

# Display the DataFrame
print(cm_df)
```

	negative	neutral	positive
negative	9	24	43
neutral	12	17	52
positive	9	12	35

```
## performance metrics report

from sklearn.metrics import classification_report

# Get predicted class labels
y_pred = np.argmax(y_prob, axis=1)

# Generate the classification report
report = classification_report(y_test_int, y_pred, target_names=['negative', 'neu

print(report)
```

	precision	recall	f1-score	support
negative	0.30	0.12	0.17	76
neutral	0.32	0.21	0.25	81
positive	0.27	0.62	0.38	56
accuracy			0.29	213
macro avg	0.30	0.32	0.27	213
weighted avg	0.30	0.29	0.26	213

Statistical Analysis

N=.20*1070=214

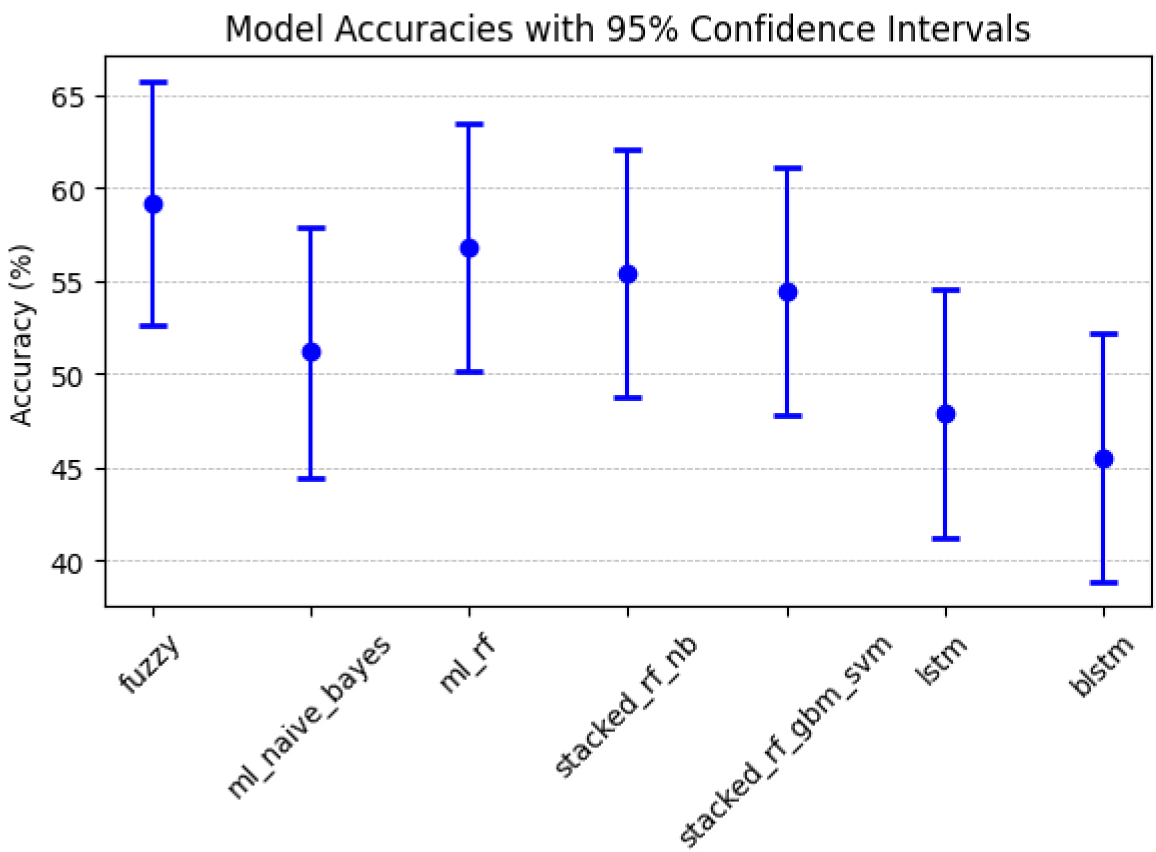
where .20= test set

```
import math

N = 214

def compute_CI(accuracy_percent):
    p = accuracy_percent / 100.0
    SE = math.sqrt(p * (1-p) / N)
    MOE = 1.96 * SE
    lower_bound = p - MOE
    upper_bound = p + MOE
    return lower_bound * 100, upper_bound * 100

models = {
    "fuzzy": 59.15,
    "ml_naive_bayes": 51.17,
    "ml_rf": 56.81,
    "stacked_rf_nb": 55.40,
    "stacked_rf_gbm_svm": 54.46,
    "lstm": 47.89,
    "blstm": 45.54
}

CI_dict = {model: compute_CI(accuracy) for model, accuracy in models.items()}

for model, ci in CI_dict.items():
    print(f"{model}: {ci[0]:.2f}% to {ci[1]:.2f}%")
```

```
fuzzy: 52.56% to 65.74%
ml_naive_bayes: 44.47% to 57.87%
ml_rf: 50.17% to 63.45%
stacked_rf_nb: 48.74% to 62.06%
stacked_rf_gbm_svm: 47.79% to 61.13%
lstm: 41.20% to 54.58%
blstm: 38.87% to 52.21%
```

