SENTIMENT ANALYSIS OF AMAZON PRODUCT REVIEWS FOR CONSUMER INSIGHTS

DSCI 6004-2: Natural Language Processing

Khaled Sayed, PhD – Assistant Professor

TEAM:

MADHAVI KANCHAM

VINAYKUMAR REDDY MOKU

NITHISH KUMAR MAYAVAN

Project Overview

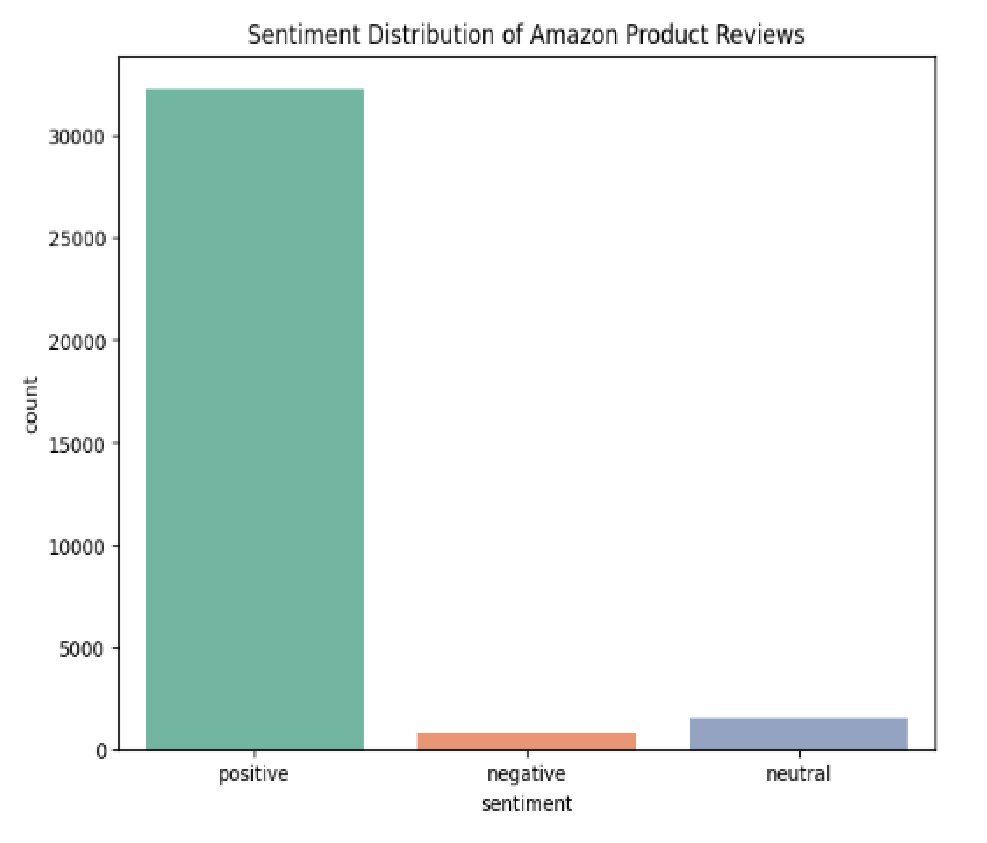
- Classify Amazon product reviews as positive, neutral or negative using sentiment analysis.
- Predict consumer satisfaction and identify trends in reviews to provide insights for product improvement and marketing strategies.
- Leverage NLP and machine learning algorithms to analyze largescale review data and automate sentiment classification.

