SENTIMENT ANALYSIS FOR MARKETING USING MACHINE LEARNING

TEAM MEMBERS

310821104003-ABINAYA TS
310821104034-HARINI M
310821104042-JOTHIKA K
310821104043-JULIYA A

Phase-2 INNOVATION

Project:Sentiment Analysis for Marketing

INTRODUCTION:

Fine-tuning pre-trained sentiment analysis models is a powerful technique for improving the accuracy of sentiment predictions. Pre-trained models like BERT and RoBERTa have been trained on massive datasets of text and code, and they have learned to represent language in a way that is useful for many different NLP tasks, including sentiment analysis.

PRE-TRAINED MODELS:

Fine-tuning involves taking a pre-trained model and training it further on a smaller dataset of labeled data specific to your task. This allows the model to learn the nuances of your dataset and improve its performance on your task.

❖ Here are some advanced techniques for fine-tuning pre-trained sentiment analysis models:

- **Use a domain-specific dataset:** If you have a dataset of labeled data that is specific to your domain, such as product reviews or social media posts, using this dataset for fine-tuning will likely improve the model's performance on your task.
- **Use transfer learning:** If you don't have a domain-specific dataset, you can use a pre-trained model that has been trained on a similar domain. This is known as transfer learning. For example, if you are doing sentiment analysis on product reviews, you could fine-tune a model that has been pre-trained on a dataset of product reviews.
- **Freeze the lower layers:** When fine-tuning a pre-trained model, it is common to freeze the lower layers of the model. These layers have learned general representations of language that are useful for many different tasks, so freezing them prevents them from being overwritten by the fine-tuning process.
- **Use a smaller learning rate:** Pre-trained models have been trained on massive datasets, so they are very sensitive to changes in the parameters. When fine-tuning, it is important to use a smaller learning rate to prevent the model from overfitting to your fine-tuning dataset.
- **Use a regularization technique:** Regularization techniques help to prevent overfitting by adding a penalty to the loss function. This penalty encourages the model to learn simpler and more generalizable representations. Common regularization techniques include L1 and L2 regularization.

❖ Here is a general workflow for fine-tuning a pre-trained sentiment analysis model:

1. Choose a pre-trained model. BERT and RoBERTa are popular choices for sentiment analysis.
2. Prepare your fine-tuning dataset. This dataset should be labeled with the sentiment of each text sample.
3. Tokenize your data using the tokenizer provided by the pre-trained model library.
4. Create a dataloader to load your data into batches.
5. Define the model architecture. This will involve adding a classification layer on top of the pre-trained model.
6. Choose an optimizer and loss function.
7. Compile the model.
8. Train the model on your fine-tuning dataset.
9. Evaluate the model on a held-out test dataset

❖ Here are the key steps:

1. <u>Install Required Libraries:</u>

First, make sure you have the necessary libraries installed, including `transformers`, `torch`, and `scikit-learn`.

2. <u>Data Preparation:</u>

You'll need a labeled dataset for sentiment analysis. Ensure your dataset is in a format that includes text samples and corresponding sentiment labels (e.g., positive, negative, neutral).

3. <u>Load Pre-trained Model:</u>

You can choose from various pre-trained BERT-based models like BERT, RoBERTa, or others. Load the model and tokenizer from the Hugging Face Transformers library.

4. Data Preprocessing:

Tokenize and preprocess your dataset. This includes converting text to input features compatible with the model.

5. Fine-tuning the Model:

Define your training loop, loss function, and optimizer. Fine-tune the model on your sentiment dataset.
(device)

6. Evaluate the Model:

After training, evaluate the model on a separate validation dataset to assess its performance

7. Inference:

Use the fine-tuned model for sentiment analysis on new text

samples.

8. Hyperparameter Tuning:

You may need to experiment with hyperparameters such as batch size, learning rate, and the number of training epochs to optimize model performance.

9. Save and Load the Model:

You can save your fine-tuned model for future use and load it when needed.

