ASSIGNMENT - I

Name : Nithin Ashwa L
Roll No : 717822I135
Dept : B.Tech, AD – "A"
Subject : SLP

Contextual Embeddings.

Question:

Implement a contextual embedding model such as ELMo or BERT for sentence classification or natural language inference tasks. Fine-tune the pre-trained model on a specific downstream task and evaluate its performance

GitHub Link:

https://github.com/NithinAsh/Contextual-Embeddings

Code:

Install package:

```
!pip install datasets
```

```
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from datas
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.11/dist-packages (from
Collecting xxhash (from datasets)
  Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metada
Collecting multiprocess<0.70.17 (from datasets)
  Downloading multiprocess-0.70.16-py311-none-any.whl.metadata (7.2 kB)
Collecting fsspec<=2024.9.0,>=2023.1.0 (from fsspec[http]<=2024.9.0,>=2023.1.0->datasets)
  Downloading fsspec-2024.9.0-py3-none-any.whl.metadata (11 kB)

Downloading datasets-3.2.0-py3-none-any.whl (480 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 480.6/480.6 kB 7.8 MB/s eta 0:00:00
Downloading dill-0.3.8-py3-none-any.whl (116 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 7.1 MB/s eta 0:00:00
Downloading fsspec-2024.9.0-py3-none-any.whl (179 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 179.3/179.3 kB 12.2 MB/s eta 0:00:00
Downloading multiprocess-0.70.16-py311-none-any.whl (143 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.5/143.5 kB 7.3 MB/s eta 0:00:00
Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (194 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.8/194.8 kB 14.0 MB/s eta 0:00:00
Installing collected packages: xxhash, fsspec, dill, multiprocess, datasets
  Attempting uninstall: fsspec
    Found existing installation: fsspec 2024.10.0
    Uninstalling fsspec-2024.10.0:
      Successfully uninstalled fsspec-2024.10.0

Successfully installed datasets-3.2.0 dill-0.3.8 fsspec-2024.9.0 multiprocess-0.70.16 xxhash-3.5.0
```

Import package:

```
import torch
from sentence_transformers import SentenceTransformer
from sklearn.linear_model import LogisticRegression
from datasets import load_dataset
import numpy as np
```

```
# Load the dataset
dataset = load_dataset('glue', 'sst2')
small_train_dataset = dataset['train'].select(range(1000))
small_validation_dataset = dataset['validation'].select(range(100))
```

```
# Initialize the sentence transformer model
model = SentenceTransformer('all-MiniLM-L6-v2')
```

```
# Function to generate embeddings
def generate_embeddings(dataset):
    sentences = [sample['sentence'] for sample in dataset]
    embeddings = model.encode(sentences, convert_to_numpy=True)
    labels = np.array([sample['label'] for sample in dataset])
    return embeddings, labels
```

```
# Prepare training and validation data
train_embeddings, train_labels = generate_embeddings(small_train_dataset)
val_embeddings, val_labels = generate_embeddings(small_validation_dataset)
```

```
# Train a logistic regression classifier
classifier = LogisticRegression()
classifier.fit(train_embeddings, train_labels)
```

```
# Evaluate the model
accuracy = classifier.score(val_embeddings, val_labels)
print(f'Validation Accuracy: {accuracy:.4f}')
```

```
Validation Accuracy: 0.7700
['logistic_regression_sentiment.pkl']
```

```
# Save the model
import joblib
joblib.dump(classifier, 'logistic_regression_sentiment.pkl')
```

```
README.md: 100% 35.3k/35.3k [00:00<00:00, 1.37MB/s]
train-00000-of-00001.parquet: 100% 3.11M/3.11M [00:00<00:00, 9.11MB/s]
validation-00000-of-00001.parquet: 100% 72.8k/72.8k [00:00<00:00, 1.20MB/s]
test-00000-of-00001.parquet: 100% 148k/148k [00:00<00:00, 3.57MB/s]
Generating train split: 100% 67349/67349 [00:00<00:00, 240826.92 examples/s]
Generating validation split: 100% 872/872 [00:00<00:00, 12241.18 examples/s]
Generating test split: 100% 1821/1821 [00:00<00:00, 28003.55 examples/s]
modules.json: 100% 349/349 [00:00<00:00, 3.67kB/s]
config_sentence_transformers.json: 100% 116/116 [00:00<00:00, 1.45kB/s]
README.md: 100% 10.7k/10.7k [00:00<00:00, 168kB/s]
sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 417B/s]
config.json: 100% 612/612 [00:00<00:00, 8.42kB/s]
model.safetensors: 100% 90.9M/90.9M [00:01<00:00, 85.0MB/s]
tokenizer_config.json: 100% 350/350 [00:00<00:00, 6.94kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 3.33MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 6.88MB/s]
special_tokens_map.json: 100% 112/112 [00:00<00:00, 1.62kB/s]
1_Pooling/config.json: 100% 190/190 [00:00<00:00, 2.74kB/s]
Validation Accuracy: 0.7700
['logistic_regression_sentiment.pkl']
```

```python
# Save the trained model
model.save_pretrained('trained_model')
tokenizer.save_pretrained('trained_model')
```

```
('trained_model/tokenizer_config.json',
 'trained_model/special_tokens_map.json',
 'trained_model/vocab.txt',
 'trained_model/added_tokens.json')
```

