review-sentiment-analysis

November 8, 2023

1 Importing necessary libraries

```
[109]: %pip install transformers[sentencepiece]
```

```
Requirement already satisfied: transformers[sentencepiece] in
/usr/local/lib/python3.10/dist-packages (4.35.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from transformers[sentencepiece]) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in
/usr/local/lib/python3.10/dist-packages (from transformers[sentencepiece])
(0.17.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from transformers[sentencepiece]) (1.23.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers[sentencepiece])
(23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
packages (from transformers[sentencepiece]) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers[sentencepiece])
(2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from transformers[sentencepiece]) (2.31.0)
Requirement already satisfied: tokenizers<0.15,>=0.14 in
/usr/local/lib/python3.10/dist-packages (from transformers[sentencepiece])
(0.14.1)
Requirement already satisfied: safetensors>=0.3.1 in
/usr/local/lib/python3.10/dist-packages (from transformers[sentencepiece])
(0.4.0)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-
packages (from transformers[sentencepiece]) (4.66.1)
Requirement already satisfied: sentencepiece!=0.1.92,>=0.1.91 in
/usr/local/lib/python3.10/dist-packages (from transformers[sentencepiece])
(0.1.99)
Requirement already satisfied: protobuf in /usr/local/lib/python3.10/dist-
packages (from transformers[sentencepiece]) (3.20.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from huggingface-hub<1.0,>=0.16.4->transformers[sentencepiece]) (2023.6.0)
```

```
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.16.4->transformers[sentencepiece]) (4.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from
requests->transformers[sentencepiece]) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->transformers[sentencepiece]) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from
requests->transformers[sentencepiece]) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from
requests->transformers[sentencepiece]) (2023.7.22)
```

[110]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer, BertForSequenceClassification, AdamW
import torch
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import accuracy_score, classification_report,␣
  ↪confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from tqdm.notebook import tqdm
```

[111]:
```
import warnings
warnings.filterwarnings("ignore")
```

2 Creating a Sample Dataset for the task

[112]:
```
data = {
    'text': [
        "This product is amazing! I love it.",
        "The worst shopping experience ever. I'm very disappointed.",
        "Not bad, but could be better.",
        "Fast and efficient service. Highly recommended.",
        "Terrible quality and horrible customer service.",
        "Average product, nothing exceptional.",
        "Outstanding quality and exceptional customer support.",
        "I'm satisfied with my purchase. It met my expectations.",
        "I regret buying this. Such a waste of money.",
        "Good value for the price.",
        "I couldn't be happier with my purchase. It's perfect!",
        "Poor customer service and slow shipping.",
        "It's an okay product. Not great, not terrible.",
        "Prompt delivery and great product quality.",
        "The item was defective and the return process was a nightmare."
    ],
    'sentiment': ['positive', 'negative', 'neutral', 'positive', 'negative',␣
 ↪'neutral', 'positive', 'positive', 'negative', 'positive', 'positive',␣
 ↪'negative', 'positive', 'negative', 'negative']
}
```

3 Exploratory Data Analysis

3.1 Converting data into a proper dataframe

[113]:
```
df = pd.read_csv('/content/reviews.csv',error_bad_lines=False, engine="python")
df.head()
```

[113]:
```
   Unnamed: 0  Clothing ID  Age                    Title  \
0           0          767   33                      NaN
1           1         1080   34                      NaN
2           2         1077   60  Some major design flaws
3           3         1049   50        My favorite buy!
4           4          847   47         Flattering shirt

                                         Review Text  Rating  Recommended IND  \
0  Absolutely wonderful - silky and sexy and comf…       4                1
1  Love this dress!  it's sooo pretty.  i happene…       5                1
2  I had such high hopes for this dress and reall…       3                0
3  I love, love, love this jumpsuit. it's fun, fl…       5                1
4  This shirt is very flattering to all due to th…       5                1

   Positive Feedback Count   Division Name Department Name Class Name
0                        0       Initmates        Intimate  Intimates
1                        4         General         Dresses    Dresses
2                        0         General         Dresses    Dresses
3                        0  General Petite         Bottoms      Pants
4                        6         General            Tops    Blouses
```

3.2 Making a single column with both Title and Review text

[114]:
```
df = df[['Title', 'Review Text', 'Rating']]
df.head()
```

[114]:
```
                     Title                                        Review Text  \
0                      NaN  Absolutely wonderful - silky and sexy and comf…
1                      NaN  Love this dress!  it's sooo pretty.  i happene…
2  Some major design flaws  I had such high hopes for this dress and reall…
```