PROGRAM:

Input:

```
import numpy as np
import pandas as pd
import touch from transformers import BertTokenizer,
BertForSequenceClassification, AdamW
from torch.utils.data import DataLoader, TensorDataset

# Sample dataset

data = pd.DataFrame({
     'text': ["I love this product!", "This is
terrible.", "It's okay.", "I'm not sure."],
     'sentiment': ['positive', 'negative',
'neutral', 'neutral']
})

# Define hyperparameters

batch_size = 2
learning_rate = 2e-5
num_epochs = 3

# Tokenizer and Model

model_name = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(model_name)
model =
BertForSequenceClassification.from_pretrained(model_name,
num_labels=3)

# Data preprocessing

def preprocess_data(data):
    input_ids = []
    attention_masks = []
    labels = []
```

```
    for index, row in data.iterrows():
        text = row['text']
        label = row['sentiment']
    inputs = tokenizer(text, padding='max_length',
 max_length=64, truncation=True, return_tensors='pt')
         input_ids.append(inputs['input_ids'])
        attention_masks.append(inputs['attention_mask'])

        if label == 'positive':
            labels.append(0)
        elif label == 'negative':
            labels.append(1)
        else:
            labels.append(2)

    input_ids = torch.cat(input_ids, dim=0)
    attention_masks = torch.cat(attention_masks, dim=0)
    labels = torch.tensor(labels)

    return input_ids, attention_masks, labels

input_ids, attention_masks, labels = preprocess_data(data)

# Create DataLoader

dataset = TensorDataset(input_ids, attention_masks,
labels)
dataloader = DataLoader(dataset, batch_size=batch_size,
shuffle=True)

# Optimizer

optimizer = AdamW(model.parameters(), lr=learning_rate)

# Training loop

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

```
    total_loss = 0.0
    for batch in dataloader:
        optimizer.zero_grad()
        input_ids, attention_mask, label = batch
        outputs = model(input_ids,
attention_mask=attention_mask, labels=label)
            loss = outputs.loss
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    avg_loss = total_loss / len(dataloader)
    print(f"Epoch {epoch + 1}/{num_epochs}, Loss:
{avg_loss:.4f}")

# Validation

model.eval()
with torch.no_grad():
    val_inputs = preprocess_data(pd.DataFrame({'text': ["I
like it.", "This is bad."], 'sentiment': ['positive',
'negative']}))
    val_input_ids, val_attention_masks, val_labels =
val_inputs

    val_dataset = TensorDataset(val_input_ids,
val_attention_masks, val_labels)
    val_dataloader = DataLoader(val_dataset,
batch_size=batch_size)

    val_accuracy = 0.0
    for batch in val_dataloader:
        input_ids, attention_mask, label = batch
        outputs = model(input_ids,
attention_mask=attention_mask)
        logits = outputs.logits
        preds = np.argmax(logits.detach().cpu().numpy(),
axis=1)
        labels = label.cpu().numpy()
```

```
        val_accuracy += (preds == labels).mean()

    avg_val_accuracy = val_accuracy / len(val_dataloader)
    print(f"Validation Accuracy: {avg_val_accuracy:.2%}")
# Inference

def predict_sentiment(text):
    inputs = tokenizer(text, padding='max_length',
max_length=64, truncation=True, return_tensors='pt')
logits = model(**inputs).logits
    sentiment = np.argmax(logits.detach().cpu().numpy())
    return sentiment

text_to_analyze = "This is great!"
sentiment = predict_sentiment(text_to_analyze)
sentiment_mapping = {0: 'positive', 1: 'negative', 2:
'neutral'}
print(f"Predicted Sentiment:
{sentiment_mapping[sentiment]}")
```

EXPECTED OUTPUT:

```
Epoch 1/3, Loss: 1.1831
Epoch 2/3, Loss: 0.4912
Epoch 3/3, Loss: 0.2282
Validation Accuracy: 50.00%
Predicted Sentiment: positive
```