Dataset

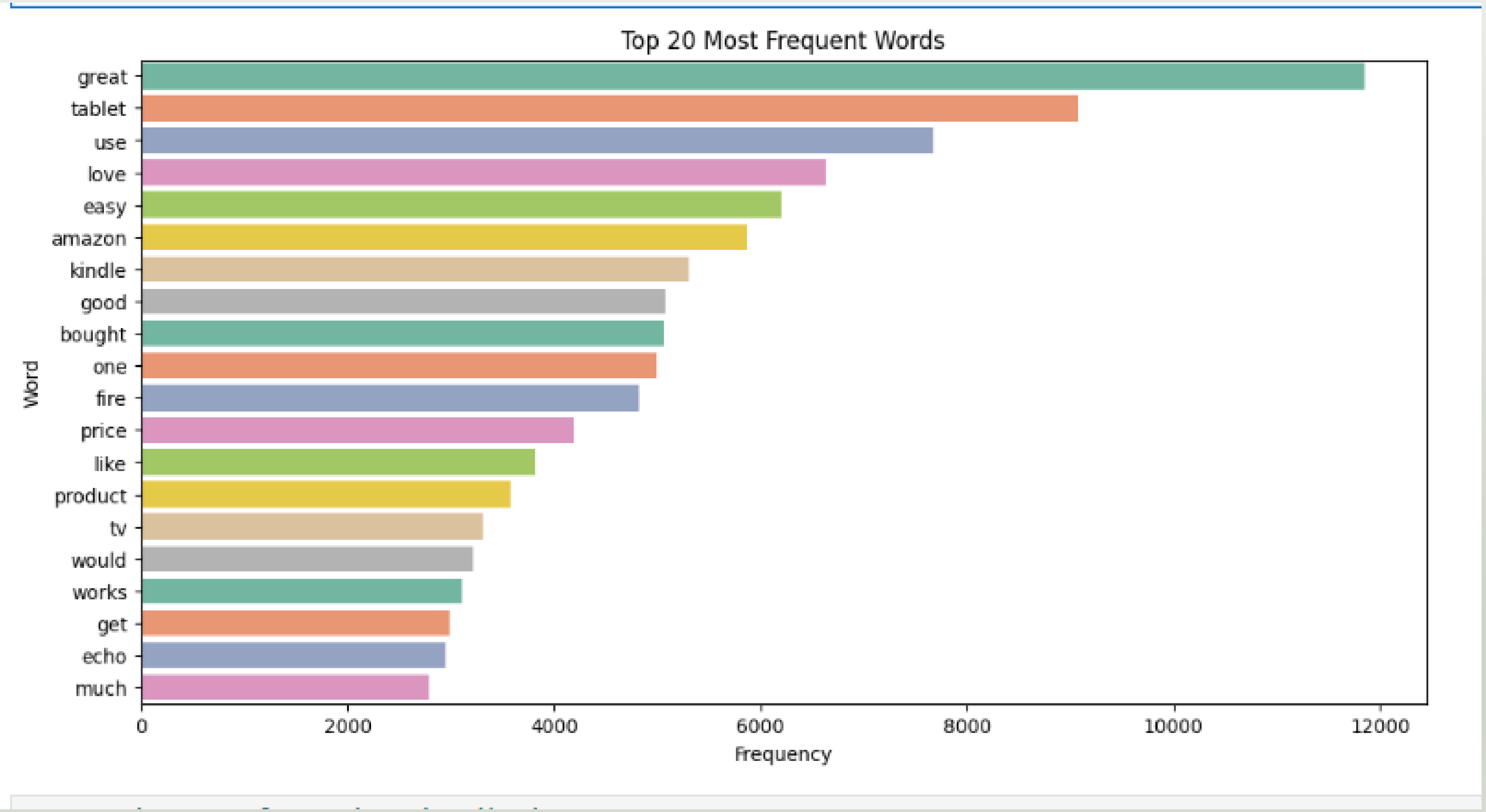
The dataset for this study was collected from Amazon through Kaggle with over 34000 rows and 21 variables on reviews. Available at: https://www.kaggle.com/datasets/bittlingmayer/amazonreviews

