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())
```

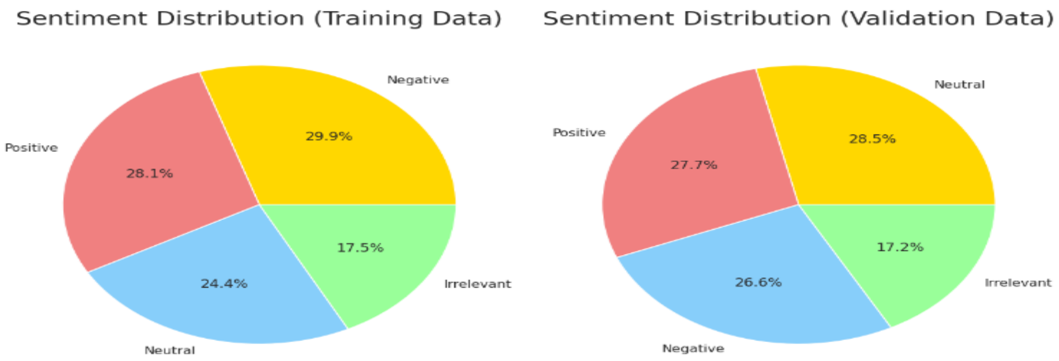
```
Tweet ID         0
Entity           0
Sentiment        0
Tweet Content    0
dtype: int64

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

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_c
ounts_validation], axis=1)
combined_counts.fillna(0, inplace=True)
combined_counts.columns = ['Test Data', 'Validation Data']  combine
d_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.inde
x, 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_valid
ation.index, autopct='%1.1f%%', colors=['gold', 'lightcoral', 'lightsky
blue','#99FF99'])
ax2.set_title('Sentiment Distribution (Validation Data)', fontsize=20)
plt.tight_layout()
plt.show()
```

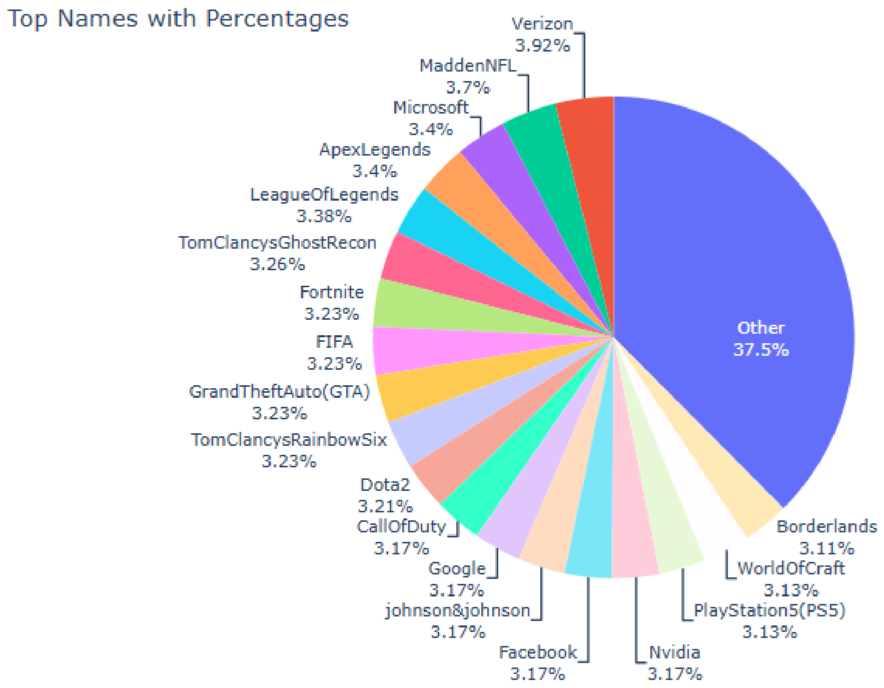