```
3         My favorite buy!  I love, love, love this jumpsuit. it's fun, fl…
4         Flattering shirt  This shirt is very flattering to all due to th…

   Rating
0       4
1       5
2       3
3       5
4       5
```

```
[115]: df.Title.fillna("", inplace=True)
       df['Review Text'].fillna("", inplace=True)
       df.head()
```

```
[115]:                      Title                                        Review Text  \
       0                            Absolutely wonderful - silky and sexy and comf…
       1                            Love this dress!  it's sooo pretty.  i happene…
       2  Some major design flaws  I had such high hopes for this dress and reall…
       3         My favorite buy!  I love, love, love this jumpsuit. it's fun, fl…
       4         Flattering shirt  This shirt is very flattering to all due to th…

          Rating
       0       4
       1       5
       2       3
       3       5
       4       5
```

```
[116]: df['text'] = df['Title'] + ' -- ' + df['Review Text']
       df.head()
```

```
[116]:                      Title                                        Review Text  \
       0                            Absolutely wonderful - silky and sexy and comf…
       1                            Love this dress!  it's sooo pretty.  i happene…
       2  Some major design flaws  I had such high hopes for this dress and reall…
       3         My favorite buy!  I love, love, love this jumpsuit. it's fun, fl…
       4         Flattering shirt  This shirt is very flattering to all due to th…

          Rating                                               text
       0       4   -- Absolutely wonderful - silky and sexy and …
       1       5   -- Love this dress!  it's sooo pretty.  i hap…
       2       3  Some major design flaws -- I had such high hop…
       3       5  My favorite buy! -- I love, love, love this ju…
       4       5  Flattering shirt -- This shirt is very flatter…
```

3.3 Mapping ratings to sentiments

```
[117]: sentiment_mapping = {
           1: 'very negative',
           2: 'negative',
           3: 'neutral',
           4: 'positive',
           5: 'very positive'
       }
       df['sentiment'] = df['Rating'].map(sentiment_mapping)

       df.head()
```

```
[117]:                     Title                                        Review Text  \
       0                           Absolutely wonderful - silky and sexy and comf…
       1                           Love this dress!  it's sooo pretty.  i happene…
       2  Some major design flaws  I had such high hopes for this dress and reall…
       3         My favorite buy!  I love, love, love this jumpsuit. it's fun, fl…
       4         Flattering shirt  This shirt is very flattering to all due to th…

          Rating                                              text      sentiment
       0       4   -- Absolutely wonderful - silky and sexy and …       positive
       1       5   -- Love this dress!  it's sooo pretty.  i hap…  very positive
       2       3  Some major design flaws -- I had such high hop…        neutral
       3       5  My favorite buy! -- I love, love, love this ju…  very positive
       4       5  Flattering shirt -- This shirt is very flatter…  very positive
```

3.4 Checking for any remaining null values

```
[118]: df.isna().sum()
```

```
[118]: Title          0
       Review Text    0
       Rating         0
       text           0
       sentiment      0
       dtype: int64
```

3.5 Taking only the relevant columns

```
[119]: df = df[['text', 'sentiment']]
       df.head()
```

```
[119]:                                              text      sentiment
       0   -- Absolutely wonderful - silky and sexy and …       positive
       1   -- Love this dress!  it's sooo pretty.  i hap…  very positive
       2  Some major design flaws -- I had such high hop…        neutral
```