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34660 entries, 0 to 34659
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   id                    34660 non-null  object 
 1   name                  27900 non-null  object 
 2   asins                 34658 non-null  object 
 3   brand                 34660 non-null  object 
 4   categories            34660 non-null  object 
 5   keys                  34660 non-null  object 
 6   manufacturer          34660 non-null  object 
 7   reviews.date          34621 non-null  object 
 8   reviews.dateAdded     24039 non-null  object 
 9   reviews.dateSeen      34660 non-null  object 
 10  reviews.didPurchase   1 non-null      object 
 11  reviews.doRecommend   34066 non-null  object 
 12  reviews.id            1 non-null      float64
 13  reviews.numHelpful    34131 non-null  float64
 14  reviews.rating        34627 non-null  float64
 15  reviews.sourceURLs    34660 non-null  object 
 16  reviews.text          34659 non-null  object 
 17  reviews.title         34654 non-null  object 
 18  reviews.userCity      0 non-null      float64
 19  reviews.userProvince  0 non-null      float64
 20  reviews.username      34653 non-null  object 
dtypes: float64(5), object(16)
memory usage: 5.6+ MB
```

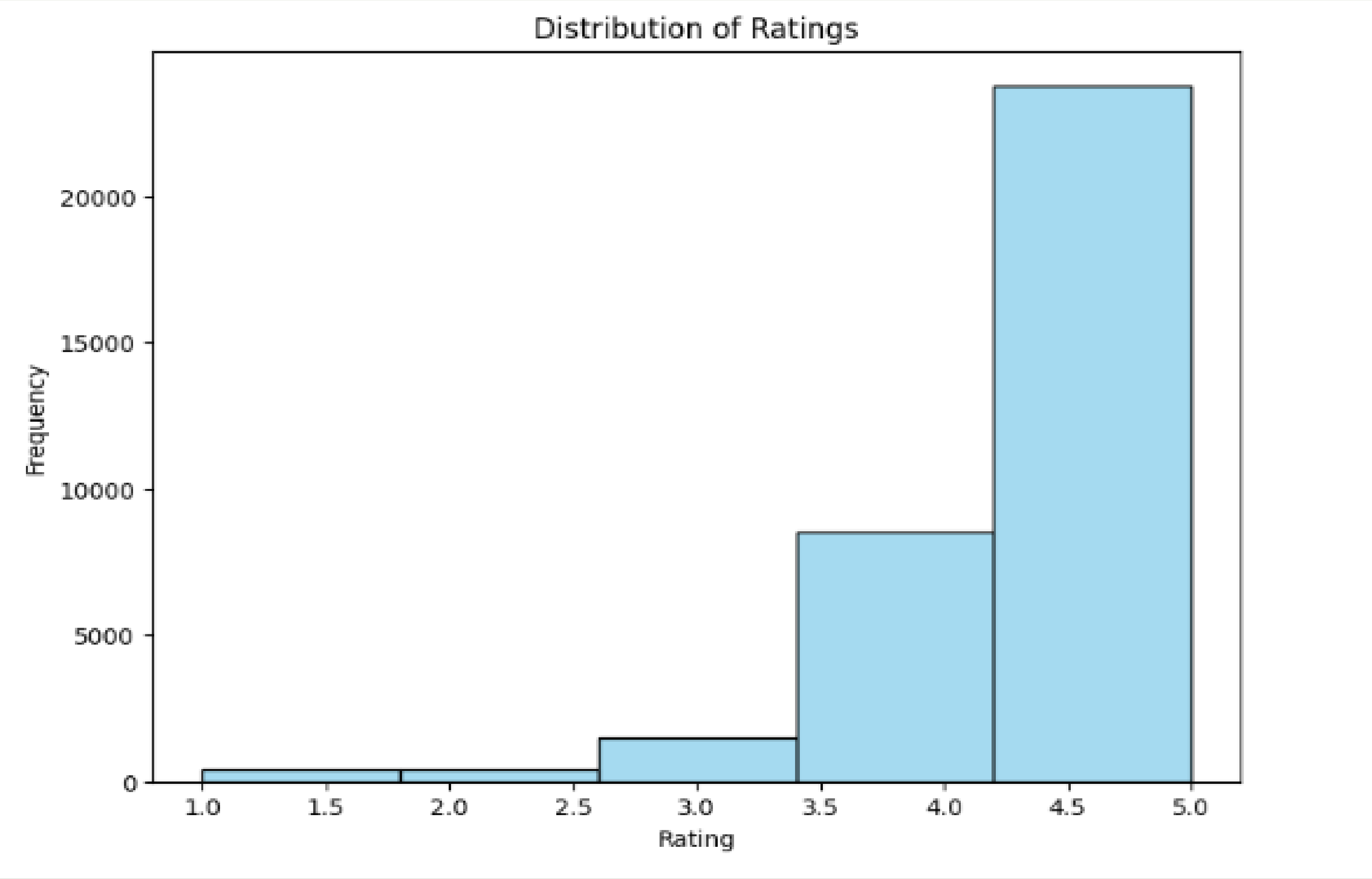
Sentiment Distribution



Most Frequent Words



Distribution of Ratings



Preprocessing

- Tokenization and Vectorization: Used TfidfVectorizer(max_features=10000) to transform preprocessed reviews into TF-IDF feature vectors.
- Sentiment Label Encoding: Converted sentiment labels (negative, neutral, positive) into numeric values (0, 1, 2).
- Data Splitting: Split the dataset into training (80%) and testing (20%) sets using train_test_split with stratified sampling.

