```
from sentence_transformers import SentenceTransformer, InputExample,
losses
from torch.utils.data import DataLoader
from datasets import load_dataset
import torch

# Set random seed for reproducibility
torch.manual_seed(42)

# Load the PAWS dataset from Hugging Face
dataset = load_dataset("google-research-datasets/paws",
"labeled_final")

# Prepare training and validation examples separately
train_examples = []
val_examples = []

# Process training data
for item in dataset['train']:
    sentence1 = item['sentence1']
    sentence2 = item['sentence2']
    label = float(item['label'])
    train_examples.append(InputExample(texts=[sentence1, sentence2],
label=label))

# Process validation data
for item in dataset['validation']:
    sentence1 = item['sentence1']
    sentence2 = item['sentence2']
    label = float(item['label'])
    val_examples.append(InputExample(texts=[sentence1, sentence2],
label=label))

# Initialize the pre-trained model distilroberta-base-v2
model = SentenceTransformer('all-MiniLM-L12-v2')
#NV-Embed-v2
# Create DataLoaders for training and validation
train_dataloader = DataLoader(
    train_examples,
    shuffle=True,
    batch_size=256,
    collate_fn=model.smart_batching_collate
)

val_dataloader = DataLoader(
    val_examples,
    shuffle=False,
    batch_size=256,
    collate_fn=model.smart_batching_collate
)
```

```
# Define the loss function
train_loss = losses.ContrastiveLoss(model)

# Create evaluator
from sentence_transformers.evaluation import
EmbeddingSimilarityEvaluator
evaluator =
EmbeddingSimilarityEvaluator.from_input_examples(val_examples,
name='paws-validation')

# Configure training parameters
num_epochs = 10
warmup_steps = int(len(train_dataloader) * 0.1)  # 10% of training
steps

# Fine-tune the model
model.fit(
    train_objectives=[(train_dataloader, train_loss)],
    evaluator=evaluator,
    epochs=num_epochs,
    warmup_steps=warmup_steps,
    output_path="output/paws_finetuned_model",
    show_progress_bar=True
)

# Save the fine-tuned model
model.save("output/paws_finetuned_model")

# Function to calculate similarity between sentences
def calculate_similarity(model, sentence1, sentence2):
    embeddings = model.encode([sentence1, sentence2],
convert_to_tensor=True)
    similarity = torch.cosine_similarity(embeddings[0], embeddings[1],
dim=0)
    return similarity.item()

# Test the model
test_sentences = [
    ("The cat sat on the mat.", "The mat was sat on by the cat."),
    ("He is going to school.", "He is cooking food.")
]

print("\nTesting the fine-tuned model:")
for s1, s2 in test_sentences:
    similarity = calculate_similarity(model, s1, s2)
    print(f"\nSentence 1: '{s1}'")
    print(f"Sentence 2: '{s2}'")
    print(f"Similarity score: {similarity:.4f}")
```

```
<IPython.core.display.HTML object>
```

{"model_id":"cc9668311c6f43deb0bf154ed9b9c32b","version_major":2,"version_minor":0}

```
Testing the fine-tuned model:

Sentence 1: 'The cat sat on the mat.'
Sentence 2: 'The mat was sat on by the cat.'
Similarity score: 0.8542

Sentence 1: 'He is going to school.'
Sentence 2: 'He is cooking food.'
Similarity score: 0.5833
```

```
from sentence_transformers import SentenceTransformer, InputExample,
losses
from torch.utils.data import DataLoader
import torch
from datasets import load_dataset
from sklearn.metrics import accuracy_score,
precision_recall_fscore_support
import numpy as np

# Load the trained model
model = SentenceTransformer('output/paws_finetuned_model')

# Load test set from PAWS dataset
dataset = load_dataset("google-research-datasets/paws",
"labeled_final")
test_data = dataset['test']

# Convert test data to numpy arrays
test_sentences1 = np.array(test_data['sentence1'])
test_sentences2 = np.array(test_data['sentence2'])
test_labels = np.array(test_data['label'])

# Function to predict similarity and convert to binary prediction
def predict_paraphrase(model, sentences1, sentences2, threshold=0.5):
    # Encode all sentences
    embeddings1 = model.encode(sentences1, convert_to_tensor=True,
batch_size=32)
    embeddings2 = model.encode(sentences2, convert_to_tensor=True,
batch_size=32)

    # Calculate cosine similarities
    similarities = torch.nn.functional.cosine_similarity(embeddings1,
embeddings2)
    print(similarities)
    # Convert similarities to predictions (0 or 1)
```

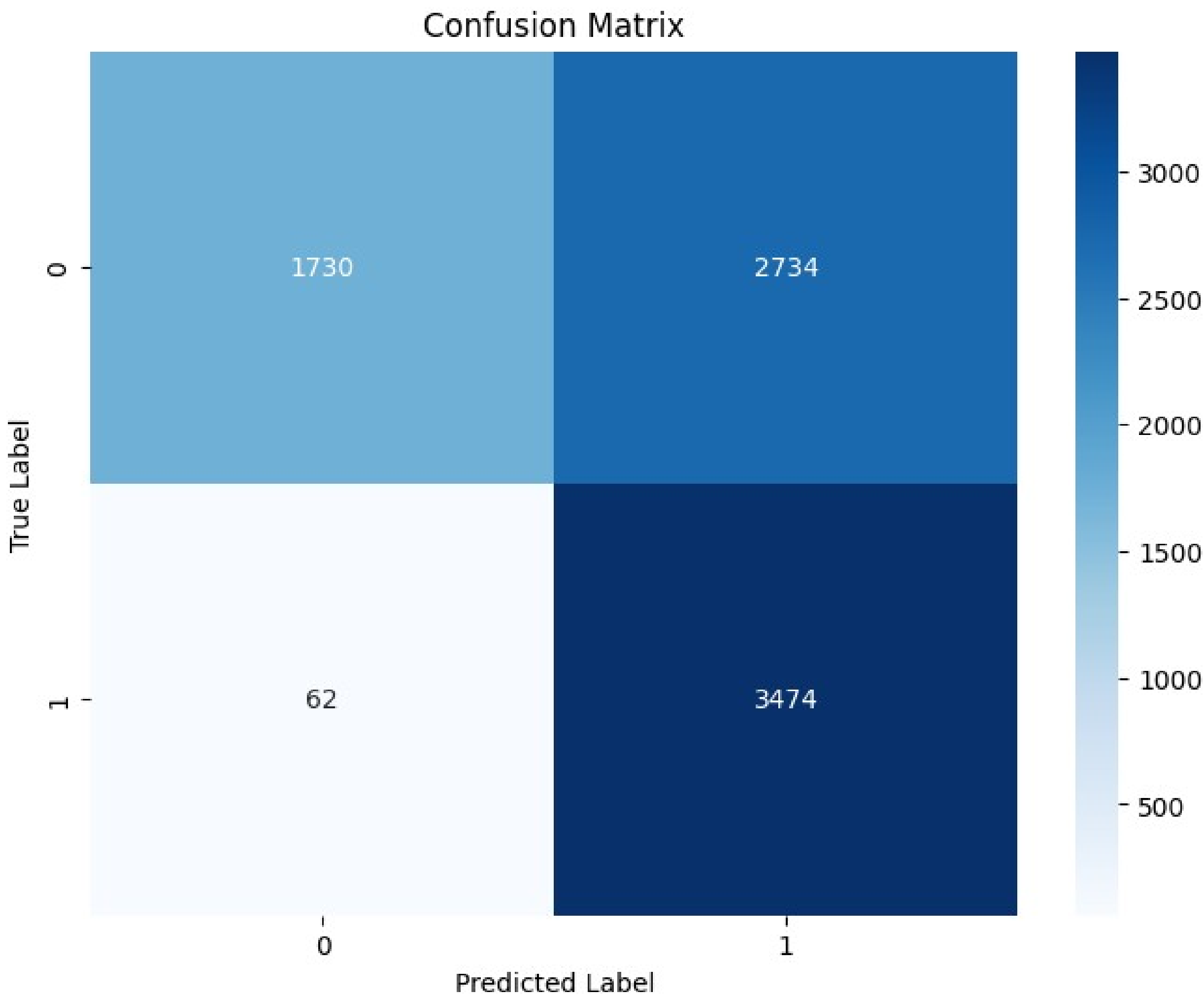
```
    predictions = (similarities > threshold).cpu().numpy().astype(int)
    return predictions, similarities.cpu().numpy()

# Make predictions on test set
print("Making predictions on test set...")