```
import matplotlib.pyplot as plt

# Models
models = ['fuzzy', 'ml_naive_bayes', 'ml_rf', 'stacked_rf_nb', 'stacked_rf_gbm_sv

# Mean accuracies
accuracies = [59.15, 51.17, 56.81, 55.40, 54.46, 47.89, 45.54]

# Lower bounds of the CIs
lower_bounds = [52.56, 44.47, 50.17, 48.74, 47.79, 41.22, 38.87]

# Upper bounds of the CIs
upper_bounds = [65.74, 57.87, 63.45, 62.06, 61.13, 54.56, 52.21]

# Errors (distance from mean accuracy to the boundaries of CI)
errors = [(accuracies[i] - lower_bounds[i], upper_bounds[i] - accuracies[i]) for

# Plotting
plt.figure(figsize=(6,4.5))
plt.errorbar(models, accuracies, yerr=list(zip(*errors)), fmt='o', capsize=5, cap
plt.ylabel('Accuracy (%)')
plt.title('Model Accuracies with 95% Confidence Intervals')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

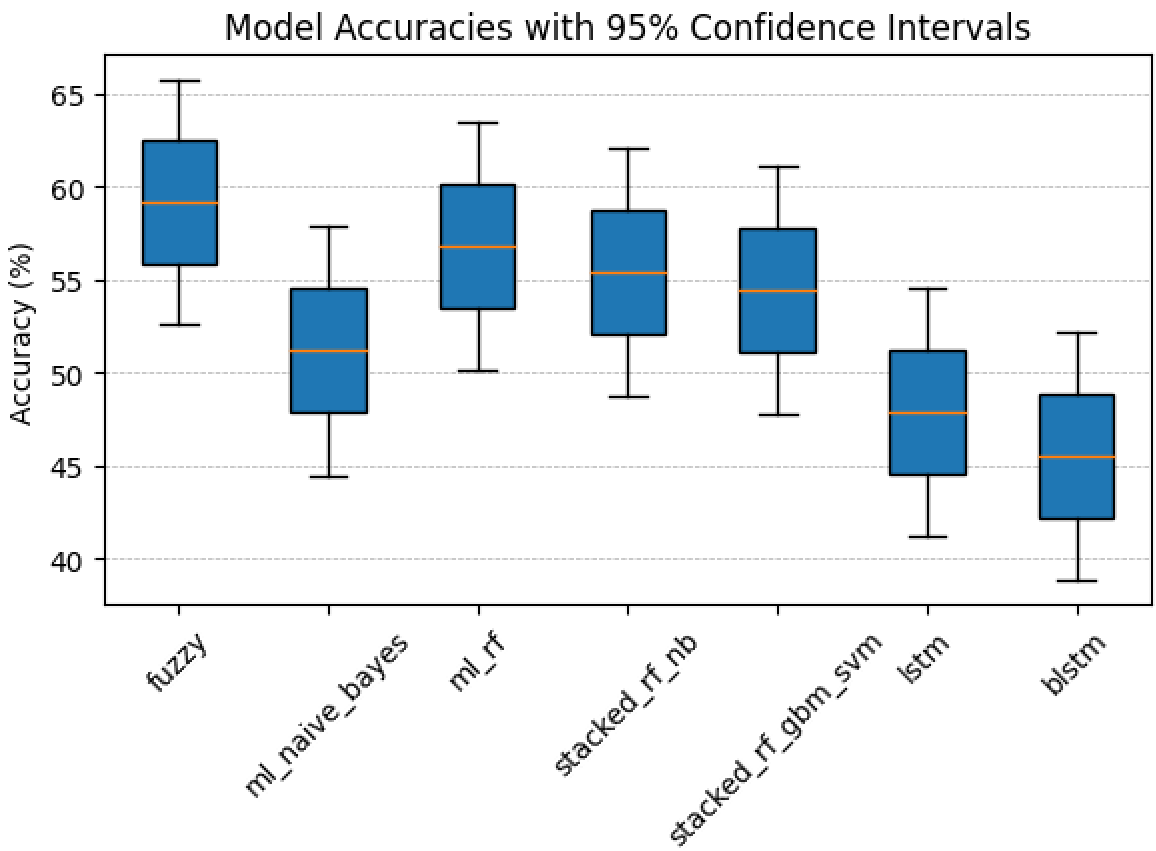
```
<ipython-input-1-873622dc23d4>:20: UserWarning: marker is redundantly defined
  plt.errorbar(models, accuracies, yerr=list(zip(*errors)), fmt='o', capsize=!
```