```
Training data shape: (27700, 10000)
Test data shape: (6926, 10000)
```

Feedforward Neural Network (Multilayer Perceptron)

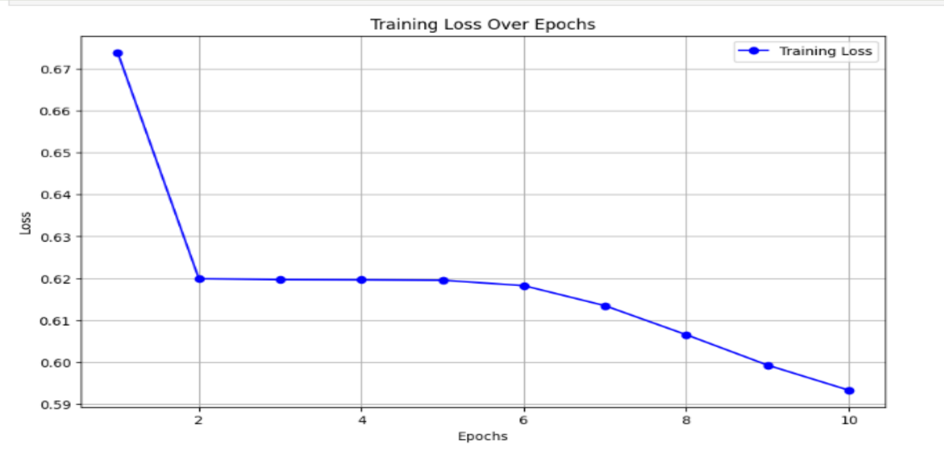
- The SentimentNN class defines a simple feedforward neural network with one hidden layer (fc1), ReLU activation, and an output layer (fc2) followed by a softmax activation for multi-class classification.
- It processes the input data (features) to predict sentiment labels (negative, neutral, positive).
- Loss and Optimization: Uses CrossEntropyLoss for multi-class classification and Adam optimizer for efficient training.
- Training Setup: Input dimension matches feature count, 128 hidden units, 3 output classes (negative, neutral, positive), with a learning rate of 0.001

14]:

```
class SentimentNN(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(SentimentNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_dim, output_dim)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.softmax(x)
        return x

# Define the model, loss function, and optimizer
input_dim = X_train.shape[1]  # Number of features
hidden_dim = 128  # Number of hidden units
output_dim = 3  # Three classes: negative, neutral, positive

model = SentimentNN(input_dim, hidden_dim, output_dim)

# Set up loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Neural Network Performance

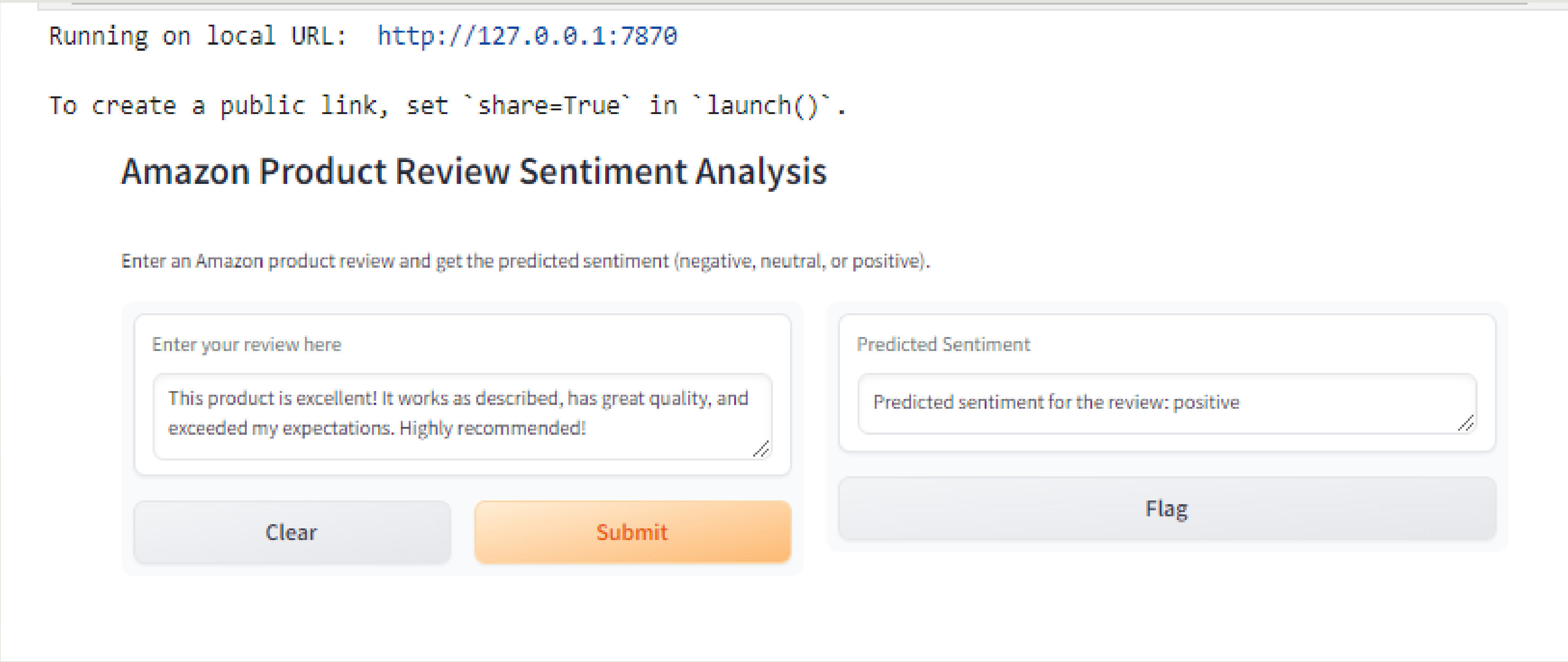
- The model shows a steady decrease in loss from 0.6738 to 0.5934 over 10 epochs, indicating improvement during training.
- Despite the low loss, performance is skewed, with the model achieving 93.17% accuracy overall, excelling in positive sentiment prediction but struggling with negative (45% precision, 27% recall) and neutral (30% precision, 11% recall) classes.



Test Accuracy: 0.9317

	precision	recall	f1-score	support
negative	0.45	0.27	0.33	162
neutral	0.30	0.11	0.16	300
positive	0.95	0.99	0.97	6464
accuracy			0.93	6926
macro avg	0.57	0.45	0.49	6926
weighted avg	0.91	0.93	0.92	6926

Neural Network deployment



LTSM MODEL

```python
class LSTMModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim, n_layers, dropout_prob):
        super(LSTMModel, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim)

        self.lstm = nn.LSTM(embedding_dim, hidden_dim, n_layers, batch_first=True, dropout=dropout_prob)

        self.fc = nn.Linear(hidden_dim, output_dim)

        # Dropout layer to prevent overfitting
        self.dropout = nn.Dropout(dropout_prob)

    def forward(self, x):
        embedded = self.embedding(x)
        lstm_out, (hidden, cell) = self.lstm(embedded)
        # We use the last hidden state as the representation for the entire sequence
        hidden = hidden[-1, :, :]
        out = self.fc(self.dropout(hidden))
        return out
```

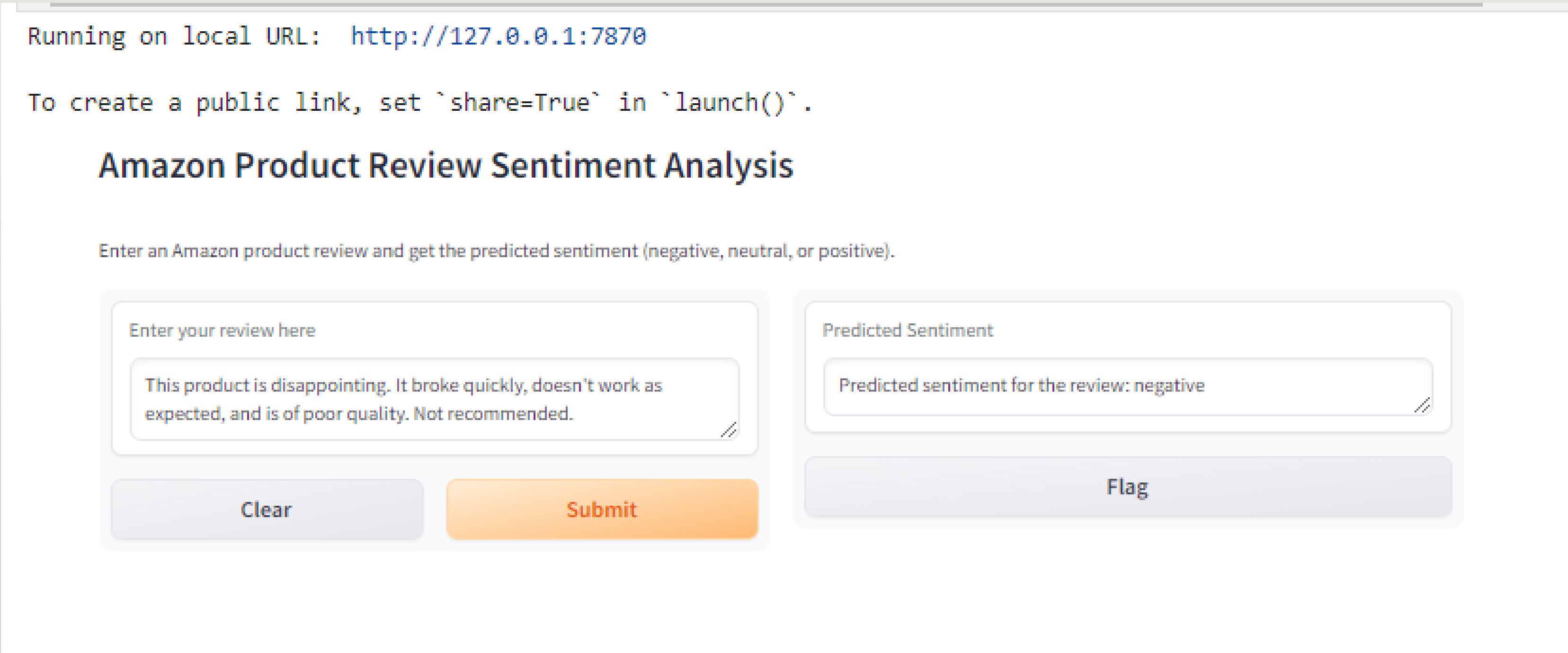
The LSTMModel defines an embedding layer, an LSTM layer, a fully connected layer, and a dropout layer to process sequential input and output sentiment predictions.