1. Dataset Preparation

- **Data Collection** – Gather data from various sources such as databases, APIs, or files.
- **Data Cleaning** – Handle missing values, remove duplicates, and correct inconsistencies.
- **Data Transformation** – Convert data into a suitable format, including normalization and encoding.
- **Feature Engineering** – Create new relevant features to improve model performance.

2. Model and Tokenizer Setup

- **Load Pre-trained Model** – Import a pre-trained model (e.g., BERT, GPT) from libraries like Hugging Face's Transformers.
- **Initialize Tokenizer** – Load a tokenizer that matches the model to convert text into tokenized input.

• **Configure Model Parameters** – Set parameters such as max sequence length, attention mask, and padding.

• **Prepare for Inference/Training** – Convert tokenized data into tensor format for model processing.

3. Model Training

• **Define Training Loop** – Iterate through the dataset, process batches, and update model weights.

• **Loss Calculation** – Use an appropriate loss function (e.g., CrossEntropyLoss for classification).

• **Optimization** – Apply optimizers like Adam or SGD to update model parameters.

• **Evaluation & Logging** – Monitor performance using validation data and track metrics like accuracy or loss.

4. Model Evaluation

• **Load Test Data** – Use a separate dataset to assess model performance.

• **Compute Metrics** – Measure accuracy, precision, recall, and F1-score based on predictions.

• **Generate Predictions** – Compare predicted outputs with ground truth labels.

• **Analyze Errors** – Identify misclassifications and areas for improvement.

5. Output

• **Predictions** – The model generates outputs based on the input data, such as class labels, probabilities, or text sequences.

• **Performance Metrics** – Evaluation metrics like accuracy, loss, precision, recall, and F1-score indicate the model's effectiveness.

• **Error Analysis** – Examining misclassifications or incorrect outputs helps refine the model for better performance.

6. Accuracy

• **Definition** – Accuracy is the ratio of correctly predicted instances to the total instances, expressed as:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}} \times 100$$

• **Example Output**:

Mathematica

Model Accuracy: 92.5%

• **Interpretation**:

- A higher accuracy (e.g., 90%+) indicates good performance.
- Low accuracy suggests issues like insufficient training, imbalanced data, or model overfitting.

• **Additional Metrics**: Accuracy alone may not be enough; precision, recall, and F1-score provide deeper insights.

Key Features and Components:

- **Pre-trained Model** – Utilizes a deep learning model trained on large datasets for transfer learning.
- **Tokenizer** – Converts text into numerical tokens suitable for model processing.
- **Training Pipeline** – Includes data loading, loss calculation, optimization, and backpropagation.
- **Evaluation Metrics** – Measures performance using accuracy, precision, recall, and F1-score.

This project demonstrates how transformer-based models, like T5, can be applied to real-world tasks such as generating structured queries (SQL) from unstructured natural language input.

Streamlit Deploy:

Code:

```
import streamlit as st
import joblib
from sentence_transformers import SentenceTransformer
import numpy as np

# Load the trained model
try:
    classifier = joblib.load('logistic_regression_sentiment.pkl')
    model = SentenceTransformer('all-MiniLM-L6-v2')
except Exception as e:
    st.error(f"Error loading model: {e}")

# Define a function to make predictions
def predict_sentiment(text):
    embedding = model.encode([text], convert_to_numpy=True)
    predicted_class = classifier.predict(embedding)[0]
    confidence = classifier.predict_proba(embedding).max()  # Get confidence score
    return "Positive" if predicted_class == 1 else "Negative", confidence

# Streamlit app layout
st.title("Sentiment Analysis with Contextual Embeddings")
st.write("Enter a movie review to analyze its sentiment:")
```

```python
# Text input for user
user_input = st.text_area("Review:")

# Button to make prediction
if st.button("Analyze"):
    if user_input:
        sentiment, confidence = predict_sentiment(user_input)
        st.write(f"Sentiment: {sentiment} (Confidence: {confidence:.2f})")
    else:
        st.write("Please enter a review.")
```

Streamlit Output: