Algorithm and Dataset

Project Title: Emoji Prediction for Text

Dataset Used:

We utilized the "Twitter Emoji Prediction" dataset in Kaggle, which can be accessed at:

https://www.kaggle.com/datasets/hariharasudhanas/twitter-emoji-prediction

Dataset Description:

- The dataset includes labeled text examples where each example is accompanied by a corresponding emoji (label).

- There are two primary columns:

  - TEXT – input text data (user posts, tweets, or statements).

  - Label – the numerical label for an emoji.

- A secondary mapping file (usually labeled mapping.csv) translates the numeric label into the actual emoji.

Algorithm Used:

We used a fine-tuned DistilBERT (Distilled BERT) model for sequence classification, utilizing the Hugging Face Transformers library. The model is trained to classify the input text as one of a number of emoji categories.

Step-by-Step Breakdown:

1.. Import Required Libraries:

```python

import pandas as pd

import torch

from sklearn.preprocessing import LabelEncoder

from transformers import DistilBertTokenizerFast, DistilBertForSequenceClassification, Trainer, TrainingArguments

```

2. Load and Preprocess the Dataset:

```python
train_df = pd.read_csv('train.csv')
mapping_df = pd.read_csv('mapping.csv')
train_df.rename(columns={'TEXT': 'text', 'Label': 'label'}, inplace=True)
train_df['label'] = train_df['label'].astype(int)
label_to_emoji = dict(zip(mapping_df['number'], mapping_df['emoticons']))
```

3. Tokenization using DistilBERT Tokenizer:

```python
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')
train_encodings = tokenizer(list(train_df['text']), truncation=True, padding=True)
```

4.Prepare Dataset Class for Trainer:

```python
class EmojiDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels
    def __len__(self):
```

```
return len(self.labels)
    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item
```

5. Load Model and Define Training Arguments:

```python

model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=len(label_to_emoji))

training_args = TrainingArguments(

    output_dir='./results',

```

num_train_epochs=3,

    per_device_train_batch_size=16,

    per_device_eval_batch_size=16,

    warmup_steps=500,

    weight_decay=0.01,

    logging_dir='./logs',

    logging_steps=10,

    evaluation_strategy='epoch',

    save_strategy='epoch')

```

### 6.Train the Model:

```python

train_dataset = EmojiDataset(train_encodings, list(train_df['label']))

trainer = Trainer(model=model, args=training_args, train_dataset=train_dataset)

trainer.train()

```

7.Save and Use the Model for Prediction:

```python

model.save_pretrained('./emoji_model')

```
tokenizer.save_pretrained('./emoji_model')

def predict_emoji(text):
    inputs = tokenizer(text, return_tensors='pt', truncation=True, padding=True)
    with torch.no_grad():
outputs = model(**inputs)
    logits = outputs.logits
    predicted_class = torch.argmax(logits, dim=1).item()
    return label_to_emoji[predicted_class]
```

Inputs and Outputs:

Inputs:

- A string of text input from the user (e.g., "I am so happy today!")

Outputs:

- A single emoji that best represents the emotion or intent of the text.

```
Enter a sentence (or type 'exit' to quit): I am so Happy today!
Predicted Emoji: 😊
```

Conditions and Loops:

- The primary loop is training (`trainer.train()`), in which batches of input are run through the model over several epochs.

- Conditions are utilized in token padding and truncation, and in determining the predicted label with `torch.argmax`.