```
3  My favorite buy! -- I love, love, love this ju…  very positive
4  Flattering shirt -- This shirt is very flatter…  very positive
```

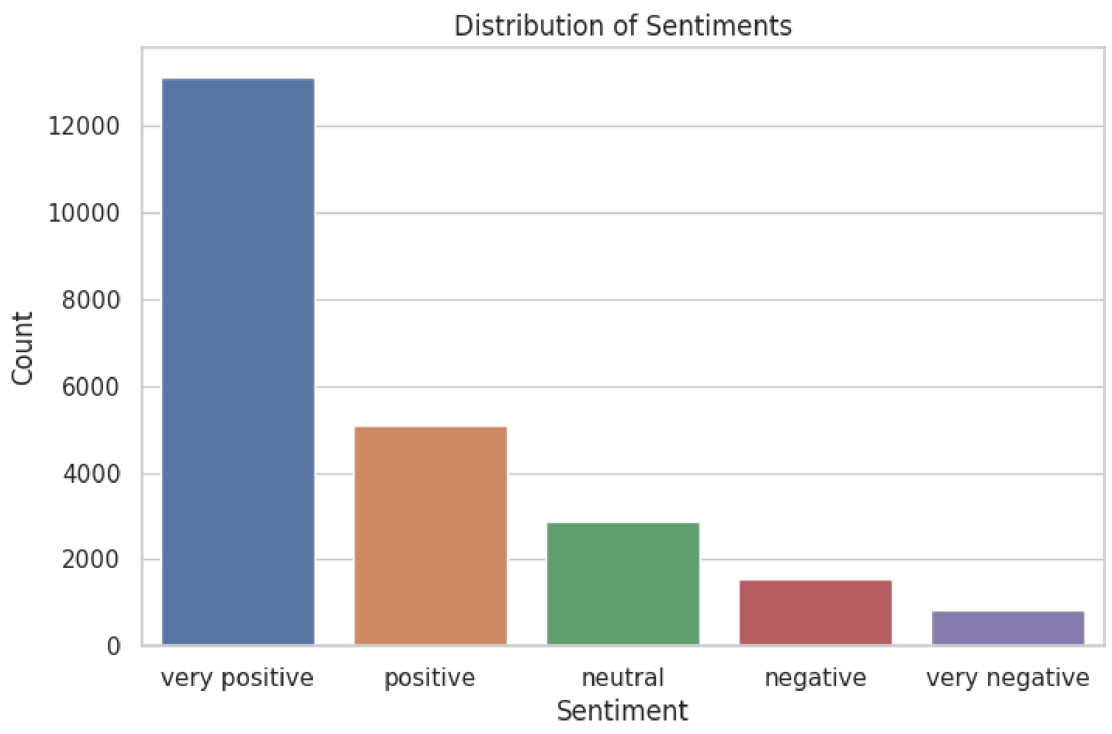
3.6 Getting full information of the dataframe

[120]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23486 entries, 0 to 23485
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   text       23486 non-null  object
 1   sentiment  23486 non-null  object
dtypes: object(2)
memory usage: 367.1+ KB
```

3.7 Plotting the distribution of sentiments in our dataset

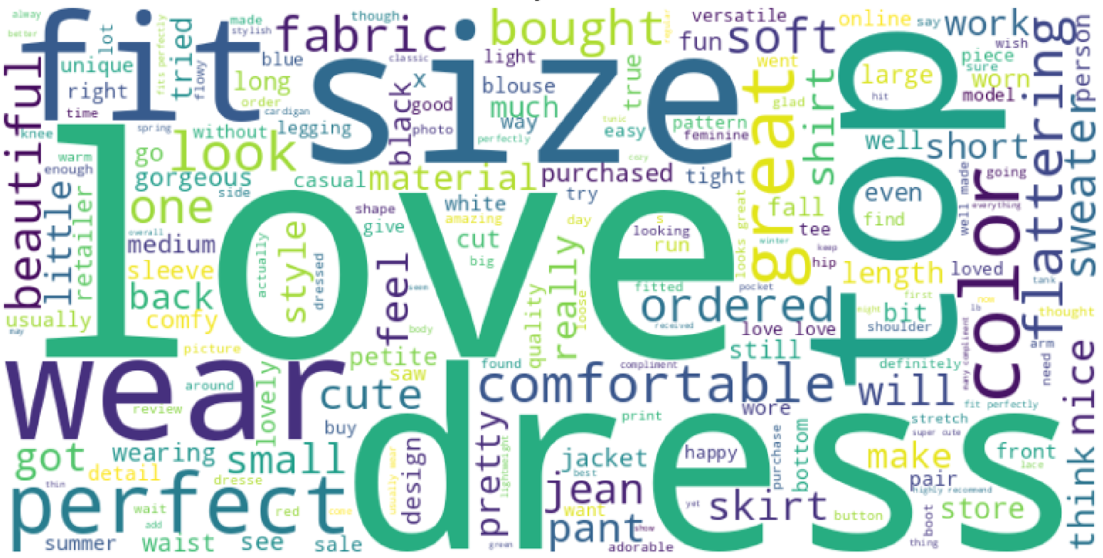
[121]:
```
sns.set(style="whitegrid")
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='sentiment', order=['very positive', 'positive',␣
  ↪'neutral', 'negative', 'very negative'])
plt.title('Distribution of Sentiments')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()
```



3.8 Creating word clouds for different types of reviews

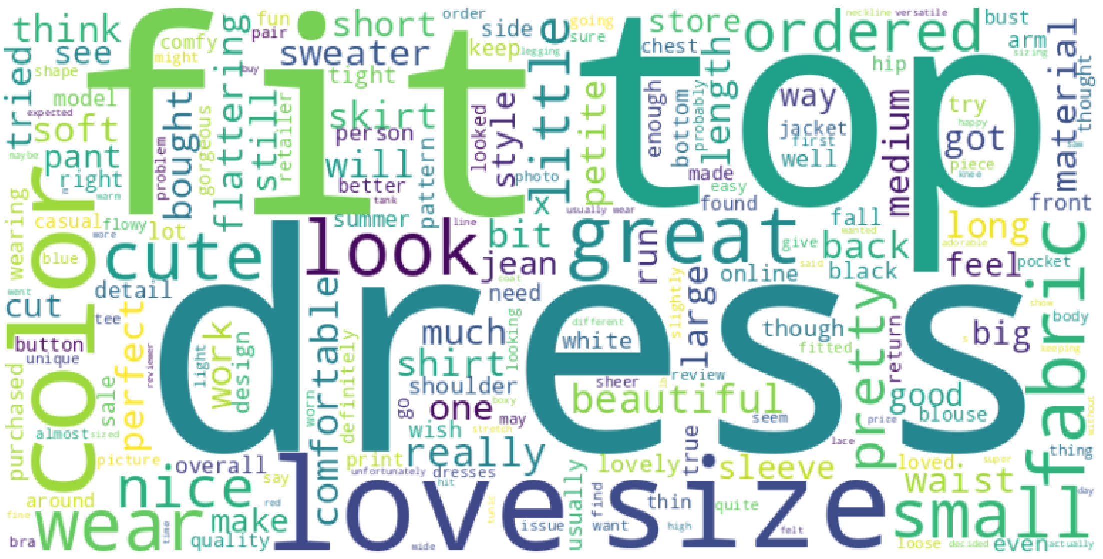
[122]:
```
very_positive_reviews = " ".join(df[df['sentiment'] == 'very positive']['text'])
wordcloud = WordCloud(width=800, height=400, background_color='white').
  ↪generate(very_positive_reviews)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title('Word Cloud for Very Positive Sentiments')
plt.show()
```

Word Cloud for Very Positive Sentiments



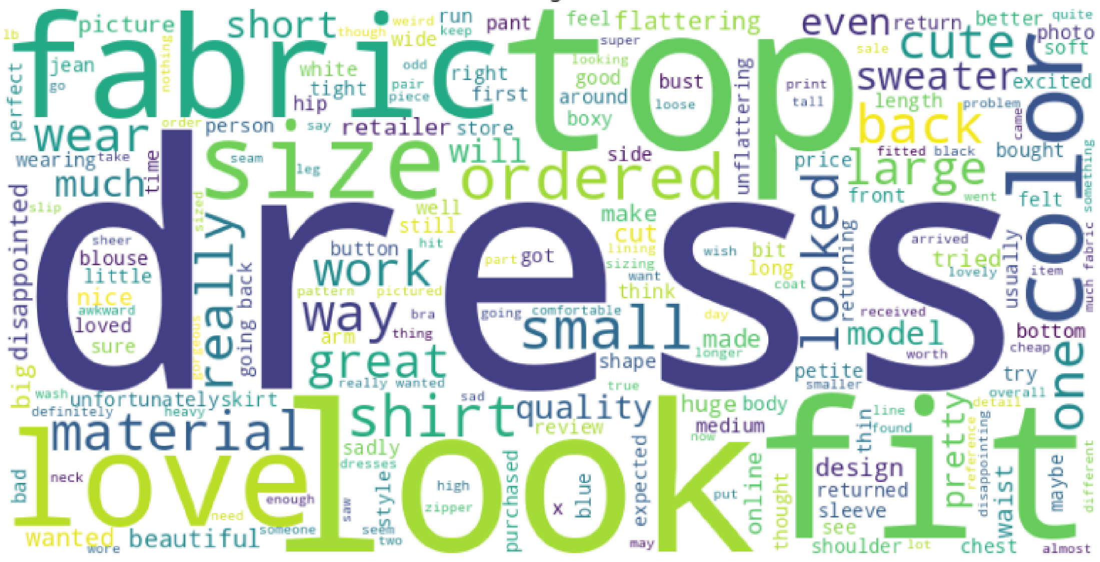
```
[123]: positive_reviews = " ".join(df[df['sentiment'] == 'positive']['text'])
       wordcloud = WordCloud(width=800, height=400, background_color='white').
        ↪generate(positive_reviews)

       plt.figure(figsize=(10, 5))
       plt.imshow(wordcloud, interpolation='bilinear')
       plt.axis("off")
       plt.title('Word Cloud for Positive Sentiments')
       plt.show()
```