LTSM MODEL Performance

The model achieves 95% accuracy, performing excellently on positive sentiment (95% precision, 100% recall), but poorly on negative and neutral sentiments, resulting in low precision, recall, and F1-scores for those classes.

	precision	recall	f1-score	support
negative	0.00	0.00	0.00	149
neutral	0.10	0.00	0.01	222
positive	0.95	1.00	0.97	6555
accuracy			0.95	6926
macro avg	0.35	0.33	0.33	6926
weighted avg	0.90	0.95	0.92	6926

LTSM Deployment

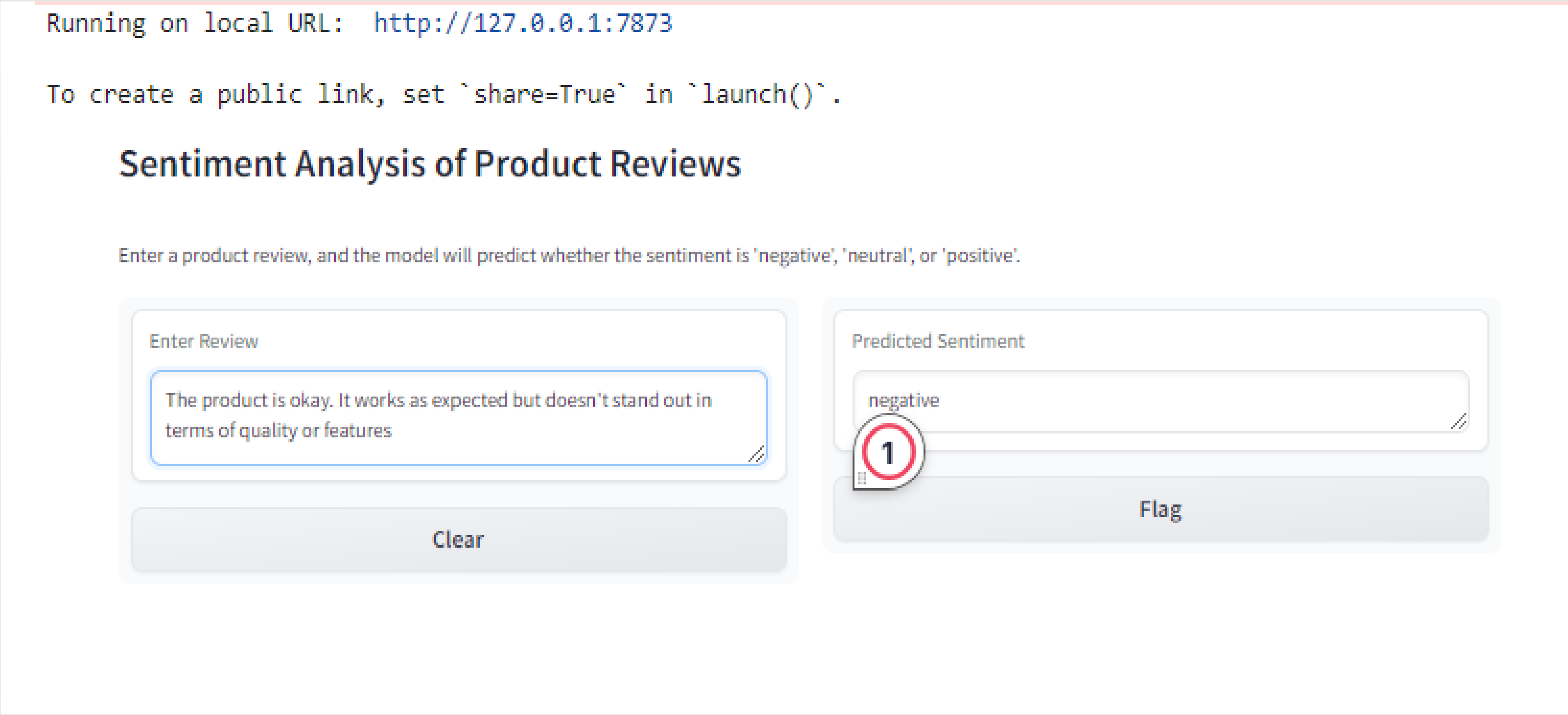


DistilBERT Model

The model loads a pretrained DistilBERT model and tokenizer for sequence classification, moving the model to the appropriate device (GPU/CPU) for efficient processing

```
