sentiment-analysis-distilbert

November 28, 2024

# 1 Sentiment Analysis Using DistilBERT

## 1.1 Project Details


- **Name**: Shrinidhi Krpete Thimmegowda
- **Email**: krpetesh@msu.edu

- **Dataset**:
  - **Source**: Amazon Product Reviews
  - **Size**: 1000 reviews
  - **Class Distribution**:
    * 500 Positive (Label: 1)
    * 500 Negative (Label: 0)

[1]:
```
import numpy as np
import pandas as pd
from transformers import DistilBertTokenizer, DistilBertModel, AdamW,␣
↪get_scheduler
from torch.utils.data import DataLoader, TensorDataset, random_split
import torch
from tqdm import tqdm
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

```
c:\Users\Admin\AppData\Local\Programs\Python\Python312\Lib\site-
packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Step 2: Load data Analize

[ ]:
```
with open('C:/Users/Admin/Downloads/archive (9)/sentiment labelled sentences/
↪amazon_cells_labelled.txt', 'r') as file:
    # Read all the content of the file
    data = file.read()
```

[3]:
```
# Split the data into lines
lines = data.strip().split("\n")
```

```python
# Split each line into two parts: review and sentiment
rows = [line.rsplit("\t", 1) for line in lines]

# Create a DataFrame
df = pd.DataFrame(rows, columns=["Review", "Sentiment"])

# Convert the 'Sentiment' column to integers
df["Sentiment"] = df["Sentiment"].astype(int)

# Display the DataFrame
print(df)
```

```
                                                Review  Sentiment
0    So there is no way for me to plug it in here i…          0
1                          Good case, Excellent value.          1
2                               Great for the jawbone.          1
3    Tied to charger for conversations lasting more…          0
4                                    The mic is great.          1
..                                                 …        …
995  The screen does get smudged easily because it …          0
996  What a piece of junk.. I lose more calls on th…          0
997                         Item Does Not Match Picture.          0
998  The only thing that disappoint me is the infra…          0
999  You can not answer calls with the unit, never …          0

[1000 rows x 2 columns]
```

[4]:
```python
df.dtypes
```

[4]:
```
Review       object
Sentiment     int32
dtype: object
```

[5]:
```python
df['Sentiment'].unique()
```

[5]:
```
array([0, 1])
```

[6]:
```python
texts = list(df["Review"])  # Reviews
labels = list(df["Sentiment"])  # Sentiments (0 or 1)
```

Step 2: Tokenize Data

[7]:
```python
tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")
inputs = tokenizer(
    texts,
    padding=True,
```

```
    truncation=True,
    max_length=128,
    return_tensors="pt"
)
labels = torch.tensor(labels)
```

Step 3: Create Dataset and DataLoader

[8]:
```
dataset = TensorDataset(inputs["input_ids"], inputs["attention_mask"], labels)
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16)
```

Step 4: Initialize DistilBERT

[9]:
```
distilbert_model = DistilBertModel.from_pretrained("distilbert-base-uncased")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
distilbert_model.to(device)
```

[9]:
```
DistilBertModel(
  (embeddings): Embeddings(
    (word_embeddings): Embedding(30522, 768, padding_idx=0)
    (position_embeddings): Embedding(512, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (transformer): Transformer(
    (layer): ModuleList(
      (0-5): 6 x TransformerBlock(
        (attention): DistilBertSdpaAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
          (activation): GELUActivation()
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
```

```
        )
      )
    )
  )
```