```python
# Calculate the value counts of 'Entity'
entity_counts = train['Entity'].value_counts()
top_names = entity_counts.head(19)

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']]
```

```
# 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['Se
ntiment_label'][:10]],  # Sentiment
            ['lightgreen' if s == 'Positive' else 'lightpink' if s == 'Negative'
             else 'lightyellow' if s == 'Neutral' else 'lightgray' for s in data_train['Se
ntiment_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

```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]],  # 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

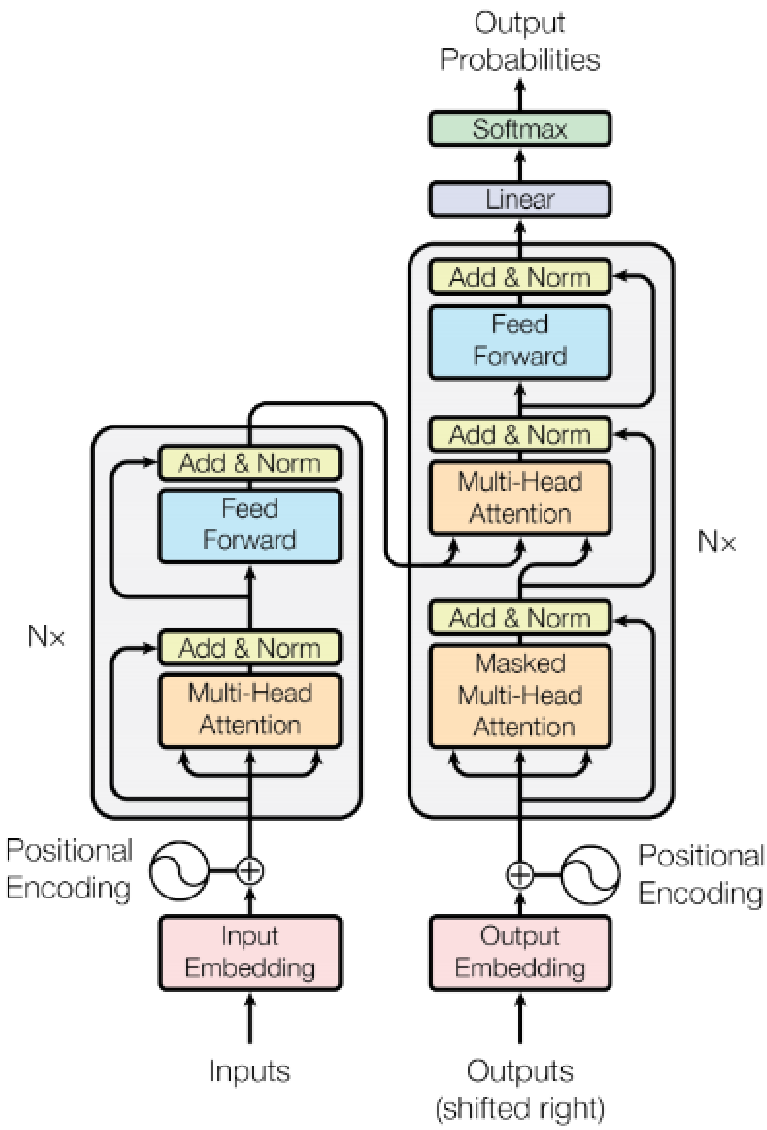
- **Strong performance:** BERT has achieved state-of-the-art results on a wide range of NLP tasks, including question answering, text classification, named entity recognition, and more.

- **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
```

```python
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
```

```python
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())
```

