Sarcasm Detection in YouTube Comments using Transformers

This notebook contains the code used for the Transformer model performances in detecting Sarcasm in YouTube comments

Import libraries and requirements

```
!pip install emoji==0.6.0
```

Show hidden output

```
!pip install -U datasets numpy
```

Show hidden output

```
import pandas as pd
import torch
import torch.nn as nn
import numpy as np
from datasets import Dataset
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer,
    EarlyStoppingCallback
)
from sklearn.metrics import (
    accuracy_score,
    f1_score,
    precision_score,
    recall_score,
    confusion_matrix,
    classification_report,
    precision_recall_fscore_support
)
from sklearn.model_selection import train_test_split
import torch.nn.functional as F
```

Dataset Preparation

```
# Load dataset
df = pd.read_csv("/content/youtube_comments2.csv", quotechar='"')

# Convert boolean labels to integers
df['sarcastic'] = df['sarcastic'].astype(int)
df = df[['video_id','comment', 'sarcastic']].dropna()
```

Data statistics

```
# Sarcasm percentage per video
sarcasm_stats = (df.groupby('video_id')['sarcastic'].agg(['count', 'sum']).rename
                                                                          'su
)

sarcasm_stats['sarcasm_percentage'] = (100 * sarcasm_stats['sarcastic_comments']
print(sarcasm_stats)

# Get average sarcastic percentage
avg = sarcasm_stats['sarcasm_percentage'].mean()
print(f"Average Sarcasm Percentage: {avg}%")
```

```
             total_comments  sarcastic_comments  sarcasm_percentage
video_id
4VGd-pvSc0w             300                  79               26.33
Ryq4lLnTmog             285                  76               26.67
a1rpr0Afhfg             300                  56               18.67
eBfw5NMgizU             300                  74               24.67
Average Sarcasm Percentage: 24.085%
```

Model Performance without Context Integration

BERTweet

```
# Train/test split
train_df, test_df = train_test_split(df, test_size=0.15, stratify=df['sarcastic']

# Convert to Hugging Face Datasets
train_ds = Dataset.from_pandas(train_df.reset_index(drop=True))
test_ds = Dataset.from_pandas(test_df.reset_index(drop=True))

# Load tokenizer - switched to RoBERTa
model_name = "vinai/bertweet-base"
```

```
tokenizer = AutoTokenizer.from_pretrained(model_name, use_fast=False)

# Tokenization
def tokenize(batch):
    return tokenizer(batch['comment'], truncation=True, padding="max_length", max

train_ds = train_ds.map(tokenize, batched=True)
test_ds = test_ds.map(tokenize, batched=True)

# Rename columns
train_ds = train_ds.rename_column("sarcastic", "labels")
test_ds = test_ds.rename_column("sarcastic", "labels")
train_ds.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels
test_ds.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'

# Calculate improved class weights
neg_count = (train_df['sarcastic'] == 0).sum()
pos_count = (train_df['sarcastic'] == 1).sum()
total = neg_count + pos_count
smoothing = 0.1  # Prevents extreme weights
pos_weight = (total - pos_count + smoothing) / (pos_count + smoothing)

# Load model
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels

# Move to device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# Custom Focal Loss implementation
class FocalLoss(nn.Module):
    def __init__(self, alpha=pos_weight, gamma=2.0):
        super().__init__()
        self.alpha = alpha
        self.gamma = gamma

    def forward(self, inputs, targets):
        ce_loss = F.cross_entropy(inputs, targets, reduction='none')
        pt = torch.exp(-ce_loss)
        loss = self.alpha * (1-pt)**self.gamma * ce_loss
        return loss.mean()

# Initialize loss with class weights
loss_fn = FocalLoss(alpha=pos_weight)

# Custom Trainer
class WeightedTrainer(Trainer):
    def __init__(self, *args, loss_fn=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.loss_fn = loss_fn

    def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
        labels = inputs.pop("labels")
```

```
            outputs = model(**inputs)
            logits = outputs.logits
            loss = self.loss_fn(logits, labels)
            return (loss, outputs) if return_outputs else loss

    # Metrics with focus on sarcastic class
    def compute_metrics(pred):
        labels = pred.label_ids
        preds = pred.predictions.argmax(-1)
        return {
            "accuracy": accuracy_score(labels, preds),
            "precision_1": precision_score(labels, preds, pos_label=1, zero_division=
            "recall_1": recall_score(labels, preds, pos_label=1, zero_division=0),
            "f1_1": f1_score(labels, preds, pos_label=1, zero_division=0),
        }

    # Improved training arguments
    training_args = TrainingArguments(
        output_dir="./bertweet_model",
        eval_strategy="epoch",
        save_strategy="epoch",
        learning_rate=2e-5,
        warmup_ratio=0.1,
        lr_scheduler_type="linear",
        per_device_train_batch_size=32,
        per_device_eval_batch_size=128,
        num_train_epochs=10,
        weight_decay=0.01,
        logging_dir="./logs",
        logging_steps=50,
        load_best_model_at_end=True,
        metric_for_best_model="eval_f1_1",
        greater_is_better=True,
    )

    # Trainer
    trainer = WeightedTrainer(
        model=model,
        args=training_args,
        train_dataset=train_ds,
        eval_dataset=test_ds,
        compute_metrics=compute_metrics,
        loss_fn=loss_fn,
    )

    # Train
    trainer.train()

    # Final evaluation
    eval_results = trainer.evaluate()
    print(eval_results)

    # Detailed analysis
```

```
from sklearn.metrics import classification_report
preds = trainer.predict(test_ds)
y_true = preds.label_ids
y_pred = np.argmax(preds.predictions, axis=1)
print("\nBERTweet Classification Report:")
print(classification_report(y_true, y_pred, target_names=["Not Sarcastic", "Sarca

# Confusion Matrix
cm = confusion_matrix(y_true, y_pred)
print("\nConfusion Matrix:")
print(cm)
```

Map: 100% 1007/1007 [00:00<00:00, 3640.25 examples/s]

Map: 100% 178/178 [00:00<00:00, 2690.81 examples/s]

```
Some weights of RobertaForSequenceClassification were not initialized from th
You should probably TRAIN this model on a down-stream task to be able to use
```

[320/320 01:19, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	Precision 1	Recall 1	F1 1
1	No log	0.431034	0.758427	0.000000	0.000000	0.000000
2	0.446000	0.394476	0.758427	0.500000	0.046512	0.085106
3	0.446000	0.360045	0.792135	0.593750	0.441860	0.506667
4	0.364900	0.422698	0.786517	0.608696	0.325581	0.424242
5	0.216200	0.396691	0.808989	0.600000	0.627907	0.613636
6	0.216200	0.613200	0.803371	0.600000	0.558140	0.578313
7	0.103300	0.657587	0.808989	0.604651	0.604651	0.604651
8	0.056500	0.805874	0.803371	0.625000	0.465116	0.533333
9	0.056500	0.875741	0.803371	0.605263	0.534884	0.567901
10	0.025700	0.859338	0.803371	0.605263	0.534884	0.567901

```
{'eval_loss': 0.39669135212898254, 'eval_accuracy': 0.8089887640449438, 'eval_

BERTweet Classification Report:
              precision    recall  f1-score   support

Not Sarcastic       0.88      0.87      0.87       135
    Sarcastic       0.60      0.63      0.61        43

     accuracy                           0.81       178
    macro avg       0.74      0.75      0.74       178
 weighted avg       0.81      0.81      0.81       178


Confusion Matrix:
[[117  18]
 [ 16  27]]
```