Word Cloud for Positive Sentiments



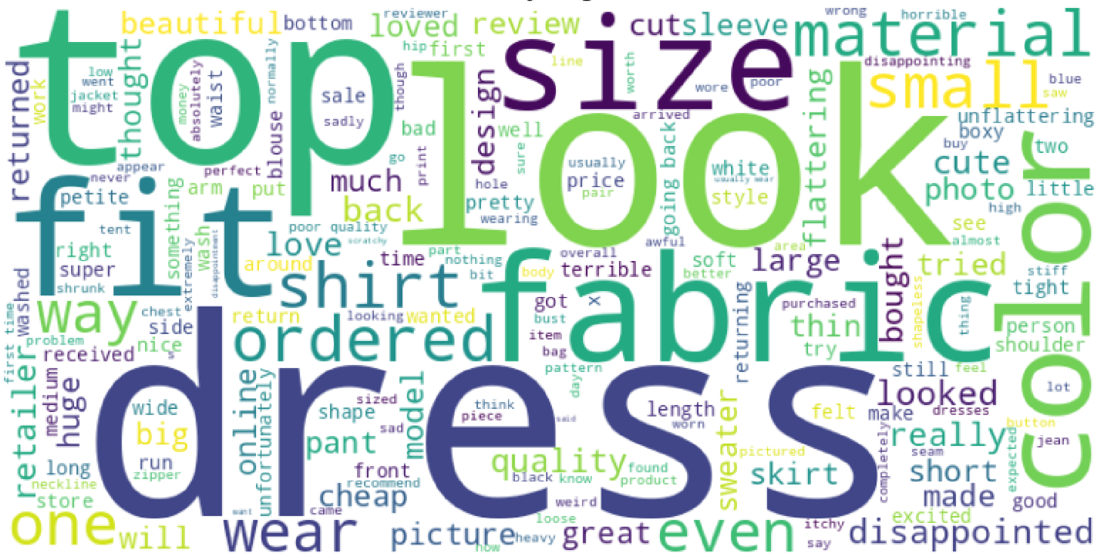
```
[124]: negative_reviews = " ".join(df[df['sentiment'] == 'negative']['text'])
       wordcloud = WordCloud(width=800, height=400, background_color='white').
        ↪generate(negative_reviews)

       plt.figure(figsize=(10, 5))
       plt.imshow(wordcloud, interpolation='bilinear')
       plt.axis("off")
       plt.title('Word Cloud for Negative Sentiments')
       plt.show()
```