# Load the pretrained DistilBERT model and tokenizer
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=3)
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

DistilBERT deployment



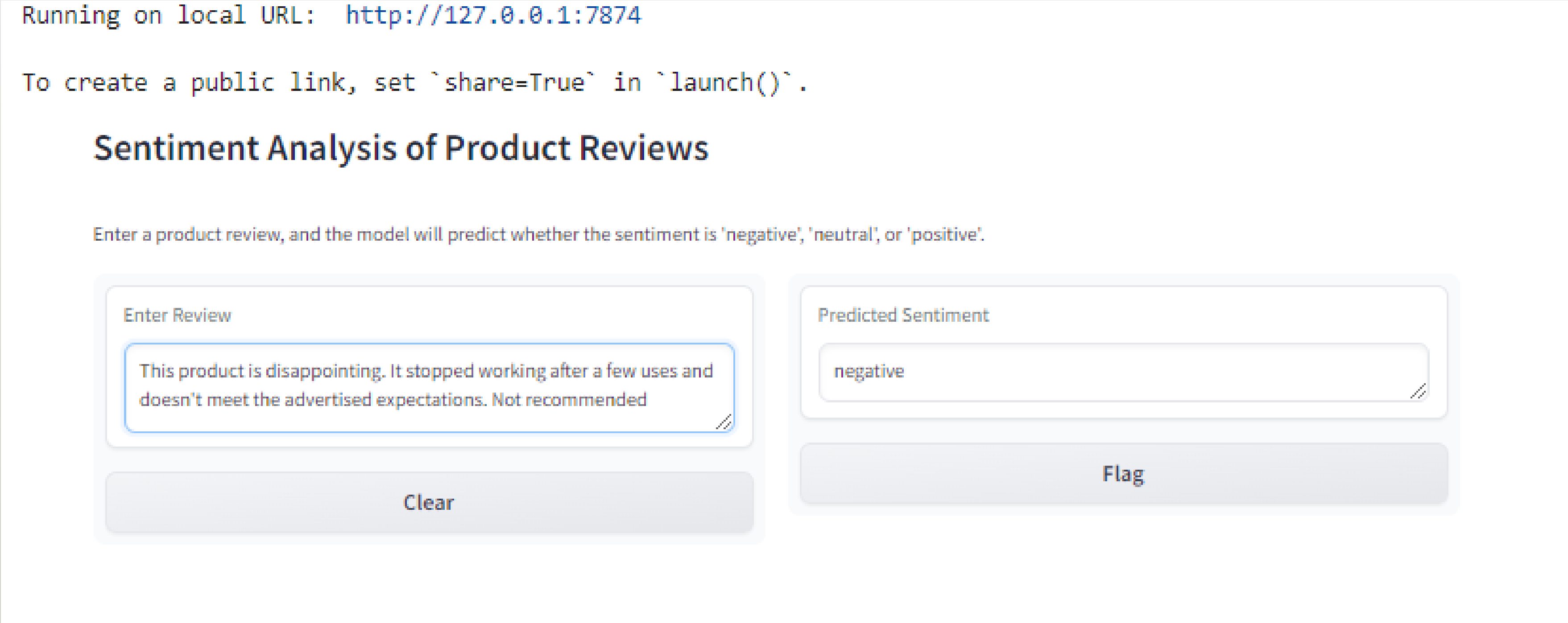
BERT Model for sentimental

The BertForSequenceClassification model consists of a BertModel with embeddings, encoder layers, and a pooler, followed by a dropout layer and a classifier that outputs predictions for three classes.

The encoder uses 12 transformer layers, each with self-attention and feed-forward components, and the pooler produces the final hidden state for classification.

```
[30]: 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=3, bias=True)
)
```

BERT Model for sentimental



THANK YOU