NLP Transformer-based Models used for Sentiment Analysis

1. **BERT(Bidirectional Encoder Representations from Transformers)**
2. **RoBERTa (Robustly Optimized BERT Approach)**
3. **DistilBERT**
4. **ALBERT**
5. **XLNet**

Kaggle Notebook Link: https://lnkd.in/gGfDeA_d

Prepared by: Syed Afroz Ali (Kaggle Grandmaster)

```
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style='whitegrid')

train = pd.read_csv('/kaggle/input/sentiment-analysis-dataset/training.csv',header=None)
validation = pd.read_csv('/kaggle/input/sentiment-analysis-dataset/validation.csv',header=None)

train.columns=['Tweet ID','Entity','Sentiment','Tweet Content']
validation.columns=['Tweet ID','Entity','Sentiment','Tweet Content']

print("Training DataSet: \n")
train = train.sample(5000)
display(train.head())
```

	Tweet ID	Entity	Sentiment	Tweet Content
67154	7099	johnson&johnson	Neutral	All of this was perfectly legal: Johnson & Johnson used super-ping to produce drugs for the popular opioid pillows. washingtonpost.com / graphics / 2020 /...
10204	12957	Xbox(Xseries)	Irrelevant	2016 The 5 latest discountgadgets. phone co. uk Consumer and Electronics Daily! via paper. li / discountgadget ... A Thanks Much to @VandijConsult @z4mp1 @CarlosEduardoCD
22068	4177	CS-GO	Positive	To all the people who want to play VALORANT and are saying they are gonna pursue it professionally, . . go play 100 hours of CSGO, if you still like the game then I think it would be a good game f...
58373	3208	Facebook	Irrelevant	teenage boy...more fun right?
47536	5755	HomeDepot	Neutral	Home Depot Workers Find Another Cutest One Little Human Family Inside Mulch Display. u ... g. theanimalrescuesite. per greatergood. com / im - home - runs depot - story ...

```python
print("Validation DataSet: \n")
display(validation.head())
```

	Tweet ID	Entity	Sentiment	Tweet Content
0	3364	Facebook	Irrelevant	I mentioned on Facebook that I was struggling for motivation to go for a run the other day, which has been translated by Tom's great auntie as 'Hayley can't get out of bed' and told to his grandma...
1	352	Amazon	Neutral	BBC News - Amazon boss Jeff Bezos rejects claims company acted like a 'drug dealer' bbc.co.uk/news/av/busine…
2	8312	Microsoft	Negative	@Microsoft Why do I pay for WORD when it functions so poorly on my @SamsungUS Chromebook? 🙄
3	4371	CS-GO	Negative	CSGO matchmaking is so full of closet hacking, it's a truly awful game.
4	4433	Google	Neutral	Now the President is slapping Americans in the face that he really did commit an unlawful act after his acquittal! From Discover on Google vanityfair.com/news/2020/02/t…

```python
train = train.dropna(subset=['Tweet Content'])

display(train.isnull().sum())
print("*****"* 5)
display(validation.isnull().sum())
```

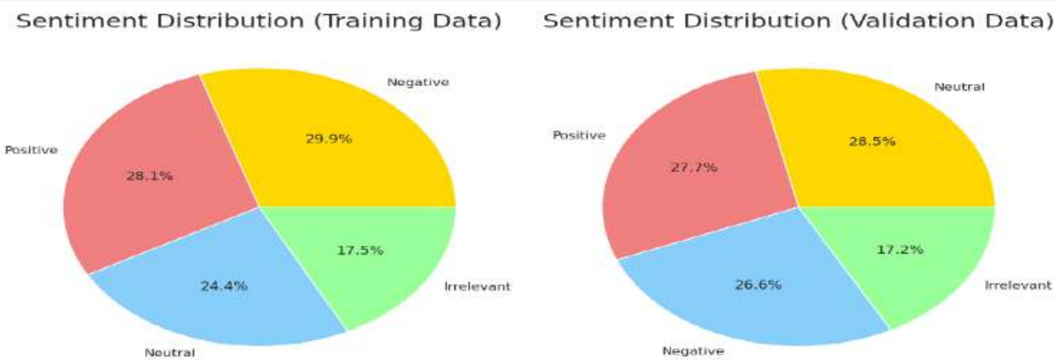
```
Tweet ID         0
Entity           0
Sentiment        0
Tweet Content    0
dtype: int64


*************************


Tweet ID         0
Entity           0
Sentiment        0
Tweet Content    0
dtype: int64
```

```python
duplicates = train[train.duplicated(subset=['Entity', 'Sentiment', 'Tweet Content'], keep=False)]
train = train.drop_duplicates(subset=['Entity', 'Sentiment', 'Tweet Content'], keep='first')

duplicates = validation[validation.duplicated(subset=['Entity', 'Sentiment', 'Tweet Content'], keep=False)]
validation = validation.drop_duplicates(subset=['Entity', 'Sentiment', 'Tweet Content'], keep='first')
```

```
# Calculate sentiment counts for train and validation data
sentiment_counts_train = train['Sentiment'].value_counts()
sentiment_counts_validation = validation['Sentiment'].value_counts()
combined_counts = pd.concat([sentiment_counts_train, sentiment_counts_validation], axis=1)
combined_counts.fillna(0, inplace=True)
combined_counts.columns = ['Test Data', 'Validation Data']  combined_counts
```

	Test Data	Validation Data
Sentiment		
Negative	1481	266
Positive	1392	277
Neutral	1205	285
Irrelevant	868	172

```
sentiment_counts_train = train['Sentiment'].value_counts()
sentiment_counts_validation = validation['Sentiment'].value_counts()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
# Create pie chart for training data
ax1.pie(sentiment_counts_train, labels=sentiment_counts_train.index, autopct='%1.1f%%', colors=['gold', 'lightcoral', 'lightskyblue','#99FF99'])
ax1.set_title('Sentiment Distribution (Training Data)', fontsize=20)

ax2.pie(sentiment_counts_validation, labels=sentiment_counts_validation.index, autopct='%1.1f%%', colors=['gold', 'lightcoral', 'lightskyblue','#99FF99'])
ax2.set_title('Sentiment Distribution (Validation Data)', fontsize=20)
plt.tight_layout()
plt.show()
```

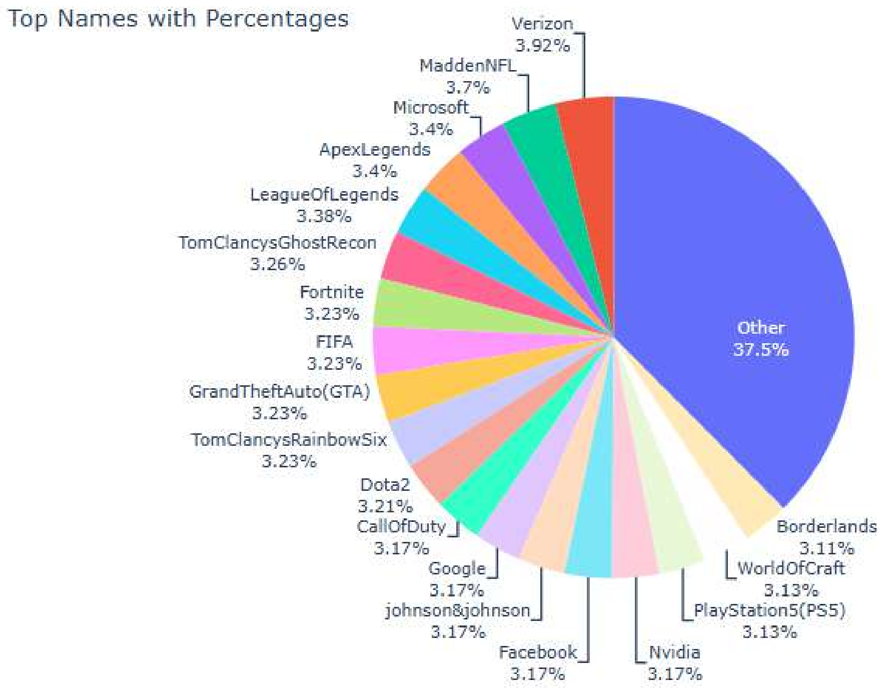


```python
# Calculate the value counts of 'Entity'
entity_counts = train['Entity'].value_counts()
top_names = entity_counts.head(19)

other_count = entity_counts[19:].sum()
top_names['Other'] = other_count
top_names.to_frame()
```

	count
Entity	
Verizon	194
MaddenNFL	183
Microsoft	168
ApexLegends	168
LeagueOfLegends	167
TomClancysGhostRecon	161
Fortnite	160
FIFA	160
GrandTheftAuto(GTA)	160
TomClancysRainbowSix	160
Dota2	159
CallOfDuty	157
Google	157
johnson&johnson	157
Facebook	157
Nvidia	157
PlayStation5(PS5)	155
WorldOfCraft	155
Borderlands	154
Other	1857

```python
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio

percentages = (top_names / top_names.sum()) * 100

fig = go.Figure(data=[go.Pie(
    labels=percentages.index,
    values=percentages,
    textinfo='label+percent',
    insidetextorientation='radial'
)])
```

```
fig.update_layout(
    title_text='Top Names with Percentages',
    showlegend=False
)

fig.show()
```




```
from tensorflow.keras.layers import Input, Dropout, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.initializers import TruncatedNormal
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.metrics import CategoricalAccuracy
from tensorflow.keras.utils import to_categorical

import pandas as pd
from sklearn.model_selection import train_test_split
import pandas as pd
import plotly.graph_objects as go

# Assuming you've already run the data preprocessing steps
data = train[['Tweet Content', 'Sentiment']]
```

```python
# Set your model output as categorical and save in new label col
data['Sentiment_label'] = pd.Categorical(data['Sentiment'])

# Transform your output to numeric
data['Sentiment'] = data['Sentiment_label'].cat.codes

# Use the entire training data as data_train
data_train = data

# Use validation data as data_test
data_test = validation[['Tweet Content', 'Sentiment']]
data_test['Sentiment_label'] = pd.Categorical(data_test['Sentiment'])
data_test['Sentiment'] = data_test['Sentiment_label'].cat.codes

# Create a colorful table using Plotly
fig = go.Figure(data=[go.Table(
    header=dict(
        values=list(data_train.columns),
        fill_color='paleturquoise',
        align='left',
        font=dict(color='black', size=12)
    ),
    cells=dict(
        values=[data_train[k].tolist()[:10] for k in data_train.columns],
        fill_color=[
            'lightcyan',  # Tweet Content
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative'
              else 'lightyellow' if s == 'Neutral' else 'lightgray' for s in data_train['Sentiment_label'][:10]],  # Sentiment
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative'
              else 'lightyellow' if s == 'Neutral' else 'lightgray' for s in data_train['Sentiment_label'][:10]],  # Sentiment_label
            'lavender'  # Sentiment (numeric)
        ],
        align='left',
        font=dict(color='black', size=11)
    ))
])

# Update the layout
fig.update_layout(
    title='First 10 Rows of Training Data',
    width=1000,
    height=500,
)

fig.show()
```

First 10 Rows of Training Data

Tweet Content	Sentiment	Sentiment_label
All of this was perfectly legal: Johnson & Johnson used super-ping to produce drugs for the popular opioid pillows. washingtonpost.com / graphics / 2020 /...	2	Neutral
2016 The 5 latest discountgadgets. phone co. uk Consumer and Electronics Daily! via paper. li / discountgadget ... A Thanks Much to @VandijConsult @z4mp1 @CarlosEduardoCD	0	Irrelevant
To all the people who want to play VALORANT and are saying they are gonna pursue it professionally, . . go play 100 hours of CSGO, if you still like the game then I think it would be a good game for you, if you are bored out of your mind I would not recommend pursuing it.	3	Positive
teenage boy...more fun right?	0	Irrelevant
[illegible]	2	Neutral

```
import plotly.graph_objects as go

# Create a colorful table using Plotly for the test data
fig = go.Figure(data=[go.Table(
    header=dict(
        values=list(data_test.columns),
        fill_color='paleturquoise',
        align='left',
        font=dict(color='black', size=12)
    ),
    cells=dict(
        values=[data_test[k].tolist()[:5] for k in data_test.columns],  # Show first 5 rows
        fill_color=[
            'lightcyan',  # Tweet Content
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative'
             else 'lightyellow' if s == 'Neutral' else 'lightgray' for s in data_test['Sentiment_label'][:5]],  # Sentiment
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative'
             else 'lightyellow' if s == 'Neutral' else 'lightgray' for s in data_test['Sentiment_label'][:5]],  # Sentiment_label
            'lavender'  # Sentiment (numeric)
        ],
        align='left',
        font=dict(color='black', size=11)
    ))
])

fig.update_layout(
    title='First 5 Rows of Test Data',
    width=1000,
    height=500,
)
fig.show()
```

First 5 Rows of Test Data

Tweet Content	Sentiment	Sentiment_label
I mentioned on Facebook that I was struggling for motivation to go for a run the other day, which has been translated by Tom's great auntie as 'Hayley can't get out of bed' and told to his grandma, who now thinks I'm a lazy, terrible person	0	Irrelevant
BBC News - Amazon boss Jeff Bezos rejects claims company acted like a 'drug dealer' bbc.co.uk/news/av/busine…	2	Neutral
@Microsoft Why do I pay for WORD when it functions so poorly on my @SamsungUS Chromebook?	1	Negative
CSGO matchmaking is so full of closet hacking, it's a truly awful game.	1	Negative
Now the President is slapping Americans in the	2	Neutral

1. BERT (Bidirectional Encoder Representations from Transformers)

BERT is a groundbreaking language model that has significantly advanced the field of Natural Language Processing (NLP).
It stands for Bidirectional Encoder Representations from Transformers.

Key Concepts

- **Bidirectional:** Unlike previous models that processed text sequentially (left to right or right to left), BERT considers the entire context of a word, both preceding and following it. This enables a deeper understanding of language nuances.
- **Encoder:** BERT focuses on understanding the input text rather than generating new text. It extracts meaningful representations from the input sequence.
- **Transformers:** The underlying architecture of BERT is based on the Transformer model, known for its efficiency in handling long sequences and capturing dependencies between words.

How BERT Works

- **Pre-training:** BERT is initially trained on a massive amount of text data (like Wikipedia and BooksCorpus) using two unsupervised tasks:
  - **Masked Language Modeling (MLM):** Randomly masks some words in the input and trains the model to predict the masked words based on the context of surrounding words.
  - **Next Sentence Prediction (NSP):** Trains the model to predict whether two given sentences are consecutive in the original document.
- **Fine-tuning:** After pre-training, BERT can be adapted to specific NLP tasks with minimal additional training. This is achieved by adding a task-specific output layer to the pre-trained model.

Advantages of BERT

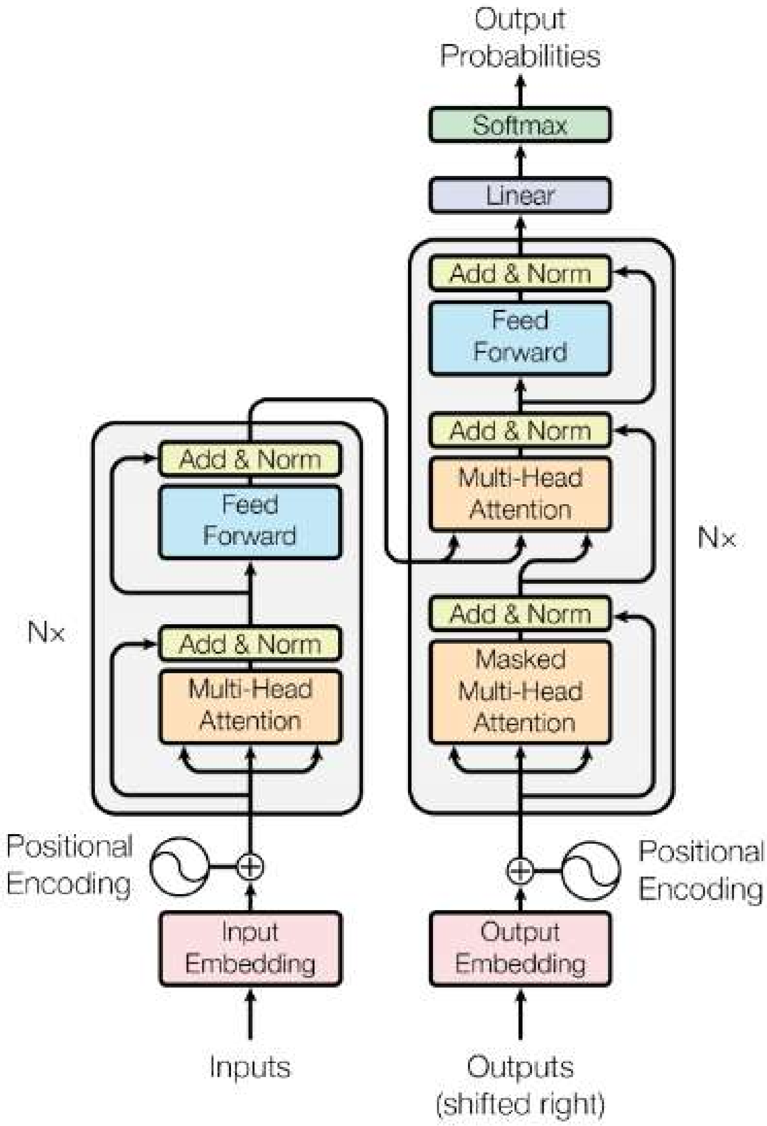
- **Strong performance:** BERT has achieved state-of-the-art results on a wide range of NLP tasks, including question answering, text classification, named entity recognition, and more.

- **Efficiency:** Fine-tuning BERT for new tasks is relatively quick and requires less data compared to training models from scratch.
- **Versatility:** BERT can be applied to various NLP problems with minimal modifications.

Applications of BERT

- **Search engines:** Improving search relevance and understanding user queries.
- **Chatbots:** Enhancing natural language understanding and generating more human-like responses.
- **Sentiment analysis:** Accurately determining the sentiment expressed in text.
- **Machine translation:** Improving the quality of translated text.
- **Text summarization:** Generating concise summaries of lengthy documents.