Word Cloud for Negative Sentiments



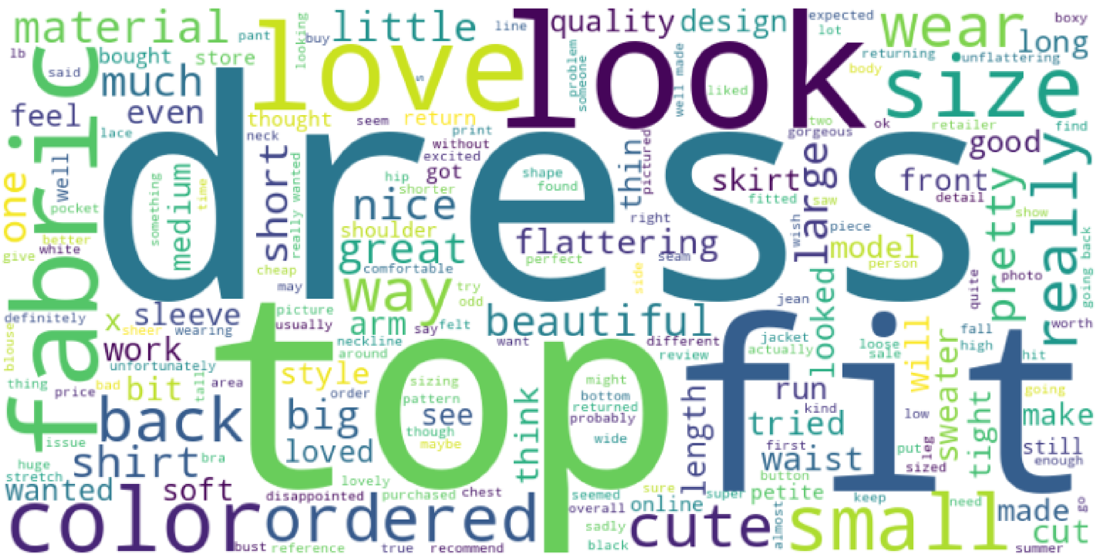
```
[125]: very_negative_reviews = " ".join(df[df['sentiment'] == 'very negative']['text'])
       wordcloud = WordCloud(width=800, height=400, background_color='white').
        ↪generate(very_negative_reviews)

       plt.figure(figsize=(10, 5))
       plt.imshow(wordcloud, interpolation='bilinear')
       plt.axis("off")
       plt.title('Word Cloud for Very Negative Sentiments')
       plt.show()
```

Word Cloud for Very Negative Sentiments



```
[126]: neutral_reviews = " ".join(df[df['sentiment'] == 'neutral']['text'])
       wordcloud = WordCloud(width=800, height=400, background_color='white').
        ↪generate(neutral_reviews)

       plt.figure(figsize=(10, 5))
       plt.imshow(wordcloud, interpolation='bilinear')
       plt.axis("off")
       plt.title('Word Cloud for Neutral Sentiments')
       plt.show()
```

Word Cloud for Neutral Sentiments



4 Splitting the dataset

4.1 Firstly only taking a certain subset of the dataset with equal number of observations of all classes

[127]:
```
subset_size = 50
grouped = df.groupby('sentiment')

# Initialize an empty list to store the samples
samples = []

# Sample one instance from each group
for group_name, group_data in grouped:
    sample = group_data.sample(n=subset_size, random_state=42)
    samples.append(sample)

subset_df = pd.concat(samples)
subset_df.reset_index(drop=True, inplace=True)
subset_df.head()
```

[127]:
```
                                                text sentiment
0  Snug and unflattering -- Would be flattering o…  negative
1  The sleeves… -- I was aware of the split s…  negative
2  Huge - swallowed me whole -- I had high hopes …  negative
3  So baggy -- I grabbed this dress to try on in …  negative
4  Not for tall ladies -- This sweater is a cute …  negative
```

[128]:
```
X = subset_df['text']
y = subset_df['sentiment']
```

[129]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

4.1.1 Mapping sentiment into numeric labels

[130]:
```
label_mapping = {'very positive':4, 'positive': 3, 'neutral': 2, 'negative': 1,␣
 ↪'very negative':0}
y_train = y_train.map(label_mapping)
y_test = y_test.map(label_mapping)
```

5 Model Training

5.1 Initializing Model

[131]:
```
model_name = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=5)
```

```
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

5.2 Getting encodings for the reviews

[132]:
```
train_encodings = tokenizer(X_train.tolist(), truncation=True, padding=True,␣
 ↪return_tensors='pt', max_length=64)
test_encodings = tokenizer(X_test.tolist(), truncation=True, padding=True,␣
 ↪return_tensors='pt', max_length=64)
```

5.3 Creating dataset and Data loaders

[133]:
```
train_dataset = TensorDataset(train_encodings['input_ids'],␣
 ↪train_encodings['attention_mask'], torch.tensor(y_train.tolist()))
test_dataset = TensorDataset(test_encodings['input_ids'],␣
 ↪test_encodings['attention_mask'], torch.tensor(y_test.tolist()))
train_loader = DataLoader(train_dataset, batch_size=3, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=3, shuffle=False)
```

5.4 Initializing optimizer and checking for cuda compatibility

[134]:
```
optimizer = AdamW(model.parameters(), lr=1e-5)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