RoBERTa

```
# Train/test split
train_df, test_df = train_test_split(df, test_size=0.15, stratify=df['sarcastic']

# Convert to Hugging Face Datasets
train_ds = Dataset.from_pandas(train_df.reset_index(drop=True))
test_ds = Dataset.from_pandas(test_df.reset_index(drop=True))

# Load tokenizer - switched to RoBERTa
model_name = "roberta-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Tokenization
def tokenize(batch):
    return tokenizer(batch['comment'], truncation=True, padding="max_length", max

train_ds = train_ds.map(tokenize, batched=True)
test_ds = test_ds.map(tokenize, batched=True)

# Rename columns
train_ds = train_ds.rename_column("sarcastic", "labels")
test_ds = test_ds.rename_column("sarcastic", "labels")
train_ds.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels
test_ds.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'

# Calculate improved class weights
neg_count = (train_df['sarcastic'] == 0).sum()
pos_count = (train_df['sarcastic'] == 1).sum()
total = neg_count + pos_count
smoothing = 0.1  # Prevents extreme weights
pos_weight = (total - pos_count + smoothing) / (pos_count + smoothing)

# Load model
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels

# Move to device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# Custom Focal Loss implementation
class FocalLoss(nn.Module):
    def __init__(self, alpha=pos_weight, gamma=2.0):
        super().__init__()
        self.alpha = alpha
        self.gamma = gamma

    def forward(self, inputs, targets):
        ce_loss = F.cross_entropy(inputs, targets, reduction='none')
        pt = torch.exp(-ce_loss)
        loss = self.alpha * (1-pt)**self.gamma * ce_loss
```

```
            return loss.mean()

    # Initialize loss with class weights
    loss_fn = FocalLoss(alpha=pos_weight)

    # Custom Trainer
    class WeightedTrainer(Trainer):
        def __init__(self, *args, loss_fn=None, **kwargs):
            super().__init__(*args, **kwargs)
            self.loss_fn = loss_fn

        def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
            labels = inputs.pop("labels")
            outputs = model(**inputs)
            logits = outputs.logits
            loss = self.loss_fn(logits, labels)
            return (loss, outputs) if return_outputs else loss

    # Metrics with focus on sarcastic class
    def compute_metrics(pred):
        labels = pred.label_ids
        preds = pred.predictions.argmax(-1)
        return {
            "accuracy": accuracy_score(labels, preds),
            "precision_1": precision_score(labels, preds, pos_label=1, zero_division=
            "recall_1": recall_score(labels, preds, pos_label=1, zero_division=0),
            "f1_1": f1_score(labels, preds, pos_label=1, zero_division=0),
        }

    # improved training arguments
    training_args = TrainingArguments(
        output_dir="./Roberta_model",
        eval_strategy="epoch",
        save_strategy="epoch",
        learning_rate=2e-5,
        warmup_ratio=0.1,
        lr_scheduler_type="linear",
        per_device_train_batch_size=32,
        per_device_eval_batch_size=128,
        num_train_epochs=10,
        weight_decay=0.01,
        logging_dir="./logs",
        logging_steps=50,
        load_best_model_at_end=True,
        metric_for_best_model="eval_f1_1",
        greater_is_better=True,
    )

    # Trainer
    trainer = WeightedTrainer(
        model=model,
        args=training_args,
        train_dataset=train_ds,
```

```
        eval_dataset=test_ds,
        compute_metrics=compute_metrics,
        loss_fn=loss_fn,
        callbacks=[EarlyStoppingCallback(early_stopping_patience=3)]
    )

    # Train
    trainer.train()

    # Final evaluation
    eval_results = trainer.evaluate()
    print(eval_results)

    # Detailed analysis
    from sklearn.metrics import classification_report
    preds = trainer.predict(test_ds)
    y_true = preds.label_ids
    y_pred = np.argmax(preds.predictions, axis=1)
    print("\nRoBERTa Classification Report:")
    print(classification_report(y_true, y_pred, target_names=["Not Sarcastic", "Sarca

    # Confusion Matrix
    cm = confusion_matrix(y_true, y_pred)
    print("\nConfusion Matrix:")
    print(cm)
```

Map: 100% 1007/1007 [00:00<00:00, 12960.32 examples/s]

Map: 100% 178/178 [00:00<00:00, 7147.58 examples/s]

```
Some weights of RobertaForSequenceClassification were not initialized from th
You should probably TRAIN this model on a down-stream task to be able to use
```

[288/320 01:10 < 00:07, 4.08 it/s, Epoch 9/10]

Epoch	Training Loss	Validation Loss	Accuracy	Precision 1	Recall 1	F1 1
1	No log	0.446647	0.758427	0.000000	0.000000	0.000000
2	0.454600	0.432821	0.758427	0.000000	0.000000	0.000000
3	0.454600	0.381067	0.764045	0.538462	0.162791	0.250000
4	0.410800	0.489402	0.808989	0.695652	0.372093	0.484848
5	0.248700	0.543468	0.808989	0.621622	0.534884	0.575000
6	0.248700	0.853092	0.848315	0.766667	0.534884	0.630137
7	0.116300	1.020838	0.808989	0.615385	0.558140	0.585366
8	0.062000	1.216191	0.797753	0.589744	0.534884	0.560976
9	0.062000	1.372678	0.831461	0.685714	0.558140	0.615385

```
{'eval_loss': 0.8530915379524231, 'eval_accuracy': 0.848314606741573, 'eval_p

RoBERTa Classification Report:
              precision    recall  f1-score   support

Not Sarcastic       0.86      0.95      0.90       135
    Sarcastic       0.77      0.53      0.63        43

     accuracy                           0.85       178
    macro avg       0.82      0.74      0.77       178
 weighted avg       0.84      0.85      0.84       178


Confusion Matrix:
[[128   7]
 [ 20  23]]
```

Model Performance with Context Integration

Format model input : [Video Context] [SEP] [User Comment]

BERTweet

```
# Context per video
video_contexts = {
```

```
        "4VGd-pvSc0w": "A woman is addicted to drinking paint.",
        "a1rpr0Afhfg": "A woman is addicted to eating soap.",
        "Ryq4lLnTmog": "A woman is addicted to eating mattresses.",
        "eBfw5NMgizU": "A woman is addicted to bathing in bleach.",
    }

    # Add video context and combined input
    df['video_context'] = df['video_id'].map(video_contexts)
    df['combined_text'] = df['video_context'] + " [SEP] " + df['comment']


    # Train/test split
    train_df, test_df = train_test_split(df, test_size=0.15, stratify=df['sarcastic']

    # Convert to Hugging Face Datasets
    train_ds = Dataset.from_pandas(train_df.reset_index(drop=True))
    test_ds = Dataset.from_pandas(test_df.reset_index(drop=True))

    # Load tokenizer - switched to RoBERTa
    model_name = "vinai/bertweet-base"
    tokenizer = AutoTokenizer.from_pretrained(model_name)

    # Tokenization
    def tokenize(batch):
        return tokenizer(batch['combined_text'], truncation=True, padding="max_length

    train_ds = train_ds.map(tokenize, batched=True)
    test_ds = test_ds.map(tokenize, batched=True)

    train_ds = train_ds.rename_column("sarcastic", "labels")
    test_ds = test_ds.rename_column("sarcastic", "labels")


    train_ds.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels
    test_ds.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'

    # Calculate improved class weights
    neg_count = (train_df['sarcastic'] == 0).sum()
    pos_count = (train_df['sarcastic'] == 1).sum()
    total = neg_count + pos_count
    smoothing = 0.1  # Prevents extreme weights
    pos_weight = (total - pos_count + smoothing) / (pos_count + smoothing)

    # Load model
    model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels

    # Move to device
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = model.to(device)

    # Custom Focal Loss implementation
    class FocalLoss(nn.Module):
        def __init__(self, alpha=pos_weight, gamma=1.0):
```

```
            super().__init__()
            self.alpha = alpha
            self.gamma = gamma

        def forward(self, inputs, targets):
            ce_loss = F.cross_entropy(inputs, targets, reduction='none')
            pt = torch.exp(-ce_loss)
            loss = self.alpha * (1-pt)**self.gamma * ce_loss
            return loss.mean()

    # Initialize loss with class weights
    loss_fn = FocalLoss(alpha=pos_weight)

    # Custom Trainer
    class WeightedTrainer(Trainer):
        def __init__(self, *args, loss_fn=None, **kwargs):
            super().__init__(*args, **kwargs)
            self.loss_fn = loss_fn

        def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
            labels = inputs.pop("labels")
            outputs = model(**inputs)
            logits = outputs.logits
            loss = self.loss_fn(logits, labels)
            return (loss, outputs) if return_outputs else loss

    # Metrics with focus on sarcastic class
    def compute_metrics(pred):
        labels = pred.label_ids
        preds = pred.predictions.argmax(-1)
        return {
            "accuracy": accuracy_score(labels, preds),
            "precision_1": precision_score(labels, preds, pos_label=1, zero_division=
            "recall_1": recall_score(labels, preds, pos_label=1, zero_division=0),
            "f1_1": f1_score(labels, preds, pos_label=1, zero_division=0),
        }

    # Improved training arguments
    training_args = TrainingArguments(
        output_dir="./bertweet_video_context_model90",
        eval_strategy="epoch",
        save_strategy="epoch",
        learning_rate=2e-5,
        warmup_ratio=0.1,
        lr_scheduler_type="linear",
        per_device_train_batch_size=32,
        per_device_eval_batch_size=128,
        num_train_epochs=10,
        weight_decay=0.01,
        logging_dir="./logs",
        logging_steps=50,
        load_best_model_at_end=True,
        metric_for_best_model="eval_f1_1",
```

```
        greater_is_better=True,
    )

    # Trainer
    trainer = WeightedTrainer(
        model=model,
        args=training_args,
        train_dataset=train_ds,
        eval_dataset=test_ds,
        compute_metrics=compute_metrics,
        loss_fn=loss_fn,
    )

    # Train
    trainer.train()

    # Final evaluation
    eval_results = trainer.evaluate()
    print(eval_results)

    # Detailed analysis
    from sklearn.metrics import classification_report
    preds = trainer.predict(test_ds)
    y_true = preds.label_ids
    y_pred = np.argmax(preds.predictions, axis=1)
    print("\nBERTweet + Context Classification Report:")
    print(classification_report(y_true, y_pred, target_names=["Not Sarcastic", "Sarca

    # Confusion Matrix
    cm = confusion_matrix(y_true, y_pred)
    print("\nConfusion Matrix:")
    print(cm)
```

Map: 100% 1007/1007 [00:00<00:00, 2923.00 examples/s]

Map: 100% 178/178 [00:00<00:00, 2415.41 examples/s]

```
Some weights of RobertaForSequenceClassification were not initialized from th
You should probably TRAIN this model on a down-stream task to be able to use
```

[320/320 01:20, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	Precision 1	Recall 1	F1 1
1	No log	0.865266	0.758427	0.000000	0.000000	0.000000
2	0.878800	0.818152	0.764045	1.000000	0.023256	0.045455
3	0.878800	0.691963	0.735955	0.388889	0.162791	0.229508
4	0.805800	0.611728	0.814607	0.678571	0.441860	0.535211
5	0.522700	0.605110	0.853933	0.707317	0.674419	0.690476
6	0.522700	0.723444	0.853933	0.729730	0.627907	0.675000
7	0.244700	0.803142	0.853933	0.666667	0.790698	0.723404
8	0.120000	0.985890	0.853933	0.742857	0.604651	0.666667
9	0.120000	1.058801	0.848315	0.735294	0.581395	0.649351
10	0.080200	1.042888	0.859551	0.736842	0.651163	0.691358

```
{'eval_loss': 0.8031416535377502, 'eval_accuracy': 0.8539325842696629, 'eval_
```

```
BERTweet + Context Classification Report:
              precision    recall  f1-score   support

Not Sarcastic       0.93      0.87      0.90       135
    Sarcastic       0.67      0.79      0.72        43

     accuracy                           0.85       178
    macro avg       0.80      0.83      0.81       178
 weighted avg       0.87      0.85      0.86       178


Confusion Matrix:
[[118  17]
 [  9  34]]
```

