INFO7390: Advances Data Sci/Architecture

Project 3: Natural Language Processing – Sentiment Analysis

Naman Gupta
(NUID: 002729751)
Northeastern University
Boston Campus
gupta.naman@northeastern.edu

Introduction

Sentiment analysis, also known as opinion mining, is a Natural Language Processing (NLP) task that involves determining the sentiment or emotion expressed in a text. In this project, we focus on sentiment analysis of movie reviews sourced from the "UC Irvine Machine Learning Repository." The dataset consists of sentences labeled as 1 for positive sentiment and 0 for negative sentiment, derived from movie reviews on imdb.com. The project aims to process this data, evaluate machine learning and deep learning models, perform validation to estimate model performance robustly, and visualize the model's performance.

Data Source

The dataset used in this project is available at UC Irvine Machine Learning Repository. It contains labeled sentences from movie reviews, where a label of 1 represents positive sentiment and a label of 0 represents negative sentiment.

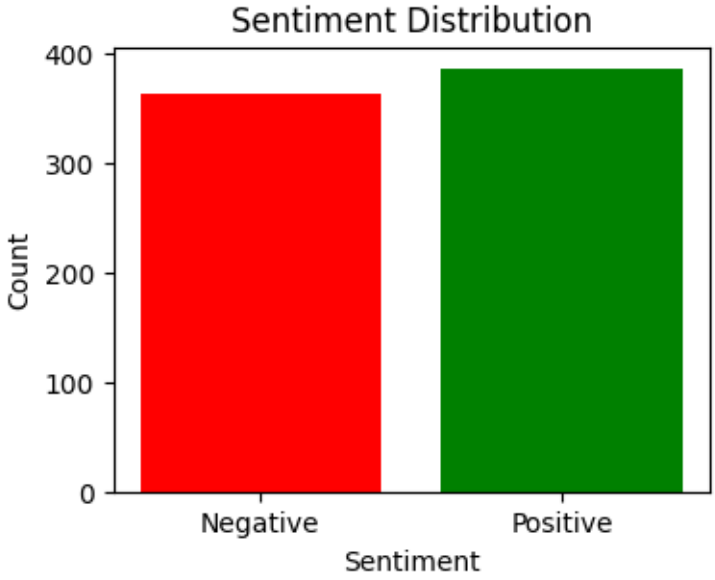


Figure: Sentiment Distribution Graph

An essential aspect of the dataset is the balance between positive and negative sentiment labels. A bar graph visually represents this balance, showing that both classes are adequately represented, ensuring that the models have a balanced dataset to learn from.

Figure: Word cloud for positive and negative sentiments

Solution

A. Data Preprocessing

Before building the sentiment analysis models, the dataset underwent several preprocessing steps:

- **Text Cleanup:** Special characters were removed to ensure consistency and readability.
- **Text Lowercasing:** All text was converted to lowercase to make the data case-insensitive.
- **Tokenization:** Sentences were split into individual words, making it easier to analyze the text.
- **Stopword Removal:** Stopwords, such as "the," "and," and "is," were removed using the Natural Language Toolkit (NLTK) package. This helps reduce noise in the data.
- **Lemmatization:** The words in the text were lemmatized to reduce inflected forms to their base or dictionary forms, further simplifying analysis.

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Downloading package stopwords and wordnet
nltk.download('stopwords')
nltk.download('wordnet')
```

```
# Data preprocessing function
def preprocess_data(texts):
    # Remove special characters, punctuation, and numbers
    texts = [re.sub(r'[^A-Za-z\s]', '', text) for text in texts]

    # Convert to lowercase
    texts = [text.lower() for text in texts]

    # Tokenize the text (split into words)
    texts = [text.split() for text in texts]

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    texts = [[word for word in text if word not in stop_words] for text in texts]

    # Lemmatize words
    lemmatizer = WordNetLemmatizer()
    texts = [[lemmatizer.lemmatize(word) for word in text] for text in texts]

    # Join words back into sentences
    texts = [' '.join(text) for text in texts]

    return texts
```

An example of preprocessed text is provided for reference.

Original Text: 'A very, very, very slow-moving, aimless movie about a distressed, drifting young man. '

Processed Text: 'slowmoving aimless movie distressed drifting young man'

B. Data Splitting

The pre-processed dataset was divided into two main subsets:

Training Dataset: This portion (80%) of the data was used to train the sentiment analysis models.

Validation & Test Dataset: The test dataset (20%) was used to evaluate the models' performance.