5.5 Model is ready to train

[135]:
```
model.to(device)
model.train()
```

[135]:
```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=5, bias=True)
)
```

5.6 Initializing list to store training losses through epochs

[136]:
```
train_losses = []
```

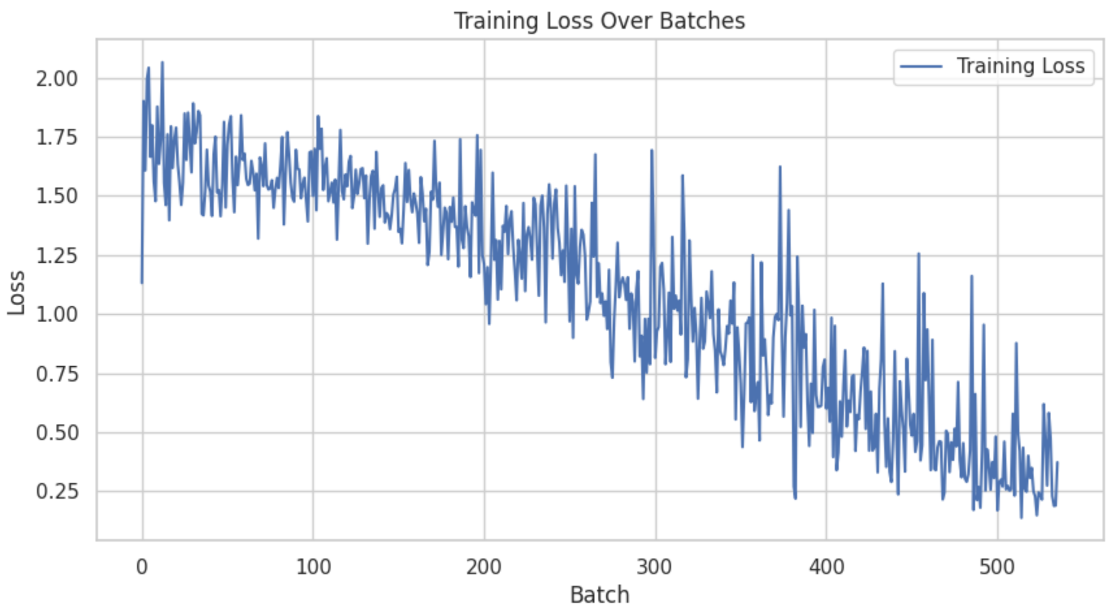
5.7 Training

[137]:
```
for epoch in range(8):
    for batch in tqdm(train_loader, desc=f'Epoch {epoch+1}'):
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device),␣
 ↪attention_mask.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()

        train_losses.append(loss.item())
```

```
Epoch 1:   0%|          | 0/67 [00:00<?, ?it/s]

Epoch 2:   0%|          | 0/67 [00:00<?, ?it/s]

Epoch 3:   0%|          | 0/67 [00:00<?, ?it/s]

Epoch 4:   0%|          | 0/67 [00:00<?, ?it/s]

Epoch 5:   0%|          | 0/67 [00:00<?, ?it/s]

Epoch 6:   0%|          | 0/67 [00:00<?, ?it/s]

Epoch 7:   0%|          | 0/67 [00:00<?, ?it/s]

Epoch 8:   0%|          | 0/67 [00:00<?, ?it/s]
```

5.8 Plotting training loss values

[138]:
```
plt.figure(figsize=(10, 5))
plt.plot(range(len(train_losses)), train_losses, label="Training Loss")
plt.xlabel("Batch")
plt.ylabel("Loss")
plt.legend()
plt.title("Training Loss Over Batches")
plt.show()
```



6 Model Evaluation

[139]:
```
model.eval()
all_preds = []
with torch.no_grad():
    for batch in test_loader:
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device),␣
 ↪attention_mask.to(device), labels.to(device)

        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        preds = torch.argmax(logits, dim=1).cpu().numpy()
        all_preds.extend(preds)
```