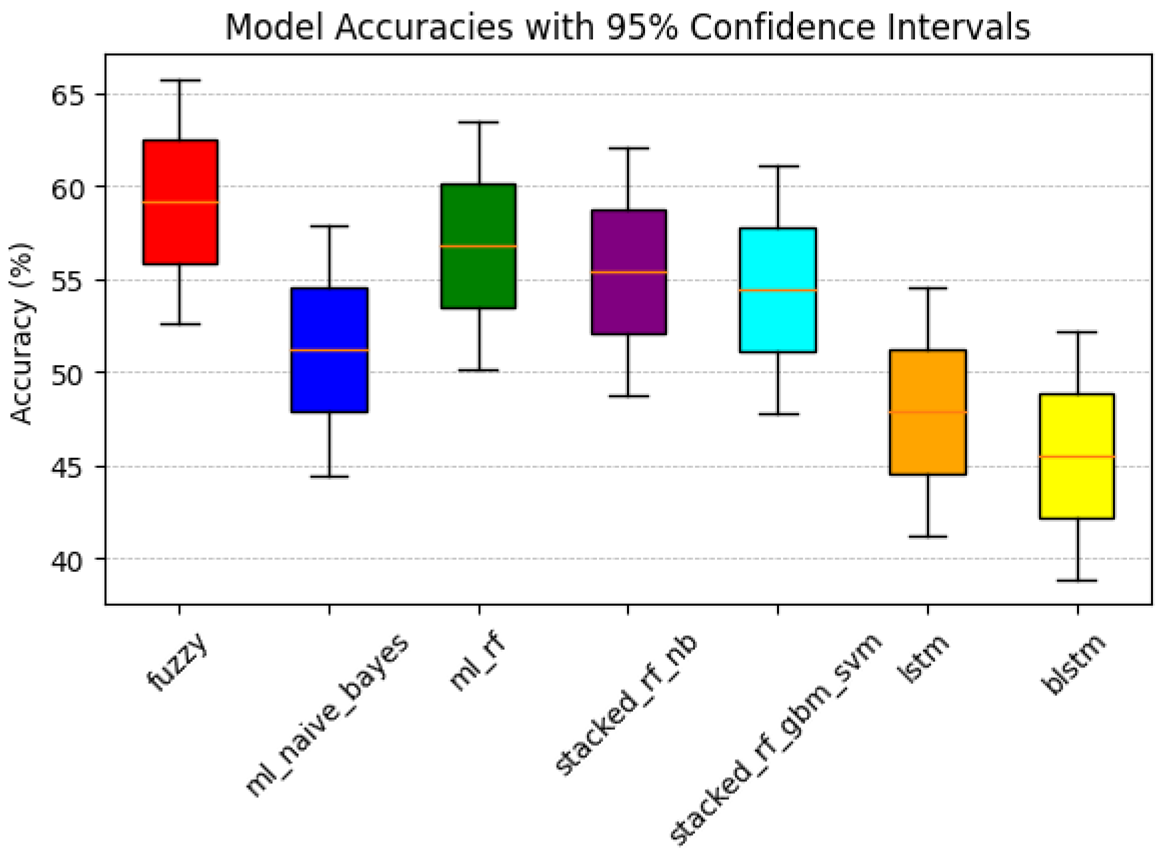
```
import matplotlib.pyplot as plt

## models
models = ['fuzzy', 'ml_naive_bayes', 'ml_rf', 'stacked_rf_nb', 'stacked_rf_gbm_sv

# mean accuracies
accuracies = [59.15, 51.17, 56.81, 55.40, 54.46, 47.89, 45.54]

## lower bounds of the CIs
lower_bounds = [52.56, 44.47, 50.17, 48.74, 47.79, 41.22, 38.87]

## upper bounds of the CIs
upper_bounds = [65.74, 57.87, 63.45, 62.06, 61.13, 54.56, 52.21]

## errors (distance from mean accuracy to the boundaries of CI)
errors = [(accuracies[i] - lower_bounds[i], upper_bounds[i] - accuracies[i]) for

colors = ['red', 'green', 'blue', 'cyan', 'magenta', 'yellow', 'black']

plt.figure(figsize=(6,4.5))

for i, model in enumerate(models):
    plt.errorbar(model, accuracies[i], yerr=[[errors[i][0]], [errors[i][1]]], fmt

plt.ylabel('Accuracy (%)')
plt.title('Model Accuracies with 95% Confidence Intervals')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

```
<ipython-input-2-00c90593efd8>:23: UserWarning: marker is redundantly defined
  plt.errorbar(model, accuracies[i], yerr=[[errors[i][0]], [errors[i][1]]], fı
```

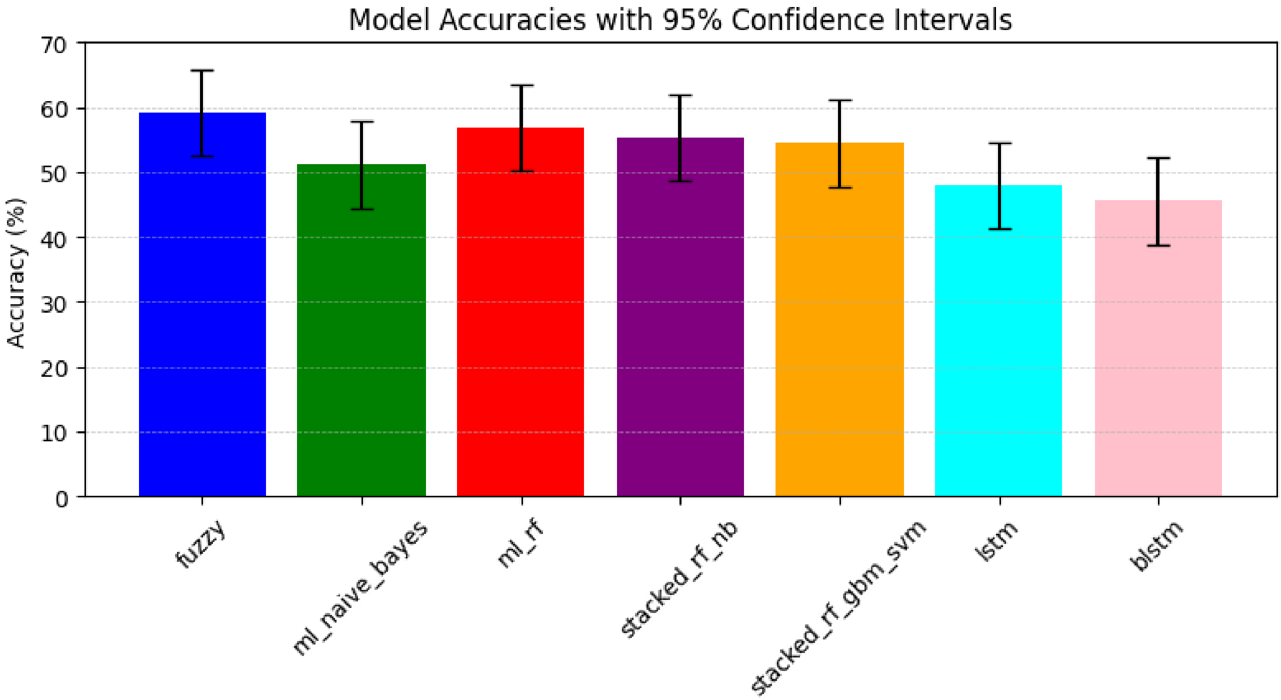
```
import matplotlib.pyplot as plt
import numpy as np