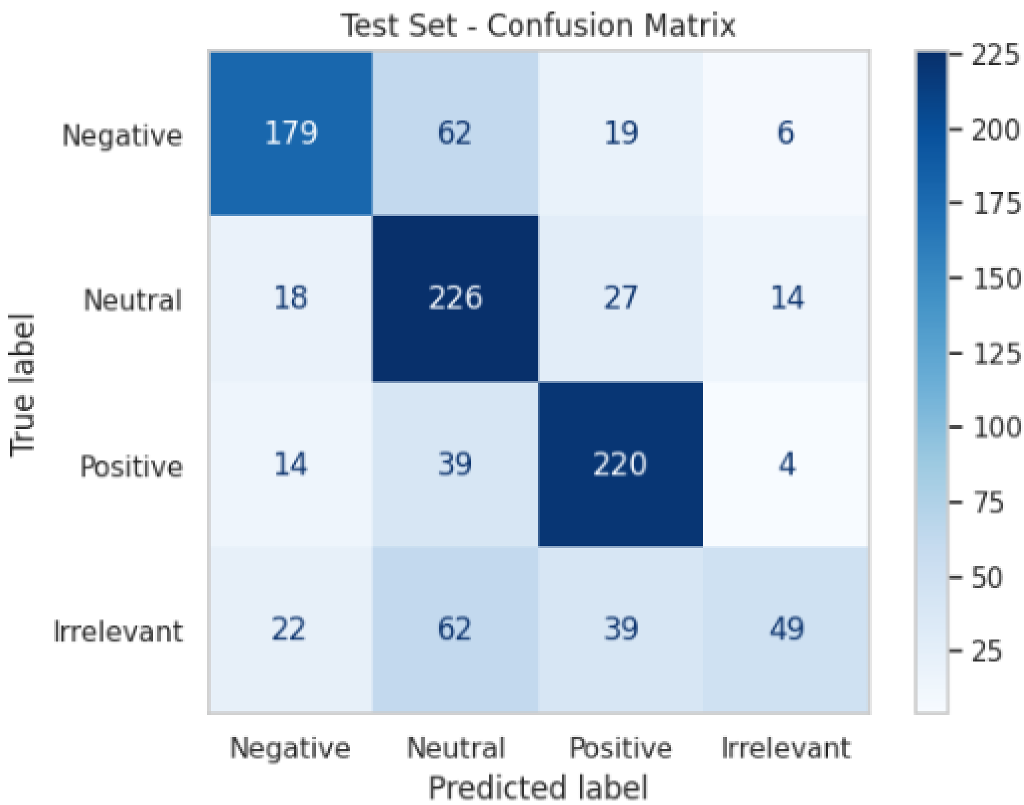
```
    accuracy = accuracy_score(test_true, test_preds)
    print(f'Epoch {epoch + 1}/{num_epochs}, Test Accuracy: {accuracy:.4f}')


# 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
```

```
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)
```