predictions, similarities = predict_paraphrase(model, test_sentences1,
test_sentences2)

# Calculate metrics
accuracy = accuracy_score(test_labels, predictions)
precision, recall, f1, _ =
precision_recall_fscore_support(test_labels, predictions,
average='binary')

# Print overall metrics
print("\nTest Set Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")

# Print some example predictions
print("\nExample Predictions:")
for i in range(min(5, len(test_sentences1))):
    print(f"\nSentence 1: {test_sentences1[i]}")
    print(f"Sentence 2: {test_sentences2[i]}")
    print(f"True Label: {test_labels[i]}")
    print(f"Predicted Label: {predictions[i]}")
    print(f"Similarity Score: {similarities[i]:.4f}")

# Calculate confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(test_labels, predictions)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()

# Save detailed results to a file
import pandas as pd

results_df = pd.DataFrame({
    'Sentence1': test_sentences1,
    'Sentence2': test_sentences2,
    'True_Label': test_labels,
```

```python
    'Predicted_Label': predictions,
    'Similarity_Score': similarities
})

# Save to CSV
results_df.to_csv('test_results.csv', index=False)
print("\nDetailed results have been saved to 'test_results.csv'")

# Analysis of error cases
error_cases = results_df[results_df['True_Label'] !=
results_df['Predicted_Label']]
print(f"\nNumber of misclassified cases: {len(error_cases)}")

# Print a few error cases
print("\nSample Error Cases:")
for _, case in error_cases.head().iterrows():
    print(f"\nSentence 1: {case['Sentence1']}")
    print(f"Sentence 2: {case['Sentence2']}")
    print(f"True Label: {case['True_Label']}")
    print(f"Predicted Label: {case['Predicted_Label']}")
    print(f"Similarity Score: {case['Similarity_Score']:.4f}")

# Calculate performance across different similarity score ranges
score_ranges = [(0, 0.2), (0.2, 0.4), (0.4, 0.6), (0.6, 0.8), (0.8,
1.0)]
print("\nPerformance across similarity score ranges:")
for low, high in score_ranges:
    mask = (similarities >= low) & (similarities < high)
    if np.any(mask):
        range_acc = accuracy_score(test_labels[mask],
predictions[mask])
        n_samples = np.sum(mask)
        print(f"Range {low:.1f}-{high:.1f}: Accuracy = {range_acc:.4f}
(n={n_samples})")

# Find optimal threshold
print("\nFinding optimal threshold...")
thresholds = np.arange(0, 1.1, 0.1)
best_accuracy = 0
best_threshold = 0.5

for threshold in thresholds:
    threshold_predictions = (similarities > threshold).astype(int)
    accuracy = accuracy_score(test_labels, threshold_predictions)
    print(f"Threshold: {threshold:.1f}, Accuracy: {accuracy:.4f}")

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_threshold = threshold
```

```
print(f"\nBest threshold: {best_threshold:.2f}")
print(f"Best accuracy: {best_accuracy:.4f}")
```

```
Making predictions on test set...
tensor([0.6468, 0.2180, 0.9507,  ..., 0.9978, 0.7373, 0.9852],
device='cuda:0')

Test Set Metrics:
Accuracy: 0.6505
Precision: 0.5596
Recall: 0.9825
F1 Score: 0.7131

Example Predictions:

Sentence 1: This was a series of nested angular standards , so that
measurements in azimuth and elevation could be done directly in polar
coordinates relative to the ecliptic .
Sentence 2: This was a series of nested polar scales , so that
measurements in azimuth and elevation could be performed directly in
angular coordinates relative to the ecliptic .
True Label: 0
Predicted Label: 1
Similarity Score: 0.6468

Sentence 1: His father emigrated to Missouri in 1868 but returned when
his wife became ill and before the rest of the family could also go to
America .
Sentence 2: His father emigrated to America in 1868 , but returned
when his wife became ill and before the rest of the family could go to
Missouri .
