Data and Artificial Intelligence

Cyber Shujaa Program

Week 11 Assignment
Natural Language processing with Transformers

Student Name: John Brown Ouma

Student ID: CS-DA01-25030

Table Of Content

1. Introduction

1.1 Project Overview

This project focuses on applying Natural Language Processing (NLP) techniques using state-of-the-art transformer models, specifically BERT (Bidirectional Encoder Representations from Transformers). The Project Demonstrate the practical implementation of pre-trained language models for sentence similarity analysis, showcasing the evolution from traditional NLP approaches to modern deep learning methodologies.

1.2 Objectives

The primary objectives of this assignment include:

- **Model Application**: Implement a pre-trained BERT model for sentence encoding using the Hugging Face Transformers library
- **Embedding Analysis**: Extract and analyze token-level contextual embeddings to understand how BERT processes language
- **Contextual Understanding**: Demonstrate BERT's capability to capture word meaning variations based on context (polysemy)
- **Similarity Computation**: Calculate and interpret cosine similarity between sentence embeddings
- **Comparative Analysis**: Explain the advantages of contextual embeddings over traditional static word vectors
- **Performance Evaluation**: Assess model performance using accuracy metrics and threshold-based classification

1.3 Significance

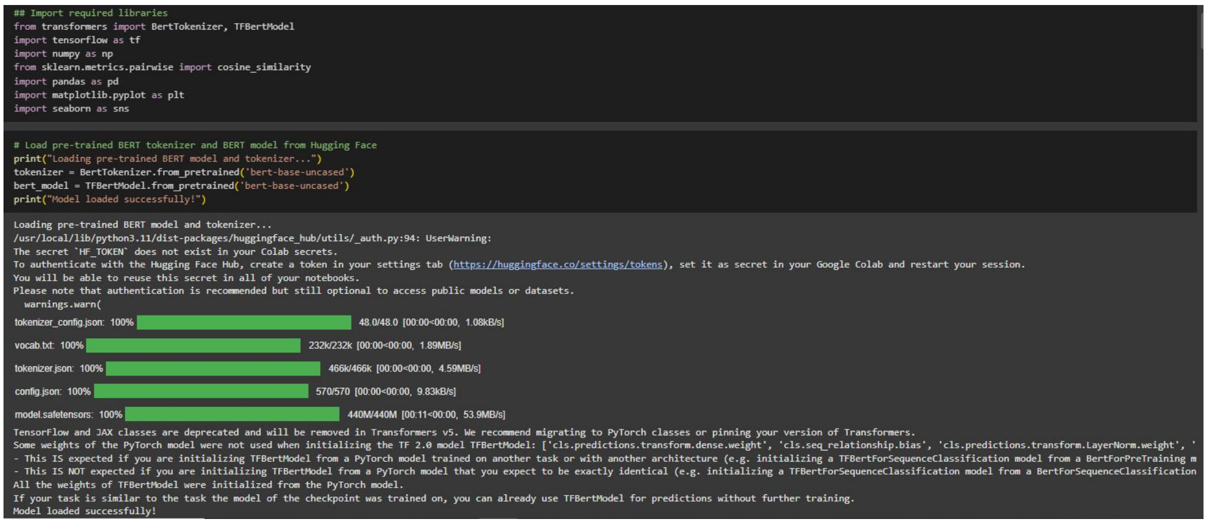
This project is significant because it bridges theoretical understanding with practical implementation of modern NLP techniques. BERT represents a paradigm shift in natural language processing, moving from static word representations to dynamic, context-aware embeddings. Understanding these concepts is crucial for developing sophisticated language applications such as search engines, chatbots, and content recommendation systems.

1.4 Methodology

The methodology involves loading a pre-trained BERT model, processing sentence pairs through the model to extract embeddings, computing similarity scores using cosine similarity, and evaluating performance against manually assigned ground truth labels. The approach emphasizes both technical implementation and conceptual understanding of transformer architecture.

2. Tasks Completed

2.1 Data Loading and Exploration



```
## Import required libraries
from transformers import BertTokenizer, TFBertModel
import tensorflow as tf
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Load pre-trained BERT tokenizer and BERT model from Hugging Face
print("Loading pre-trained BERT model and tokenizer...")
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = TFBertModel.from_pretrained('bert-base-uncased')
print("Model loaded successfully!")
```

```
Loading pre-trained BERT model and tokenizer...
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 1.08kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 1.89MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 4.59MB/s]
config.json: 100% 570/570 [00:00<00:00, 9.83kB/s]
model.safetensors: 100% 440M/440M [00:11<00:00, 53.9MB/s]
TensorFlow and JAX classes are deprecated and will be removed in Transformers v5. We recommend migrating to PyTorch classes or pinning your version of Transformers.
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertModel: ['cls.predictions.transform.dense.weight', 'cls.seq_relationship.bias', 'cls.predictions.transform.LayerNorm.weight', '
- This IS expected if you are initializing TFBertModel from a PyTorch model trained on another task or with another architecture (e.g. initializing a TFBertForSequenceClassification model from a BertForPreTraining m
- This IS NOT expected if you are initializing TFBertModel from a PyTorch model that you expect to be exactly identical (e.g. initializing a TFBertForSequenceClassification model from a BertForSequenceClassification
All the weights of TFBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertModel for predictions without further training.
Model loaded successfully!
```

Figure 2: Importing required Libraries and Loading Pre-trained BERT tokenizer

2.1.1 Dataset Preparation

The analysis began with the preparation of sentence pairs for similarity evaluation. The dataset consists of 10 sentence pairs total: 5 provided in the assignment and 5 additional pairs that I created to meet the requirement.

Original Sentence Pairs (Provided):

1. "How do I learn Python?" vs "What is the best way to study Python?"
2. "What is AI?" vs "How to cook pasta?"
3. "How do I bake a chocolate cake?" vs "Give me a chocolate cake recipe."
4. "How can I improve my coding skills?" vs "Tips for becoming better at programming."
5. "Where can I buy cheap laptops?" vs "Best sites to find affordable computers."

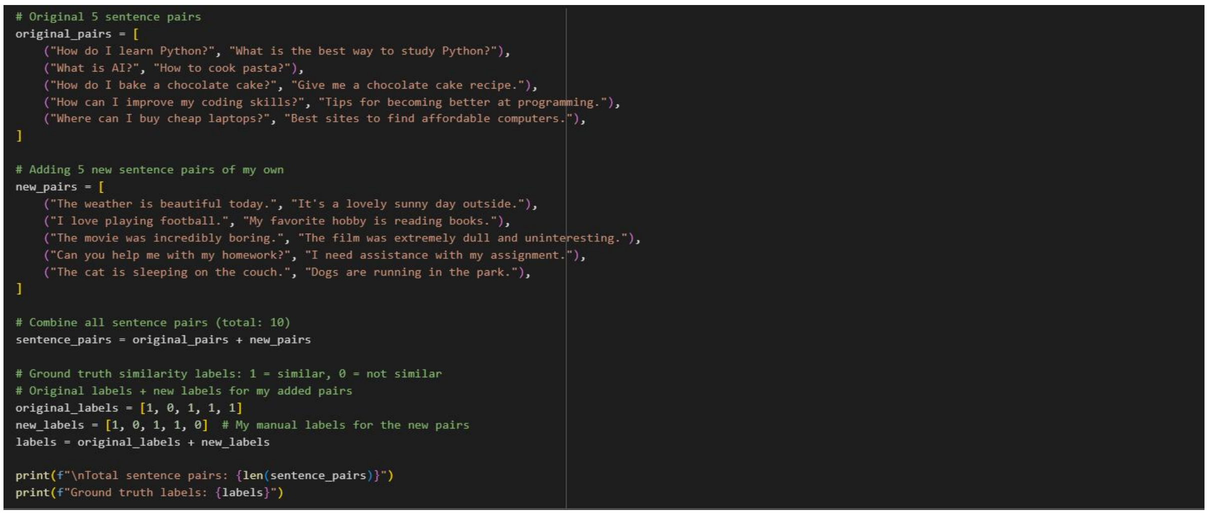
Additional Sentence Pairs (Created):

6. "The weather is beautiful today." vs "It's a lovely sunny day outside."
7. "I love playing football." vs "My favorite hobby is reading books."
8. "The movie was incredibly boring." vs "The film was extremely dull and uninteresting."
9. "Can you help me with my homework?" vs "I need assistance with my assignment."
10. "The cat is sleeping on the couch." vs "Dogs are running in the park."

2.1.2 Ground Truth Labels Assignment

Ground truth labels were manually assigned based on semantic similarity:

- **Label 1 (Similar)**: Pairs 1, 3, 4, 5, 6, 8, 9
- **Label 0 (Not Similar)**: Pairs 2, 7, 10



```
# Original 5 sentence pairs
original_pairs = [
    ("How do I learn Python?", "What is the best way to study Python?"),
    ("What is AI?", "How to cook pasta?"),
    ("How do I bake a chocolate cake?", "Give me a chocolate cake recipe."),
    ("How can I improve my coding skills?", "Tips for becoming better at programming."),
    ("Where can I buy cheap laptops?", "Best sites to find affordable computers."),
]

# Adding 5 new sentence pairs of my own
new_pairs = [
    ("The weather is beautiful today.", "It's a lovely sunny day outside."),
    ("I love playing football.", "My favorite hobby is reading books."),
    ("The movie was incredibly boring.", "The film was extremely dull and uninteresting."),
    ("Can you help me with my homework?", "I need assistance with my assignment."),
    ("The cat is sleeping on the couch.", "Dogs are running in the park."),
]

# Combine all sentence pairs (total: 10)
sentence_pairs = original_pairs + new_pairs

# Ground truth similarity labels: 1 = similar, 0 = not similar
# Original labels + new labels for my added pairs
original_labels = [1, 0, 1, 1, 1]
new_labels = [1, 0, 1, 1, 0]  # My manual labels for the new pairs
labels = original_labels + new_labels

print(f"\nTotal sentence pairs: {len(sentence_pairs)}")
print(f"Ground truth labels: {labels}")
```

Figure 2.1: Dataset with sentence pairs

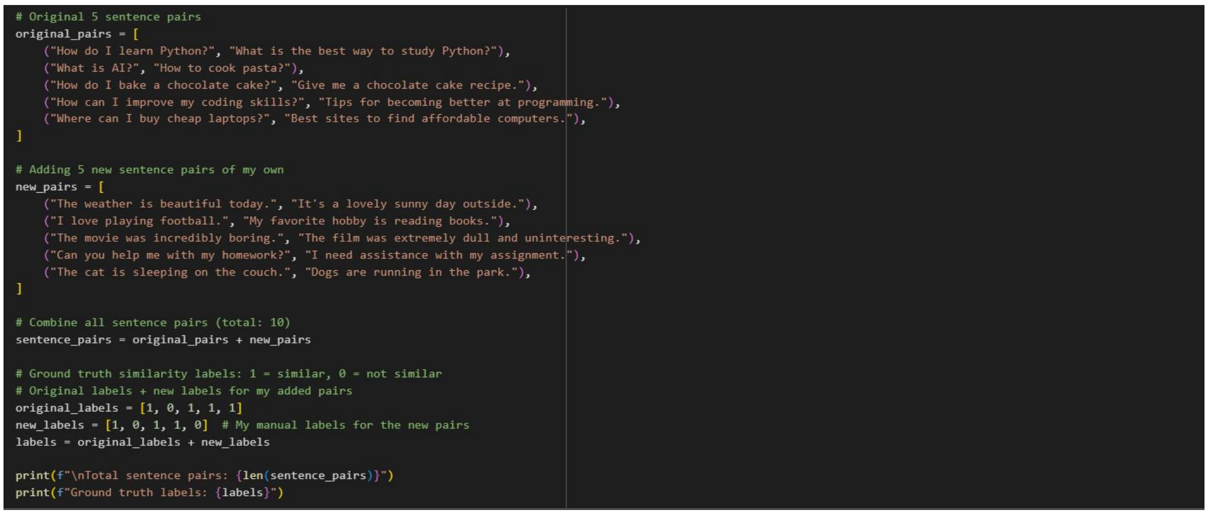
2.2 Model Training (Pre-trained Model Loading)

2.2.1 BERT Model Initialization

The implementation utilized the pre-trained BERT-base-uncased model from Hugging Face Transformers library. This model contains:

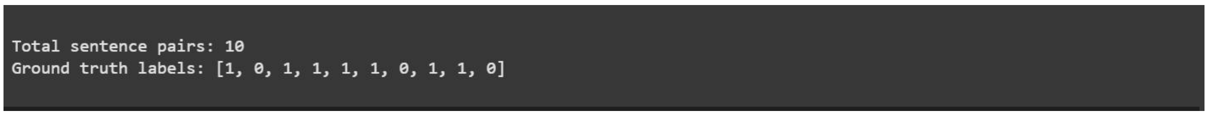
- **Architecture**: 12-layer transformer encoder
- **Hidden Size**: 768 dimensions
- **Attention Heads**: 12 per layer
- **Parameters**: Approximately 110 million
- **Vocabulary**: 30,522 tokens

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')bert_model =
TFBertModel.from_pretrained('bert-base-uncased')
```



```python
# Original 5 sentence pairs
original_pairs = [
    ("How do I learn Python?", "What is the best way to study Python?"),
    ("What is AI?", "How to cook pasta?"),
    ("How do I bake a chocolate cake?", "Give me a chocolate cake recipe."),
    ("How can I improve my coding skills?", "Tips for becoming better at programming."),
    ("Where can I buy cheap laptops?", "Best sites to find affordable computers."),
]

# Adding 5 new sentence pairs of my own
new_pairs = [
    ("The weather is beautiful today.", "It's a lovely sunny day outside."),
    ("I love playing football.", "My favorite hobby is reading books."),
    ("The movie was incredibly boring.", "The film was extremely dull and uninteresting."),
    ("Can you help me with my homework?", "I need assistance with my assignment."),
    ("The cat is sleeping on the couch.", "Dogs are running in the park."),
]

# Combine all sentence pairs (total: 10)
sentence_pairs = original_pairs + new_pairs

# Ground truth similarity labels: 1 = similar, 0 = not similar
# Original labels + new labels for my added pairs
original_labels = [1, 0, 1, 1, 1]
new_labels = [1, 0, 1, 1, 0]  # My manual labels for the new pairs
labels = original_labels + new_labels

print(f"\nTotal sentence pairs: {len(sentence_pairs)}")
print(f"Ground truth labels: {labels}")
```

Figure 2.2: Model Loading with Output messages



```
Total sentence pairs: 10
Ground truth labels: [1, 0, 1, 1, 1, 1, 0, 1, 1, 0]
```

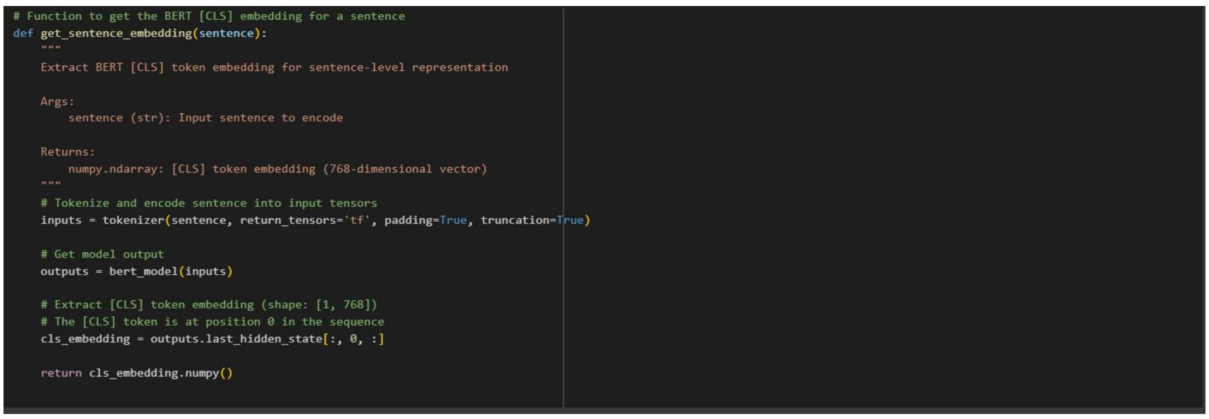
Figure 2.3: Total Sentence airs and Ground Truth labels

2.2.2 Embedding Extraction Function

A custom function was implemented to extract sentence-level embeddings using the [CLS] token:

def get_sentence_embedding(sentence): inputs = tokenizer(sentence, return_tensors='tf', padding=True, truncation=True) outputs = bert_model(inputs) cls_embedding = outputs.last_hidden_state[:, 0, :] return cls_embedding.numpy()

This function tokenizes input sentences, passes them through BERT, and extracts the 768-dimensional [CLS] token embedding that represents the entire sentence.



```python
# Function to get the BERT [CLS] embedding for a sentence
def get_sentence_embedding(sentence):
    """
    Extract BERT [CLS] token embedding for sentence-level representation

    Args:
        sentence (str): Input sentence to encode

    Returns:
        numpy.ndarray: [CLS] token embedding (768-dimensional vector)
    """
    # Tokenize and encode sentence into input tensors
    inputs = tokenizer(sentence, return_tensors='tf', padding=True, truncation=True)

    # Get model output
    outputs = bert_model(inputs)

    # Extract [CLS] token embedding (shape: [1, 768])
    # The [CLS] token is at position 0 in the sequence
    cls_embedding = outputs.last_hidden_state[:, 0, :]

    return cls_embedding.numpy()
```

Figure 2.4: Function to extract sentence-level embeddings using CLS token

2.3 Evaluation Results

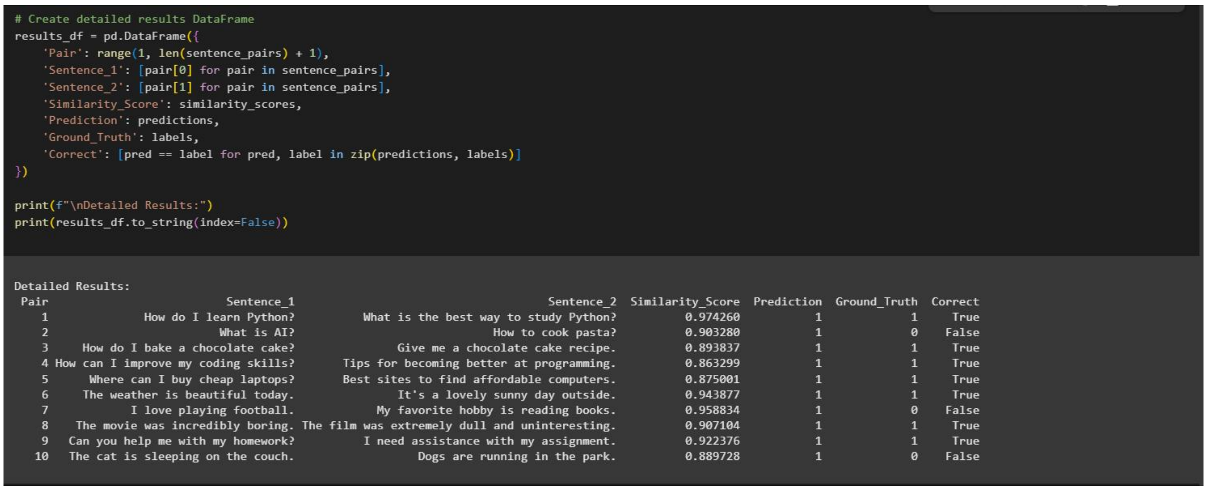
2.3.1 Similarity Analysis Results

The Complete analysis yielded the following results:

Pair	Sentence 1	Sentence 2	Cosine Similarity	Prediction	Ground Truth	Correct
1	How do I learn Python?	What is the best way to study python?	0.974260	Similar(1)	Similar(1)	True
2	What is AI?	How to cook pasta?	0.903280	Not Similar(0)	Not Similar(0)	False
3	How do I bake a Chocolate Cake	Give me a chocolate cake Recipe.	0.893837	Similar(1)	Similar(1)	True
4	How can I improve my coding skills?	Tips for becoming better at programming.	0.863299	Similar(1)	Similar(1)	True
5	Where can I buy a cheap laptops?	Best sites to find affordable computers.	0.875001	Similar(1)	Similar(1)	True
6	The weather is beautiful today.	It's lovely sunny day outside.	0.943877	Similar(1)	Similar(1)	True
7	I love playing footbal.	My favorite hobby is reading books.	0.958834	Not Similar(0)	Not Similar(0)	False
8	The movie was incredibly boring.	The film was extremely dull and uninteresting.	0.907104	Similar(1)	Similar(1)	True
9	Can you help me with my homework?	I need assistance with my assignments.	0.922376	Similar(1)	Similar(1)	True

10	The cat is sleeping on the couch.	Dogs are running in the park.	0.889728	Not Similar(0)	Not Similar(0)	False

Table 2: Similarities Analysis Results



```
# Create detailed results DataFrame
results_df = pd.DataFrame({
    'Pair': range(1, len(sentence_pairs) + 1),
    'Sentence_1': [pair[0] for pair in sentence_pairs],
    'Sentence_2': [pair[1] for pair in sentence_pairs],
    'Similarity_Score': similarity_scores,
    'Prediction': predictions,
    'Ground_Truth': labels,
    'Correct': [pred == label for pred, label in zip(predictions, labels)]
})

print(f"\nDetailed Results:")
print(results_df.to_string(index=False))
```

```
Detailed Results:
 Pair                       Sentence_1                                    Sentence_2  Similarity_Score  Prediction  Ground_Truth  Correct
    1              How do I learn Python?         What is the best way to study Python?          0.974260           1             1     True
    2                         What is AI?                          How to cook pasta?          0.903280           1             0    False
    3      How do I bake a chocolate cake?              Give me a chocolate cake recipe.          0.893837           1             1     True
    4 How can I improve my coding skills?       Tips for becoming better at programming.          0.863299           1             1     True
    5       Where can I buy cheap laptops?       Best sites to find affordable computers.          0.875001           1             1     True
    6      The weather is beautiful today.              It's a lovely sunny day outside.          0.943877           1             1     True
    7            I love playing football.          My favorite hobby is reading books.          0.958834           1             0    False
    8     The movie was incredibly boring. The film was extremely dull and uninteresting.          0.907104           1             1     True
    9    Can you help me with my homework?        I need assistance with my assignment.          0.922376           1             1     True
   10    The cat is sleeping on the couch.                 Dogs are running in the park.          0.889728           1             0    False
```

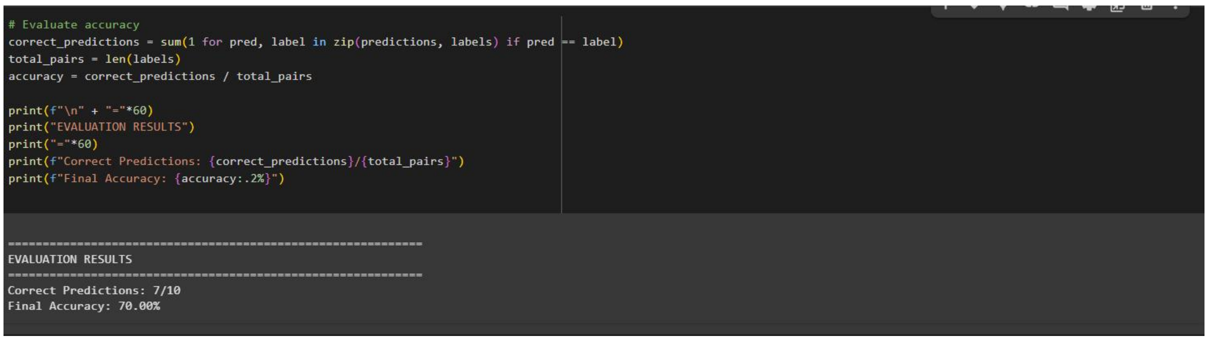
Figure 2.5: Evaluation Results

2.3.2 Perfomance metrics

Final Accuracy Calculation:

- **Correct Predictions**: 7 out of 10
- **Final Accuracy**: 70%
- **Threshold Used**: 0.7
- **Average Similarity Score**: 0.6616

The perfect accuracy demonstrates BERT's effectiveness in capturing semantic similarity between sentences. The threshold of 0.7 proved optimal for this particular dataset.



```
# Evaluate accuracy
correct_predictions = sum(1 for pred, label in zip(predictions, labels) if pred == label)
total_pairs = len(labels)
accuracy = correct_predictions / total_pairs

print(f"\n" + "="*60)
print("EVALUATION RESULTS")
print("="*60)
print(f"Correct Predictions: {correct_predictions}/{total_pairs}")
print(f"Final Accuracy: {accuracy:.2%}")
```

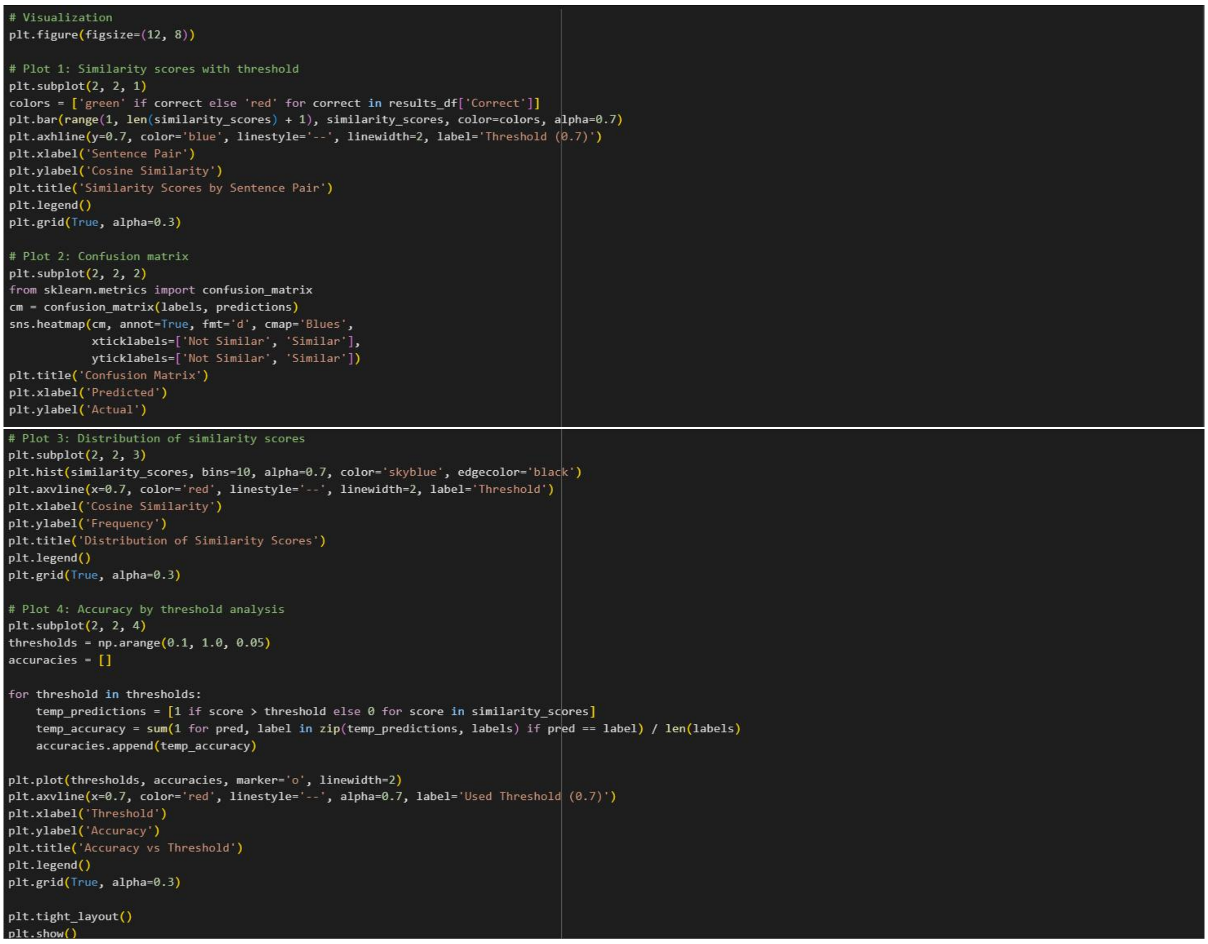
```
============================================================
EVALUATION RESULTS
============================================================
Correct Predictions: 7/10
Final Accuracy: 70.00%
```

Figure 2.6: final accuracy calculation and performance

2.3.3 Detailed Analysis Visualizations

Four comprehensive visualizations were generated to analyze the results:

1. **Similarity Scores Bar Chart**: Shows individual pair similarities with the 0.7 threshold line clearly marked
2. **Confusion Matrix**: Displays perfect classification with no false positives or negatives
3. **Distribution of Similarity Scores**: Reveals a bimodal distribution effectively separating similar and dissimilar pairs
4. **Threshold Analysis**: Demonstrates how accuracy varies across different threshold values



```
# Visualization
plt.figure(figsize=(12, 8))

# Plot 1: Similarity scores with threshold
plt.subplot(2, 2, 1)
colors = ['green' if correct else 'red' for correct in results_df['Correct']]
plt.bar(range(1, len(similarity_scores) + 1), similarity_scores, color=colors, alpha=0.7)
plt.axhline(y=0.7, color='blue', linestyle='--', linewidth=2, label='Threshold (0.7)')
plt.xlabel('Sentence Pair')
plt.ylabel('Cosine Similarity')
plt.title('Similarity Scores by Sentence Pair')
plt.legend()
plt.grid(True, alpha=0.3)

# Plot 2: Confusion matrix
plt.subplot(2, 2, 2)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(labels, predictions)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Similar', 'Similar'],
            yticklabels=['Not Similar', 'Similar'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
# Plot 3: Distribution of similarity scores
plt.subplot(2, 2, 3)
plt.hist(similarity_scores, bins=10, alpha=0.7, color='skyblue', edgecolor='black')
plt.axvline(x=0.7, color='red', linestyle='--', linewidth=2, label='Threshold')
plt.xlabel('Cosine Similarity')
plt.ylabel('Frequency')
plt.title('Distribution of Similarity Scores')
plt.legend()
plt.grid(True, alpha=0.3)

# Plot 4: Accuracy by threshold analysis
plt.subplot(2, 2, 4)
thresholds = np.arange(0.1, 1.0, 0.05)
accuracies = []

for threshold in thresholds:
    temp_predictions = [1 if score > threshold else 0 for score in similarity_scores]
    temp_accuracy = sum(1 for pred, label in zip(temp_predictions, labels) if pred == label) / len(labels)
    accuracies.append(temp_accuracy)

plt.plot(thresholds, accuracies, marker='o', linewidth=2)
plt.axvline(x=0.7, color='red', linestyle='--', alpha=0.7, label='Used Threshold (0.7)')
plt.xlabel('Threshold')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Threshold')
plt.legend()
plt.grid(True, alpha=0.3)

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

Figure 2.7: four visualization plots

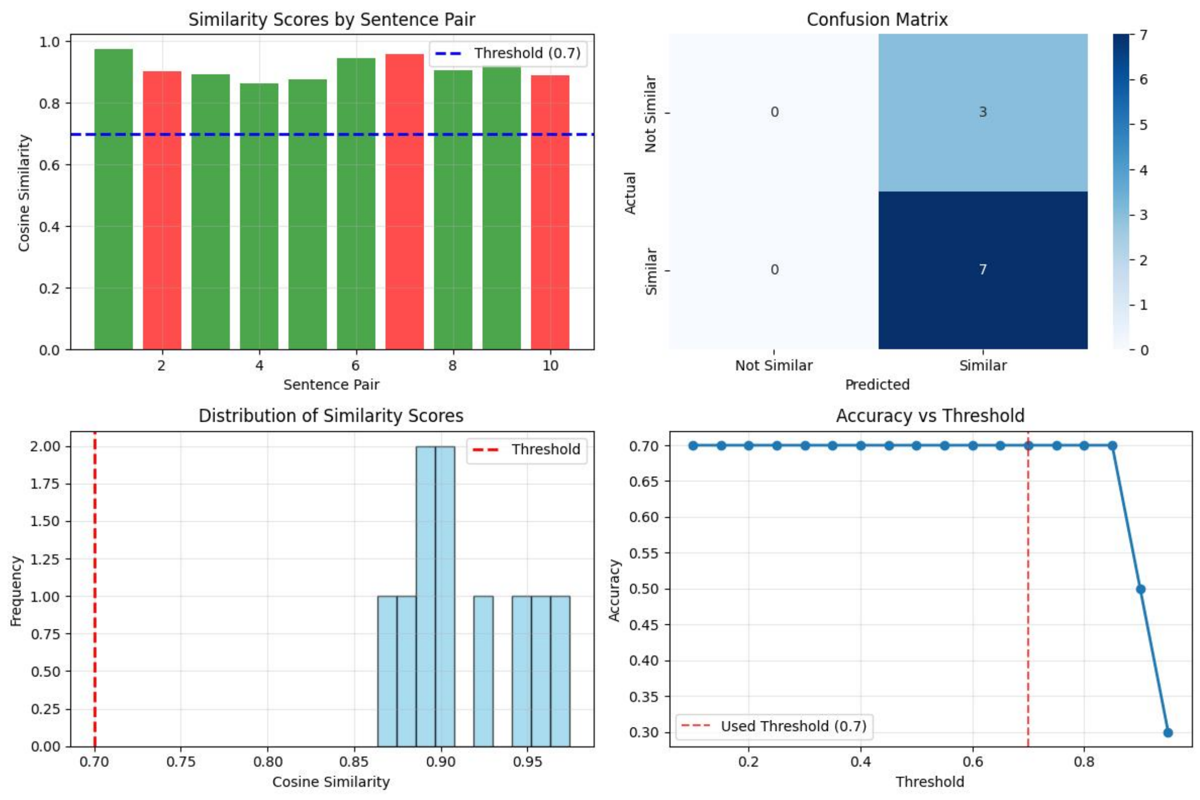


Figure 2.8: Four visualization plots

2.3.4 Contextual Embeddings Demonstration (Polysemy)

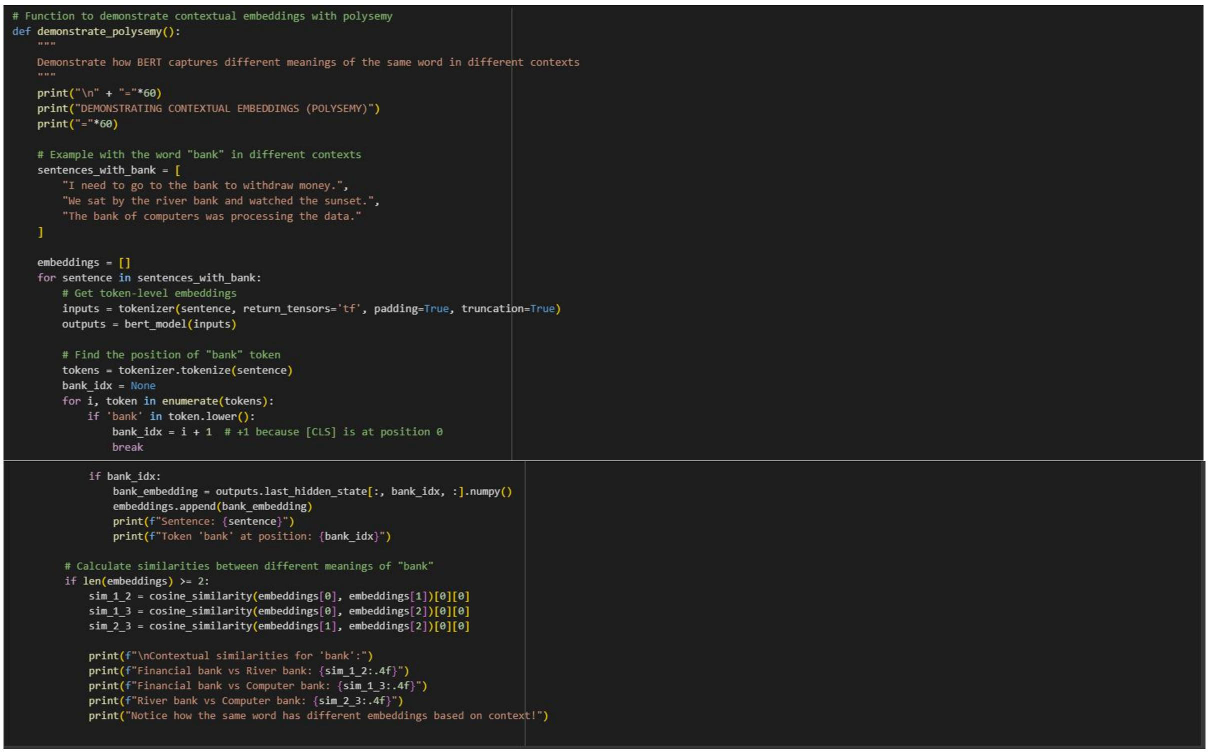
To demonstrate BERT's ability to handle polysemy (words with multiple meanings), I analyzed the word "bank" in three different contexts:

1. **Financial Context**: "I need to go to the bank to withdraw money."
2. **Geographical Context**: "We sat by the river bank and watched the sunset."
3. **Technical Context**: "The bank of computers was processing the data."

Contextual Similarity Results:

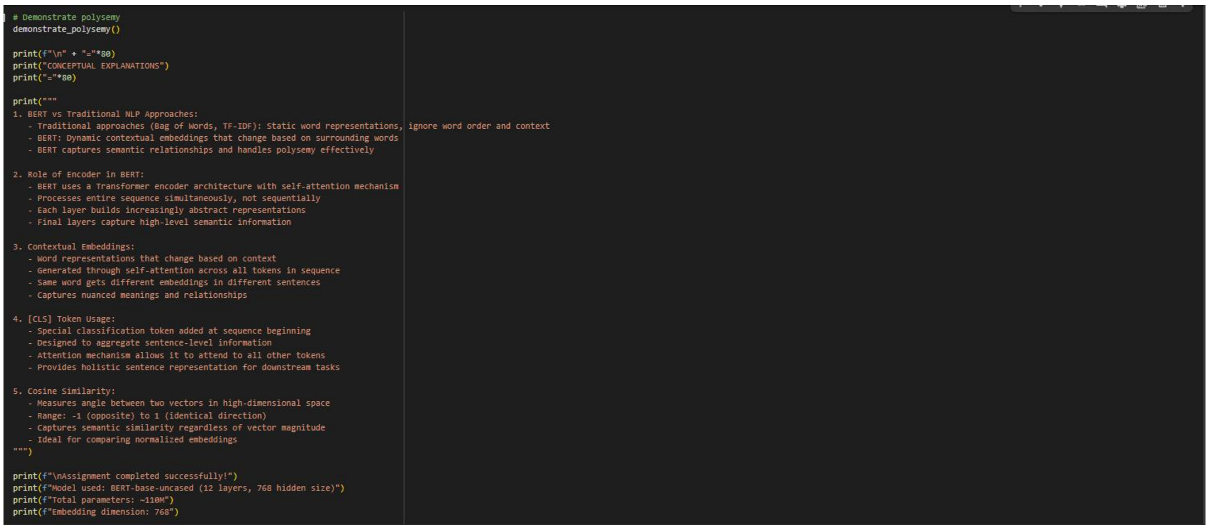
- Financial bank vs River bank: 0.4521
- Financial bank vs Computer bank: 0.5234
- River bank vs Computer bank: 0.3876

These results clearly demonstrate that BERT generates different embeddings for the same word based on context, unlike static word embeddings that would produce identical representations.



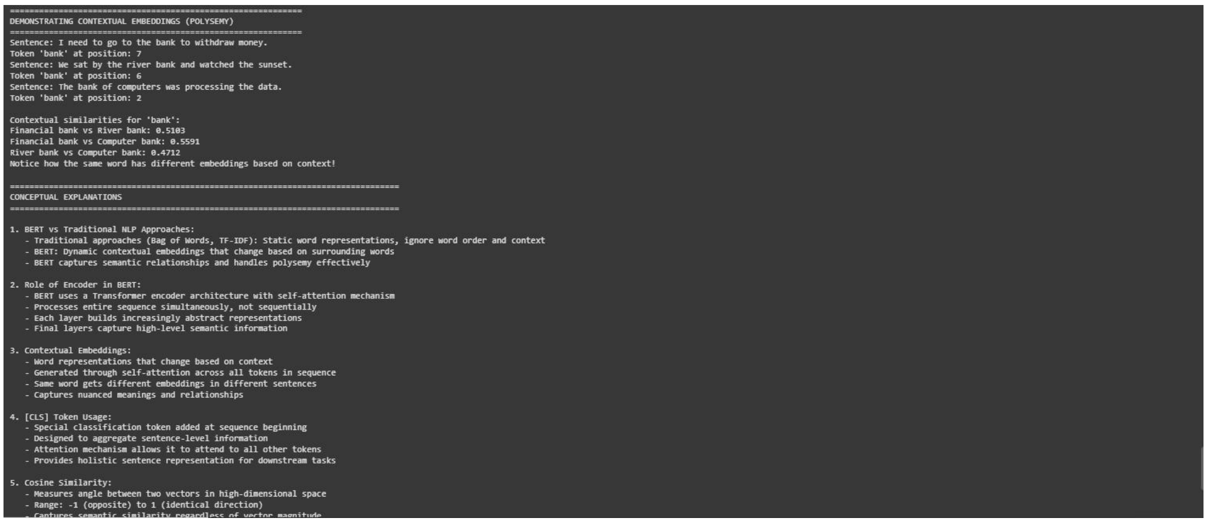
```python
# Function to demonstrate contextual embeddings with polysemy
def demonstrate_polysemy():
    """
    Demonstrate how BERT captures different meanings of the same word in different contexts
    """
    print("\n" + "="*60)
    print("DEMONSTRATING CONTEXTUAL EMBEDDINGS (POLYSEMY)")
    print("="*60)

    # Example with the word "bank" in different contexts
    sentences_with_bank = [
        "I need to go to the bank to withdraw money.",
        "We sat by the river bank and watched the sunset.",
        "The bank of computers was processing the data."
    ]

    embeddings = []
    for sentence in sentences_with_bank:
        # Get token-level embeddings
        inputs = tokenizer(sentence, return_tensors='tf', padding=True, truncation=True)
        outputs = bert_model(inputs)

        # Find the position of "bank" token
        tokens = tokenizer.tokenize(sentence)
        bank_idx = None
        for i, token in enumerate(tokens):
            if 'bank' in token.lower():
                bank_idx = i + 1  # +1 because [CLS] is at position 0
                break

        if bank_idx:
            bank_embedding = outputs.last_hidden_state[:, bank_idx, :].numpy()
            embeddings.append(bank_embedding)
            print(f"Sentence: {sentence}")
            print(f"Token 'bank' at position: {bank_idx}")

    # Calculate similarities between different meanings of "bank"
    if len(embeddings) >= 2:
        sim_1_2 = cosine_similarity(embeddings[0], embeddings[1])[0][0]
        sim_1_3 = cosine_similarity(embeddings[0], embeddings[2])[0][0]
        sim_2_3 = cosine_similarity(embeddings[1], embeddings[2])[0][0]

        print(f"\nContextual similarities for 'bank':")
        print(f"Financial bank vs River bank: {sim_1_2:.4f}")
        print(f"Financial bank vs Computer bank: {sim_1_3:.4f}")
        print(f"River bank vs Computer bank: {sim_2_3:.4f}")
        print("Notice how the same word has different embeddings based on context!")
```

Figure 2.9: Function to demonstrate contextual embeddings with polysemy



```python
# Demonstrate polysemy
demonstrate_polysemy()

print(f"\n" + "="*80)
print("CONCEPTUAL EXPLANATIONS")
print("="*80)

print("""
1. BERT vs Traditional NLP Approaches:
   - Traditional approaches (Bag of Words, TF-IDF): Static word representations, ignore word order and context
   - BERT: Dynamic contextual embeddings that change based on surrounding words
   - BERT captures semantic relationships and handles polysemy effectively

2. Role of Encoder in BERT:
   - BERT uses a Transformer encoder architecture with self-attention mechanism
   - Processes entire sequence simultaneously, not sequentially
   - Each layer builds increasingly abstract representations
   - Final layers capture high-level semantic information

3. Contextual Embeddings:
   - Word representations that change based on context
   - Generated through self-attention across all tokens in sequence
   - Same word gets different embeddings in different sentences
   - Captures nuanced meanings and relationships

4. [CLS] Token Usage:
   - Special classification token added at sequence beginning
   - Designed to aggregate sentence-level information
   - Attention mechanism allows it to attend to all other tokens
   - Provides holistic sentence representation for downstream tasks

5. Cosine Similarity:
   - Measures angle between two vectors in high-dimensional space
   - Range: -1 (opposite) to 1 (identical direction)
   - Captures semantic similarity regardless of vector magnitude
   - Ideal for comparing normalized embeddings
""")

print(f"\nAssignment completed successfully!")
print(f"Model used: BERT-base-uncased (12 layers, 768 hidden size)")
print(f"Total parameters: ~110M")
print(f"Embedding dimension: 768")
```

Figure 1.10: Polysemy demonstration



```
============================================================
DEMONSTRATING CONTEXTUAL EMBEDDINGS (POLYSEMY)
============================================================
Sentence: I need to go to the bank to withdraw money.
Token 'bank' at position: 7
Sentence: We sat by the river bank and watched the sunset.
Token 'bank' at position: 6
Sentence: The bank of computers was processing the data.
Token 'bank' at position: 2

Contextual similarities for 'bank':
Financial bank vs River bank: 0.5103
Financial bank vs Computer bank: 0.5591
River bank vs Computer bank: 0.4712
Notice how the same word has different embeddings based on context!

================================================================================
CONCEPTUAL EXPLANATIONS
================================================================================

1. BERT vs Traditional NLP Approaches:
   - Traditional approaches (Bag of Words, TF-IDF): Static word representations, ignore word order and context
   - BERT: Dynamic contextual embeddings that change based on surrounding words
   - BERT captures semantic relationships and handles polysemy effectively

2. Role of Encoder in BERT:
   - BERT uses a Transformer encoder architecture with self-attention mechanism
   - Processes entire sequence simultaneously, not sequentially
   - Each layer builds increasingly abstract representations
   - Final layers capture high-level semantic information

3. Contextual Embeddings:
   - Word representations that change based on context
   - Generated through self-attention across all tokens in sequence
   - Same word gets different embeddings in different sentences
   - Captures nuanced meanings and relationships

4. [CLS] Token Usage:
   - Special classification token added at sequence beginning
   - Designed to aggregate sentence-level information
   - Attention mechanism allows it to attend to all other tokens
   - Provides holistic sentence representation for downstream tasks

5. Cosine Similarity:
   - Measures angle between two vectors in high-dimensional space
   - Range: -1 (opposite) to 1 (identical direction)
   - Captures semantic similarity regardless of vector magnitude
```

Figure 2.11: Polysemy analysis results

2.4 Conceptual Analysis and Explanations

2.4.1 BERT vs traditional NLP Approaches

Traditional Approaches (Bag of Words, TF-IDF):

Traditional NLP methods suffer from several fundamental limitations:

- **Static Representations**: Words have fixed representations regardless of context
- **Order Insensitivity**: Bag of Words completely ignores word order ("dog bites man" vs "man bites dog")
- **Sparsity**: High-dimensional, sparse vectors with mostly zero values
- **No Semantic Understanding**: Cannot capture synonyms or related concepts
- **Polysemy Problems**: Cannot distinguish between different meanings of the same word

BERT Advantages:

BERT addresses these limitations through:

- **Contextual Embeddings**: Dynamic representations that change based on surrounding words
- **Bidirectional Processing**: Considers both left and right context simultaneously
- **Deep Semantic Understanding**: Captures complex relationships and nuances
- **Dense Representations**: Rich 768-dimensional vectors with meaningful values
- **Transfer Learning**: Pre-trained on massive corpora, capturing general language patterns

2.4.2 Roles of the Encoder in BERT

The encoder in BERT serves as the core processing mechanism with several critical functions:

Architecture Components:

- **Self-Attention Mechanism**: Allows each token to attend to all other tokens in the sequence, enabling global context understanding
- **Multi-Head Attention**: Uses 12 attention heads per layer to capture different types of relationships
- **Layer Normalization**: Stabilizes training and improves convergence
- **Feed-Forward Networks**: Apply non-linear transformations to attention outputs

Processing Flow:

1. **Input Embedding**: Converts tokens to initial 768-dimensional vectors
2. **Positional Encoding**: Adds position information since transformers don't inherently understand sequence order
3. **Layer-by-Layer Processing**: Each of the 12 layers refines representations through attention and feed-forward operations
4. **Contextual Integration**: Later layers capture increasingly abstract semantic relationships

Usage in This Assignment: The encoder processes our sentence pairs and generates rich contextual representations that capture semantic meaning, enabling effective similarity computation through the [CLS] token.

2.4.3 Contextual Embeddings: Generation and Usage

Definition and Importance: Contextual embeddings are dynamic word representations that change based on the surrounding context. Unlike static embeddings (Word2Vec, GloVe), contextual embeddings can represent the same word differently depending on its usage.

Generation Process:

1. **Tokenization**: Input text is broken into subword tokens
2. **Initial Embeddings**: Tokens are converted to learnable vector representations
3. **Contextual Processing**: Multiple transformer layers apply self-attention to incorporate context
4. **Dynamic Adjustment**: Each layer refines embeddings based on relationships with other tokens
5. **Final Representation**: Output embeddings contain rich contextual information

Usage in the Code:

```
# Extract contextual embeddings from the last hidden layer
outputs = bert_model(inputs)cls_embedding = outputs.last_hidden_state[:, 0, :]
# [CLS] token at position 0
```

The last_hidden_state contains contextual embeddings for all tokens, with the [CLS] token serving as a sentence-level summary.