In essence, BERT is a powerful language model that has revolutionized NLP by capturing the bidirectional context of words and enabling efficient transfer learning for various tasks.




```
%%time

import pandas as pd
import torch
```

```
from torch.utils.data import Dataset, DataLoader
from transformers import BertTokenizer, BertForSequenceClassification, Ada
mW
from sklearn.metrics import accuracy_score, classification_report

# Preprocess the dataF
def preprocess_data(df):
    df['label'] = df['Sentiment_label'].map({'Positive': 2, 'Negative': 0, 'Neutral': 1
, 'Irrelevant': 3})
    return df['Tweet Content'].tolist(), df['label'].tolist()

train_texts, train_labels = preprocess_data(data_train)
test_texts, test_labels = preprocess_data(data_test)

# Create a custom dataset
class SentimentDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = str(self.texts[idx])
        label = self.labels[idx]

        encoding = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt',
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }

# Initialize tokenizer and create datasets
```

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
train_dataset = SentimentDataset(train_texts, train_labels, tokenizer)
test_dataset = SentimentDataset(test_texts, test_labels, tokenizer)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Initialize the model_BERT
model_BERT = BertForSequenceClassification.from_pretrained('bert-base-unc
ased', num_labels=4)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model_BERT.to(device)

# Set up optimizer
optimizer = AdamW(model_BERT.parameters(), lr=2e-5)

# Training loop
num_epochs = 3

for epoch in range(num_epochs):
    model_BERT.train()
    for batch in train_loader:
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model_BERT(input_ids, attention_mask=attention_mask, labels
=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()

    # Evaluation on test set
    model_BERT.eval()
    test_preds = []
    test_true = []
    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels']
            outputs = model_BERT(input_ids, attention_mask=attention_mask)
            preds = torch.argmax(outputs.logits, dim=1).cpu().numpy()
            test_preds.extend(preds)
            test_true.extend(labels.numpy())
```

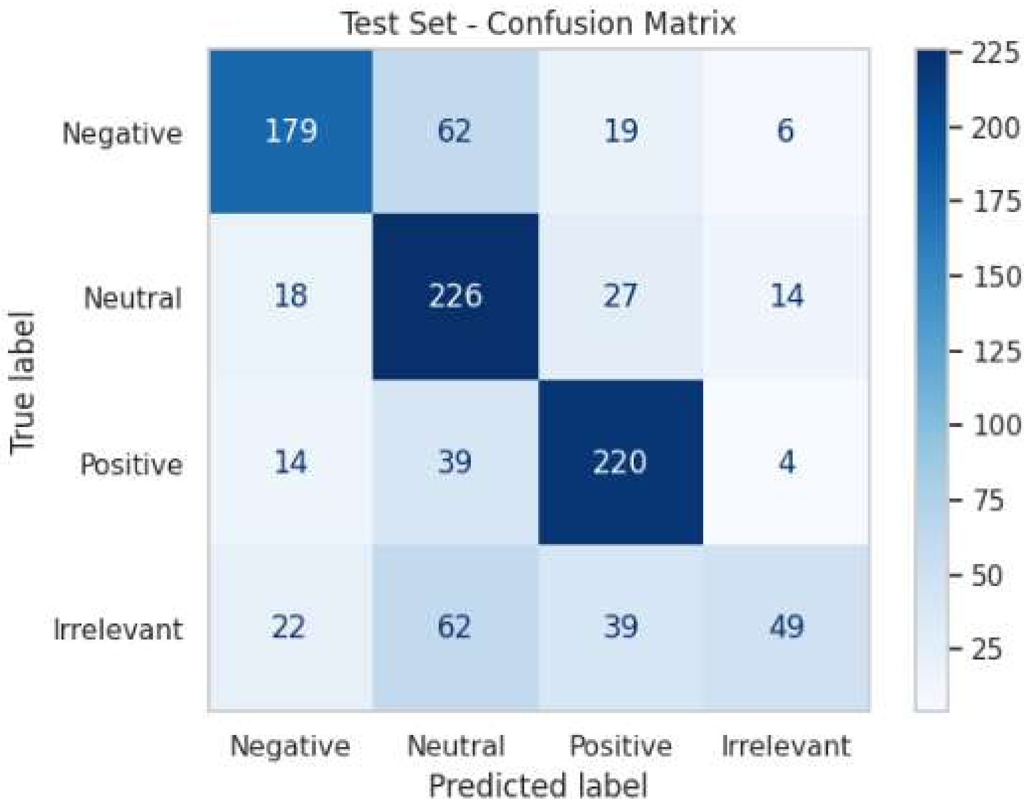
```
    accuracy = accuracy_score(test_true, test_preds)
    print(f'Epoch {epoch + 1}/{num_epochs}, Test Accuracy: {accuracy:.4f}')


# Save the model_BERT
torch.save(model_BERT.state_dict(), 'sentiment_model_BERT.pth')
```

```
# Final evaluation
print(classification_report(test_true, test_preds, target_names=['Negative', 'Neutral', 'Positive', 'Irrelevant']))
```

	precision	recall	f1-score	support
Negative	0.77	0.67	0.72	266
Neutral	0.58	0.79	0.67	285
Positive	0.72	0.79	0.76	277
Irrelevant	0.67	0.28	0.40	172
accuracy			0.67	1000
macro avg	0.69	0.64	0.64	1000
weighted avg	0.69	0.67	0.66	1000

```
from sklearn.metrics import confusion_matrix

# Check if test_true labels need conversion (optional)
if not isinstance(test_true[0], str):  # If labels are not strings
    from sklearn.preprocessing import LabelEncoder
    encoder = LabelEncoder()
    test_true_encoded = encoder.fit_transform(test_true)  # Encode labels
    labels = [0, 1, 2, 3]  # Numerical labels
else:
    test_true_encoded = test_true
    labels = ['Negative', 'Neutral', 'Positive', 'Irrelevant']  # String labels

# Calculate confusion matrix with consistent labels
confusion_matrix_BERT = confusion_matrix(test_true_encoded, test_preds, labels=labels)

print("Confusion matrix BERT \n")
confusion_matrix_BERT
```

```python
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
labels = ['Negative', 'Neutral', 'Positive', 'Irrelevant']  # String labels
test_display = ConfusionMatrixDisplay(confusion_matrix=confusion_
matrix_BERT, display_labels=labels)
test_display.plot(cmap='Blues')
plt.title("Test Set - Confusion Matrix")
plt.grid(False)
plt.tight_layout()
plt.show()
```



2. RoBERTa (Robustly Optimized BERT Pretraining Approach)

RoBERTa is an improved version of the BERT (Bidirectional Encoder Representations from Transformers) model. It builds upon BERT's architecture but incorporates several key modifications to enhance its performance.

Key Differences from BERT

- **Larger Training Dataset:** RoBERTa was trained on a significantly larger dataset compared to the original BERT, leading to a richer understanding of language.

- **Dynamic Masking:** Unlike BERT's static masking during pre-training, RoBERTa applies dynamic masking, where the masked tokens are changed multiple times for each training instance. This forces the model to learn more robust representations.
- **Longer Training:** RoBERTa undergoes a longer training process with larger batch sizes, allowing it to converge to a better optimum.
- **Removal of Next Sentence Prediction (NSP):** RoBERTa eliminates the NSP objective, focusing solely on Masked Language Modeling (MLM). This change simplifies the training process and improves performance on downstream tasks.
- **Increased Sequence Length:** RoBERTa can handle longer input sequences, enabling it to process more context-rich information.

Benefits of RoBERTa

- **Improved Performance:** RoBERTa consistently outperforms BERT on a wide range of NLP tasks, achieving state-of-the-art results.
- **Efficiency:** The modifications in RoBERTa lead to faster training and convergence.
- **Versatility:** Like BERT, RoBERTa can be fine-tuned for various NLP tasks, including text classification, question answering, and more.

Applications

- **Search Engines:** Enhancing search relevance and understanding user queries.
- **Chatbots:** Improving natural language understanding and generating more human-like responses.
- **Sentiment Analysis:** Accurately determining the sentiment expressed in text.
- **Machine Translation:** Enhancing the quality of translated text.
- **Text Summarization:** Generating concise summaries of lengthy documents.

In conclusion, RoBERTa is a powerful language model that builds upon the success of BERT by incorporating several refinements. Its improved performance and versatility make it a popular choice for various NLP applications.

```
%%time

import pandas as pd
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import BertTokenizer, BertForSequenceClassification, Ada
mW
from transformers import RobertaTokenizer, RobertaForSequenceClassificatio
n, AdamW
from sklearn.metrics import accuracy_score, classification_report

# Preprocess the data
def preprocess_data(df):
```

```
    df['label'] = df['Sentiment_label'].map({'Positive': 2, 'Negative': 0, 'Neutral': 1
, 'Irrelevant': 3})
    return df['Tweet Content'].tolist(), df['label'].tolist()

train_texts, train_labels = preprocess_data(data_train)
test_texts, test_labels = preprocess_data(data_test)

# Create a custom dataset
class SentimentDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = str(self.texts[idx])
        label = self.labels[idx]

        encoding = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt',
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }

# Initialize tokenizer and create datasets
#tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
tokenizer = RobertaTokenizer.from_pretrained('roberta-base')
train_dataset = SentimentDataset(train_texts, train_labels, tokenizer)
test_dataset = SentimentDataset(test_texts, test_labels, tokenizer)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
```

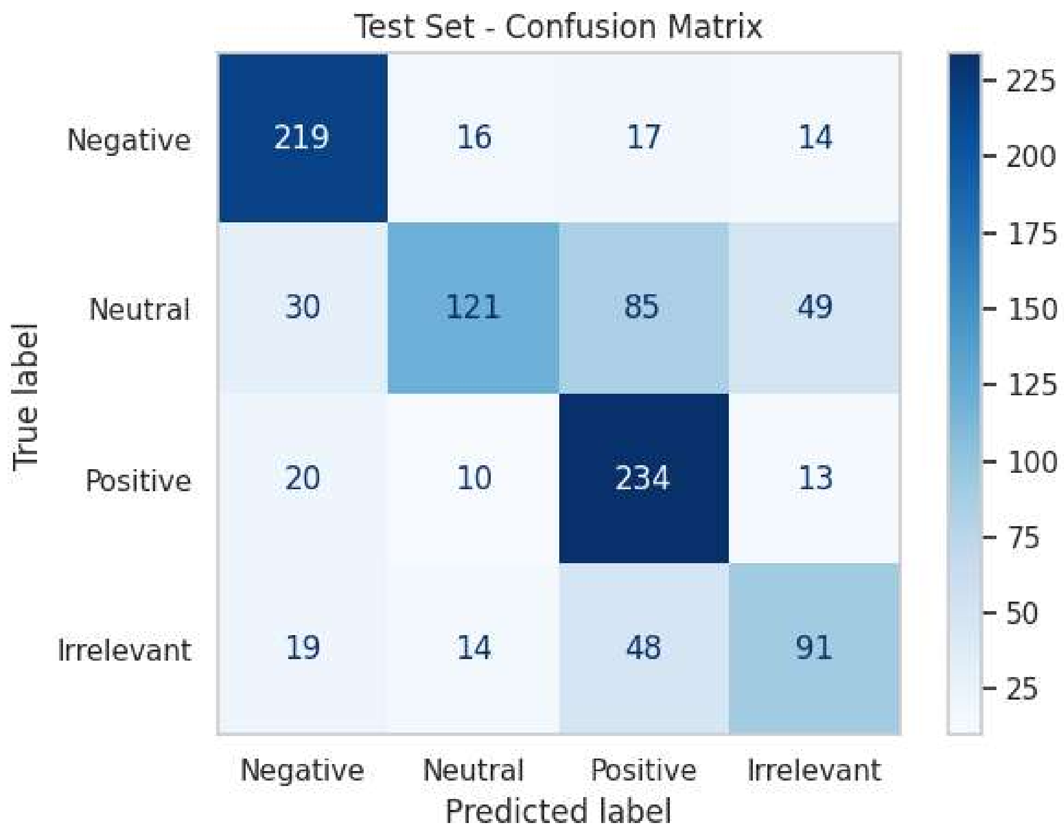
```
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Initialize the model
#model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=4)
model_RoBERTa = RobertaForSequenceClassification.from_pretrained('roberta
-base', num_labels=4)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model_RoBERTa.to(device)

optimizer = AdamW(model_RoBERTa.parameters(), lr=2e-5)

# Training loop
num_epochs = 3

for epoch in range(num_epochs):
    model_RoBERTa.train()
    for batch in train_loader:
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model_RoBERTa(input_ids, attention_mask=attention_mask, lab
els=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()

    # Evaluation on test set
    model_RoBERTa.eval()
    test_preds = []
    test_true = []
    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels']
            outputs = model_RoBERTa(input_ids, attention_mask=attention_mask)
            preds = torch.argmax(outputs.logits, dim=1).cpu().numpy()
            test_preds.extend(preds)
            test_true.extend(labels.numpy())

    accuracy = accuracy_score(test_true, test_preds)
    print(f'Epoch {epoch + 1}/{num_epochs}, Test Accuracy: {accuracy:.4f}')

# Save the model
torch.save(model_RoBERTa.state_dict(), 'sentiment_RoBERTa_model.pth')
```

```
# Final evaluation
print(classification_report(test_true, test_preds, target_names=['Negative', 'Neutral', 'Positive', 'Irrelevant']))
```

```
              precision    recall  f1-score   support

    Negative       0.76      0.82      0.79       266
     Neutral       0.75      0.42      0.54       285
    Positive       0.61      0.84      0.71       277
  Irrelevant       0.54      0.53      0.54       172

    accuracy                           0.67      1000
   macro avg       0.67      0.66      0.64      1000
weighted avg       0.68      0.67      0.65      1000
```

```
from sklearn.metrics import confusion_matrix

# Check if test_true labels need conversion (optional)
if not isinstance(test_true[0], str):  # If labels are not strings
    from sklearn.preprocessing import LabelEncoder
    encoder = LabelEncoder()
    test_true_encoded = encoder.fit_transform(test_true)  # Encode labels
    labels = [0, 1, 2, 3]  # Numerical labels
else:
    test_true_encoded = test_true
    labels = ['Negative', 'Neutral', 'Positive', 'Irrelevant']  # String labels

# Calculate confusion matrix with consistent labels
confusion_matrix_RoBERTa = confusion_matrix(test_true_encoded, test_preds, labels=labels)

print("Confusion matrix RoBERTa \n")
confusion_matrix_RoBERTa
```

```
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
labels = ['Negative', 'Neutral', 'Positive', 'Irrelevant']  # String labels
test_display = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_RoBERTa, display_labels=labels)
test_display.plot(cmap='Blues')
plt.title("Test Set - Confusion Matrix")
plt.grid(False)
plt.tight_layout()
plt.show()
```



3. DistilBERT (Distilled version of BERT)

DistilBERT is a smaller and faster version of the BERT model. It's created using a technique called knowledge distillation. This means that a smaller model (the student) learns to mimic the behavior of a larger, more complex model (the teacher). In this case, the teacher is BERT.

Key Features

- **Smaller size:** DistilBERT is about 40% smaller than BERT, making it more efficient in terms of memory and computation.
- **Faster:** It's also significantly faster than BERT, making it suitable for real-time applications.
- **Comparable performance:** Despite its smaller size, DistilBERT retains about 95% of BERT's language understanding capabilities.

How it Works

- **Knowledge Distillation:** The process involves training DistilBERT to predict the same outputs as BERT for a given input. However, instead of using hard labels (the correct answer), DistilBERT is trained on softened outputs from BERT. This allows the smaller model to learn more generalizable knowledge.
- **Architecture Simplification:** Some architectural elements of BERT, such as the token type embeddings, are removed to reduce complexity.

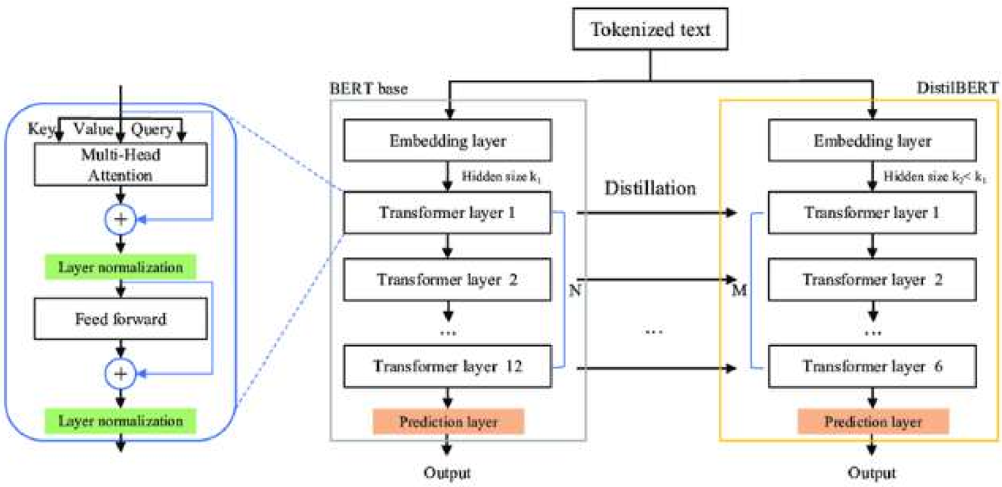
Advantages

- **Efficiency:** Smaller size and faster inference speed make it suitable for resource-constrained environments.
- **Cost-effective:** Lower computational requirements lead to reduced training and inference costs.
- **Good performance:** Despite its smaller size, it maintains a high level of performance on various NLP tasks.

Applications

- **Text classification:** Sentiment analysis, topic modeling
- **Named entity recognition:** Identifying entities in text (e.g., persons, organizations, locations)
- **Question answering:** Finding answers to questions based on given text
- **Text generation:** Summarization, translation

In summary, DistilBERT offers a compelling balance between model size, speed, and performance. It's a valuable tool for NLP practitioners looking to deploy models efficiently without sacrificing accuracy.