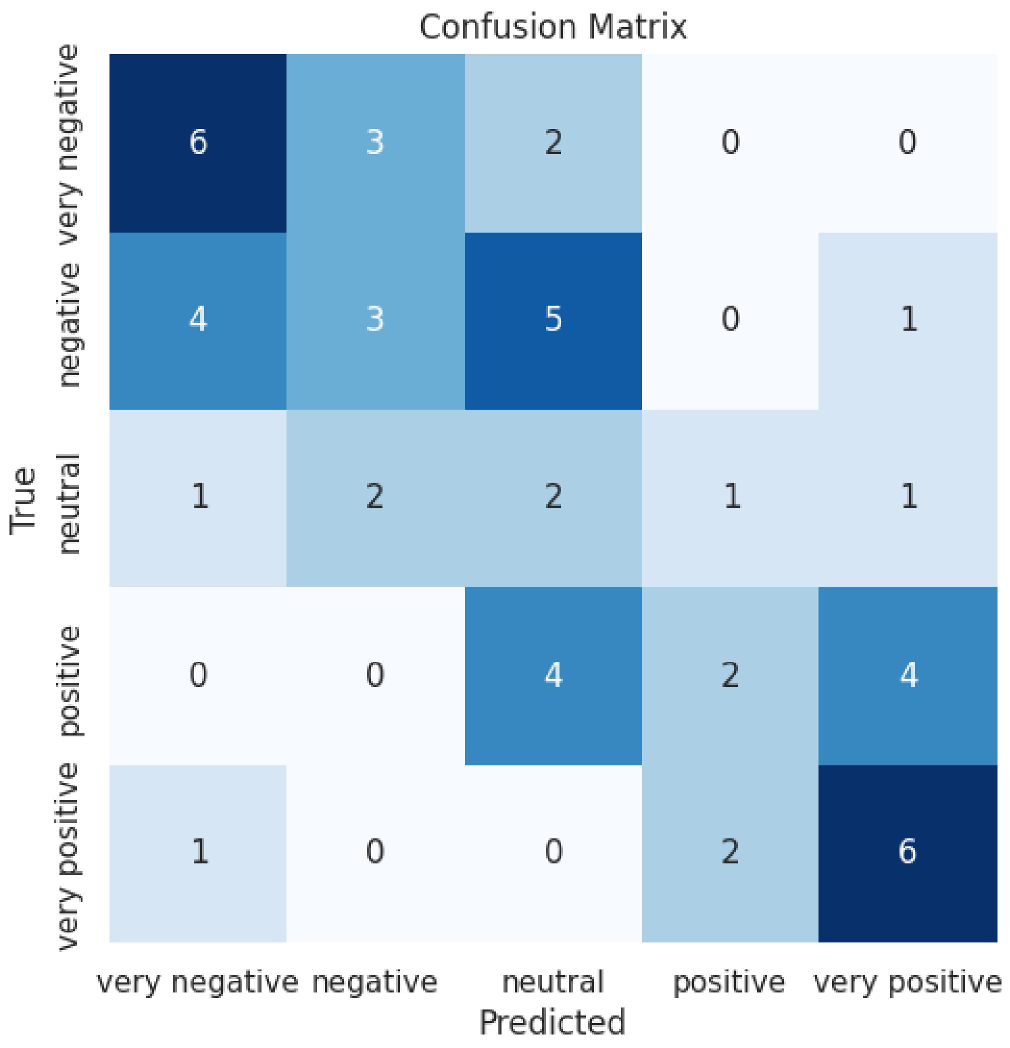
6.1 Displaying Model Results

[140]:
```
accuracy = accuracy_score(y_test, all_preds)
report = classification_report(y_test, all_preds)

print(f"Accuracy: {accuracy}\n\n")
print("Classification Report:")
print(report)
```

```
Accuracy: 0.38
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.55      0.52        11
           1       0.38      0.23      0.29        13
           2       0.15      0.29      0.20         7
           3       0.40      0.20      0.27        10
           4       0.50      0.67      0.57         9

    accuracy                           0.38        50
   macro avg       0.39      0.39      0.37        50
weighted avg       0.40      0.38      0.37        50
```

[141]:
```
confusion = confusion_matrix(y_test, all_preds)

plt.figure(figsize=(8, 6))
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues', cbar=False,␣
 ↪square=True,
            xticklabels=['very negative', 'negative', 'neutral', 'positive',␣
 ↪'very positive'],
            yticklabels=['very negative', 'negative', 'neutral', 'positive',␣
 ↪'very positive'])

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')

plt.show()
```


Confusion Matrix
True
Predicted
very negative
negative
neutral
positive
very positive
6
3
2
0
0
4
3
5
0
1
1
2
2
1
1
0
0
4
2
4
1
0
0
2
6