Step 5: Fine-Tune DistilBERT

```
[10]: optimizer = AdamW(distilbert_model.parameters(), lr=2e-5)
num_training_steps = len(train_loader) * 3  # 3 epochs
scheduler = get_scheduler("linear", optimizer=optimizer, num_warmup_steps=0,␣
 ↪num_training_steps=num_training_steps)

distilbert_model.train()
for epoch in range(3):  # Adjust number of epochs as needed
    progress_bar = tqdm(train_loader, desc=f"Epoch {epoch+1}")
    for batch in progress_bar:
        input_ids, attention_mask, _ = [b.to(device) for b in batch]

        # Forward pass
        outputs = distilbert_model(input_ids, attention_mask=attention_mask)
        embeddings = outputs.last_hidden_state[:, 0, :]  # First token␣
 ↪([CLS]-like embedding)

        # Compute dummy loss (optional for fine-tuning purposes)
        loss = embeddings.norm(2)  # Example dummy loss
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        scheduler.step()

distilbert_model.eval()
```

```
c:\Users\Admin\AppData\Local\Programs\Python\Python312\Lib\site-
packages\transformers\optimization.py:591: FutureWarning: This implementation of
AdamW is deprecated and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True`
to disable this warning
  warnings.warn(
Epoch 1: 100%|      | 50/50 [02:26<00:00,  2.93s/it]
Epoch 2: 100%|      | 50/50 [02:09<00:00,  2.60s/it]
Epoch 3: 100%|      | 50/50 [02:04<00:00,  2.49s/it]
```

```
[10]: DistilBertModel(
  (embeddings): Embeddings(
    (word_embeddings): Embedding(30522, 768, padding_idx=0)
    (position_embeddings): Embedding(512, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
```

```
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (transformer): Transformer(
    (layer): ModuleList(
      (0-5): 6 x TransformerBlock(
        (attention): DistilBertSdpaAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
          (activation): GELUActivation()
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
      )
    )
  )
)
```

Step 6: Extract Fine-Tuned Embeddings

[11]:
```
with torch.no_grad():
    outputs = distilbert_model(**inputs)
    embeddings = outputs.last_hidden_state[:, 0, :].cpu().numpy()  # Extract␣
 ↪[CLS]-like embeddings
```

Step 7: Train Logistic Regression

[12]:
```
# Split data into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(embeddings, labels.numpy(),␣
 ↪test_size=0.2, random_state=42)

# Train logistic regression
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train, y_train)
```

[12]:
```
LogisticRegression(max_iter=1000)
```

Step 8: Evaluate Logistic Regression

[13]:

```
y_pred = logistic_model.predict(X_val)
accuracy = accuracy_score(y_val, y_pred)
print(f"Validation Accuracy: {accuracy:.4f}")
print("Classification Report:")
print(classification_report(y_val, y_pred, target_names=["Negative",␣
↪"Positive"]))
```

```
Validation Accuracy: 0.8900
Classification Report:
              precision    recall  f1-score   support

    Negative       0.87      0.90      0.88        93
    Positive       0.91      0.88      0.90       107

    accuracy                           0.89       200
   macro avg       0.89      0.89      0.89       200
weighted avg       0.89      0.89      0.89       200
```

## 1.2 Results Summary

### 1.2.1 Validation Metrics

- **Validation Accuracy**: **89.0%**
  - The model correctly classified 89% of the validation dataset.

### 1.2.2 Class-Specific Metrics

- **Negative Class (Label: 0)**:
  - **Precision**: 87% – Out of all reviews predicted as `Negative`, 87% were actually `Negative`.
  - **Recall**: 90% – Out of all actual `Negative` reviews, 90% were correctly identified.
  - **F1-Score**: 88% – A balanced performance metric combining precision and recall.
- **Positive Class (Label: 1)**:
  - **Precision**: 91% – Out of all reviews predicted as `Positive`, 91% were actually `Positive`.
  - **Recall**: 88% – Out of all actual `Positive` reviews, 88% were correctly identified.
  - **F1-Score**: 90% – Strong overall performance in identifying `Positive` reviews.

### 1.2.3 Overall Metrics

- **Macro Average**:
  - Precision: 89%
  - Recall: 89%
  - F1-Score: 89%
- **Weighted Average**:
  - Precision: 89%
  - Recall: 89%
  - F1-Score: 89%

## 1.3 Key Takeaways

- The model achieves a strong balance between precision, recall, and F1-score across both classes.
- **Positive reviews** are identified with slightly higher precision (91%), reducing false positives.
- **Negative reviews** achieve slightly higher recall (90%), reducing false negatives.
- The balanced metrics indicate the model effectively handles both classes with consistent performance.