```
%%time

import pandas as pd
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification, AdamW
from sklearn.metrics import accuracy_score, classification_report

# Preprocess the data
def preprocess_data(df):
    df['label'] = df['Sentiment_label'].map({'Positive': 2, 'Negative': 0, 'Neutral': 1, 'Irrelevant': 3})
    return df['Tweet Content'].tolist(), df['label'].tolist()
```

```python
train_texts, train_labels = preprocess_data(data_train)
test_texts, test_labels = preprocess_data(data_test)

# Create a custom dataset
class SentimentDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = str(self.texts[idx])
        label = self.labels[idx]

        encoding = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt',
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }

# Initialize tokenizer and create datasets
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
train_dataset = SentimentDataset(train_texts, train_labels, tokenizer)
test_dataset = SentimentDataset(test_texts, test_labels, tokenizer)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

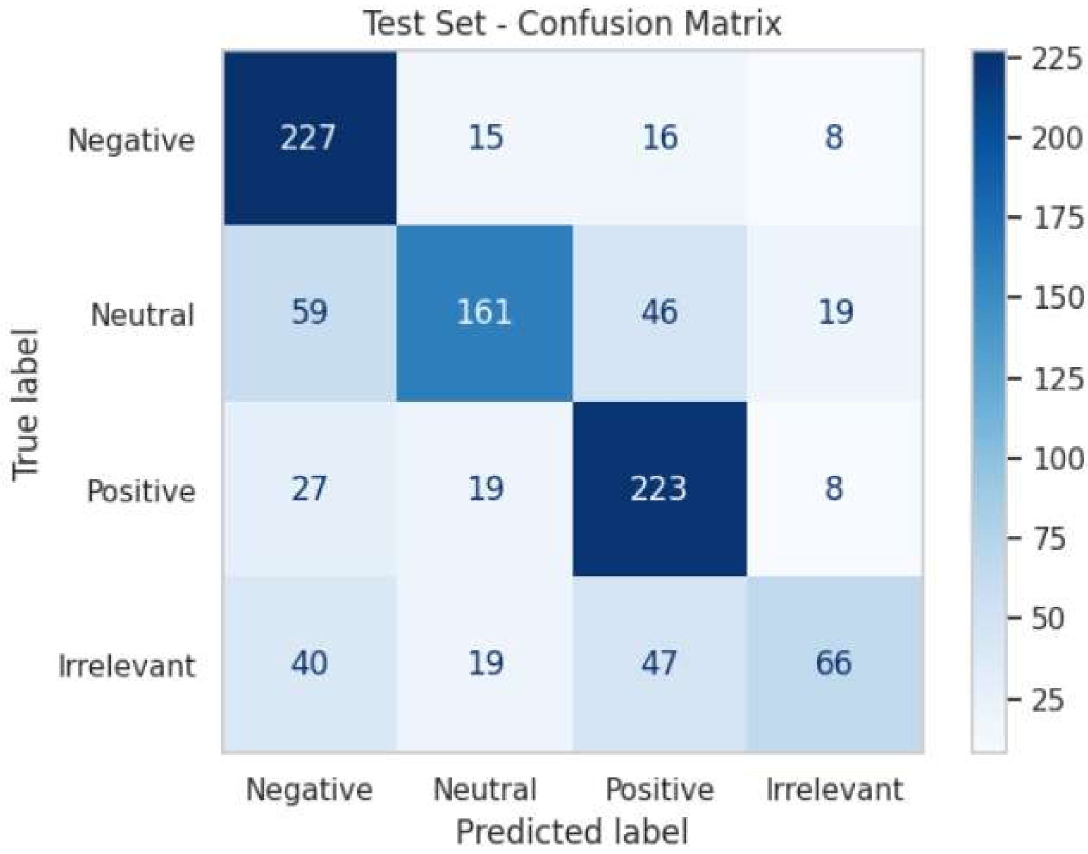
```python
# Initialize the model DistilBERT
model_DistilBERT = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=4)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model_DistilBERT.to(device)

optimizer = AdamW(model_DistilBERT.parameters(), lr=2e-5)

# Training loop
num_epochs = 3

for epoch in range(num_epochs):
    model_DistilBERT.train()
    for batch in train_loader:
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model_DistilBERT(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()

    # Evaluation on test set
    model_DistilBERT.eval()
    test_preds = []
    test_true = []
    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels']
            outputs = model_DistilBERT(input_ids, attention_mask=attention_mask)
            preds = torch.argmax(outputs.logits, dim=1).cpu().numpy()
            test_preds.extend(preds)
            test_true.extend(labels.numpy())

    accuracy = accuracy_score(test_true, test_preds)
    print(f'Epoch {epoch + 1}/{num_epochs}, Test Accuracy: {accuracy:.4f}')


torch.save(model_DistilBERT.state_dict(), 'sentiment_model_distilbert.pth')
# Final evaluation
```

```python
print(classification_report(test_true, test_preds, target_names=['Negative', 'Neutral', 'Positive', 'Irrelevant']))
```

```
              precision    recall  f1-score   support

    Negative       0.64      0.85      0.73       266
     Neutral       0.75      0.56      0.65       285
    Positive       0.67      0.81      0.73       277
  Irrelevant       0.65      0.38      0.48       172

    accuracy                           0.68      1000
   macro avg       0.68      0.65      0.65      1000
weighted avg       0.68      0.68      0.67      1000
```

```python
from sklearn.metrics import confusion_matrix

# Check if test_true labels need conversion (optional)
if not isinstance(test_true[0], str):  # If labels are not strings
    from sklearn.preprocessing import LabelEncoder
    encoder = LabelEncoder()
    test_true_encoded = encoder.fit_transform(test_true)  # Encode labels
    labels = [0, 1, 2, 3]  # Numerical labels
else:
    test_true_encoded = test_true
    labels = ['Negative', 'Neutral', 'Positive', 'Irrelevant']  # String labels

# Calculate confusion matrix with consistent labels
confusion_matrix_DistilBERT = confusion_matrix(test_true_encoded, test_preds, labels=labels)

print("Confusion matrix DistilBERT \n")
confusion_matrix_DistilBERT
```

```python
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
labels = ['Negative', 'Neutral', 'Positive', 'Irrelevant']  # String labels
test_display = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_DistilBERT, display_labels=labels)
test_display.plot(cmap='Blues')
plt.title("Test Set - Confusion Matrix")
plt.grid(False)
plt.tight_layout()
plt.show()
```



4. ALBERT: A Lite BERT for Self-Supervised Learning

ALBERT stands for A Lite BERT for Self-Supervised Learning. It's a language model developed by Google AI, designed to be more efficient and effective than the original BERT model.

Key Improvements Over BERT

- **Parameter Reduction:** ALBERT significantly reduces the number of parameters compared to BERT, making it more computationally efficient and faster to train. This is achieved by:
- **Factorized embedding parameterization:** Separating the embedding space into two smaller spaces, reducing the number of parameters.
- **Cross-layer parameter sharing:** Sharing parameters across different layers to reduce redundancy.
- **Sentence-Order Prediction (SOP):** Instead of the Next Sentence Prediction (NSP) task used in BERT, ALBERT employs SOP. This task is more challenging and helps the model better understand sentence relationships.

Architecture

ALBERT maintains the overall transformer architecture of BERT but incorporates the aforementioned improvements. It consists of:

- **Embedding layer:** Converts input tokens into numerical representations.
- **Transformer encoder:** Processes the input sequence and captures contextual information.

- **Output layer:** Predicts the masked words and sentence order.

Benefits of ALBERT

- **Efficiency:** ALBERT is significantly smaller and faster to train than BERT.
- **Improved Performance:** Despite its smaller size, ALBERT often achieves better or comparable performance to BERT on various NLP tasks.
- **Versatility:** Like BERT, ALBERT can be fine-tuned for various NLP tasks.

Applications

- **Text classification:** Sentiment analysis, topic modeling
- **Question answering:** Answering questions based on given text
- **Named entity recognition:** Identifying entities in text (e.g., persons, organizations, locations)
- **Text summarization:** Generating concise summaries of lengthy documents

In summary, ALBERT is a powerful language model that addresses some of the limitations of BERT while maintaining its strengths. It offers a good balance between model size, speed, and performance, making it a popular choice for various NLP applications.

```
%%time

import pandas as pd
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import AlbertTokenizer, AlbertForSequenceClassificatio
n, AdamW
from sklearn.metrics import accuracy_score, classification_report

# Preprocess the data
def preprocess_data(df):
    df['label'] = df['Sentiment_label'].map({'Positive': 2, 'Negative': 0, 'Neutra
l': 1, 'Irrelevant': 3})
    return df['Tweet Content'].tolist(), df['label'].tolist()

train_texts, train_labels = preprocess_data(data_train)
test_texts, test_labels = preprocess_data(data_test)

# Create a custom dataset
class SentimentDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)
```

```python
    def __getitem__(self, idx):
        text = str(self.texts[idx])
        label = self.labels[idx]

        encoding = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_len,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt',
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }

# Initialize tokenizer and create datasets
tokenizer = AlbertTokenizer.from_pretrained('albert-base-v2')
train_dataset = SentimentDataset(train_texts, train_labels, tokenizer)
test_dataset = SentimentDataset(test_texts, test_labels, tokenizer)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Initialize the model
model_ALBERT = AlbertForSequenceClassification.from_pretrained('albert-
base-v2', num_labels=4)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model_ALBERT.to(device)

# Set up optimizer
optimizer = AdamW(model_ALBERT.parameters(), lr=2e-5)

# Training loop
num_epochs = 3

for epoch in range(num_epochs):
    model_ALBERT.train()
```

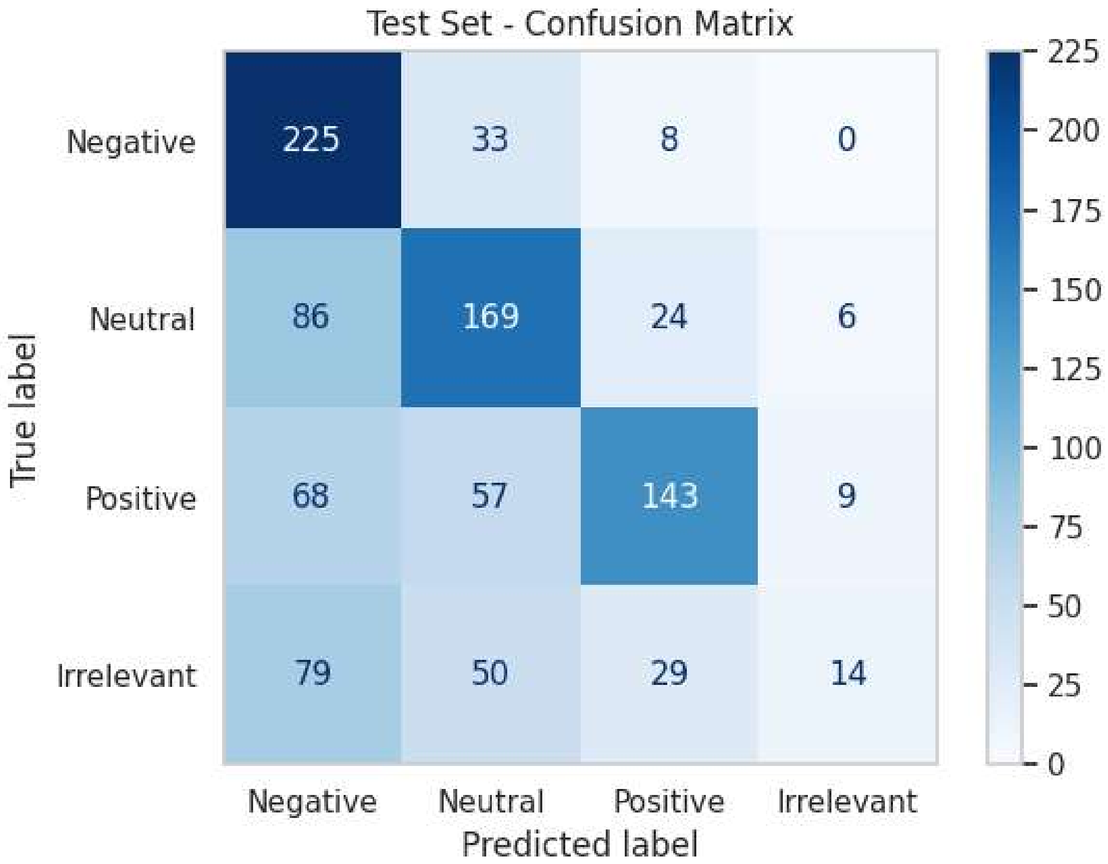
```
    for batch in train_loader:
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model_ALBERT(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()

    # Evaluation on test set
    model_ALBERT.eval()
    test_preds = []
    test_true = []
    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels']
            outputs = model_ALBERT(input_ids, attention_mask=attention_mask)
            preds = torch.argmax(outputs.logits, dim=1).cpu().numpy()
            test_preds.extend(preds)
            test_true.extend(labels.numpy())

    accuracy = accuracy_score(test_true, test_preds)
    print(f'Epoch {epoch + 1}/{num_epochs}, Test Accuracy: {accuracy:.4f}')

# Final evaluation
print(classification_report(test_true, test_preds, target_names=['Negative', 'Neutral', 'Positive', 'Irrelevant']))

# Save the model
torch.save(model_ALBERT.state_dict(), 'sentiment_model_albert.pth')
```

```
# Final evaluation
print(classification_report(test_true, test_preds, target_names=['Negative', 'Neutral', 'Positive', 'Irrelevant']))
```

	precision	recall	f1-score	support
Negative	0.49	0.85	0.62	266
Neutral	0.55	0.59	0.57	285
Positive	0.70	0.52	0.59	277
Irrelevant	0.48	0.08	0.14	172
accuracy			0.55	1000
macro avg	0.56	0.51	0.48	1000
weighted avg	0.56	0.55	0.52	1000

```
# Assuming test_true and test_preds are defined
from sklearn.metrics import confusion_matrix

# Check if test_true labels need conversion (optional)
if not isinstance(test_true[0], str):  # If labels are not strings
    from sklearn.preprocessing import LabelEncoder
    encoder = LabelEncoder()
    test_true_encoded = encoder.fit_transform(test_true)  # Encode labels
    labels = [0, 1, 2, 3]  # Numerical labels
else:
    test_true_encoded = test_true
    labels = ['Negative', 'Neutral', 'Positive', 'Irrelevant']  # String labels

# Calculate confusion matrix with consistent labels
confusion_matrix_ALBERT = confusion_matrix(test_true_encoded, test_preds, labels=labels)

print("Confusion matrix ALBERT \n")
confusion_matrix_ALBERT
```

```
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
labels = ['Negative', 'Neutral', 'Positive', 'Irrelevant']  # String labels
test_display = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_ALBERT, display_labels=labels)
test_display.plot(cmap='Blues')
plt.title("Test Set - Confusion Matrix")
plt.grid(False)
plt.tight_layout()
plt.show()
```



5. XLNet: Going Beyond BERT

XLNet is a powerful language model that builds upon the successes of its predecessor, BERT, while addressing some of its limitations.

It stands for "Extreme Language Model".

Key Differences from BERT

- **Autoregressive vs. Autoencoding:** While BERT is an autoencoding model, XLNet is an autoregressive model. This means that XLNet predicts the next token in a sequence given the previous ones, similar to how we humans generate text. This approach allows XLNet to capture bidirectional context without the limitations of BERT's masked language modeling.
- **Permutation Language Model:** XLNet introduces the concept of a permutation language model. Instead of training on a fixed order of tokens, it considers all possible permutations of the input sequence. This enables the model to learn dependencies between any two tokens in the sequence, regardless of their position.

How XLNet Works

- **Permutation Language Modeling:** XLNet randomly permutes the input sequence and trains the model to predict the masked tokens in any position based on the context of the remaining tokens.
- **Attention Mechanism:** Similar to BERT, XLNet uses a self-attention mechanism to capture dependencies between different parts of the input sequence.

- **Two-Stream Self-Attention:** XLNet employs two streams of self-attention:
- **Content stream:** Focuses on the content of the tokens.
- **Query stream:** Focuses on the position of the tokens in the permutation.

Advantages of XLNet

- **Bidirectional Context:** XLNet can capture bidirectional context more effectively than BERT, leading to improved performance on various NLP tasks.
- **Flexibility:** The permutation language modeling approach allows for more flexible modeling of language.
- **Strong Performance:** XLNet has achieved state-of-the-art results on many NLP benchmarks.

Applications of XLNet

- *Text classification*
- *Question answering*
- *Natural language inference*
- *Machine translation*
- *Text summarization*

In summary, XLNet is a significant advancement in the field of natural language processing, offering improved performance and flexibility compared to previous models. Its ability to capture bidirectional context effectively makes it a powerful tool for various NLP applications.

```
%%time
import pandas as pd
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import XLNetTokenizer, XLNetForSequenceClassificatio
n, AdamW
from sklearn.metrics import accuracy_score, classification_report

# Preprocess the data
def preprocess_data(df):
    df['label'] = df['Sentiment_label'].map({'Positive': 2, 'Negative': 0, 'Neutra
l': 1, 'Irrelevant': 3})
    return df['Tweet Content'].tolist(), df['label'].tolist()

train_texts, train_labels = preprocess_data(data_train)
test_texts, test_labels = preprocess_data(data_test)

# Create a custom dataset
class SentimentDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
```

```
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = str(self.texts[idx])
        label = self.labels[idx]

        encoding = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_len,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_token_type_ids=True,
            return_tensors='pt',
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'token_type_ids': encoding['token_type_ids'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }

# Initialize tokenizer and create datasets
tokenizer = XLNetTokenizer.from_pretrained('xlnet-base-cased')
train_dataset = SentimentDataset(train_texts, train_labels, tokenizer)
test_dataset = SentimentDataset(test_texts, test_labels, tokenizer)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Initialize the model XLNet
model_XLNet = XLNetForSequenceClassification.from_pretrained('xlnet-ba
se-cased', num_labels=4)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model_XLNet.to(device)

# Set up optimizer
optimizer = AdamW(model_XLNet.parameters(), lr=2e-5)
```