RoBERTa

```
# Context per video
video_contexts = {
    "4VGd-pvSc0w": "Video shows a woman Heather who is addicted to drinking pain
    "a1rpr0Afhfg": "Video shows a woman Tempestt who is addicted to eating soap.
    "Ryq4lLnTmog": "Video shows a woman Jennifer who is addicted to eating mattr
    "eBfw5NMgizU": "Video shows a woman Gloria who is addicted to bathing in ble
}
```

```
# Add video context and combined input
df['video_context'] = df['video_id'].map(video_contexts)
df['combined_text'] = df['video_context'] + " [SEP] " + df['comment']


# Train/test split
train_df, test_df = train_test_split(df, test_size=0.15, stratify=df['sarcastic'

# Convert to Hugging Face Datasets
train_ds = Dataset.from_pandas(train_df.reset_index(drop=True))
test_ds = Dataset.from_pandas(test_df.reset_index(drop=True))

# Load tokenizer - switched to RoBERTa
model_name = "roberta-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Tokenization
def tokenize(batch):
    return tokenizer(batch['combined_text'], truncation=True, padding="max_lengt

train_ds = train_ds.map(tokenize, batched=True)
test_ds = test_ds.map(tokenize, batched=True)

train_ds = train_ds.rename_column("sarcastic", "labels")
test_ds = test_ds.rename_column("sarcastic", "labels")


train_ds.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label
test_ds.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels

# Calculate improved class weights
neg_count = (train_df['sarcastic'] == 0).sum()
pos_count = (train_df['sarcastic'] == 1).sum()
total = neg_count + pos_count
smoothing = 0.1  # Prevents extreme weights
pos_weight = (total - pos_count + smoothing) / (pos_count + smoothing)

# Load model
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_label

# Move to device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# Custom Focal Loss implementation
class FocalLoss(nn.Module):
    def __init__(self, alpha=pos_weight, gamma=2.0):
        super().__init__()
        self.alpha = alpha
        self.gamma = gamma

    def forward(self, inputs, targets):
```

```
            ce_loss = F.cross_entropy(inputs, targets, reduction='none')
            pt = torch.exp(-ce_loss)
            loss = self.alpha * (1-pt)**self.gamma * ce_loss
            return loss.mean()

    # Initialize loss with class weights
    loss_fn = FocalLoss(alpha=pos_weight)

    # Custom Trainer
    class WeightedTrainer(Trainer):
        def __init__(self, *args, loss_fn=None, **kwargs):
            super().__init__(*args, **kwargs)
            self.loss_fn = loss_fn

        def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
            labels = inputs.pop("labels")
            outputs = model(**inputs)
            logits = outputs.logits
            loss = self.loss_fn(logits, labels)
            return (loss, outputs) if return_outputs else loss

    # Metrics with focus on sarcastic class
    def compute_metrics(pred):
        labels = pred.label_ids
        preds = pred.predictions.argmax(-1)
        return {
            "accuracy": accuracy_score(labels, preds),
            "precision_1": precision_score(labels, preds, pos_label=1, zero_division
            "recall_1": recall_score(labels, preds, pos_label=1, zero_division=0),
            "f1_1": f1_score(labels, preds, pos_label=1, zero_division=0),
        }

    # Improved training arguments
    training_args = TrainingArguments(
        output_dir="./roberta_video_context_model",
        eval_strategy="epoch",
        save_strategy="epoch",
        learning_rate=2e-5,
        warmup_ratio=0.1,
        lr_scheduler_type="linear",
        per_device_train_batch_size=32,
        per_device_eval_batch_size=128,
        num_train_epochs=10,
        weight_decay=0.01,
        logging_dir="./logs",
        logging_steps=50,
        load_best_model_at_end=True,
        metric_for_best_model="eval_f1_1",
        greater_is_better=True,
    )

    # Trainer
    trainer = WeightedTrainer(
```

```
        model=model,
        args=training_args,
        train_dataset=train_ds,
        eval_dataset=test_ds,
        compute_metrics=compute_metrics,
        loss_fn=loss_fn,
    )

    # Train
    trainer.train()

    # Final evaluation
    eval_results = trainer.evaluate()
    print(eval_results)

    # Detailed analysis
    from sklearn.metrics import classification_report
    preds = trainer.predict(test_ds)
    y_true = preds.label_ids
    y_pred = np.argmax(preds.predictions, axis=1)
    print("\nRoBERTa + Context Classification Report:")
    print(classification_report(y_true, y_pred, target_names=["Not Sarcastic", "Sarc

    # Confusion Matrix
    cm = confusion_matrix(y_true, y_pred)
    print("\nConfusion Matrix:")
```

Map: 100% 1007/1007 [00:00<00:00, 9902.11 examples/s]

Map: 100% 178/178 [00:00<00:00, 4715.26 examples/s]

```
Some weights of RobertaForSequenceClassification were not initialized from
You should probably TRAIN this model on a down-stream task to be able to us
```

[320/320 01:16, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	Precision 1	Recall 1	F1 1