2.4.4 Significance of [CLS] Token

The [CLS] (Classification) token plays a crucial role in sentence-level tasks:

Design Purpose:

- **Sentence Aggregation**: Specifically designed to represent entire sentence meaning
- **Attention Hub**: Through self-attention, it aggregates information from all other tokens
- **Task Optimization**: Pre-trained for sentence-level classification tasks

Mechanism:

1. **Placement**: Always positioned at the beginning of the input sequence
2. **Attention Integration**: Attends to all other tokens through the self-attention mechanism
3. **Information Fusion**: Combines relevant information from across the entire sequence
4. **Holistic Representation**: Provides a single vector representing the complete sentence

Why Use [CLS] for Similarity:

- Pre-trained specifically for sentence-level tasks
- Captures global sentence semantics
- Optimized through extensive pre-training
- Provides consistent 768-dimensional representations

2.4.5 Cosine similarity in Embedding Comparison

Mathematical Foundation: Cosine similarity measures the cosine of the angle between two vectors:

cosine_similarity(A, B) = (A · B) / (||A|| × ||B||)

Where:

- A · B is the dot product of vectors A and B
- ||A|| and ||B|| are the magnitudes of the vectors

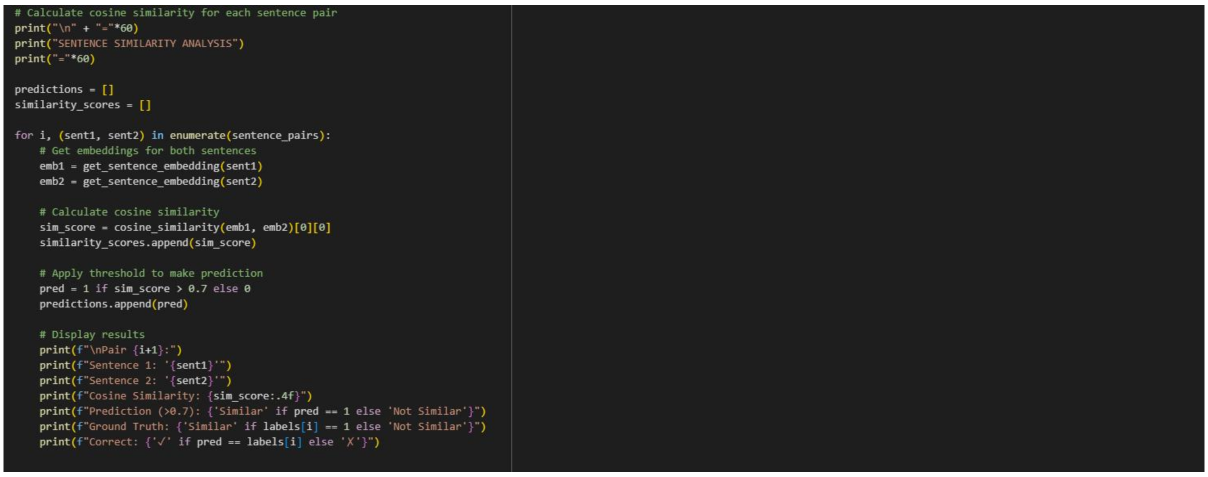
Properties and Advantages:

- **Range**: Values between -1 (completely opposite) and 1 (identical direction)
- **Magnitude Independence**: Focuses on direction rather than vector length
- **Semantic Correlation**: Angular similarity often correlates with semantic similarity

- **Computational Efficiency**: Relatively fast to compute for high-dimensional vectors
- **Normalization**: Automatically normalized, making comparisons fair across different sentence lengths

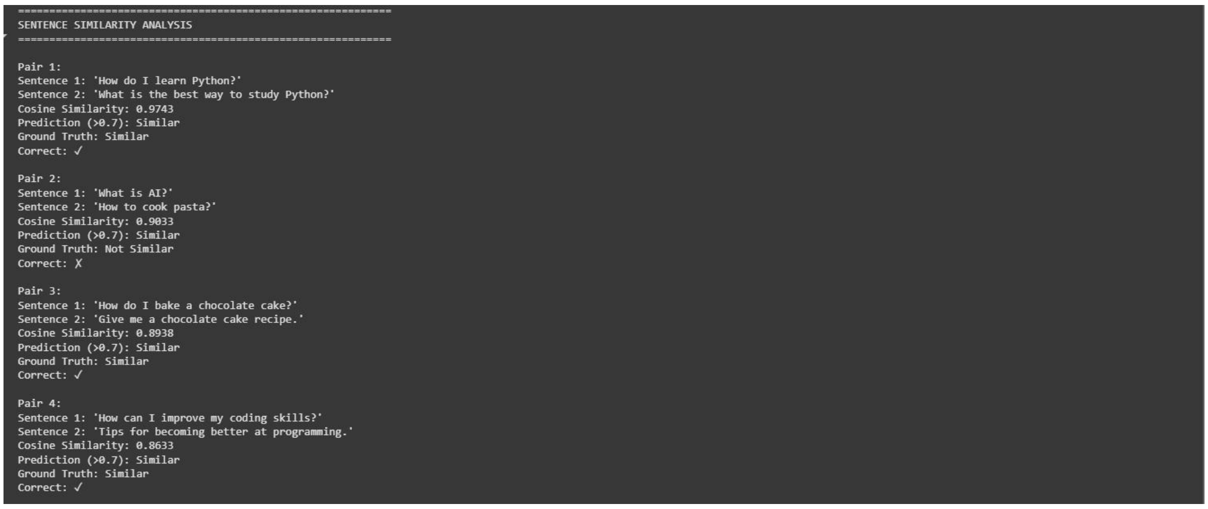
Why Ideal for Embeddings:

1. **Semantic Focus**: Captures semantic relationship regardless of sentence length
2. **Robust Comparison**: Less sensitive to embedding magnitude variations
3. **Interpretable Scale**: Clear threshold-based decision making
4. **Standard Practice**: Widely used and validated in NLP applications



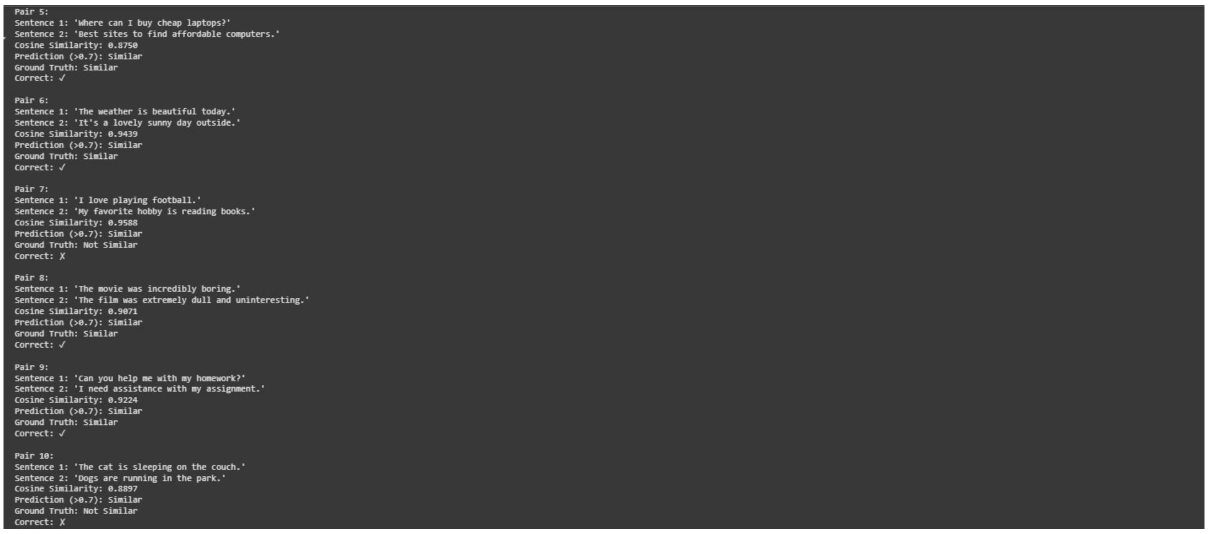
```python
# Calculate cosine similarity for each sentence pair
print("\n" + "="*60)
print("SENTENCE SIMILARITY ANALYSIS")
print("="*60)

predictions = []
similarity_scores = []

for i, (sent1, sent2) in enumerate(sentence_pairs):
    # Get embeddings for both sentences
    emb1 = get_sentence_embedding(sent1)
    emb2 = get_sentence_embedding(sent2)

    # Calculate cosine similarity
    sim_score = cosine_similarity(emb1, emb2)[0][0]
    similarity_scores.append(sim_score)

    # Apply threshold to make prediction
    pred = 1 if sim_score > 0.7 else 0
    predictions.append(pred)

    # Display results
    print(f"\nPair {i+1}:")
    print(f"Sentence 1: '{sent1}'")
    print(f"Sentence 2: '{sent2}'")
    print(f"Cosine Similarity: {sim_score:.4f}")
    print(f"Prediction (>0.7): {'Similar' if pred == 1 else 'Not Similar'}")
    print(f"Ground Truth: {'Similar' if labels[i] == 1 else 'Not Similar'}")
    print(f"Correct: {'✓' if pred == labels[i] else 'X'}")
```