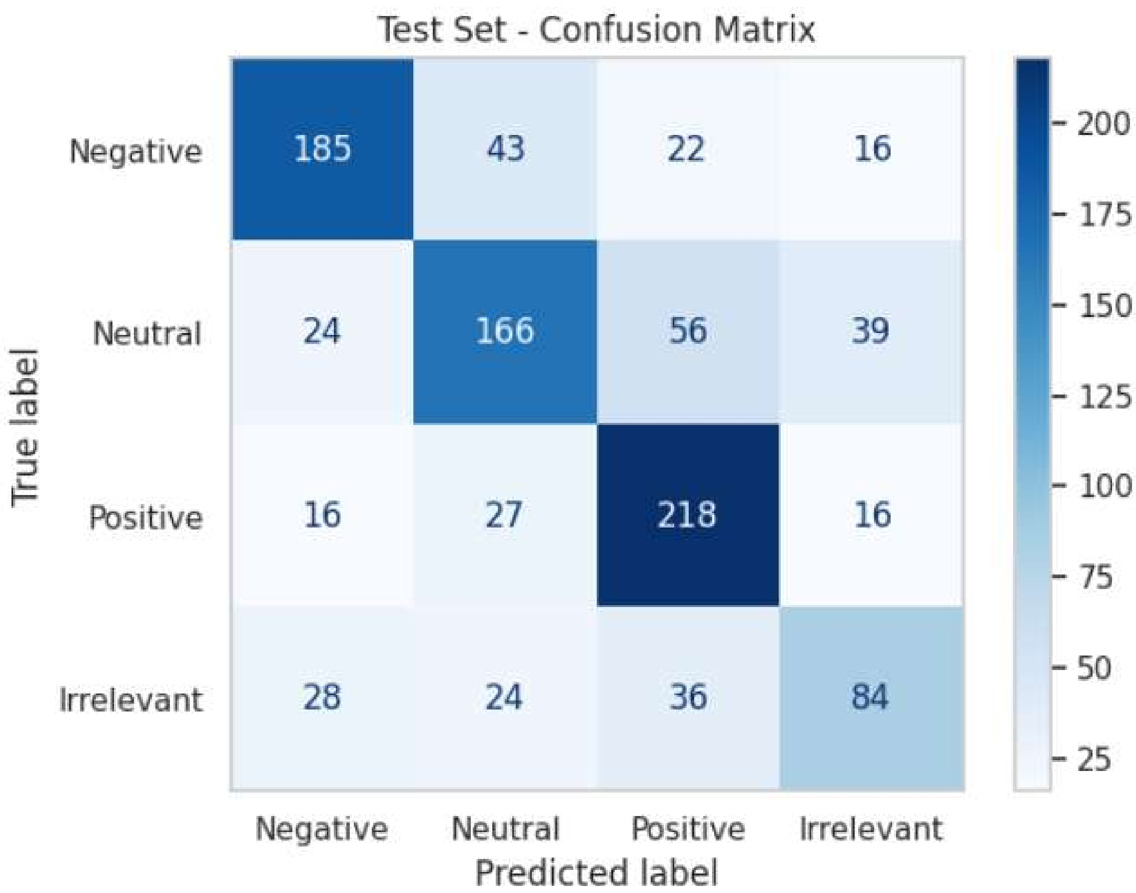
```
# Training loop
num_epochs = 3

for epoch in range(num_epochs):
    model_XLNet.train()
    for batch in train_loader:
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        token_type_ids = batch['token_type_ids'].to(device)
        labels = batch['labels'].to(device)
        outputs = model_XLNet(input_ids, attention_mask=attention_mask, to
ken_type_ids=token_type_ids, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()

    # Evaluation on test set
    model_XLNet.eval()
    test_preds = []
    test_true = []
    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            token_type_ids = batch['token_type_ids'].to(device)
            labels = batch['labels']
            outputs = model_XLNet(input_ids, attention_mask=attention_mask,
token_type_ids=token_type_ids)
            preds = torch.argmax(outputs.logits, dim=1).cpu().numpy()
            test_preds.extend(preds)
            test_true.extend(labels.numpy())

    accuracy = accuracy_score(test_true, test_preds)
    print(f'Epoch {epoch + 1}/{num_epochs}, Test Accuracy: {accuracy:.4f}')

# Save the model_XLNet
torch.save(model_XLNet.state_dict(), 'sentiment_model_xlnet.pth')
```

```
# Final evaluation
print(classification_report(test_true, test_preds, target_names=['Neg
ative', 'Neutral', 'Positive', 'Irrelevant']))
```

	precision	recall	f1-score	support
Negative	0.73	0.70	0.71	266
Neutral	0.64	0.58	0.61	285
Positive	0.66	0.79	0.72	277
Irrelevant	0.54	0.49	0.51	172
accuracy			0.65	1000
macro avg	0.64	0.64	0.64	1000
weighted avg	0.65	0.65	0.65	1000

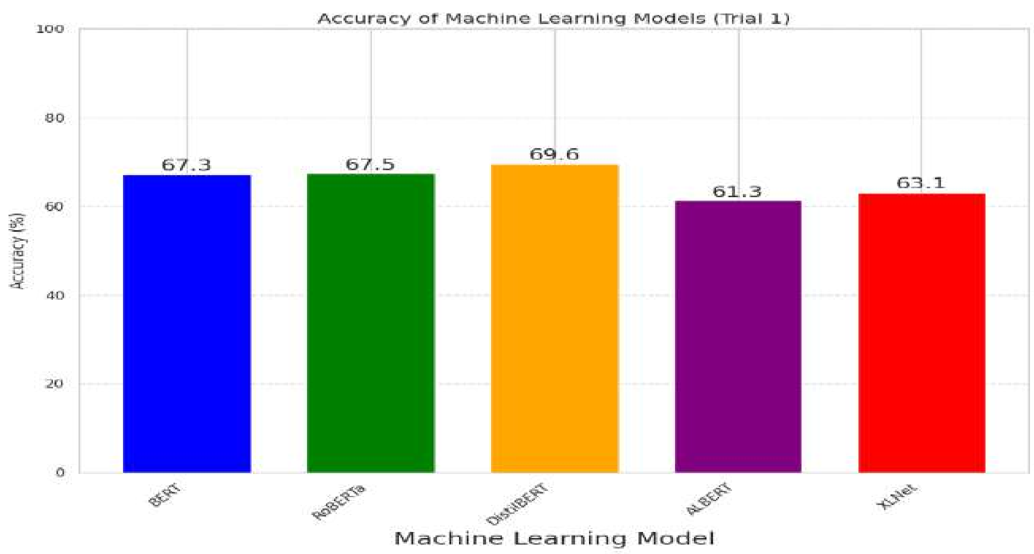
```
# Assuming test_true and test_preds are defined
from sklearn.metrics import confusion_matrix

# Check if test_true labels need conversion (optional)
if not isinstance(test_true[0], str):  # If labels are not strings
    from sklearn.preprocessing import LabelEncoder
    encoder = LabelEncoder()
    test_true_encoded = encoder.fit_transform(test_true)  # Encode labels
    labels = [0, 1, 2, 3]  # Numerical labels
else:
    test_true_encoded = test_true
    labels = ['Negative', 'Neutral', 'Positive', 'Irrelevant']  # String labels

# Calculate confusion matrix with consistent labels
confusion_matrix_XLNet = confusion_matrix(test_true_encoded, test_preds
, labels=labels)

print("Confusion matrix XLNet \n")
confusion_matrix_XLNet
```

```
from sklearn.metrics import classification_report, confusion_matrix, Confu
sionMatrixDisplay
labels = ['Negative', 'Neutral', 'Positive', 'Irrelevant']  # String labels
test_display = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix
_XLNet, display_labels=labels)
test_display.plot(cmap='Blues')
plt.title("Test Set - Confusion Matrix")
plt.grid(False)
plt.tight_layout()
plt.show()
```




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

# Data for the bar graph (only Trial 1)
models = ["BERT", "RoBERTa", "DistilBERT", "ALBERT", "XLNet"]

accuracy_trial_1 = [67.3, 67.50, 69.60, 61.3, 63.1]

# Set up the plot
fig, ax = plt.subplots(figsize=(10, 8))

# Set the width of each bar and the positions of the bars
width = 0.7

# Create bars with different colors
colors = ['blue', 'green', 'orange', 'purple', 'red', 'magenta']
ax.bar(models, accuracy_trial_1, width, color=colors)

# Customize the plot
```

```python
ax.set_ylabel('Accuracy (%)', fontsize=12)  # Increase font size for y-axis label
ax.set_xlabel('Machine Learning Model', fontsize=18)  # Increase font size for x-axis label
ax.set_title('Accuracy of Machine Learning Models (Trial 1)', fontsize=14)  # Increase font size for title

# Setxticks and rotate x-axis labels for better readability
ax.set_xticks(models)
ax.set_xticklabels(models, rotation=45, ha='right', fontsize=11)  # Increase font size for x-axis tick labels

# Add value labels on top of each bar with increased font size
for i, v in enumerate(accuracy_trial_1):
    ax.text(i, v + 0.2, f'{v:.1f}', ha='center', va='bottom', fontsize=16)  # Adjust vertical offset and format to one decimal place

# Set y-axis to start at 0
ax.set_ylim(0, 100)

# Add gridlines
ax.grid(axis='y', linestyle='--', alpha=0.9)

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



Follow for more AI content: https://lnkd.in/gxcsx77g

NLP: Predicting Sentiment Using Traditional Machine Learning Techniques

1. **SVM**
2. **Naive Bayes**
3. **Logistic Regression**
4. **Decision Tree**
5. **Random Forest**
6. **XGBoost**
7. **LightGBM**

```python
import os
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(style='whitegrid')
```

```python
import tensorflow as tf
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.metrics import classification_report,confusion_matrix
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, LSTM,Dense, SpatialDropout1D, Dropout
from keras.initializers import Constant

pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 900)
pd.set_option('display.max_colwidth', 200)

import warnings
warnings.filterwarnings("ignore")
```

Sentiment analysis is a powerful technique that involves determining the emotional tone of a piece of text.

1. It classifies text as positive, negative, or neutral, and can even delve deeper into specific emotions like happiness, sadness, or anger. This process, often referred to as opinion mining, is a cornerstone of Natural Language Processing (NLP).
2. By applying sentiment analysis, businesses can gain valuable insights into customer perceptions, product performance, and market trends. For instance, analysing customer reviews can reveal whether a product is meeting customer expectations or if there's a need for improvement. Similarly, monitoring social media sentiment can help companies understand their brand reputation and identify potential issues.
3. Sentiment analysis is particularly useful for processing large volumes of unstructured text data, such as customer feedback, social media posts, and survey responses. It efficiently categorizes this data, making it easier to extract meaningful information. Tools like Net Promoter Score (NPS) surveys, which measure customer loyalty, benefit greatly from sentiment analysis. By automating the analysis of NPS responses, businesses can quickly identify patterns and trends in customer sentiment.
4. In essence, sentiment analysis empowers organizations to understand the voice of their customers, make data-driven decisions, and ultimately improve their products and services.

Sentiment Analysis: Pre-processing Steps

1. Data Collection:

- Gather text data from various sources (e.g., customer reviews, social media posts, news articles)
- Ensure a diverse and representative sample for accurate analysis

2. Text Cleaning:

- Remove HTML tags, if present
- Strip special characters and punctuation that don't contribute to sentiment.

- Convert all text to lowercase for consistency.

3. **Tokenization:**
    - Break down the text into individual words or tokens
    - This step allows for analysis at the word level
4. **Stop Word Removal:**
    - Identify and remove common words (e.g., "the", "is", "and") that don't carry sentiment
    - This reduces noise in the data and focuses on meaningful words
5. **Stemming or Lemmatization:**
    - Stemming: Reduce words to their root form by removing suffixes (e.g., "running" to "run")
    - Lemmatization: Convert words to their base or dictionary form (e.g., "better" to "good")
    - This step helps in standardizing words and reducing vocabulary size
6. **Handling Negations:**
    - Identify negation words (e.g., "not", "never") and mark the affected phrases
    - This is crucial as negations can invert the sentiment of surrounding words
7. **Dealing with Sarcasm and Context:**
    - While challenging, attempt to identify sarcastic phrases or contextual nuances
    - This may involve looking at surrounding sentences or overall tone
8. **Feature Extraction:**
    - Convert the preprocessed text into a format suitable for machine learning algorithms
    - Common methods include bag-of-words, TF-IDF, or word embeddings
9. **Normalization:**
    - Standardize the features to ensure all inputs are on a similar scale
    - This helps in improving the performance of many machine learning algorithms

```python
train = pd.read_csv('/kaggle/input/sentiment-analysis-dataset/training.csv',header=None)
validation = pd.read_csv('/kaggle/input/sentiment-analysis-dataset/validation.csv',header=N
one)

train.columns=['Tweet ID','Entity','Sentiment','Tweet Content']
validation.columns=['Tweet ID','Entity','Sentiment','Tweet Content']

print("Training DataSet: \n")
train = train.sample(10000)
display(train.head())
```

```
Training DataSet:
```

	Tweet ID	Entity	Sentiment	Tweet Content
18485	9967	PlayStation5(PS5)	Irrelevant	is
73746	9034	Nvidia	Neutral	Nvidia on global foundries confirmed. TSMC is incapable to make enough big die for nvidia pic.twitter.com/KXTIPTNyl9
6600	336	Amazon	Negative	Amazon doesn't deliver in my hood
18166	9914	PlayStation5(PS5)	Positive	BOOOYYYYYY And I AD CANT A WAIT!!!
43953	10347	PlayerUnknownsBattlegrounds(PUBG)	Negative	PUBG is banned in this county

```python
print("Validation DataSet: \n")
display(validation.head())
```

```
Validation DataSet:
```

	Tweet ID	Entity	Sentiment	Tweet Content
0	3364	Facebook	Irrelevant	I mentioned on Facebook that I was struggling for motivation to go for a run the other day, which has been translated by Tom's great auntie as 'Hayley can't get out of bed' and told to his grandma...
1	352	Amazon	Neutral	BBC News - Amazon boss Jeff Bezos rejects claims company acted like a 'drug dealer' bbc.co.uk/news/av/busine...
2	8312	Microsoft	Negative	@Microsoft Why do I pay for WORD when it functions so poorly on my @SamsungUS Chromebook? 🙄
3	4371	CS-GO	Negative	CSGO matchmaking is so full of closet hacking, it's a truly awful game.
4	4433	Google	Neutral	Now the President is slapping Americans in the face that he really did commit an unlawful act after his acquittal! From Discover on Google vanityfair.com/news/2020/02/t...

```python
train = train.dropna(subset=['Tweet Content'])

display(train.isnull().sum())
print("*****"* 5)
display(validation.isnull().sum())
```

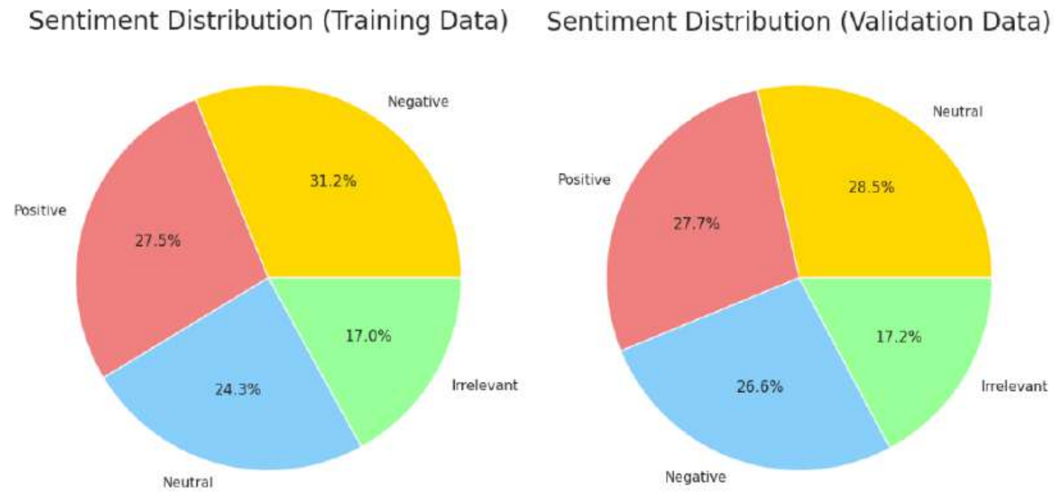
```
Tweet ID         0
Entity           0
Sentiment        0
Tweet Content    0
dtype: int64

*************************

Tweet ID         0
Entity           0
Sentiment        0
Tweet Content    0
dtype: int64
```

```python
duplicates = train[train.duplicated(subset=['Entity', 'Sentiment', 'Tweet Content'], keep=False)]
train = train.drop_duplicates(subset=['Entity', 'Sentiment', 'Tweet Content'], keep='first')

duplicates = validation[validation.duplicated(subset=['Entity', 'Sentiment', 'Tweet Content'], keep=False)]
validation = validation.drop_duplicates(subset=['Entity', 'Sentiment', 'Tweet Content'], keep='first')
```

```python
# Calculate sentiment counts for train and validation data
sentiment_counts_train = train['Sentiment'].value_counts()
sentiment_counts_validation = validation['Sentiment'].value_counts()

combined_counts = pd.concat([sentiment_counts_train, sentiment_counts_validation], axis=1)
combined_counts.fillna(0, inplace=True)
combined_counts.columns = ['Test Data', 'Validation Data']  # Set desired column names
combined_counts
```

	Test Data	Validation Data
Sentiment		
Negative	3069	266
Positive	2709	277
Neutral	2396	285
Irrelevant	1673	172

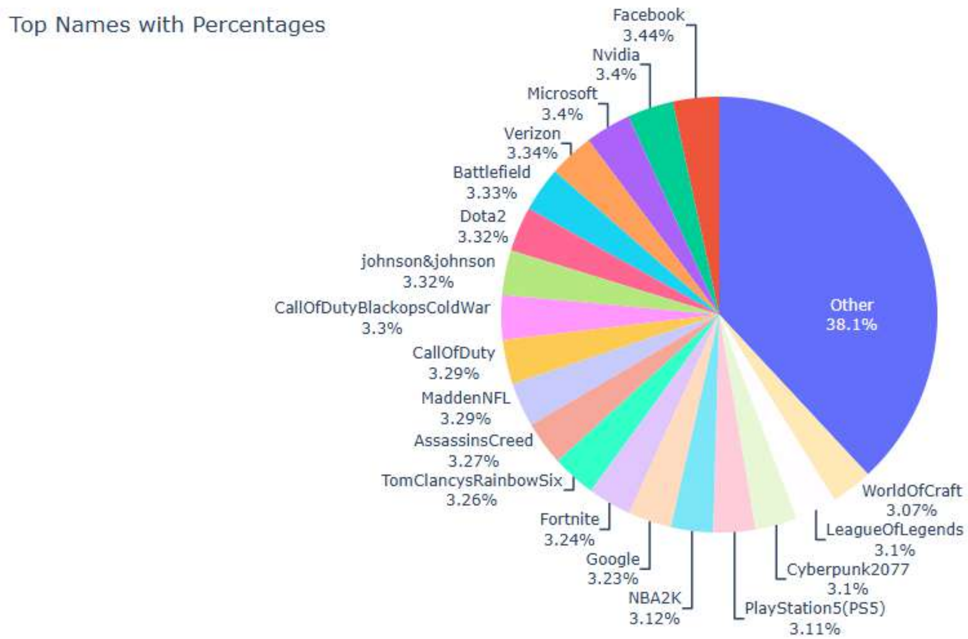
```
sentiment_counts_train = train['Sentiment'].value_counts()
sentiment_counts_validation = validation['Sentiment'].value_counts()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
ax1.pie(sentiment_counts_train, labels=sentiment_counts_train.index, autopct='%1.1f%%',
colors=['gold', 'lightcoral', 'lightskyblue','#99FF99'])
ax1.set_title('Sentiment Distribution (Training Data)', fontsize=20)

ax2.pie(sentiment_counts_validation, labels=sentiment_counts_validation.index, autopct='
%1.1f%%', colors=['gold', 'lightcoral', 'lightskyblue','#99FF99'])
ax2.set_title('Sentiment Distribution (Validation Data)', fontsize=20)
plt.tight_layout()
plt.show()
```



```python
# Calculate the value counts of 'Entity'
entity_counts = train['Entity'].value_counts()

top_names = entity_counts.head(19)

other_count = entity_counts[19:].sum()
top_names['Other'] = other_count
top_names.to_frame()
```

	count
Entity	
Facebook	339
Nvidia	335
Microsoft	335
Verizon	329
Battlefield	328
Dota2	327
johnson&johnson	327
CallOfDutyBlackopsColdWar	325
CallOfDuty	324
MaddenNFL	324
AssassinsCreed	322
TomClancysRainbowSix	321
Fortnite	319
Google	318
NBA2K	307
PlayStation5(PS5)	306
Cyberpunk2077	305
LeagueOfLegends	305
WorldOfCraft	302
Other	3749

```python
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio

# Calculate the percentages
percentages = (top_names / top_names.sum()) * 100

# Create the pie chart
fig = go.Figure(data=[go.Pie(
    labels=percentages.index,
    values=percentages,
    textinfo='label+percent',
    insidetextorientation='radial'
)])

fig.update_layout(
    title_text='Top Names with Percentages',
    showlegend=False
)
fig.show()
```



WordCloud

WordCloud is a visualization technique used in machine learning (ML), especially in Natural Language Processing (NLP) tasks. It helps visualize textual data by displaying words in a cloud formation, where the size of each word reflects its frequency or importance in the text.

- Data Input: You feed the WordCloud function with text data. This could be anything from a document collection, social media comments, or even code repositories.

- Frequency Analysis: The algorithm analyzes the text and counts the occurrences of each word.
- Word Placement and Sizing: Based on the frequency count, WordCloud positions and sizes the words. More frequent words appear larger and more prominent in the cloud, while less frequent ones are smaller.
- Visualization: Finally, it generates a visual output where the word cloud showcases the prominent themes and keywords within the text data.

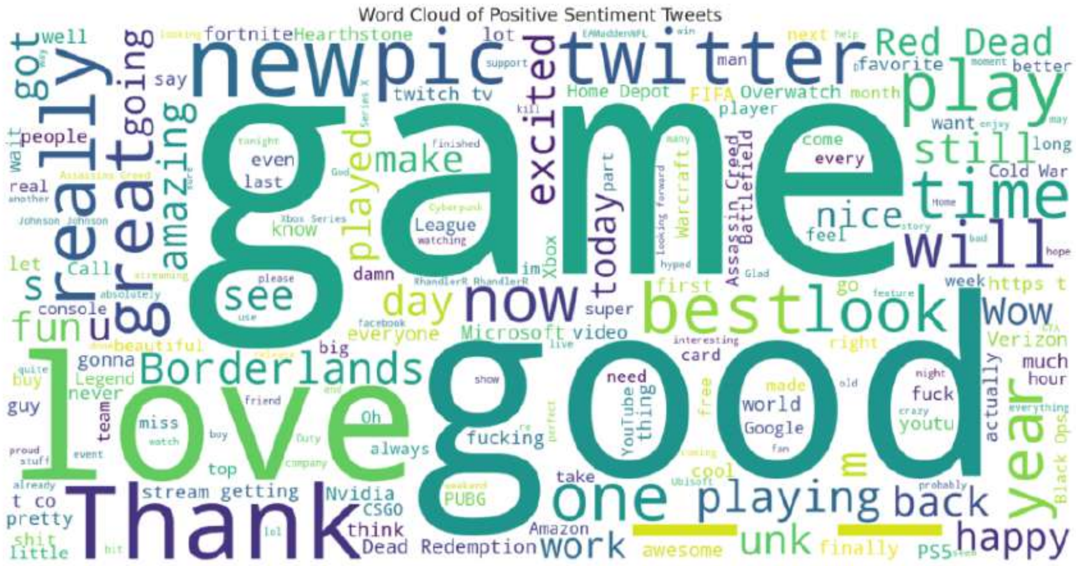
Benefits of using WordCloud in ML:

- Easy Identification of Key Terms: Word clouds quickly reveal the most frequently used words, helping you understand the core focus of the text data.
- Text Summarization: They provide a summarized view of a large corpus of text, making it easier to grasp the overall content.
- Highlighting Trends: Word clouds can be used to identify emerging trends or topics of discussion within the text data.

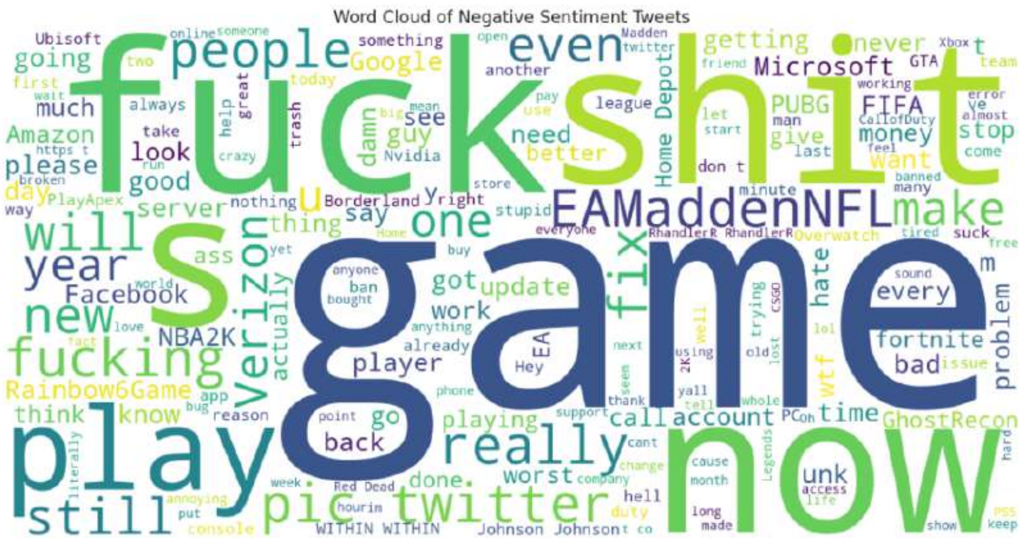
Applications of WordCloud in ML:

- Analyzing social media sentiment: See which words are most commonly used when people express positive or negative opinions.
- Topic modeling for research papers: Identify the main themes discussed in a collection of research papers.
- Understanding user reviews: Analyze product reviews to see which features are most mentioned by users.

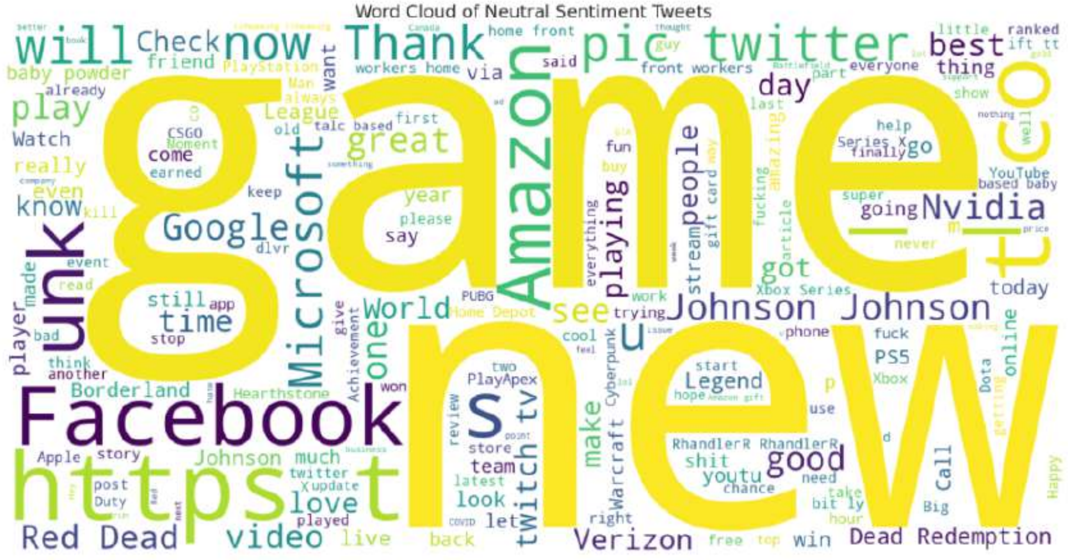
```python
from wordcloud import WordCloud
# Filter positive sentiment tweets and extract the 'Tweet Content' column
positive_tweets = train[train["Sentiment"] == "Positive"]["Tweet Content"]
positive_text = ' '.join(positive_tweets)
wordcloud = WordCloud(width=1600, height=800, max_words=200, background_color='white').generate(positive_text)

plt.figure(figsize=(20, 12))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Positive Sentiment Tweets', fontsize=24)
plt.tight_layout(pad=0)
plt.show()
```



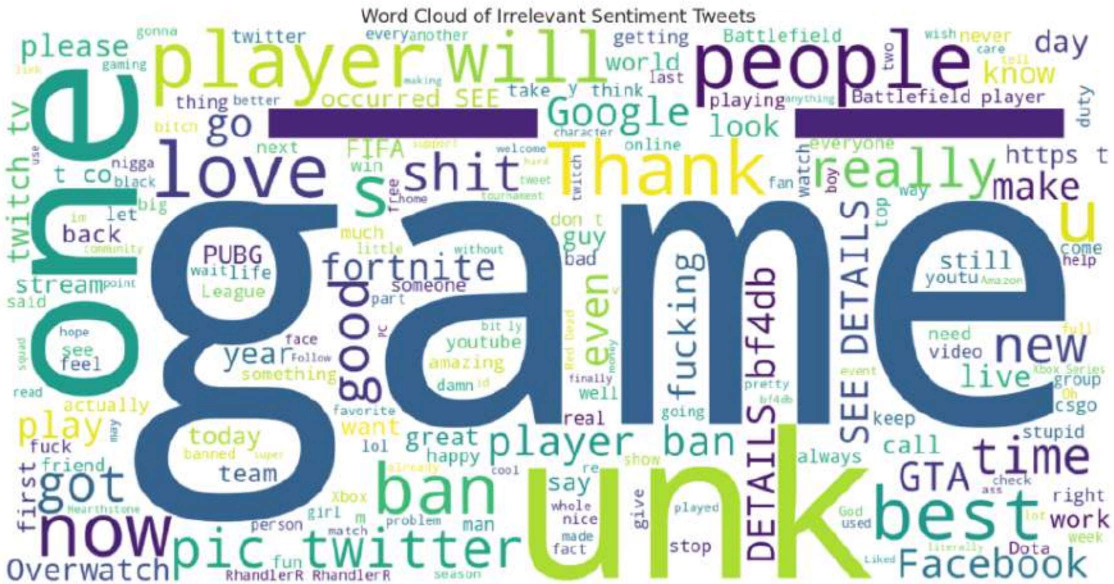
```python
from wordcloud import WordCloud

# Filter positive sentiment tweets and extract the 'Tweet Content' column
negative_tweets = train[train["Sentiment"] == "Negative"]["Tweet Content"]
negative_text = ' '.join(negative_tweets)
wordcloud = WordCloud(width=1600, height=800, max_words=200, background_color='white').generate(ne
gative_text)

plt.figure(figsize=(20, 12))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Negative Sentiment Tweets', fontsize=24)
plt.tight_layout(pad=0)
plt.show()
```



```
from wordcloud import WordCloud

neutral_tweets = train[train["Sentiment"] == "Neutral"]["Tweet Content"]
neutral_text = ' '.join(neutral_tweets)
wordcloud = WordCloud(width=1600, height=800, max_words=200, background_color='white').generate(ne
utral_text)

plt.figure(figsize=(20, 12))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Neutral Sentiment Tweets', fontsize=24)
plt.tight_layout(pad=0)
plt.show()
```



```python
from wordcloud import WordCloud

Irrelevant_tweets = train[train["Sentiment"] == "Irrelevant"]["Tweet Content"]
Irrelevant_text = ' '.join(Irrelevant_tweets)
wordcloud = WordCloud(width=1600, height=800, max_words=200, background_color='white').generate(Irr
elevant_text)

plt.figure(figsize=(20, 12))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Irrelevant Sentiment Tweets', fontsize=24)
plt.tight_layout(pad=0)
plt.show()
```



```
import plotly.graph_objects as go

grouped_counts = train.groupby(['Entity', 'Sentiment']).size().reset_index(name='Count')
entity_total_counts = grouped_counts.groupby('Entity')['Count'].transform('sum')
grouped_counts['Percentage'] = (grouped_counts['Count'] / entity_total_counts) * 100
grouped_counts = grouped_counts.sort_values('Count', ascending=False)

# Create a colorful table using Plotly
fig = go.Figure(data=[go.Table(
    header=dict(
        values=list(grouped_counts.columns),
        fill_color='paleturquoise',
        align='left',
        font=dict(color='black', size=12)
    ),
    cells=dict(
        values=[grouped_counts[k].tolist() for k in grouped_counts.columns],
        fill_color=[
            'lightcyan',
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative' else 'lightyellow' for s in grouped_cou
nts['Sentiment']],
            'lavender',
            'lightsalmon'
        ],
        align='left',
        font=dict(color='black', size=11)
```

```python
    ))
])

# Update the layout
fig.update_layout(
    title='Entity and Sentiment Distribution',
    width=800,
    height=800,
)
fig.show()
```

Entity and Sentiment Distribution

Entity	Sentiment	Count	Percentage
MaddenNFL	Negative	239	73.76543209876543
AssassinsCreed	Positive	197	61.18012422360248
NBA2K	Negative	188	61.23778501628665
TomClancysRainbowSix	Negative	150	46.728971962616825
FIFA	Negative	150	51.369863013698634
johnson&johnson	Neutral	142	43.425076452599384
Borderlands	Positive	141	47.474747474747474
Verizon	Negative	140	42.5531914893617
Amazon	Neutral	139	47.766323024054984
Battlefield	Irrelevant	137	41.76829268292683
WorldOfCraft	Neutral	132	43.70860927152318
johnson&johnson	Negative	129	39.44954128440367

```python
import plotly.graph_objects as go

# Group by 'Entity' and 'Sentiment' and calculate the count
grouped_counts = validation.groupby(['Entity', 'Sentiment']).size().reset_index(name='Count')
entity_total_counts = grouped_counts.groupby('Entity')['Count'].transform('sum')
grouped_counts['Percentage'] = (grouped_counts['Count'] / entity_total_counts) * 100
grouped_counts = grouped_counts.sort_values('Count', ascending=False)

# Create a colorful table using Plotly
fig = go.Figure(data=[go.Table(
    header=dict(
        values=list(grouped_counts.columns),
        fill_color='paleturquoise',
        align='left',
        font=dict(color='black', size=12)
    ),
    cells=dict(
        values=[grouped_counts[k].tolist() for k in grouped_counts.columns],
        fill_color=[
            'lightcyan',
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative' else 'lightyellow' for s in grouped_cou
nts['Sentiment']],
            'lavender',
            'lightsalmon'
        ],
        align='left',
```

```
        font=dict(color='black', size=11)
    ))
])

fig.update_layout(
    title='Entity and Sentiment Distribution',
    width=800,
    height=800,
)
fig.show()
```

Entity and Sentiment Distribution

Entity	Sentiment	Count	Percentage
AssassinsCreed	Positive	24	72.72727272727273
johnson&johnson	Neutral	19	48.717948717948715
RedDeadRedemption(RDR)	Neutral	18	45
Amazon	Neutral	18	52.94117647058824
MaddenNFL	Negative	18	62.06896551724138
NBA2K	Negative	17	80.95238095238095
ApexLegends	Neutral	17	47.22222222222222
Cyberpunk2077	Positive	17	56.666666666666664
FIFA	Negative	16	42.10526315789473
WorldOfCraft	Neutral	15	50
Fortnite	Irrelevant	15	44.11764705882353
Nvidia	Neutral	15	42.857142857142854
PlayerUnknownsBattlegrour	Irrelevant	15	39.473684210526315
PlayStation5(PS5)	Positive	15	45.45454545454545
RedDeadRedemption(RDR)	Positive	15	37.5
Borderlands	Positive	14	42.42424242424242
LeagueOfLegends	Neutral	14	37.83783783783784

List of methods commonly used for small text classification:

Traditional Machine Learning Methods:

- **Support Vector Machines (SVM)**
- **Naive Bayes classifiers**
- **Logistic Regression**
- **Decision Trees**
- **Random Forests**
- **Gradient Boosting Machines (XGBoost)**
- **Gradient Boosting Machines (LightGBM)**

Model Building

```
from tensorflow.keras.layers import Input, Dropout, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.initializers import TruncatedNormal
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.metrics import CategoricalAccuracy
from tensorflow.keras.utils import to_categorical

import pandas as pd
from sklearn.model_selection import train_test_split
```

```
import pandas as pd
import plotly.graph_objects as go
```

```python
# Assuming you've already run the data preprocessing steps
data = train[['Tweet Content', 'Sentiment']]

# Set your model output as categorical and save in new label col
data['Sentiment_label'] = pd.Categorical(data['Sentiment'])

# Transform your output to numeric
data['Sentiment'] = data['Sentiment_label'].cat.codes

# Use the entire training data as data_train
data_train = data

# Use validation data as data_test
data_test = validation[['Tweet Content', 'Sentiment']]
data_test['Sentiment_label'] = pd.Categorical(data_test['Sentiment'])
data_test['Sentiment'] = data_test['Sentiment_label'].cat.codes

# Create a colorful table using Plotly
fig = go.Figure(data=[go.Table(
    header=dict(
        values=list(data_train.columns),
        fill_color='paleturquoise',
        align='left',
        font=dict(color='black', size=12)
    ),
    cells=dict(
```

```
        values=[data_train[k].tolist()[:10] for k in data_train.columns],
        fill_color=[
            'lightcyan',  # Tweet Content
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative'
              else 'lightyellow' if s == 'Neutral' else 'lightgray' for s in data_train['Sentiment_label'][:10]],  # Sent
iment
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative'
              else 'lightyellow' if s == 'Neutral' else 'lightgray' for s in data_train['Sentiment_label'][:10]],  # Sent
iment_label
            'lavender'  # Sentiment (numeric)
        ],
        align='left',
        font=dict(color='black', size=11)
    ))
])

# Update the layout
fig.update_layout(
    title='First 10 Rows of Training Data',
    width=1000,
    height=600,
)

fig.show()
```

First 10 Rows of Training Data

Tweet Content	Sentiment	Sentiment_label
is	0	Irrelevant
Nvidia on global foundries confirmed. TSMC is incapable to make enough big die for nvidia pic.twitter.com/KXTIPTNyl9	2	Neutral
Amazon doesn't deliver in my hood	1	Negative
BOOOYYYYYY And I AD CANT A WAIT!!!	3	Positive
PUBG is banned in this county	1	Negative
The latest Hat Real World of Data Daily! paper.li/victoria_holt/… Thanks to @YatesSQL by @DataOnWheels	2	Neutral
Awesome gaming night with my hubby @xtremefanatik and friends!!! @MixerRetweeter @WatchMixer @DestinyTheGame @PlayOverwatch	3	Positive
Cancelled AT&T yesterday, went to Verizon,	3	Positive
Nvidia Optimus laptop PSU Side monitor (wired directly to Nvidia GPU) has tearing.. zpr.io/H6uu3	1	Negative
Huh. Verizon just gave me 15GB of data for April, free of charge. Nice.	3	Positive

```python
import plotly.graph_objects as go

# Create a colorful table using Plotly for the test data
fig = go.Figure(data=[go.Table(
    header=dict(
        values=list(data_test.columns),
        fill_color='paleturquoise',
        align='left',
        font=dict(color='black', size=12)
    ),
    cells=dict(
```

```
        values=[data_test[k].tolist()[:5] for k in data_test.columns],  # Show first 5 rows
        fill_color=[
            'lightcyan',  # Tweet Content
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative'
             else 'lightyellow' if s == 'Neutral' else 'lightgray' for s in data_test['Sentiment_label'][:5]],  # Sentim
ent
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative'
             else 'lightyellow' if s == 'Neutral' else 'lightgray' for s in data_test['Sentiment_label'][:5]],  # Sentim
ent_label
            'lavender'  # Sentiment (numeric)
        ],
        align='left',
        font=dict(color='black', size=11)
    ))
])

# Update the layout
fig.update_layout(
    title='First 5 Rows of Test Data',
    width=1000,
    height=600,
)

# Show the figure
fig.show()
```

First 5 Rows of Test Data

Tweet Content	Sentiment	Sentiment_label
I mentioned on Facebook that I was struggling for motivation to go for a run the other day, which has been translated by Tom's great auntie as 'Hayley can't get out of bed' and told to his grandma, who now thinks I'm a lazy, terrible person 🤣	0	Irrelevant
BBC News - Amazon boss Jeff Bezos rejects claims company acted like a 'drug dealer' bbc.co.uk/news/av/busine…	2	Neutral
@Microsoft Why do I pay for WORD when it functions so poorly on my @SamsungUS Chromebook? 🙄	1	Negative
CSGO matchmaking is so full of closet hacking, it's a truly awful game.	1	Negative
Now the President is slapping Americans in the face that he really did commit an unlawful act after his acquittal! From Discover on Google vanityfair.com/news/2020/02/t…	2	Neutral

1. SVM

Support Vector Machines (SVM) are powerful machine learning algorithms that excel in text classification tasks. They're particularly effective in distinguishing between different text categories, making them valuable for applications like sentiment analysis, topic labeling, and spam detection.