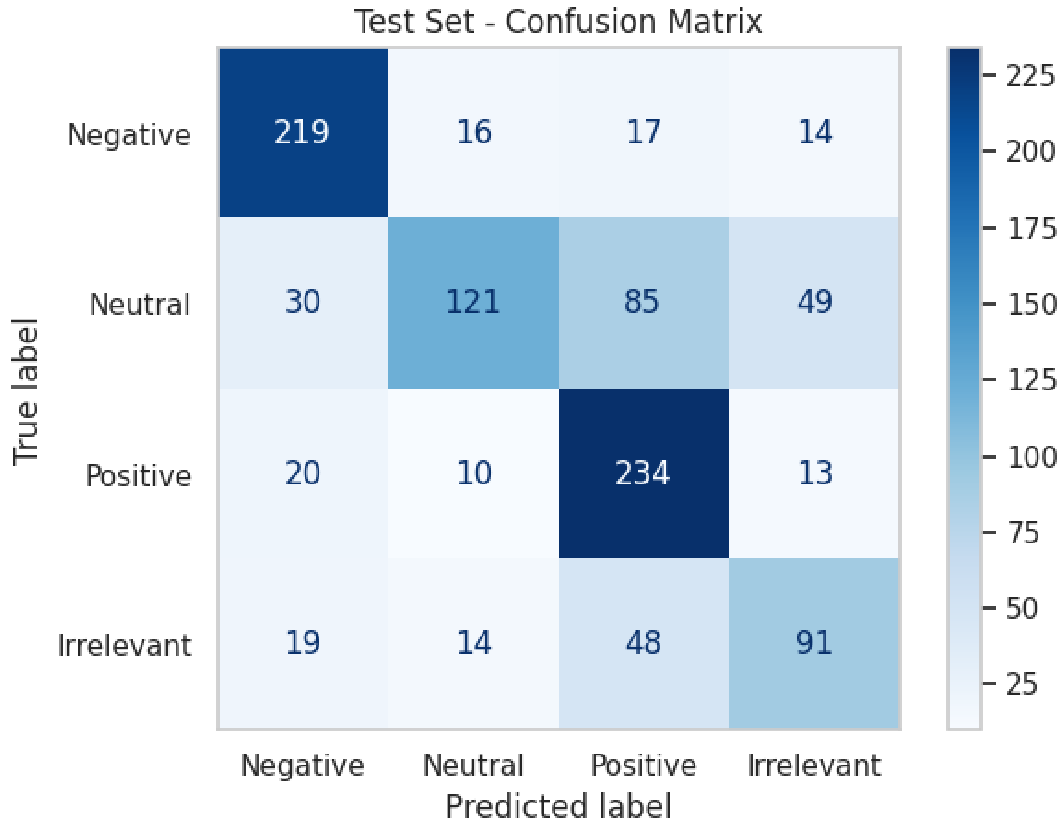
```
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# 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')
```

```python
# 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
```

```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_RoBERTa = confusion_matrix(test_true_encoded, test_preds, labels=labels)

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

```python
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.

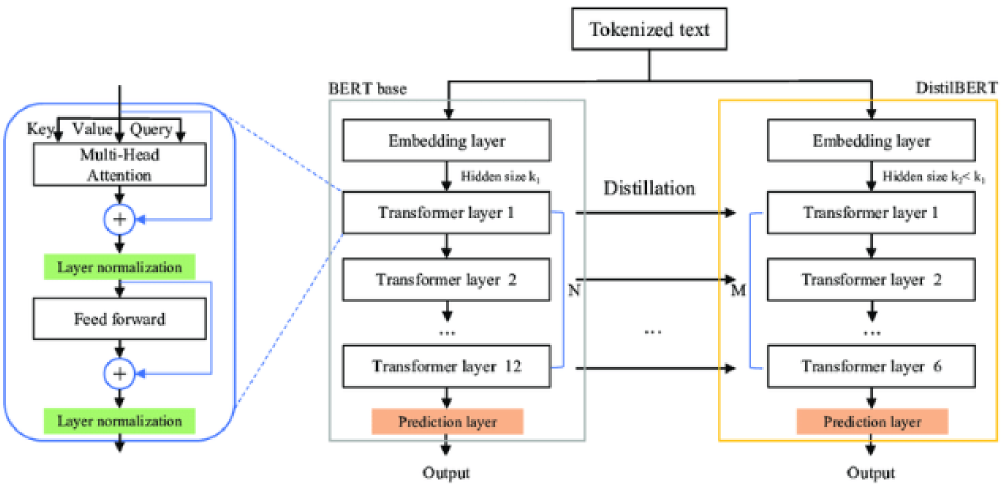
Advantages

- **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, DistilBertForSequenceClassi
fication, 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()
```

```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)
```

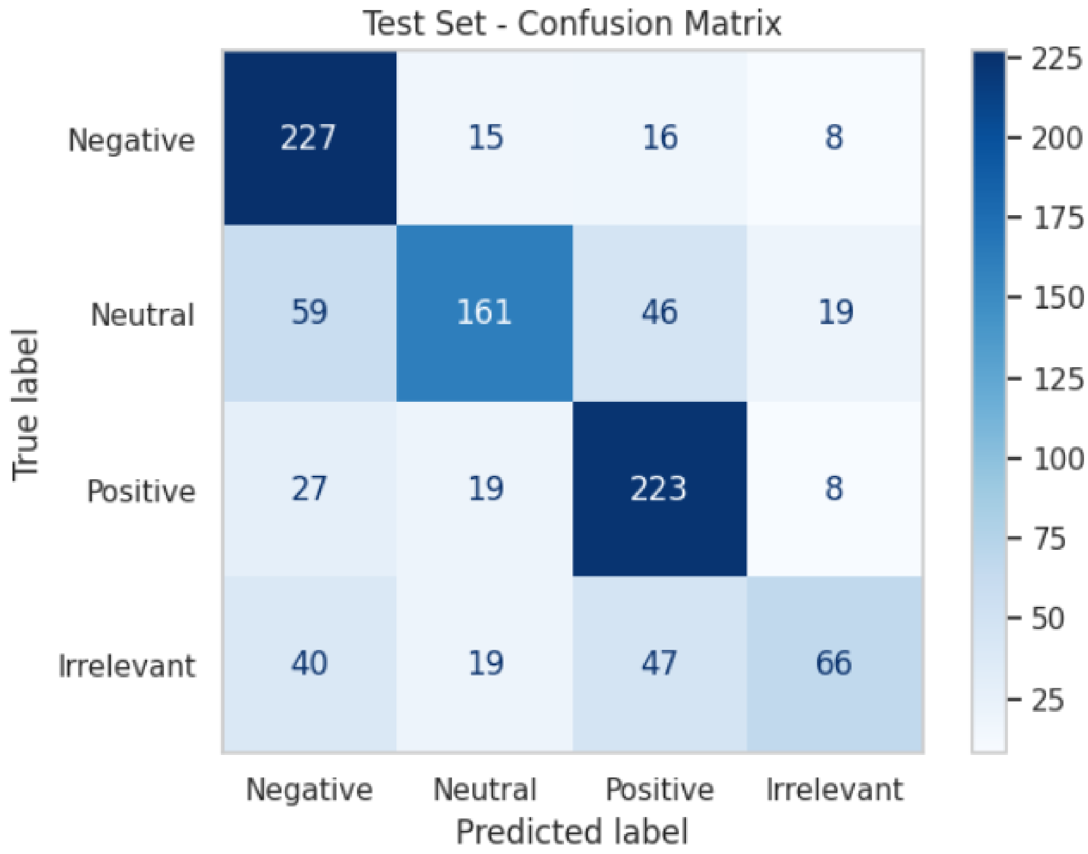
```
# Initialize the model DistilBERT
model_DistilBERT = DistilBertForSequenceClassification.from_pretrained('
distilbert-base-uncased', num_labels=4)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model_DistilBERT.to(device)

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_mas
k, 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_m
ask)
            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
```

```
print(classification_report(test_true, test_preds, target_names=['Neg
ative', '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
```

```
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_p
reds, labels=labels)

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

```
from sklearn.metrics import classification_report, confusion_matrix, Confu
sionMatrixDisplay
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()
```

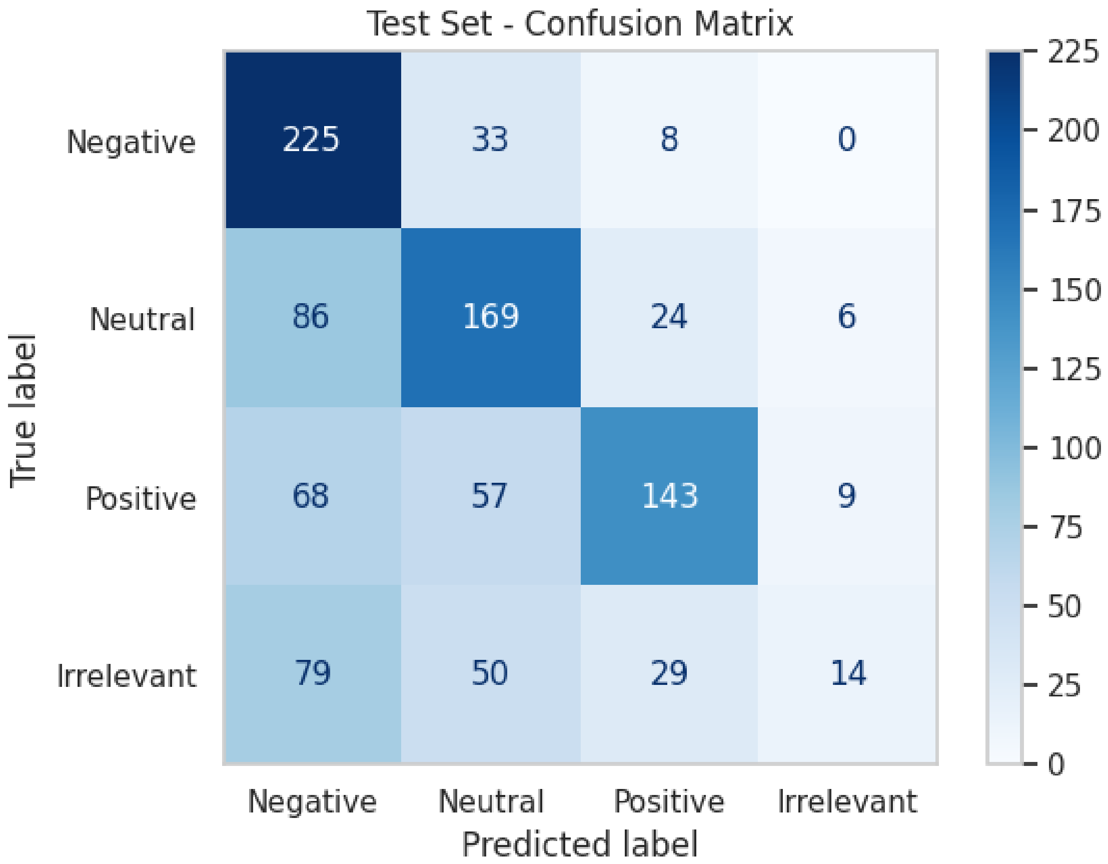
```python
    for batch in train_loader:
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model_ALBERT(input_ids, attention_mask=attention_mask, labels=labels)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