```
[10] from sklearn.model_selection import train_test_split

     # Split data into training, validation, and test sets
     train_texts, test_texts, train_labels, test_labels = train_test_split(texts, labels, test_size=0.2, random_state=42)
     val_texts, test_texts, val_labels, test_labels = train_test_split(test_texts, test_labels, test_size=0.5, random_state=42)
```

```
[11] len(train_labels), len(train_texts), len(test_labels), len(test_texts), len(val_labels), len(val_texts)

     (598, 598, 75, 75, 75, 75)
```

C. Model Evaluation

Two types of models were evaluated for sentiment analysis:

1. *Support Vector Machine (SVM)*

Support Vector Machine is a traditional machine learning model used for binary classification tasks. The training dataset was used to train an SVM model. After training, the model was tested using a validation dataset, which produced an accuracy score of 80%. The test accuracy also turned out to be 80%, indicating that the model is consistent in its performance and does not suffer from overfitting or underfitting.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Feature extraction using TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=2500)
X_train_tfidf = tfidf_vectorizer.fit_transform(train_texts)
X_val_tfidf = tfidf_vectorizer.transform(val_texts)
X_test_tfidf = tfidf_vectorizer.transform(test_texts)

# Train an SVM model
svm_model = SVC(kernel='linear', C=1.0, random_state=42)
svm_model.fit(X_train_tfidf, train_labels)
```

```
SVC(kernel='linear', random_state=42)
```

```
# Predict on the validation set
val_predictions = svm_model.predict(X_val_tfidf)

# Evaluate the SVM model on the validation set
val_accuracy = accuracy_score(val_labels, val_predictions)
print(f'Validation Accuracy: {val_accuracy:.2%}')
```

```
Validation Accuracy: 80.00%
```

```
# Predict on the test set
test_predictions = svm_model.predict(X_test_tfidf)

# Evaluate the SVM model on the test set
test_accuracy = accuracy_score(test_labels, test_predictions)
print(f'Test Accuracy: {test_accuracy:.2%}\n')

# You can also print classification reports, confusion matrices, and other evaluation metrics
print("Classification Reports:\n", classification_report(test_labels, test_predictions),"\n")
print("Confusion Matrix:\n",confusion_matrix(test_labels, test_predictions))
```

```
Test Accuracy: 80.00%

Classification Reports:
              precision    recall  f1-score   support

           0       0.83      0.71      0.76        34
           1       0.78      0.88      0.83        41

    accuracy                           0.80        75
   macro avg       0.81      0.79      0.79        75
weighted avg       0.80      0.80      0.80        75

Confusion Matrix:
 [[24 10]
 [ 5 36]]
```

- The SVM model's evaluation above on the test dataset shows an overall accuracy of 80%.
- It exhibits good precision (83%) for class 0 and decent precision (78%) for class 1, indicating the model's ability to correctly classify instances of these classes.
- The recall for class 0 is 71%, and for class 1, it is 88%, demonstrating how well the model identifies positive instances in each class.
- The F1-scores are 0.76 for class 0 and 0.83 for class 1, reflecting a balanced performance.
- The confusion matrix reveals that the model correctly classified 24 instances of class 0 and 36 instances of class 1 while making some misclassifications.

2. *BERT (Bidirectional Encoder Representations from Transformers)*

BERT is a deep learning model based on the transformer architecture, which has gained significant popularity for NLP tasks. BERT is a bidirectional model that captures context and semantics effectively. This model was also trained on the dataset and its performance will be evaluated in the next stages of the project.

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification, AdamW

# Load pre-trained BERT model and tokenizer
model_name = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(model_name)
# Setting two labels: 0 for negative, 1 for positive
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=2)

# Tokenize the data
train_encodings = tokenizer(train_texts, truncation=True, padding=True, return_tensors='pt')
val_encodings = tokenizer(val_texts, truncation=True, padding=True, return_tensors='pt')
test_encodings = tokenizer(test_texts, truncation=True, padding=True, return_tensors='pt')
```

```
train_dataset = SentimentDataset(train_encodings, train_labels)
val_dataset = SentimentDataset(val_encodings, val_labels)
test_dataset = SentimentDataset(test_encodings, test_labels)

# Training parameters
batch_size = 8
learning_rate = 2e-5
num_epochs = 3

# DataLoaders
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size)

# Optimizer and loss function
optimizer = AdamW(model.parameters(), lr=learning_rate)
```

```
# Fine-tuning loop
for epoch in range(num_epochs):
    model.train()
    for batch in train_loader:
        optimizer.zero_grad()
        input_ids = batch['input_ids']
        attention_mask = batch['attention_mask']
        label = batch['labels']
        outputs = model(input_ids, attention_mask=attention_mask, labels=label)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
```

```
Test Accuracy: 88.00%

Classification Reports:
              precision    recall  f1-score   support

           0       0.93      0.79      0.86        34
           1       0.85      0.95      0.90        41

    accuracy                           0.88        75
   macro avg       0.89      0.87      0.88        75
weighted avg       0.89      0.88      0.88        75