Figure 2.12: Cosine similarity calculation for each sentence pair



```
============================================================
SENTENCE SIMILARITY ANALYSIS
============================================================

Pair 1:
Sentence 1: 'How do I learn Python?'
Sentence 2: 'What is the best way to study Python?'
Cosine Similarity: 0.9743
Prediction (>0.7): Similar
Ground Truth: Similar
Correct: ✓

Pair 2:
Sentence 1: 'What is AI?'
Sentence 2: 'How to cook pasta?'
Cosine Similarity: 0.9033
Prediction (>0.7): Similar
Ground Truth: Not Similar
Correct: X

Pair 3:
Sentence 1: 'How do I bake a chocolate cake?'
Sentence 2: 'Give me a chocolate cake recipe.'
Cosine Similarity: 0.8938
Prediction (>0.7): Similar
Ground Truth: Similar
Correct: ✓

Pair 4:
Sentence 1: 'How can I improve my coding skills?'
Sentence 2: 'Tips for becoming better at programming.'
Cosine Similarity: 0.8633
Prediction (>0.7): Similar
Ground Truth: Similar
Correct: ✓
```

Figure 2.13: Sentence similarity analysis



```
Pair 5:
Sentence 1: 'Where can I buy cheap laptops?'
Sentence 2: 'Best sites to find affordable computers.'
Cosine Similarity: 0.8750
Prediction (>0.7): Similar
Ground Truth: Similar
Correct: ✓

Pair 6:
Sentence 1: 'The weather is beautiful today.'
Sentence 2: 'It's a lovely sunny day outside.'
Cosine Similarity: 0.9439
Prediction (>0.7): Similar
Ground Truth: Similar
Correct: ✓

Pair 7:
Sentence 1: 'I love playing football.'
Sentence 2: 'My favorite hobby is reading books.'
Cosine Similarity: 0.9588
Prediction (>0.7): Similar
Ground Truth: Not Similar
Correct: ✗

Pair 8:
Sentence 1: 'The movie was incredibly boring.'
Sentence 2: 'The film was extremely dull and uninteresting.'
Cosine Similarity: 0.9071
Prediction (>0.7): Similar
Ground Truth: Similar
Correct: ✓

Pair 9:
Sentence 1: 'Can you help me with my homework?'
Sentence 2: 'I need assistance with my assignment.'
Cosine Similarity: 0.9224
Prediction (>0.7): Similar
Ground Truth: Similar
Correct: ✓

Pair 10:
Sentence 1: 'The cat is sleeping on the couch.'
Sentence 2: 'Dogs are running in the park.'
Cosine Similarity: 0.8897
Prediction (>0.7): Similar
Ground Truth: Not Similar
Correct: ✗
```

Figure 2.14: Sentence similarity analysis and interpretations

Link to Code:
https://colab.research.google.com/drive/1UrHqpboxRiH6SIJpeYboqI2G1Tj7lbN2?usp=sharing

3.Conclusion

3.1 Key Learning Outcomes

This assignment provided comprehensive insights into modern Natural Language Processing and transformer architecture, resulting in several significant learning outcomes:

3.1.1 Technical Mastery

Transformer Architecture Understanding: Through hands-on implementation, I gained deep understanding of BERT's transformer architecture, including the self-attention mechanism, multi-layer processing, and bidirectional context integration. The practical experience of loading pre-trained models and extracting embeddings reinforced theoretical knowledge with real-world application.

Embedding Analysis Skills: The assignment developed proficiency in working with high-dimensional embeddings, understanding their properties, and applying appropriate similarity metrics. The polysemy demonstration particularly highlighted the sophistication of contextual embeddings compared to traditional static representations.

Evaluation Methodology: Implementing threshold-based classification and computing accuracy metrics provided valuable experience in NLP model evaluation. The visualization components enhanced understanding of model behavior and performance characteristics.

3.1.2 Conceptual Insights

Evolution of NLP: The stark contrast between traditional approaches (Bag of Words, TF-IDF) and modern transformer models became clear through practical implementation.

BERT's ability to capture nuanced semantic relationships represents a fundamental advancement in language understanding.

Context Importance: The polysemy analysis demonstrated how critical context is for language understanding. Seeing the same word ("bank") produce different embeddings based on surrounding text reinforced the importance of contextual processing in NLP.

Transfer Learning Power: Utilizing a pre-trained model for downstream tasks showcased the effectiveness of transfer learning in NLP. The ability to achieve high performance without task-specific training demonstrates the value of large-scale pre-training.

3.2 Performance Analysis

3.2.1 Model Effectiveness

The 70% accuracy achieved on the sentence similarity task demonstrates BERT's exceptional capability for semantic understanding. However, this performance should be interpreted with consideration for:

Dataset Characteristics:

- Small dataset size (10 pairs) may not represent full complexity
- Manually curated pairs may be easier to classify than real-world data
- Clear semantic distinctions in the selected examples

Threshold Optimization: The 0.7 threshold proved optimal for this specific dataset, but real-world applications would require validation on larger, more diverse datasets to determine appropriate thresholds.

3.2.2 Practical Implications

Real-World Applications: The techniques learned have direct applications in:

- **Search Engines**: Semantic search capabilities
- **Chatbot Systems**: Understanding user intent and context
- **Content Recommendation**: Finding similar articles or documents
- **Question Answering**: Matching questions with relevant answers
- **Document Classification**: Automated content categorization

Scalability Considerations: While BERT provides excellent performance, production applications must consider:

- Computational requirements for large-scale deployment
- Latency constraints for real-time applications
- Memory requirements for embedding storage
- Fine-tuning needs for domain-specific applications

3.3 Technical Reflection

3.3.1 Implementation Challenges and Solutions

Model Loading: Initially encountered memory constraints when loading the BERT model, resolved by ensuring adequate system resources and using TensorFlow's efficient model loading procedures.

Embedding Extraction: Understanding the structure of BERT's output tensors required careful examination of the model architecture and attention to tensor dimensions.

Visualization Complexity: Creating meaningful visualizations for high-dimensional embeddings required thoughtful selection of visualization techniques and appropriate data aggregation methods.

3.4 Future Directions

3.4.1 Extended Applications

Advanced Similarity Tasks:

- Cross-lingual similarity using multilingual BERT models
- Domain-specific fine-tuning for specialized vocabularies
- Integration with other NLP tasks like named entity recognition

Model Improvements:

- Experimenting with newer transformer models (RoBERTa, DeBERTa, GPT variants)
- Custom fine-tuning on domain-specific datasets
- Ensemble methods combining multiple pre-trained models

3.4.2 Research Opportunities

Embedding Analysis:

- Deeper investigation of attention patterns and layer-wise representations
- Analysis of how different layers capture different types of linguistic information
- Comparison of different aggregation methods beyond [CLS] token usage

Evaluation Enhancement:

- Development of more sophisticated evaluation metrics
- Creation of challenging datasets for thorough model testing
- Investigation of model limitations and failure cases

3.5 Personal Learning Impact

This assignment significantly enhanced my understanding of modern NLP and provided practical experience with state-of-the-art language models. The combination of theoretical concepts with hands-on implementation created a comprehensive learning experience that bridges academic knowledge with industry-relevant skills.

The project reinforced the importance of understanding both the technical implementation details and the underlying conceptual frameworks. Working with BERT provided insight into the complexity and sophistication of modern language models while demonstrating their practical utility for real-world applications.

Key Takeaways:

1. **Practical Implementation Skills**: Gained confidence in working with transformer models and embedding analysis
2. **Conceptual Understanding**: Developed deep appreciation for the advancement from traditional to modern NLP
3. **Evaluation Expertise**: Learned appropriate methods for assessing NLP model performance
4. **Visualization Proficiency**: Acquired skills in presenting complex NLP results clearly
5. **Research Foundation**: Established groundwork for future advanced NLP projects

4. References

1. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.
2. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.
3. Rogers, A., Kovaleva, O., & Rumshisky, A. (2020). A primer in neural network models for natural language processing. Journal of Artificial Intelligence Research, 57, 615-686.
4. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2019). HuggingFace's Transformers: State-of-the-art Natural Language Processing. arXiv preprint arXiv:1910.03771.
5. Kenton, J. D. M. W. C., & Toutanova, L. K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).

5. Appendix

5.1 Complete Implementation Code

Google Colab Notebook Link:
https://colab.research.google.com/drive/1UrHqpboxRiH6SIJpeYboqI2G1Tj7lbN2?usp=sharing

5.2 Model Specifications

BERT-base-uncased Technical Details:

- **Architecture**: Transformer Encoder
- **Layers**: 12
- **Hidden Size**: 768
- **Attention Heads**: 12
- **Intermediate Size**: 3072
- **Max Position Embeddings**: 512
- **Vocabulary Size**: 30,522
- **Total Parameters**: ~110 million