# Models
models = ['fuzzy', 'ml_naive_bayes', 'ml_rf', 'stacked_rf_nb', 'stacked_rf_gbm_sv

# Convert accuracies and confidence intervals to box plot data format
box_data = [[accuracies[i] - errors[i][0], accuracies[i], accuracies[i] + errors[

# Plotting
plt.figure(figsize=(6, 4.5))
plt.boxplot(box_data, vert=True, patch_artist=True)
plt.ylabel('Accuracy (%)')
plt.title('Model Accuracies with 95% Confidence Intervals')
plt.xticks(np.arange(1, len(models)+1), models, rotation=45)
plt.grid(axis='y', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```



```
import matplotlib.pyplot as plt
import numpy as np

# Models
models = ['fuzzy', 'ml_naive_bayes', 'ml_rf', 'stacked_rf_nb', 'stacked_rf_gbm_sv

# Convert accuracies and confidence intervals to box plot data format
box_data = [[accuracies[i] - errors[i][0], accuracies[i], accuracies[i] + errors[

# Colors
colors = ['red', 'blue', 'green', 'purple', 'cyan', 'orange', 'yellow']

# Plotting
fig, ax = plt.subplots(figsize=(6, 4.5))
bp = ax.boxplot(box_data, vert=True, patch_artist=True)

# Setting colors to each box
for patch, color in zip(bp['boxes'], colors):
    patch.set_facecolor(color)

plt.ylabel('Accuracy (%)')
plt.title('Model Accuracies with 95% Confidence Intervals')
plt.xticks(np.arange(1, len(models)+1), models, rotation=45)
plt.grid(axis='y', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```



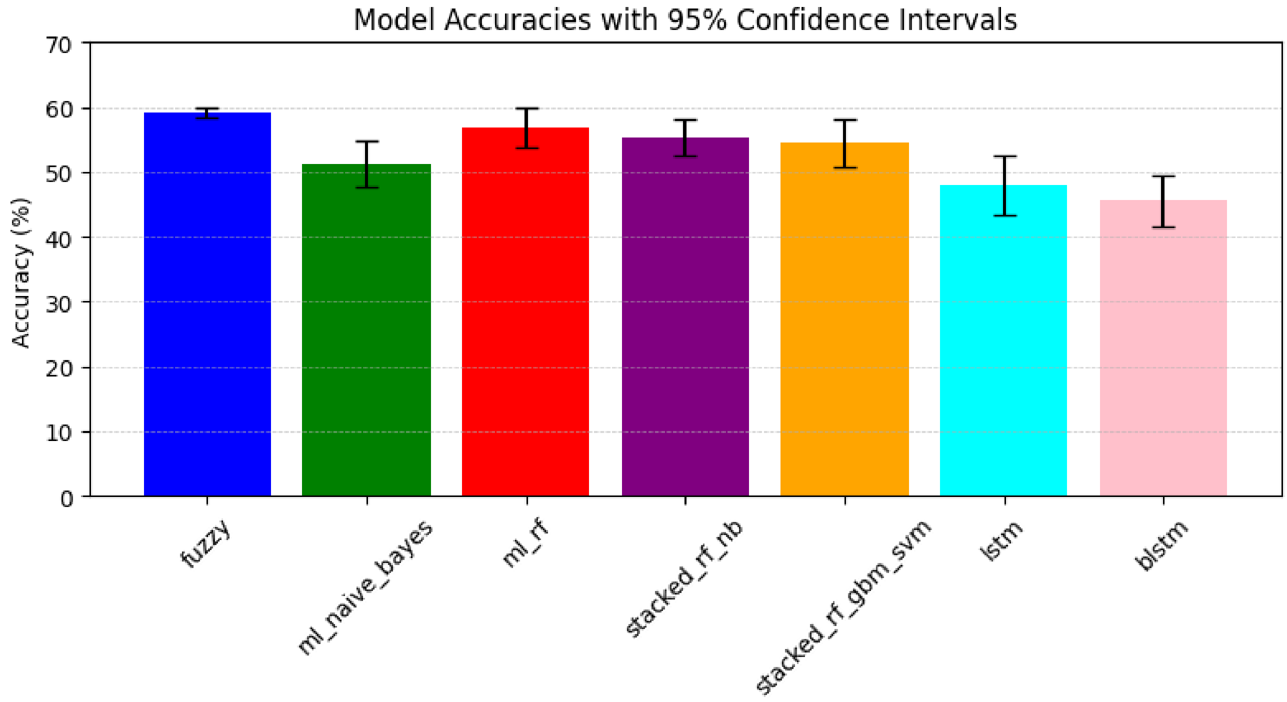
```
import matplotlib.pyplot as plt

### Models
models = ['fuzzy', 'ml_naive_bayes', 'ml_rf', 'stacked_rf_nb', 'stacked_rf_gbm_sv

### accuracies mean
accuracies = [59.15, 51.17, 56.81, 55.40, 54.46, 47.89, 45.54]

# Calculate errors for bar chart (distance from mean accuracy to the upper bound
errors = [upper_bounds[i] - accuracies[i] for i in range(len(accuracies))]

# Plotting
plt.figure(figsize=(8,4.5))
bars = plt.bar(models, accuracies, yerr=errors, capsize=5, color=['blue', 'green'


plt.ylabel('Accuracy (%)')
plt.title('Model Accuracies with 95% Confidence Intervals')
plt.xticks(rotation=45)
plt.ylim(0, 70)
plt.grid(axis='y', linestyle='--', linewidth=0.5, alpha=0.7)
plt.tight_layout()

plt.show()
```

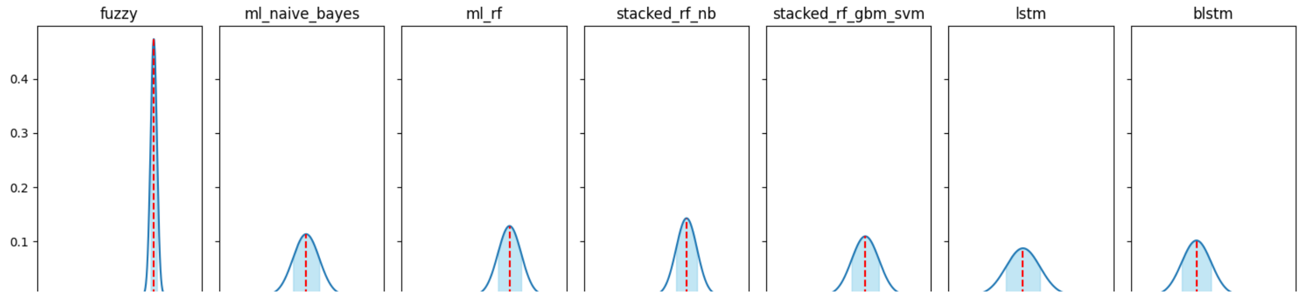


```
from scipy.stats import kruskal

models = {
    "fuzzy": [59.15],
    "ml_naive_bayes": [51.17],
    "ml_rf": [56.81],
    "stacked_rf_nb": [55.40],
    "stacked_rf_gbm_svm": [54.46],
    "lstm": [47.89],
    "blstm": [45.54]
}

h_statistic, p_value = kruskal(*models.values())

print(f"H-statistic: {h_statistic}")
print(f"P-value: {p_value}")
```

```
H-statistic: 6.0
P-value: 0.42319008112684364
```

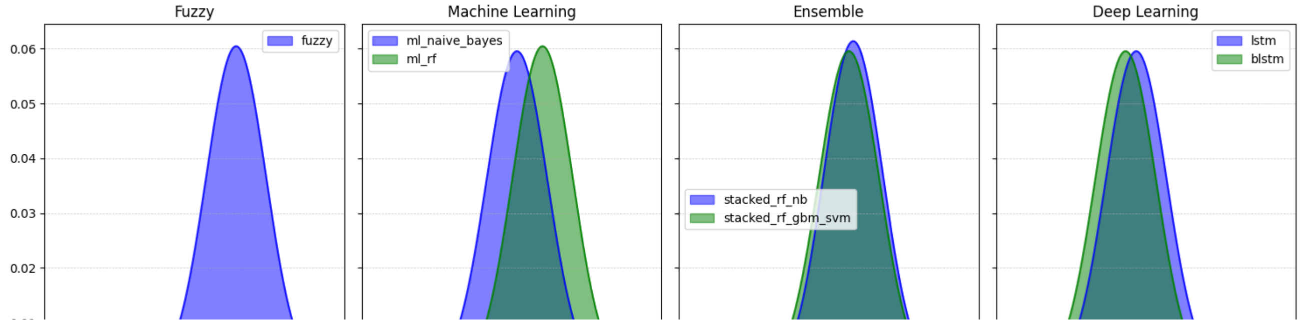
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

### Models
models = ['fuzzy', 'ml_naive_bayes', 'ml_rf', 'stacked_rf_nb', 'stacked_rf_gbm_sv

### accuracies mean
accuracies = [59.15, 51.17, 56.81, 55.40, 54.46, 47.89, 45.54]


upper_bounds = [accuracy + np.random.uniform(0, 5) for accuracy in accuracies]
errors = [upper_bounds[i] - accuracies[i] for i in range(len(accuracies))]

### Plotting bar chart with errors
plt.figure(figsize=(8,4.5))
bars = plt.bar(models, accuracies, yerr=errors, capsize=5, color=['blue', 'green'
plt.ylabel('Accuracy (%)')
plt.title('Model Accuracies with 95% Confidence Intervals')
plt.xticks(rotation=45)
plt.ylim(0, 70)
plt.grid(axis='y', linestyle='--', linewidth=0.5, alpha=0.7)
plt.tight_layout()
plt.show()

### Plotting individual bell curves
fig, axs = plt.subplots(1, len(models), figsize=(15,4), sharey=True)
x = np.linspace(30, 70, 1000)
for i, model in enumerate(models):
    mu = accuracies[i]
    sigma = errors[i]  # Using the error as standard deviation
    y = stats.norm.pdf(x, mu, sigma)
    axs[i].plot(x, y)
    axs[i].set_title(model)
    axs[i].vlines(mu, 0, max(y), colors='red', linestyles='dashed')
    axs[i].fill_between(x, y, where=((x > mu - sigma) & (x < mu + sigma)), color=

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


Model Accuracies with 95% Confidence Intervals
Accuracy (%)
0
10
20
30
40
50
60
70
fuzzy
ml_naive_bayes
ml_rf
stacked_rf_nb
stacked_rf_gbm_svm
lstm
blstm

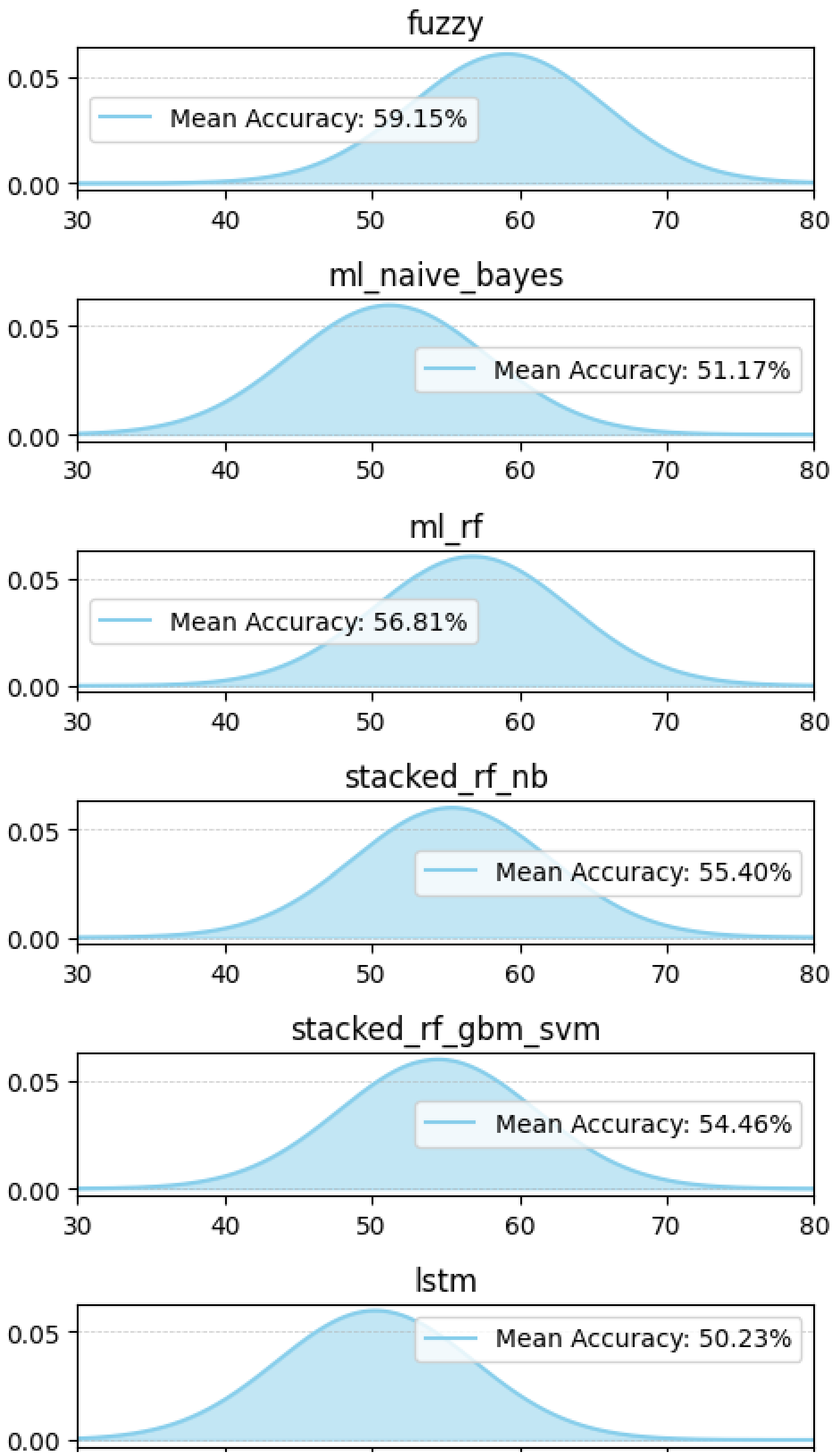

fuzzy
ml_naive_bayes
ml_rf
stacked_rf_nb
stacked_rf_gbm_svm
lstm
blstm
0.1
0.2
0.3
0.4

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

# Models and their categories
categories = ['Fuzzy', 'Machine Learning', 'Ensemble', 'Deep Learning']
models = ['fuzzy', 'ml_naive_bayes', 'ml_rf', 'stacked_rf_nb', 'stacked_rf_gbm_sv
category_map = ['Fuzzy', 'Machine Learning', 'Machine Learning', 'Ensemble', 'Ens

# accuracies mean and assumed standard deviations
accuracies = [59.15, 51.17, 56.81, 55.40, 54.46, 47.89, 45.54]
std_devs = [error for error in errors]  # Assuming error is 1 std deviation
x = np.linspace(20, 80, 1000)  # 20-80% accuracy range with 1000 points

fig, axs = plt.subplots(1, len(categories), figsize=(15, 5), sharey=True)

for cat in categories:
    relevant_models = [model for model, map_cat in zip(models, category_map) if m
    relevant_accuracies = [accuracy for accuracy, map_cat in zip(accuracies, cate
    relevant_std_devs = [std for std, map_cat in zip(std_devs, category_map) if m

    for model, accuracy, std_dev in zip(relevant_models, relevant_accuracies, rel
        y = stats.norm.pdf(x, accuracy, std_dev)
        axs[categories.index(cat)].plot(x, y, label=model)

    axs[categories.index(cat)].set_title(cat)
    axs[categories.index(cat)].legend()
    axs[categories.index(cat)].grid(axis='y', linestyle='--', linewidth=0.5, alph

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

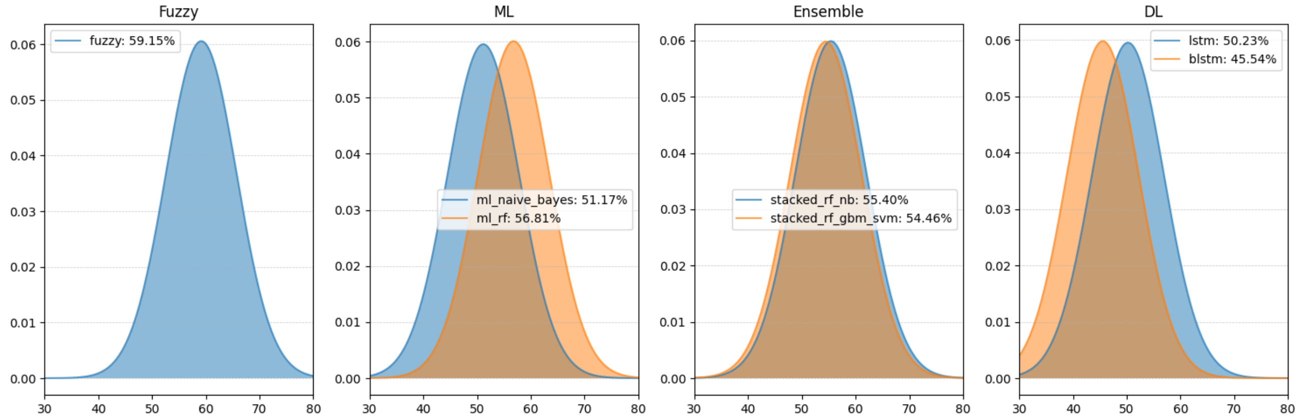

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

# Models and their categories
categories = ['Fuzzy', 'Machine Learning', 'Ensemble', 'Deep Learning']
models = ['fuzzy', 'ml_naive_bayes', 'ml_rf', 'stacked_rf_nb', 'stacked_rf_gbm_sv
category_map = ['Fuzzy', 'Machine Learning', 'Machine Learning', 'Ensemble', 'Ens

## error=[upper boundary-mean accuracy]
## # Upper bounds of the CIs [65.74, 57.87, 63.45, 62.06, 61.13, 54.58, 52.21]
# accuracies mean and assumed standard deviations
accuracies = [59.15, 51.17, 56.81, 55.40, 54.46, 47.89, 45.54]
errors = [6.6, 6.7, 6.6, 6.5, 6.7, 6.7, 6.7]  ## error=[upper bounds- accuracies]
std_devs = [error for error in errors]
x = np.linspace(20, 80, 1000)

fig, axs = plt.subplots(1, len(categories), figsize=(15, 5), sharey=True)

colors = ['blue', 'green', 'red', 'purple', 'orange', 'cyan', 'pink']

for cat in categories:
    relevant_models = [model for model, map_cat in zip(models, category_map) if m
    relevant_accuracies = [accuracy for accuracy, map_cat in zip(accuracies, cate
    relevant_std_devs = [std for std, map_cat in zip(std_devs, category_map) if m

    for model, accuracy, std_dev, color in zip(relevant_models, relevant_accuraci
        y = stats.norm.pdf(x, accuracy, std_dev)
        axs[categories.index(cat)].fill_between(x, y, color=color, alpha=0.5, lab
        axs[categories.index(cat)].plot(x, y, color=color, alpha=0.8)

    axs[categories.index(cat)].set_title(cat)
    axs[categories.index(cat)].legend()
    axs[categories.index(cat)].grid(axis='y', linestyle='--', linewidth=0.5, alph

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