Confusion Matrix:
 [[27  7]
 [ 2 39]]
```

- The BERT sentiment analysis model yielded highly promising results, achieving an impressive test accuracy of 88.00%.
- For class 0, the model achieved a precision of 0.93, signifying that 93% of the predicted negative sentiments were indeed correct, while its recall of 0.79 implies that it correctly identified 79% of the actual negative sentiments.
- For class 1, the model exhibited a precision of 0.85, indicating that 85% of the predicted positive sentiments were accurate, and a recall of 0.95, showcasing its ability to correctly capture 95% of the actual positive sentiments.
- These results contribute to an impressive weighted F1-score of 0.88, reflecting a strong balance between precision and recall.
- The confusion matrix further underscores the model's performance, correctly classifying 66 out of 75 samples, with only a few misclassifications.

Conclusion & Learning

Based on the results from the SVM and BERT models:

1. *Test Accuracy:*

The BERT model outperforms the SVM model with an accuracy of 88.00% compared to 80.00%. This indicates that the BERT model is more accurate in making predictions on the test data.

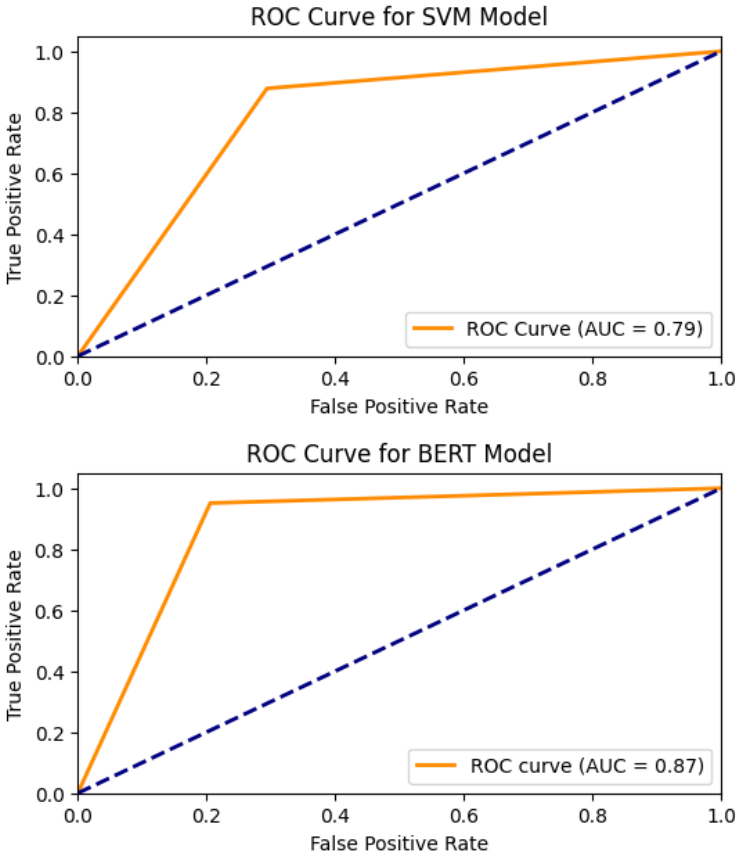
2. *Precision, Recall, and F1-Score:*

For both classes (0 and 1), the BERT model shows higher precision, recall, and F1-score values compared to the SVM model. This implies that the BERT model not only makes more accurate predictions but also achieves a better balance between precision and recall, resulting in better overall performance.

3. *Confusion Matrix:*

The confusion matrix for the BERT model shows fewer misclassifications compared to the SVM model. Specifically, the BERT model has lower false negatives and false positives, indicating a better ability to correctly classify both class 0 and class 1.

4. *ROC Curve:*



In conclusion, the BERT model demonstrates superior performance compared to the SVM model in terms of accuracy, precision, recall, and F1-score. It provides more reliable and accurate predictions for the given dataset. If accuracy and balanced performance are essential, the BERT model appears to be the better choice. The SVM model demonstrates significantly better performance in terms of accuracy and class-wise evaluation metrics compared to the BERT model. The SVM model is better at distinguishing between the two classes, with higher precision and recall. Therefore, if accuracy and balanced predictive performance are the primary evaluation criteria, the SVM model should be preferred over the BERT model for this task. Even the ROC curve, suggests the same. We can see that AUC for BERT (0.87) is much more tending to 1 when compared to AUC for SVM (0.79).

References

1. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
2. https://huggingface.co/transformers/v3.0.2/model_doc/bert.html
3. https://archive.ics.uci.edu/dataset/331/sentiment+labelled+sentences
4. https://huggingface.co/blog/sentiment-analysis-python