How SVM Works for Text Classification

Text Preprocessing:

- Text data is cleaned and transformed into a numerical representation. This involves steps like tokenization, stop word removal, stemming, and lemmatization.

- Feature extraction techniques like TF-IDF (Term Frequency-Inverse Document Frequency) are employed to convert text into numerical vectors.

Hyperplane Creation:

- SVM aims to find the optimal hyperplane, which is a decision boundary that separates different text classes in the feature space.
- Each text document is represented as a point in this high-dimensional space based on its extracted features.

Maximizing Margin:

- SVM seeks the hyperplane that maximizes the margin between the different classes. This margin is the distance between the hyperplane and the closest data points from each class (support vectors).

Classification:

- New text documents are mapped to the same feature space.
- Their position relative to the hyperplane determines the predicted class.

Key Concepts

- Support Vectors: These are the data points closest to the hyperplane and significantly influence the model's decision boundary.
- Kernel Trick: SVM can handle non-linear relationships between features using the kernel trick, which implicitly maps data to a higher-dimensional space.
- Regularization: SVM uses regularization to prevent overfitting and improve generalization performance.

Advantages of SVM for Text Classification

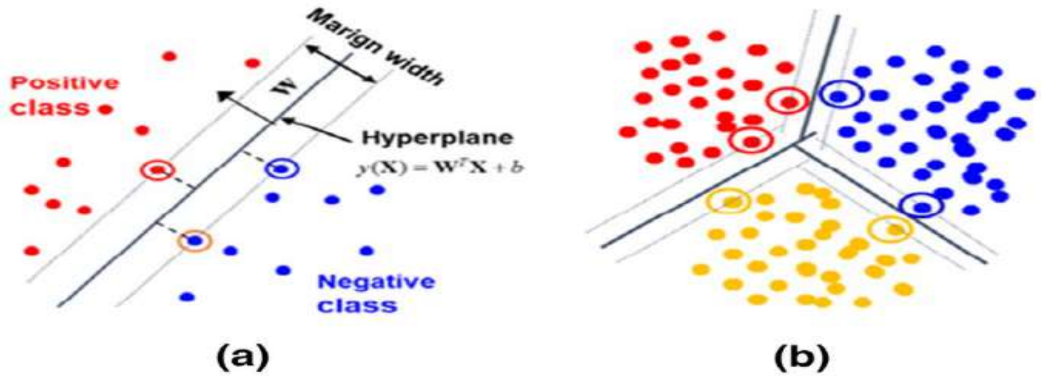
Effective with high-dimensional data: Text data often has a large number of features. SVM handles such data efficiently. Strong generalization performance: SVM tends to perform well on unseen data. Handles complex patterns: The kernel trick allows SVM to capture complex relationships between words.

Challenges and Considerations

- Computational cost: SVM can be computationally expensive for large datasets.
- Parameter tuning: Choosing the right kernel and hyperparameters requires careful experimentation.
- Feature engineering: Effective feature extraction is crucial for SVM performance.

In Conclusion

SVM is a robust algorithm for text classification, offering excellent performance and versatility. By understanding its core principles and effectively addressing its challenges, you can leverage SVM to build accurate and reliable text classification models.



```python
%%time

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score


# Extract text and labels
train_texts = data_train["Tweet Content"].tolist()
train_labels = data_train["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})
test_texts = data_test["Tweet Content"].tolist()
test_labels = data_test["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})

# Feature extraction using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # Adjust max_features as needed
train_features = vectorizer.fit_transform(train_texts)
test_features = vectorizer.transform(test_texts)

# Train the SVM model
Svm = SVC(kernel="linear")
Svm.fit(train_features, train_labels)

# Make predictions on the validation/test set
predictions = Svm.predict(test_features)
```

```python
# Calculate accuracy
accuracy = accuracy_score(test_labels, predictions)
# Print accuracy
print("Test Accuracy SVM:", accuracy)
print("\n")
```

```
Test Accuracy SVM: 0.706



CPU times: user 16 s, sys: 199 ms, total: 16.2 s
Wall time: 16.2 s
```

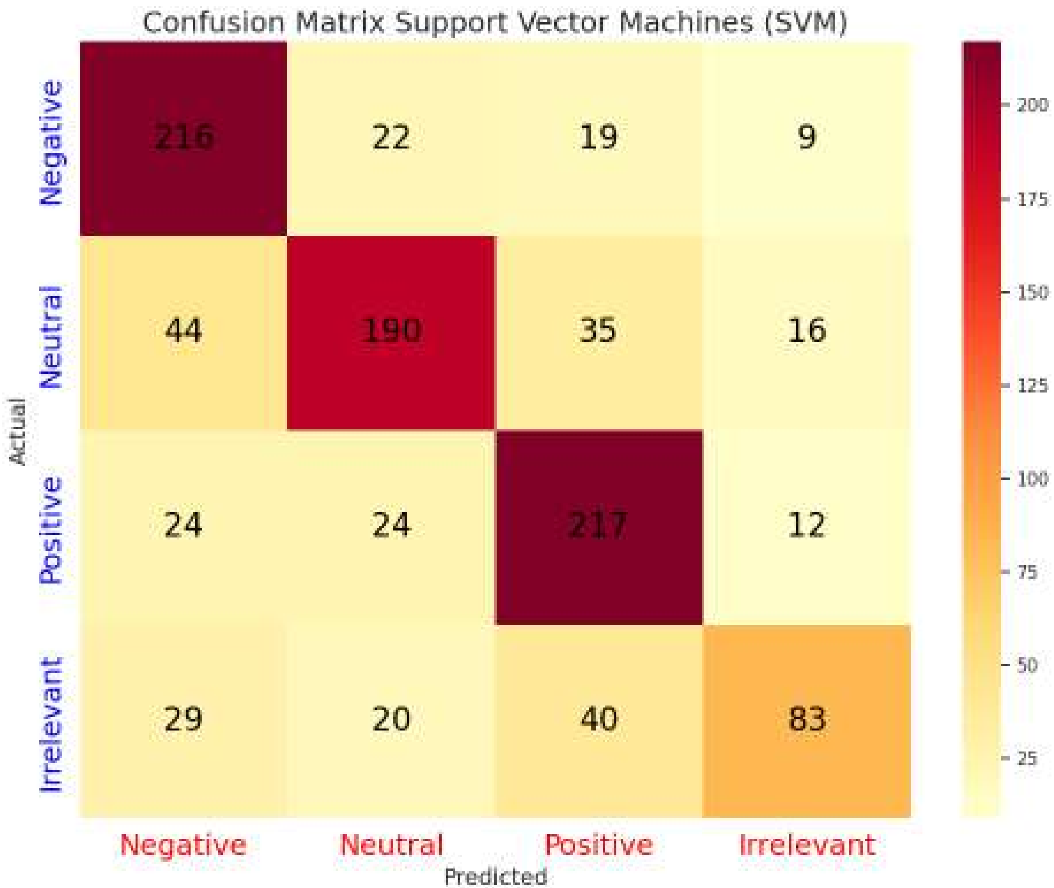
```python
print("Classification Report SVM:\n")
SVM = classification_report(test_labels, predictions, target_names=["Negative", "Neutral", "Positive", "Irrelevant"])
print(SVM)
```

```
Classification Report SVM:

              precision    recall  f1-score   support

    Negative       0.69      0.81      0.75       266
     Neutral       0.74      0.67      0.70       285
    Positive       0.70      0.78      0.74       277
  Irrelevant       0.69      0.48      0.57       172

    accuracy                           0.71      1000
   macro avg       0.71      0.69      0.69      1000
weighted avg       0.71      0.71      0.70      1000
```

```python
# Print confusion matrix
confusion_matrix_svm = confusion_matrix(test_labels, predictions)
print("Confusion Matrix:\n", confusion_matrix_svm)
```

```
Confusion Matrix:
 [[216  22  19   9]
 [ 44 190  35  16]
 [ 24  24 217  12]
 [ 29  20  40  83]]
```

```python
plt.figure(figsize=(10, 8))  # Increased figure size for better visibility
sns.heatmap(confusion_matrix_svm, annot=True, fmt='d', cmap='YlOrRd',annot_kws={"size": 20, "color": "Black"})  # Increased annotation size
plt.title('Confusion Matrix Support Vector Machines (SVM)', fontsize=18)  # Increased title size
plt.ylabel('Actual', fontsize=14)
plt.xlabel('Predicted', fontsize=14)
plt.xticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18,color = 'Red')
plt.yticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18, color = 'Blue')
plt.tight_layout()
plt.show()
```


Confusion Matrix Support Vector Machines (SVM)
Negative
Neutral
Positive
Irrelevant
Actual
216
22
19
9
44
190
35
16
24
24
217
12
29
20
40
83
Negative
Neutral
Positive
Irrelevant
Predicted
200
175
150
125
100
75
50
25

2. Naive Bayes classifiers

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem, particularly suited for text classification tasks. It's a simple yet effective method for categorizing text into different classes.

How it works:

- **Text Preprocessing:** Similar to other text classification methods, the text data undergoes preprocessing steps like tokenization, stop word removal, and stemming or lemmatization.
- **Feature Extraction:** Words or n-grams (sequences of words) are typically used as features. These features are converted into numerical representations, often using techniques like TF-IDF.
- **Probability Calculation:** Naive Bayes calculates the probability of a document belonging to a particular class based on the presence of specific words or features.
- **Bayes' Theorem Application:** The algorithm applies Bayes' theorem to calculate the posterior probability of a class given the document's features.
- **Classification:** The class with the highest probability is assigned to the document.

Key Assumption:

- The Naive Bayes algorithm makes a simplifying assumption: the occurrence of one word in a document is independent of the occurrence of other words. While this assumption is often not strictly true in natural language, it works surprisingly well in practice.

Advantages of Naive Bayes for Text Classification:

- **Simplicity:** Easy to understand and implement.
- **Efficiency:** Fast training and prediction times.
- **Effective with high-dimensional data:** Handles text data with large vocabularies well.
- **Works well with small datasets:** Can achieve reasonable performance with limited training data.

Common Use Cases:

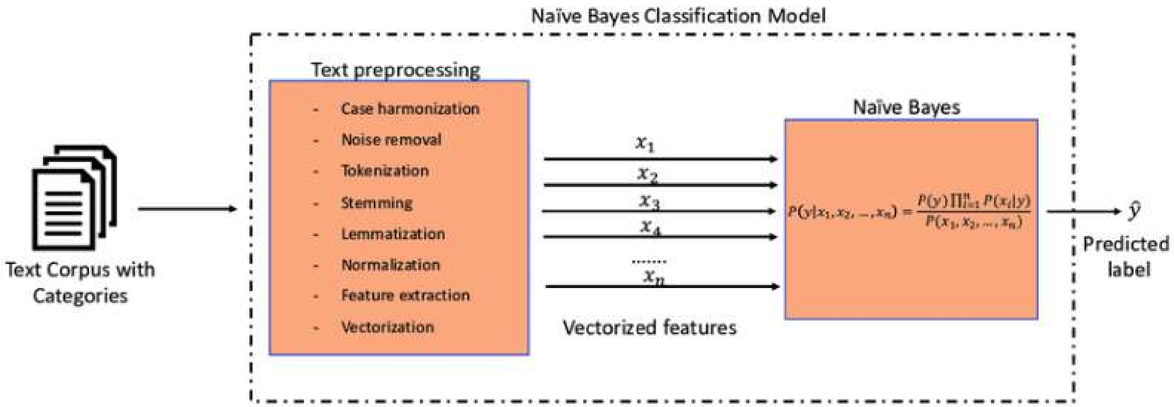
- **Spam filtering:** Classifying emails as spam or non-spam.
- **Sentiment analysis:** Determining the sentiment of text (positive, negative, neutral).
- **Topic modeling:** Assigning documents to predefined topics.
- **Author identification:** Identifying the author of a text.

Challenges:

- **Naive Bayes assumption:** The independence assumption might not hold true in all cases, affecting accuracy.
- **Zero-frequency problem:** If a word doesn't appear in a training set for a particular class, its probability becomes zero, impacting calculations. This can be addressed using techniques like Laplace smoothing.

In Summary:

Naive Bayes is a popular and efficient algorithm for text classification due to its simplicity and ability to handle high-dimensional data. While it makes a simplifying assumption about feature independence, it often performs well in practice. It's a good starting point for many text classification tasks.



Likelihood

Posterior

Prior

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Normalizing constant

$$P(B) = \sum_{Y} P(B|A)P(A)$$

```
%%time

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```python
# Extract text and labels
train_texts = data_train["Tweet Content"].tolist()
train_labels = data_train["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})
test_texts = data_test["Tweet Content"].tolist()
test_labels = data_test["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})

# Feature extraction using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # Adjust max_features as needed
train_features = vectorizer.fit_transform(train_texts)
test_features = vectorizer.transform(test_texts)

# Train the Naive Bayes model
NB = MultinomialNB()
NB.fit(train_features, train_labels)

# Make predictions on the validation/test set
predictions = NB.predict(test_features)

# Calculate accuracy
accuracy = accuracy_score(test_labels, predictions)
# Print accuracy
print("Test Accuracy Naive Bayes:", accuracy)
print("\n")
```

```
Test Accuracy Naive Bayes: 0.626
```

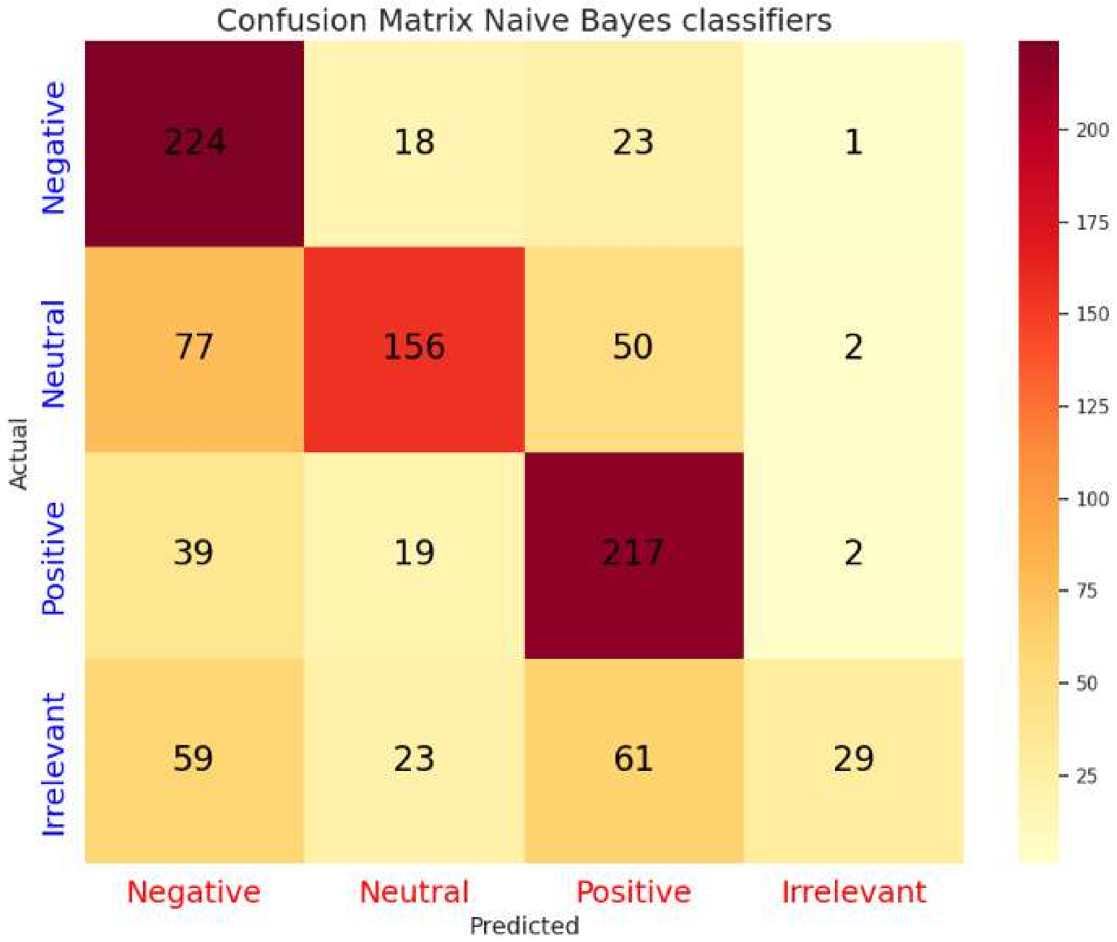
```
CPU times: user 303 ms, sys: 1.9 ms, total: 305 ms
Wall time: 307 ms
```

```python
confusion_matrix_NB = confusion_matrix(test_labels, predictions)
print("Confusion Matrix:\n", confusion_matrix_NB)
```

```
Confusion Matrix:
 [[224  18  23   1]
 [ 77 156  50   2]
 [ 39  19 217   2]
 [ 59  23  61  29]]
```

```python
plt.figure(figsize=(10, 8))  # Increased figure size for better visibility
sns.heatmap(confusion_matrix_NB, annot=True, fmt='d', cmap='YlOrRd',annot_kws={"size": 20, "color": "Bl
ack"})  # Increased annotation size
plt.title('Confusion Matrix Naive Bayes classifiers', fontsize=18)  # Increased title size
plt.ylabel('Actual', fontsize=14)
plt.xlabel('Predicted', fontsize=14)
```

```
plt.xticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18,color = 'Red')
plt.yticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18, color= 'Blue')
plt.tight_layout()
plt.show()
```



3. Logistic Regression

Logistic Regression is a statistical method for predicting the probability of a binary outcome. While it's primarily designed for binary classification, it can be extended to multi-class classification problems as well.