```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

models = ['fuzzy', 'ml_naive_bayes', 'ml_rf', 'stacked_rf_nb', 'stacked_rf_gbm_sv

### Mean accuracies
accuracies = [59.15, 51.17, 56.81, 55.40, 54.46, 50.23, 45.54]

### Lower bounds of the CIs
lower_bounds = [52.56, 44.47, 50.17, 48.74, 47.79, 43.53, 38.87]

### Upper bounds of the CIs
upper_bounds = [65.74, 57.87, 63.45, 62.06, 61.13, 56.93, 52.21]

# Compute standard deviations from the distance between means and bounds
std_devs = [(upper - lower) / 2 for upper, lower in zip(upper_bounds, lower_bound

## subplots for each model
fig, axs = plt.subplots(len(models), 1, figsize=(5, 10))
x = np.linspace(30, 80, 1000)  # Define a range of accuracies

for i, model in enumerate(models):
    ## bell curve
    axs[i].plot(x, norm.pdf(x, accuracies[i], std_devs[i]), color='skyblue', labe
    axs[i].fill_between(x, norm.pdf(x, accuracies[i], std_devs[i]), color='skyblu

    axs[i].set_title(model)
    axs[i].legend()
    axs[i].set_xlim(30, 80)
    axs[i].grid(axis='y', linestyle='--', linewidth=0.5, alpha=0.7)