True Label: 0
Predicted Label: 0
Similarity Score: 0.2180

Sentence 1: In January 2011 , the Deputy Secretary General of FIBA
Asia , Hagop Khajirian , inspected the venue together with SBP -
President Manuel V. Pangilinan .
Sentence 2: In January 2011 , FIBA Asia deputy secretary general Hagop
Khajirian along with SBP president Manuel V. Pangilinan inspected the
venue .
True Label: 1
Predicted Label: 1
Similarity Score: 0.9507

Sentence 1: Steiner argued that , in the right circumstances , the
spiritual world can be explored through direct experience by
practicing ethical and cognitive forms of rigorous self-discipline .
Sentence 2: Steiner held that the spiritual world can be researched in
the right circumstances through direct experience , by persons
```

```
practicing rigorous forms of ethical and cognitive self-discipline .
True Label: 0
Predicted Label: 1
Similarity Score: 0.7855

Sentence 1: Luciano Williames Dias ( born July 25 , 1970 ) is a
Brazilian football coach and former player .
Sentence 2: Luciano Williames Dias ( born 25 July 1970 ) is a former
football coach and Brazilian player .
True Label: 0
Predicted Label: 1
Similarity Score: 0.5325
```




```
Detailed results have been saved to 'test_results.csv'

Number of misclassified cases: 2796
```

```
Sample Error Cases:

Sentence 1: This was a series of nested angular standards , so that
measurements in azimuth and elevation could be done directly in polar
coordinates relative to the ecliptic .
Sentence 2: This was a series of nested polar scales , so that
measurements in azimuth and elevation could be performed directly in
angular coordinates relative to the ecliptic .
True Label: 0
Predicted Label: 1
Similarity Score: 0.6468

Sentence 1: Steiner argued that , in the right circumstances , the
spiritual world can be explored through direct experience by
practicing ethical and cognitive forms of rigorous self-discipline .
Sentence 2: Steiner held that the spiritual world can be researched in
the right circumstances through direct experience , by persons
practicing rigorous forms of ethical and cognitive self-discipline .
True Label: 0
Predicted Label: 1
Similarity Score: 0.7855

Sentence 1: Luciano Williames Dias ( born July 25 , 1970 ) is a
Brazilian football coach and former player .
Sentence 2: Luciano Williames Dias ( born 25 July 1970 ) is a former
football coach and Brazilian player .
True Label: 0
Predicted Label: 1
Similarity Score: 0.5325

Sentence 1: The smallest number that can be represented in two
positive and seventh ways as a sum of four different powers is
2056364173794800 .
Sentence 2: The smallest number that can be represented as a sum of
four positive seventh potences in two different ways is
2056364173794800 .
True Label: 0
Predicted Label: 1
Similarity Score: 0.7584

Sentence 1: The Villa Pesquera facilities are owned by the
Municipality of Ponce , but operated by the fishermen themselves .
Sentence 2: The facilities of Villa Pesquera are operated by the
Municipality of Ponce , but are owned by the fishermen .
True Label: 0
Predicted Label: 1
Similarity Score: 0.5598

Performance across similarity score ranges:
Range 0.0-0.2: Accuracy = 0.9571 (n=163)
```

```
Range 0.2-0.4: Accuracy = 0.9711 (n=1002)
Range 0.4-0.6: Accuracy = 0.4988 (n=1273)
Range 0.6-0.8: Accuracy = 0.1738 (n=1323)
Range 0.8-1.0: Accuracy = 0.7567 (n=4213)

Finding optimal threshold...
Threshold: 0.0, Accuracy: 0.4429
Threshold: 0.1, Accuracy: 0.4460
Threshold: 0.2, Accuracy: 0.4615
Threshold: 0.3, Accuracy: 0.5126
Threshold: 0.4, Accuracy: 0.5795
Threshold: 0.5, Accuracy: 0.6505
Threshold: 0.6, Accuracy: 0.7219
Threshold: 0.7, Accuracy: 0.7844
Threshold: 0.8, Accuracy: 0.8297
Threshold: 0.9, Accuracy: 0.8448
Threshold: 1.0, Accuracy: 0.5585

Best threshold: 0.90
Best accuracy: 0.8448
```