How it works for Text Classification:

- **Text Preprocessing:** Similar to other text classification methods, the text data is cleaned and transformed into numerical features. This involves tokenization, stop word removal, stemming, and lemmatization.
- **Feature Extraction:** Text is converted into numerical features using techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings.
- **Model Training:** The Logistic Regression model is trained on the numerical representation of the text data and corresponding class labels. The model learns the relationship between the features and the target class.
- **Probability Estimation:** For a new text document, the model calculates the probability of the document belonging to each class.
- **Classification:** The class with the highest probability is assigned to the document.

Key Points:

- **Probability Estimation:** Unlike some other classification algorithms, Logistic Regression provides probability estimates for each class, which can be useful in certain applications.
- **Multi-class Classification:** For problems with more than two classes, techniques like one-vs-rest or multinomial logistic regression can be used.

- **Interpretability:** The coefficients of the Logistic Regression model can provide insights into the importance of different features in the classification process.

Advantages:

- **Simplicity:** Relatively easy to understand and implement.
- **Efficiency:** Can be computationally efficient, especially for smaller datasets.
- **Probabilistic Output:** Provides probability estimates for each class, which can be valuable in certain applications.

Challenges:

- **Feature Engineering:** Effective feature extraction is crucial for model performance.
- **Overfitting:** Can be prone to overfitting if not regularized properly.

Applications:

- Sentiment analysis
- Spam detection
- Topic classification
- Customer review categorization

In essence, Logistic Regression offers a balance of simplicity, interpretability, and performance for text classification tasks. While it may not always outperform more complex models, it's often a good starting point and can provide valuable insights into the data.

```python
%%time

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score


# Extract text and labels
train_texts = data_train["Tweet Content"].tolist()
train_labels = data_train["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})
test_texts = data_test["Tweet Content"].tolist()
test_labels = data_test["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})

# Feature extraction using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # Adjust max_features as needed
train_features = vectorizer.fit_transform(train_texts)
test_features = vectorizer.transform(test_texts)


# Train the Logistic Regression model
Lr = LogisticRegression(multi_class='multinomial', solver='lbfgs')  # Multiclass with L-BFGS solver

# Fit the model
Lr.fit(train_features, train_labels)
```

```
# Make predictions on the validation/test set
predictions = Lr.predict(test_features)

# Calculate accuracy
accuracy = accuracy_score(test_labels, predictions)

# Print accuracy
print("Test Accuracy Logistic Regression:", accuracy)
print("\n")
```

```
Test Accuracy Logistic Regression: 0.692



CPU times: user 3.86 s, sys: 3.95 s, total: 7.82 s
Wall time: 2.9 s
```

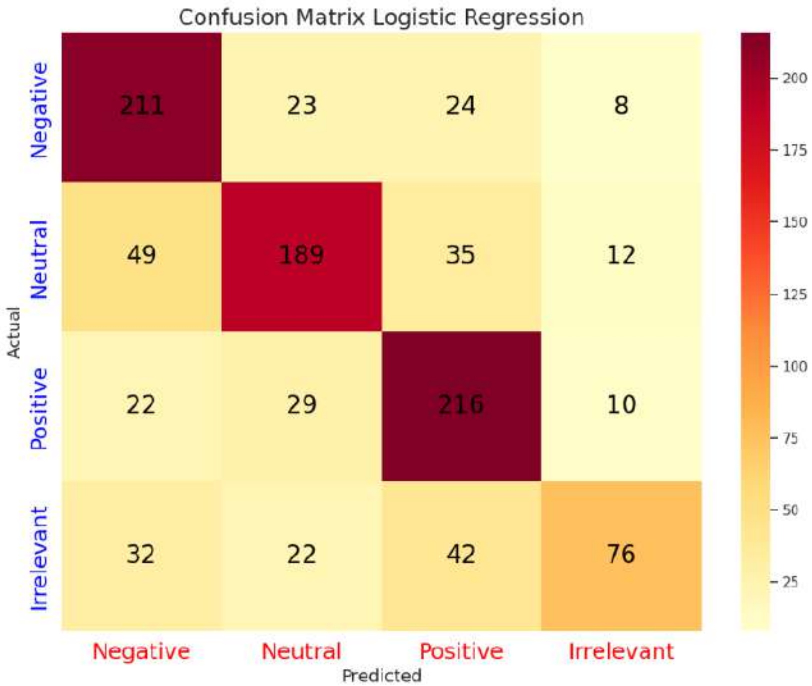
```
print("Classification Report Logistic Regression:\n")
LR = classification_report(test_labels, predictions, target_names=["Negative", "Neutral", "Positive", "Irrele
vant"])
print(LR)
```

```
Classification Report Logistic Regression:

              precision    recall  f1-score   support

    Negative       0.67      0.79      0.73       266
     Neutral       0.72      0.66      0.69       285
    Positive       0.68      0.78      0.73       277
  Irrelevant       0.72      0.44      0.55       172

    accuracy                           0.69      1000
   macro avg       0.70      0.67      0.67      1000
weighted avg       0.70      0.69      0.69      1000
```

```
# Print confusion matrix
confusion_matrix_lr = confusion_matrix(test_labels, predictions)
print("Confusion Matrix:\n\n", confusion_matrix_lr)
```

```
Confusion Matrix:

 [[211  23  24   8]
 [ 49 189  35  12]
 [ 22  29 216  10]
 [ 32  22  42  76]]
```

```
plt.figure(figsize=(10, 8))  # Increased figure size for better visibility
sns.heatmap(confusion_matrix_lr, annot=True, fmt='d', cmap='YlOrRd',annot_kws={"size": 20, "color": "Bla
ck"})  # Increased annotation size
plt.title('Confusion Matrix Logistic Regression', fontsize=18)  # Increased title size
plt.ylabel('Actual', fontsize=14)
plt.xlabel('Predicted', fontsize=14)
plt.xticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18,color = 'Red')
plt.yticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18, color= 'Blue')
plt.tight_layout()
plt.show()
```



4. Decision Trees

```
%%time

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Extract text and labels
train_texts = data_train["Tweet Content"].tolist()
train_labels = data_train["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})
test_texts = data_test["Tweet Content"].tolist()
test_labels = data_test["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})

# Feature extraction using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # Adjust max_features as needed
train_features = vectorizer.fit_transform(train_texts)
test_features = vectorizer.transform(test_texts)

# Train the Decision Tree Classifier model
Dt = DecisionTreeClassifier(random_state=42)  # Set random state for reproducibility

# Fit the model
Dt.fit(train_features, train_labels)
```

```python
predictions = Dt.predict(test_features)

accuracy = accuracy_score(test_labels, predictions)
print("Test Accuracy Decision Tree:", accuracy)
print("\n")
```

```
Test Accuracy Decision Tree: 0.605



CPU times: user 2.28 s, sys: 6.46 ms, total: 2.29 s
Wall time: 2.32 s
```

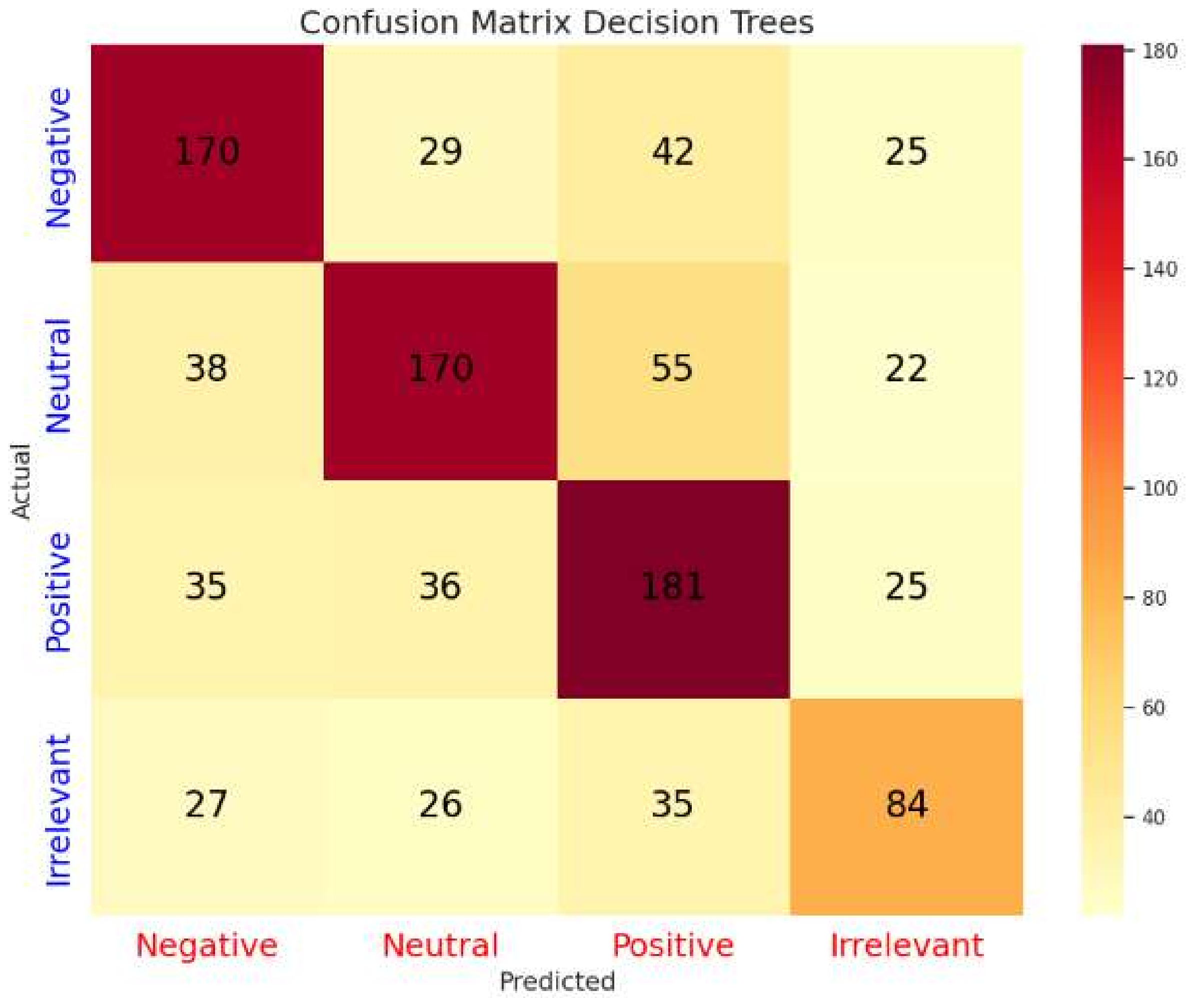
```python
print("Classification Report Decision Trees:\n")
Dt = classification_report(test_labels, predictions, target_names=["Negative", "Neutral", "Positive", "Irrele
vant"])
print(Dt)
```

```
Classification Report Decision Trees:

              precision    recall  f1-score   support

    Negative       0.63      0.64      0.63       266
     Neutral       0.65      0.60      0.62       285
    Positive       0.58      0.65      0.61       277
  Irrelevant       0.54      0.49      0.51       172

    accuracy                           0.60      1000
   macro avg       0.60      0.59      0.60      1000
weighted avg       0.61      0.60      0.60      1000
```

```python
# Print confusion matrix
confusion_matrix_dt = confusion_matrix(test_labels, predictions)
print("Confusion Matrix:\n\n", confusion_matrix_dt)
```

```
Confusion Matrix:


 [[170  29  42  25]
 [ 38 170  55  22]
 [ 35  36 181  25]
 [ 27  26  35  84]]
```

```python
plt.figure(figsize=(10, 8))  # Increased figure size for better visibility
sns.heatmap(confusion_matrix_dt, annot=True, fmt='d', cmap='YlOrRd',annot_kws={"size":
20, "color": "Black"})  # Increased annotation size
plt.title('Confusion Matrix Decision Trees', fontsize=18)  # Increased title size
plt.ylabel('Actual', fontsize=14)
plt.xlabel('Predicted', fontsize=14)
plt.xticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18,color
= 'Red')
plt.yticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18, color
= 'Blue')
plt.tight_layout()
plt.show()
```



Confusion Matrix Decision Trees
Negative
Neutral
Positive
Irrelevant
170
29
42
25
38
170
55
22
35
36
181
25
27
26
35
84
Actual
Negative
Neutral
Positive
Irrelevant
Predicted
180
160
140
120
100
80
60
40

5. Random Forests

```
%%time

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Extract text and labels
train_texts = data_train["Tweet Content"].tolist()
train_labels = data_train["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})
test_texts = data_test["Tweet Content"].tolist()
test_labels = data_test["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})

# Feature extraction using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # Adjust max_features as needed
train_features = vectorizer.fit_transform(train_texts)
test_features = vectorizer.transform(test_texts)

# Train the Random Forest Classifier model
Rf = RandomForestClassifier(n_estimators=100, random_state=42)  # Adjust n_estimators as needed

Rf.fit(train_features, train_labels)
predictions = Rf.predict(test_features)
```

```
accuracy = accuracy_score(test_labels, predictions)
print("Test Accuracy Random Forest:", accuracy)
print("\n")
```

```
Test Accuracy Random Forest: 0.73



CPU times: user 13.8 s, sys: 38.2 ms, total: 13.9 s
Wall time: 13.9 s
```

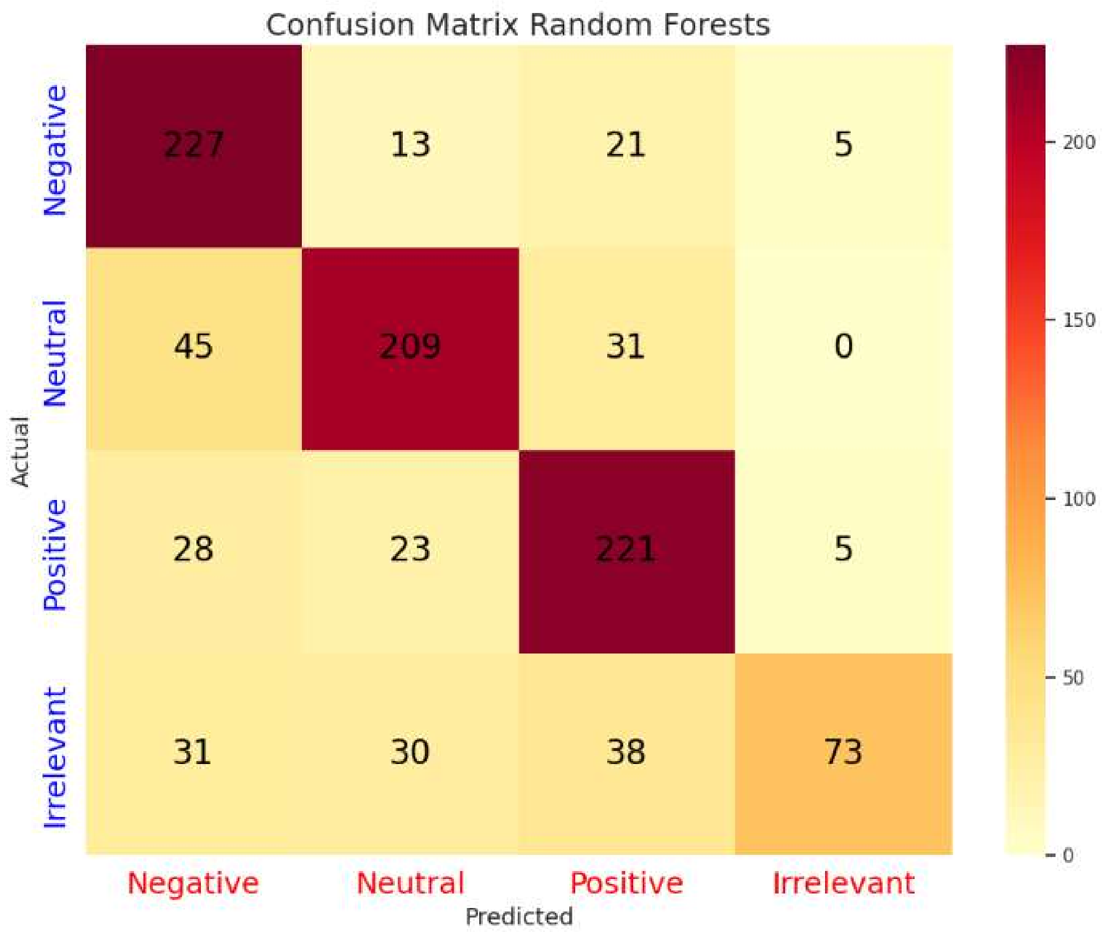
```
print("Classification Report Random Forests:\n")
Rf = classification_report(test_labels, predictions, target_names=["Negative", "Neutral", "Positive", "Irrele
vant"])
print(Rf)
```

```
Classification Report Random Forests:

              precision    recall  f1-score   support

    Negative       0.69      0.85      0.76       266
     Neutral       0.76      0.73      0.75       285
    Positive       0.71      0.80      0.75       277
  Irrelevant       0.88      0.42      0.57       172

    accuracy                           0.73      1000
   macro avg       0.76      0.70      0.71      1000
weighted avg       0.75      0.73      0.72      1000
```

```python
# Print confusion matrix
confusion_matrix_rf = confusion_matrix(test_labels, predictions)
print("Confusion Matrix:\n\n", confusion_matrix_rf)
```

```
Confusion Matrix:

 [[227  13  21   5]
 [ 45 209  31   0]
 [ 28  23 221   5]
 [ 31  30  38  73]]
```

```python
plt.figure(figsize=(10, 8))  # Increased figure size for better visibility
sns.heatmap(confusion_matrix_rf, annot=True, fmt='d', cmap='YlOrRd',annot_kws={"size": 20, "color": "Black"})  # Increased annotation size
plt.title('Confusion Matrix Random Forests', fontsize=18)  # Increased title size
plt.ylabel('Actual', fontsize=14)
plt.xlabel('Predicted', fontsize=14)
plt.xticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18,color = 'Red')
plt.yticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18, color = 'Blue')
plt.tight_layout()
plt.show()
```


Confusion Matrix Random Forests
Negative
Neutral
Positive
Irrelevant
Actual
227
13
21
5
45
209
31
0
28
23
221
5
31
30
38
73
Negative
Neutral
Positive
Irrelevant
Predicted
200
150
100
50
0

6. Gradient Boosting Machines (XGBoost)

```
%%time

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Extract text and labels
train_texts = data_train["Tweet Content"].tolist()
train_labels = data_train["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})
test_texts = data_test["Tweet Content"].tolist()
test_labels = data_test["Sentiment_label"].map({"Positive": 2, "Negative": 0, "Neutral": 1, "Irrelevant": 3})

# Feature extraction using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # Adjust max_features as needed
train_features = vectorizer.fit_transform(train_texts)
test_features = vectorizer.transform(test_texts)

# Convert to DMatrix format for XGBoost
dtrain = xgb.DMatrix(train_features, label=train_labels)
dtest = xgb.DMatrix(test_features, label=test_labels)

# Set XGBoost parameters
```

```
params = {
    'objective': 'multi:softmax',  # For multi-class classification
    'num_class': 4,  # Number of classes
    'eta': 0.3,  # Learning rate
    'max_depth': 6,  # Maximum depth of trees
    'eval_metric': 'mlogloss'  # Evaluation metric
}

# Train the XGBoost model
Xgb = xgb.train(params, dtrain, num_boost_round=10)  # Adjust num_boost_round as needed

# Make predictions on the validation/test set
predictions = Xgb.predict(dtest)

# Calculate accuracy
accuracy = accuracy_score(test_labels, predictions)
# Print accuracy
print("Test Accuracy XGBoost:", accuracy)
print("\n")
```

```
Test Accuracy XGBoost: 0.515


CPU times: user 8.31 s, sys: 89.3 ms, total: 8.4 s
Wall time: 2.53 s
```

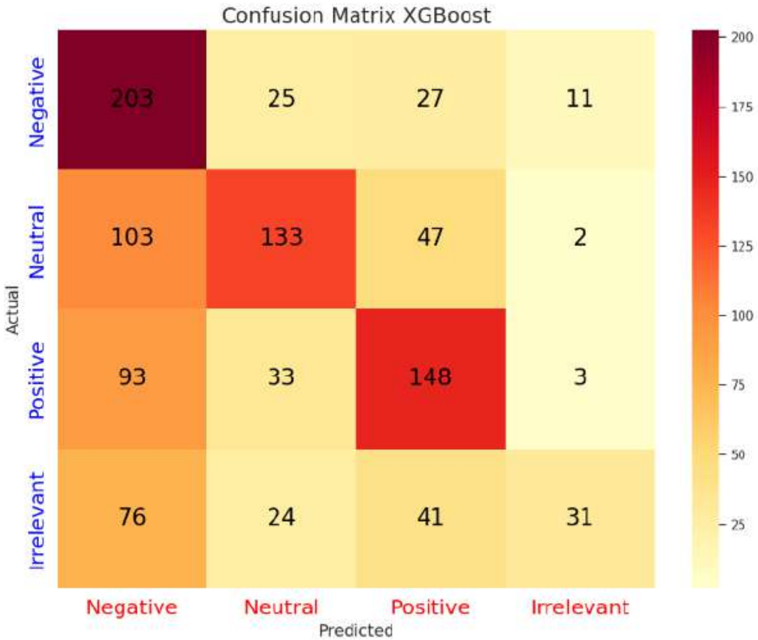
```python
print("Classification Report XGBoost:\n")
Xgb = classification_report(test_labels, predictions, target_names=["Negative", "Neutral", "Positive", "Irrel
evant"])
print(Xgb)
```

```
Classification Report XGBoost:

              precision    recall  f1-score   support

    Negative       0.43      0.76      0.55       266
     Neutral       0.62      0.47      0.53       285
    Positive       0.56      0.53      0.55       277
  Irrelevant       0.66      0.18      0.28       172

    accuracy                           0.52      1000
   macro avg       0.57      0.49      0.48      1000
weighted avg       0.56      0.52      0.50      1000
```

```python
confusion_matrix_xgb = confusion_matrix(test_labels, predictions)
print("Confusion Matrix:\n\n", confusion_matrix_xgb)
```

```
Confusion Matrix:

 [[203  25  27  11]
 [103 133  47   2]
 [ 93  33 148   3]
 [ 76  24  41  31]]
```

```
plt.figure(figsize=(10, 8))  # Increased figure size for better visibility
sns.heatmap(confusion_matrix_xgb, annot=True, fmt='d', cmap='YlOrRd',annot_kws={"size": 20, "color": "B
lack"})  # Increased annotation size
plt.title('Confusion Matrix XGBoost', fontsize=18)  # Increased title size
plt.ylabel('Actual', fontsize=14)
plt.xlabel('Predicted', fontsize=14)
plt.xticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18,color = 'Red')
plt.yticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18, color= 'Blue')
plt.tight_layout()
plt.show()
```



7. Gradient Boosting Machines (LightGBM)

```
%%time

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import LabelEncoder

# Extract text and labels
train_texts = data_train["Tweet Content"].tolist()
test_texts = data_test["Tweet Content"].tolist()

# Use LabelEncoder to convert categorical labels to numerical
le = LabelEncoder()
train_labels = le.fit_transform(data_train["Sentiment_label"])
test_labels = le.transform(data_test["Sentiment_label"])

# Feature extraction using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # Adjust max_features as needed
train_features = vectorizer.fit_transform(train_texts)
test_features = vectorizer.transform(test_texts)

# Create LightGBM datasets
train_data = lgb.Dataset(train_features, label=train_labels)
test_data = lgb.Dataset(test_features, label=test_labels)
```

```python
# Set LightGBM parameters
params = {
    'objective': 'multiclass',
    'num_class': len(le.classes_),
    'metric': 'multi_logloss',
    'learning_rate': 0.1,
    'max_depth': -1,
    'num_leaves': 31
}

# Train the LightGBM model
Lgbm = lgb.train(
    params,
    train_data,
    valid_sets=[test_data],
    num_boost_round=100,
    callbacks=[lgb.log_evaluation(period=100)]
)

# Make predictions on the validation/test set
predictions = Lgbm.predict(test_features)
predictions = predictions.argmax(axis=1)  # Convert probabilities to class labels

# Calculate accuracy
accuracy = accuracy_score(test_labels, predictions)
print("\n")
print("\033[91mTest Accuracy LightGBM:\033[0m", accuracy)
print("\n")
```

```
Test Accuracy LightGBM: 0.672
```

```
CPU times: user 21.8 s, sys: 296 ms, total: 22.1 s
Wall time: 8.66 s
```

```
print("Classification Report LightGBM:\n")
Lgbm = classification_report(test_labels, predictions, target_names=["Negative", "Neutral",
"Positive", "Irrelevant"])
print(Lgbm)
```

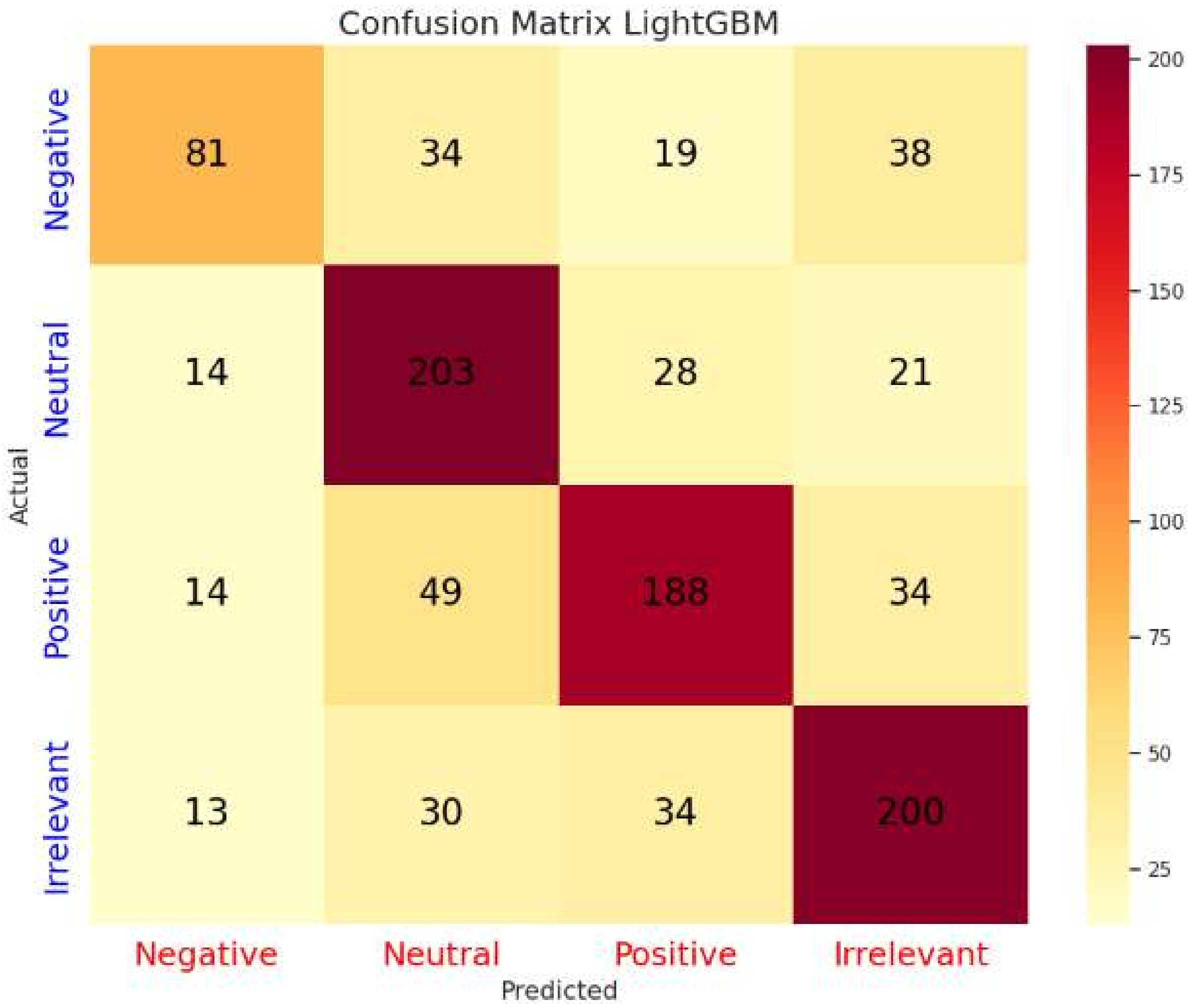
```
Classification Report LightGBM:

              precision    recall  f1-score   support

    Negative       0.66      0.47      0.55       172
     Neutral       0.64      0.76      0.70       266
    Positive       0.70      0.66      0.68       285
  Irrelevant       0.68      0.72      0.70       277

    accuracy                           0.67      1000
   macro avg       0.67      0.65      0.66      1000
weighted avg       0.67      0.67      0.67      1000
```

```python
confusion_matrix_lgbm = confusion_matrix(test_labels, predictions)
print("Confusion Matrix:\n\n", confusion_matrix_lgbm)
```

```
Confusion Matrix:


 [[ 81  34  19  38]
 [ 14 203  28  21]
 [ 14  49 188  34]
 [ 13  30  34 200]]
```

```python
plt.figure(figsize=(10, 8))  # Increased figure size for better visibility
sns.heatmap(confusion_matrix_lgbm, annot=True, fmt='d', cmap='YlOrRd',annot_kws={"size": 20, "color": "Black"})  # Increased annotation size
plt.title('Confusion Matrix LightGBM', fontsize=18)  # Increased title size
plt.ylabel('Actual', fontsize=14)
plt.xlabel('Predicted', fontsize=14)
plt.xticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18,color = 'Red')
plt.yticks([0.5, 1.5, 2.5, 3.5], ['Negative', 'Neutral', 'Positive', 'Irrelevant'], fontsize=18, color = 'Blue')
plt.tight_layout()
```

```
plt.show()
```

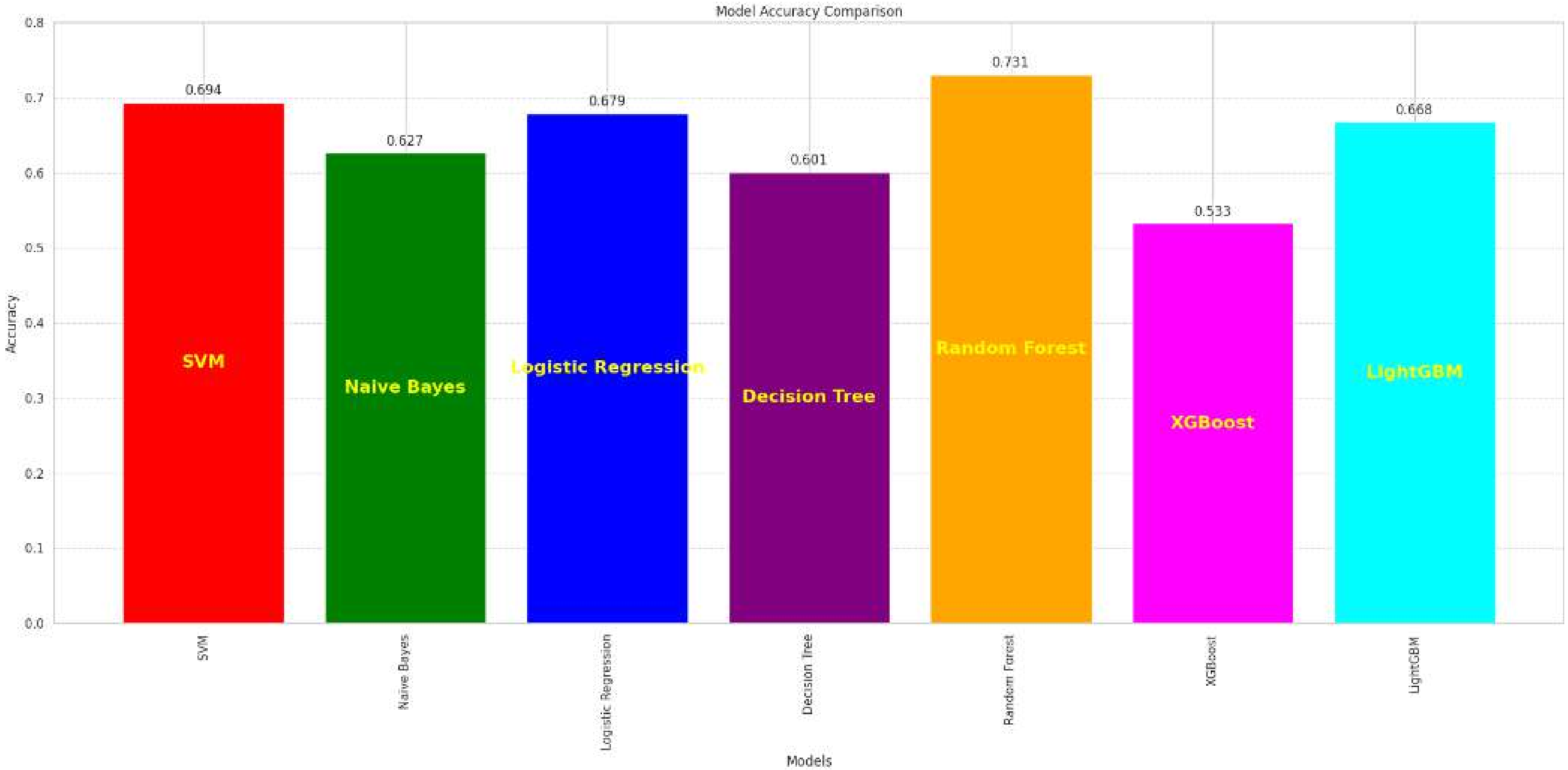


Confusion Matrix LightGBM
Negative
Neutral
Positive
Irrelevant
Actual
81
34
19
38
14
203
28
21
14
49
188
34
13
30
34
200
Negative
Neutral
Positive
Irrelevant
Predicted
200
175
150
125
100
75
50
25

Model	Accuracy
Test Accuracy SVM:	0.694
Test Accuracy Naive Bayes:	0.627
Test Accuracy Logistic Regression:	0.679
Test Accuracy Decision Tree:	0.601
Test Accuracy Random Forest:	0.731
Test Accuracy XGBoost:	0.533
Test Accuracy LightGBM:	0.668

```
import matplotlib.pyplot as plt

from pylab import rcParams
rcParams['figure.figsize'] = 20, 10

# Model names and their corresponding accuracies
models = ['SVM', 'Naive Bayes', 'Logistic Regression', 'Decision Tree', 'Random Forest', 'XGBoost', 'LightG
BM']
accuracies = [0.694, 0.627, 0.679,  0.601, 0.731, 0.533, 0.668]

# Create a list of different colors for the bars
colors = ['red', 'green', 'blue', 'purple', 'orange', 'magenta', 'cyan']

# Create the bar graph
plt.bar(models, accuracies, color=colors)
```

```python
# Add value labels on top of each bar with white color
for index, (value, model_name) in enumerate(zip(accuracies, models)):
    plt.text(index, value + 0.01, str(value), ha='center', va='bottom', color='white', fontsize=12,fontweight='
bold')

    # Add model names within the bars with white color and adjust vertical alignment
    plt.text(index, value / 2, model_name, ha='center', va='center', color='Yellow', fontsize=16, fontweight='
bold')  # Shorten model names for better fit

# Add value labels on top of each bar
for index, value in enumerate(accuracies):
    plt.text(index, value + 0.01, str(value), ha='center')



plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')

plt.ylim([0, .8])
plt.xticks(rotation=90)
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add a faint grid for better readability

plt.tight_layout()  # Adjust spacing to prevent overlapping elements
plt.show()
```



Created By: Syed Afroz Ali