```python
# 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)
```

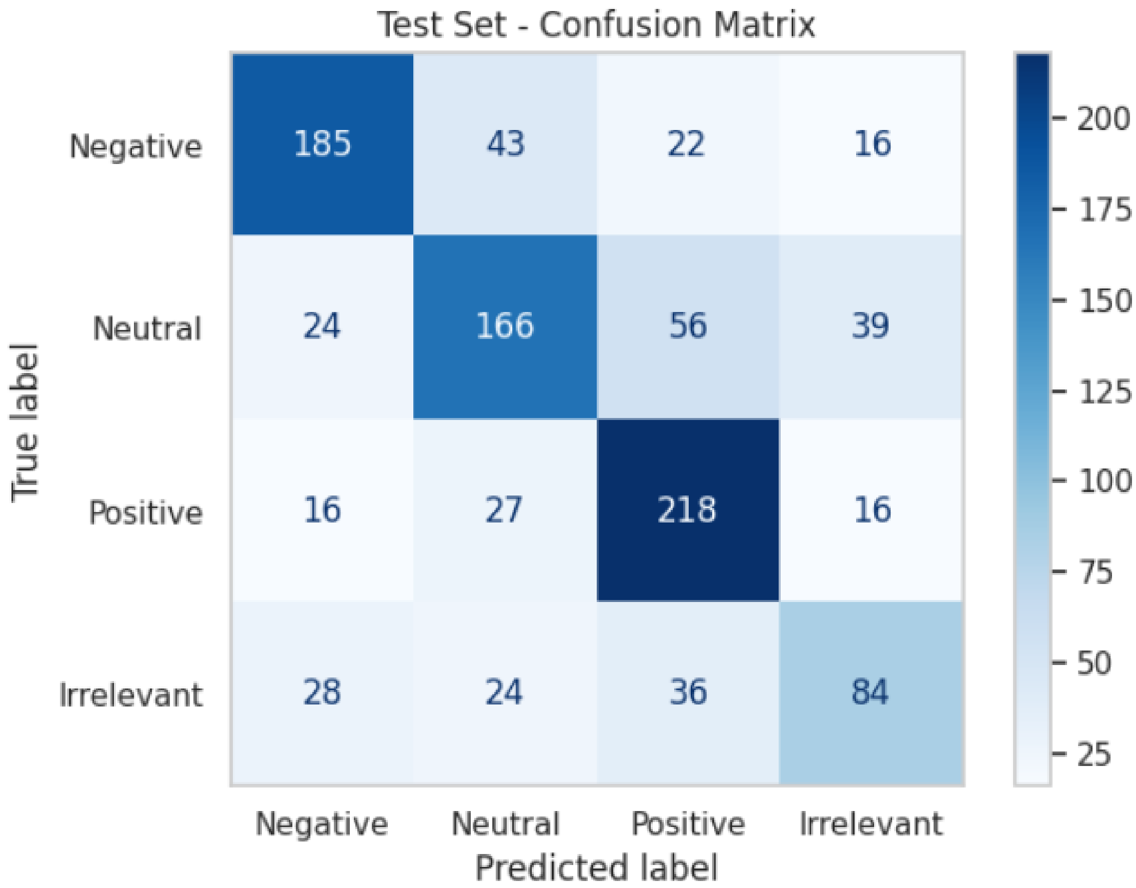
```
# Training loop
num_epochs = 3

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
```

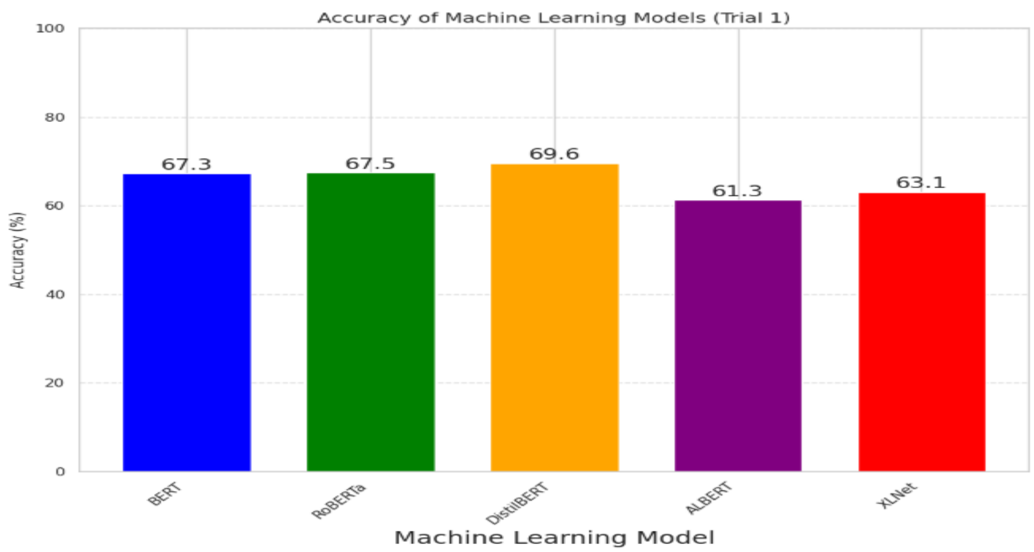
```python
# Assuming test_true and test_preds are defined
from sklearn.metrics import confusion_matrix

# 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
```

```python
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
```

```
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