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

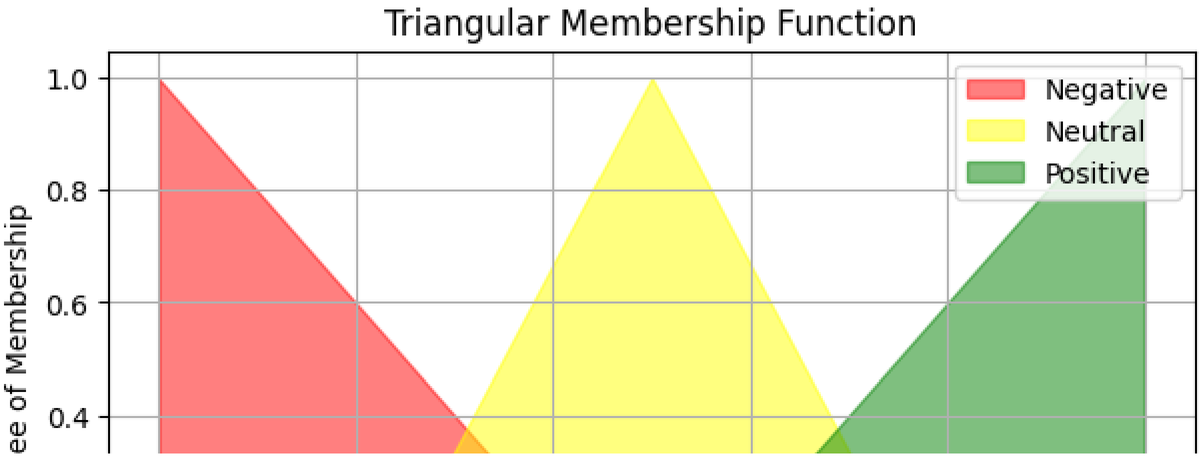

fuzzy
Mean Accuracy: 59.15%
ml_naive_bayes
Mean Accuracy: 51.17%
ml_rf
Mean Accuracy: 56.81%
stacked_rf_nb
Mean Accuracy: 55.40%
stacked_rf_gbm_svm
Mean Accuracy: 54.46%
lstm
Mean Accuracy: 50.23%
0.05
0.00
30
40
50
60
70
80

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Models and categories
models = {
    'Fuzzy': ['fuzzy'],
    'ML': ['ml_naive_bayes', 'ml_rf'],
    'Ensemble': ['stacked_rf_nb', 'stacked_rf_gbm_svm'],
    'DL': ['lstm', 'blstm']
}

# Mean accuracies
accuracies = [59.15, 51.17, 56.81, 55.40, 54.46, 50.23, 45.54]

# Lower bounds of the CIs
lower_bounds = [52.56, 44.47, 50.17, 48.74, 47.79, 43.53, 38.87]

# Upper bounds of the CIs
upper_bounds = [65.74, 57.87, 63.45, 62.06, 61.13, 56.93, 52.21]

# Compute standard deviations from the distance between means and bounds
std_devs = [(upper - lower) / 2 for upper, lower in zip(upper_bounds, lower_bound

# Create subplots for each category
fig, axs = plt.subplots(1, len(models), figsize=(15, 5))
x = np.linspace(30, 80, 1000)  # Define a range of accuracies

model_idx = 0  # Index to iterate over accuracies and std_devs
for ax, category in zip(axs, models.keys()):
    for model in models[category]:
        ax.plot(x, norm.pdf(x, accuracies[model_idx], std_devs[model_idx]), label
        ax.fill_between(x, norm.pdf(x, accuracies[model_idx], std_devs[model_idx]
        model_idx += 1
    ax.set_title(category)
    ax.legend()
    ax.set_xlim(30, 80)
    ax.grid(axis='y', linestyle='--', linewidth=0.5, alpha=0.7)

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




```
import matplotlib.pyplot as plt
import numpy as np

###triangular membership function
def trimf(x, a, b, c):
    return np.maximum(np.minimum((x - a) / (b - a), (c - x) / (c - b)), 0)

### the universe of discourse
x = np.linspace(0, 1, 1000)

##parameters for the triangular functions
params_negative = [0, 0, 0.5]
params_neutral = [0.2, 0.5, 0.8]
params_positive = [0.5, 1, 1]

##membership values
y_negative = trimf(x, *params_negative)
y_neutral = trimf(x, *params_neutral)
y_positive = trimf(x, *params_positive)

plt.figure(figsize=(7, 4))
plt.fill_between(x, y_negative, color='red', alpha=0.5, label='Negative')
plt.fill_between(x, y_neutral, color='yellow', alpha=0.5, label='Neutral')
plt.fill_between(x, y_positive, color='green', alpha=0.5, label='Positive')
plt.title('Triangular Membership Function')
plt.xlabel('Sentiment Score')
plt.ylabel('Degree of Membership')
plt.legend()
plt.grid(True)
plt.show()
```

```
<ipython-input-3-3de667961124>:6: RuntimeWarning: divide by zero encountered 
  return np.maximum(np.minimum((x - a) / (b - a), (c - x) / (c - b)), 0)
<ipython-input-3-3de667961124>:6: RuntimeWarning: invalid value encountered i
  return np.maximum(np.minimum((x - a) / (b - a), (c - x) / (c - b)), 0)
```




```
## model accuracy plot

import matplotlib.pyplot as plt

models = ["Fuzzy", "Naive Bayes", "Random Forest", "Stacked RF & NB", "Stacked RF
accuracies = [59.15, 51.17, 56.81, 55.40, 54.46, 47.89, 45.54]

plt.figure(figsize=(6, 4))
plt.bar(models, accuracies, color=['blue', 'green', 'red', 'cyan', 'magenta', 'ye
plt.xticks(rotation=45, ha="right", fontsize=10)
```