Sentiment Analysis on Review Websites Using Supervised and Zero-Shot Methods

Contributors:

- 201805017 NESLİHAN ÖZDİL
- 201805032 ARDIL SİLAN AYDIN
- 201805060 ELİF YILMAZ

1. Introduction

1.1 Project Topic

In this project, sentiment analysis was performed on reviews of the movie "Parasite" (2019) from the Letterboxd website. Sentiment analysis was conducted using two different methods: supervised learning and zero-shot learning.

1.2 Objective

The objective of this project is to perform sentiment analysis on reviews of the movie "Parasite" and compare the performance of supervised learning and zero-shot methods.

2. Literature Review

2.1 Sentiment Analysis Overview

Sentiment analysis is one of the techniques in natural language processing (NLP) and is used to detect the emotions (positive, negative, neutral) contained in texts. This technique is commonly used in analyzing user feedback and social media comments.

2.2 Supervised vs Zero-Shot Learning

• Supervised Learning: The model is trained using labeled datasets. The model is optimized with labeled data to solve a specific task.
• Zero-Shot Learning: This type of learning allows the model to solve tasks it has not encountered before without using labeled data. Zero-shot learning provides flexibility and broad task compatibility.

3. Data Collection and Preprocessing

3.1 Data Collection:Selenium

The data collection process was carried out using Selenium to scrape the Letterboxd website. The reviews were collected by navigating through the pages and waiting for dynamic content to load.

Code used for data collection:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time

# Firefox WebDriver için ayarlar
from selenium.webdriver.firefox.options import Options
options = Options()
options.add_argument("--start-maximized")
options.set_preference("general.useragent.override",
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/91.0.4472.124 Safari/537.36")

# WebDriver'ı başlat
driver = webdriver.Firefox(options=options)

# Timeout süresini artır
driver.set_page_load_timeout(300)

# İlk sayfanın linkini girin
base_url = "https://letterboxd.com/film/parasite-2019/reviews/by/activity/"

# Tüm yorumları kaydetmek için bir liste oluştur
all_reviews = []

try:
    # İlk sayfayı aç
    driver.get(base_url)
    wait = WebDriverWait(driver, 30)

    while True:  # Next butonuna basılabildiği sürece devam et
        print(f"Sayfa açıldı, yorumlar çekiliyor...")

        # Dinamik içeriklerin yüklenmesini bekle
        review_elements = wait.until(
            EC.presence_of_all_elements_located((By.CSS_SELECTOR,
"div.body-text"))
        )

        # "I can handle the truth" butonlarına tıklama
        try:
            spoiler_buttons = driver.find_elements(By.LINK_TEXT, "I can
handle the truth.")
            for button in spoiler_buttons:
                try:

driver.execute_script("arguments[0].scrollIntoView(true);", button)
                    button.click()  # Spoiler içeriğini açmak için tıkla
                    time.sleep(1)  # İçeriğin yüklenmesi için bekleme
                except Exception as e:
                    print(f"Bir hata oluştu: {e}")
        except Exception as e:
            print(f"Spoiler butonları bulunamadı: {e}")

        # "more" düğmelerine tıklama
        try:
            more_buttons = driver.find_elements(By.LINK_TEXT, "more")
            for button in more_buttons:
```

```
                try:
driver.execute_script("arguments[0].scrollIntoView(true);", button)
                    button.click()  # Yorumun tamamını görmek için tıkla
                    time.sleep(1)  # İçeriğin yüklenmesi için bekleme
                except Exception as e:
                    print(f"'more' düğmesine tıklanırken bir hata oluştu: {e}")
        except Exception as e:
            print(f"'more' düğmeleri bulunamadı: {e}")

        # Yorumları al
        reviews = [review.text for review in review_elements]
        all_reviews.extend(reviews)  # Yorumları ana listeye ekle

        # "Next" butonuna tıklama
        try:
            next_button = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, "a.next")))
            driver.execute_script("arguments[0].scrollIntoView(true);", next_button)  # Next butonunu görünür yap
            next_button.click()  # Sonraki sayfaya git
            time.sleep(5)  # Yeni sayfanın yüklenmesini bekle
        except Exception as e:
            print("Sonraki sayfa bulunamadı, işlem tamamlandı.")
            break  # Eğer "Next" butonu yoksa döngüden çık

    # Yorumları bir dosyaya kaydet
    print("Tüm yorumlar çekildi. Dosyaya kaydediliyor...")
    with open("all_reviews_3.txt", "w", encoding="utf-8") as file:
        for i, review in enumerate(all_reviews, 1):
            file.write(f"{i}. Yorum: {review}\n")

except Exception as e:
    print(f"Hata oluştu: {e}")

finally:
    # Tarayıcıyı kapat
    driver.quit()
```

The code above shows the basic structure used to scrape reviews from Letterboxd.

- **Next Button:** In the code, the "Next" button on the page is clicked to navigate to the next page. This is done using the command `next_button = wait.until(...)`.
- **"I Can Handle The Truth" Buttons:** If there are spoiler texts in the reviews, the "I can handle the truth." buttons are clicked to reveal the spoiler content.
- **"More" Buttons:** By clicking the "More" button, the full review is made visible.

After collecting the reviews, all the data is saved to the file `all_reviews_3.txt`.
A total of 3327 reviews were collected, and 2533 of them were saved in English. Only English reviews were analyzed in this project.

Code for separating English reviews:

```
from langdetect import detect
import os
```

```python
# Dosya yolları
input_file = "all_reviews_3.txt"  # Orijinal dosya
output_file = "filtered_reviews.txt"  # Filtrelenmiş dosya

def is_english(text):
    try:
        return detect(text) == 'en'  # Yorumun dilini kontrol eder
    except:
        return False  # Eğer hata alırsa İngilizce değil kabul eder

def filter_english_reviews(input_file, output_file):
    with open(input_file, "r", encoding="utf-8") as infile,
open(output_file, "w", encoding="utf-8") as outfile:
        for line in infile:
            if "Yorum:" in line:
                # Yorum başlangıcını tespit eder
                parts = line.split(": ", 1)
                if len(parts) == 2 and is_english(parts[1]):
                    outfile.write(line)

# İngilizce yorumları filtrele ve kaydet
filter_english_reviews(input_file, output_file)

print(f"İngilizce yorumlar {output_file} dosyasına kaydedildi.")
```

The filtered reviews are saved to the file "filtered_reviews.txt".

3.2 Data Preprocessing

3.2.1 Converting to CSV and Cleaning

First, the "n. Yorum" phrase at the beginning of each review in the `filtered_reviews.txt` file was removed, and the reviews were converted into a CSV file with columns for `id` and `comment`. Then, data cleaning was performed on the reviews in this CSV file.

```python
# "nth comment:" ifadesini kaldırma ve ID oluşturma
comments_cleaned = [comment.split(":", 1)[-1].strip() for comment in
comments]  # "nth comment:" kısmını kaldırıyoruz

# ID ekleyerek yeni veri yapısını oluşturma
data = {'ID': range(1, len(comments_cleaned) + 1), 'Comment':
comments_cleaned}
```

3.2.2 Cleaning and Tokenization with NLTK

The NLTK library was used to clean the words in the reviews and process the texts. NLTK is a Python library that provides a wide range of tools for text processing. The following operations were performed in this step:

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk

nltk.download('punkt_tab')
nltk.download('punkt')
nltk.download('stopwords')
# Stopwords ve noktalama işaretleri temizleme fonksiyonu
def clean_text(text):
    # Küçük harfe çevirme
    text = text.lower()
    # Emojileri ve özel karakterleri temizleme
    text = re.sub(r'[^\w\s]', '', text)  # Noktalama işaretlerini kaldırma
    text = re.sub(r'[\U00010000-\U0010ffff]', '', text)  # Emoji temizleme
    # Tokenizasyon (kelimelere ayırma)
    words = word_tokenize(text)
    # Stop-words temizleme
    stop_words = set(stopwords.words('english'))
    filtered_words = [word for word in words if word not in stop_words]
    # Temizlenmiş metni geri birleştirme
    return ' '.join(filtered_words)
# Her bir yorumu temizleme
df['Cleaned_Comment'] = df['Comment'].apply(clean_text)
```

Stop-words Removal: Meaningless words in the reviews (such as "and", "this", "that") were removed.
Punctuation and Emoji Removal: Unnecessary punctuation marks and emojis in the reviews were cleaned.
Tokenization: The reviews were split into words, making the meaningful words in each review ready for processing.

3.2.3 Lemmatization

Lemmatization was applied to the cleaned texts. In this step, words were reduced to their base forms to create a more meaningful text. This process, carried out using the WordNetLemmatizer, ensures that words are reduced to their roots without changing their meaning, making the text more consistent and suitable for analysis.

```
from nltk.stem import WordNetLemmatizer
```

```
import re

nltk.download('wordnet')
# Lemmatizer'ı başlatma
lemmatizer = WordNetLemmatizer()
  # Kelimelere ayırma (tokenization)
words = word_tokenize(text)
  # Stopwords temizleme ve lemmatization
 filtered_words = [lemmatizer.lemmatize(word) for word in words if word not
in stop_words]
# Her bir yorumu temizleme ve lemmatization uygulama
df['Lemmatized_Comment'] = df['Cleaned_Comment'].apply(clean_and_lemmatize)
```

The final version of our dataset:

	ID	Comment	Cleaned_Comment	Lemmatized_Comment	Label
0	1	Update: Now in video form	update video form	update video form	neutral
1	2	call me by your name's cum peach walked so par...	call names cum peach walked parasites killer p...	call name cum peach walked parasite killer pea...	neutral
2	3	Another Bong hit.	another bong hit	another bong hit	neutral
3	4	Our expectations were high but HOLY FUCK	expectations high holy fuck	expectation high holy fuck	negative
4	5	a question to people who rate this 4.5: what m...	question people rate 45 want literally want	question people rate 45 want literally want	neutral
5	6	One detail I noticed this time around is that ...	one detail noticed time around min mr park rea...	one detail noticed time around min mr park rea...	positive
6	7	The tent won't leak. It's from America.	tent wont leak america	tent wont leak america	neutral
7	8	The bloody napkin scene....top 3 scenes of all...	bloody napkin scenetop 3 scenes time hands	bloody napkin scenetop 3 scene time hand	negative
8	9	morse code me by your name and i'll morse code...	morse code name ill morse code mine	morse code name ill morse code mine	negative
9	10	maybe the real parasite... was the friends we ...	maybe real parasite friends made along way	maybe real parasite friend made along way	positive

3.2.4 Vectorization and Label Encoding

In the data preprocessing process, vectorization was performed to convert the review texts into numerical data. For this purpose, the TF-IDF (Term Frequency-Inverse Document Frequency) method was used. This method calculates the importance of terms in texts and represents each term with a numerical vector. Additionally, Label Encoding was applied to convert the labels associated with the reviews into numerical values.

```
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer

# Veri ön işleme: Vektörize etme ve label encoding
def preprocess_data(data):
    """Yorumları vektörize eder ve etiketleri sayısal hale getirir"""
    vectorizer = TfidfVectorizer(max_features=5000)
    X = vectorizer.fit_transform(data['Lemmatized_Comment'])
    encoder = LabelEncoder()
    y = encoder.fit_transform(data['Label'])
    return X, y, vectorizer, encoder
```

TF-IDF Vectorization: Using TfidfVectorizer, the reviews in the Lemmatized_Comment column were converted into numerical vectors. This process determines the importance of each word and represents the words with numerical representations. The **`max_features=5000`** parameter was used to select up to 5000 features (words), controlling the complexity of the model.
Label Encoding: Using the LabelEncoder class, the categorical labels associated with the reviews (e.g., positive, negative, neutral) were converted into numerical values. This process transforms the text labels into a numerical format that the model can understand.

4. Supervised Sentiment Analysis

4.1 Model Training

In this project, supervised learning methods were used to perform sentiment analysis. The **Support Vector Machine** (SVM) and **Logistic Regression** algorithms were chosen. The performance of the models was evaluated using the **Stratified K-Fold cross-validation** method, and the validation process for each model was carried out with 10-fold cross-validation. K-fold splits the data into K parts, and each time, a different part is used to test the model. The results are averaged to perform the validation.

4.1.1 Data Balancing

SMOTE (Synthetic Minority Over-sampling Technique) was used to address class imbalances in the training data. SMOTE works by increasing the data for minority classes, thereby resolving class imbalances and ensuring a more balanced learning process for the model across all classes. This method improves the overall accuracy of the model by adding synthetic examples to the minority class.

4.1.2 Model Training and Cross-Validation

Both models were trained and tested **using Stratified K-Fold cross-validation**. This method ensures reliable results by maintaining class distributions in both the training and test sets.

4.1.3 Data Splitting and Labeling

The training and test data were split using the `train_test_split` function with an **80% training and 20% testing ratio.** The **`stratify=y`** parameter ensures that the class distributions are consistent across both datasets. Labeling was performed using the LabelEncoder.

4.1.4 Model Parameters

• SVM: The model was trained with the parameters `kernel='linear'`, `C=1`, `gamma='scale'`.
• Logistic Regression: The parameters `max_iter=1000`, `C=1`, and `solver='liblinear'` were chosen.

4.1.5 Performance Evaluation

Both models were tested using the Stratified K-Fold method and evaluated based on accuracy and other metrics. The results show that both models have improved generalization capacity and that the class imbalances were addressed.

4.2 Performance Metrics

Performance metrics were calculated using criteria such as accuracy and F1-score. The obtained metrics are as follows:

• **Accuracy:**
o SVM Model: Average accuracy of 92.85%
o Logistic Regression Model: Average accuracy of 93.72%

• **F1-Score:**
o SVM Model: Average F1-score of 0.9268
o Logistic Regression Model: Average F1-score of 0.9360

These metrics show that both models have high accuracy and F1-scores, but the Logistic Regression model performed slightly better than the SVM model. This indicates that while both models are successful in classification tasks, the Logistic Regression model has better generalization in some cases.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import torch
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix, f1_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

# 1. Google Colab'da GPU kullanımını kontrol etme
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Veri yükleme
def load_data(file_path):
    """Veriyi yükler ve döndürür"""
```

```python
    data = pd.read_csv(file_path)
    return data

# Veri ön işleme: Vektörize etme ve label encoding
def preprocess_data(data):
    """Yorumları vektörize eder ve etiketleri sayısal hale getirir"""
    vectorizer = TfidfVectorizer(max_features=5000)
    X = vectorizer.fit_transform(data['Lemmatized_Comment'])

    encoder = LabelEncoder()
    y = encoder.fit_transform(data['Label'])

    return X, y, vectorizer, encoder

# Eğitim ve test verisine ayırma
def split_data(X, y):
    """Veriyi eğitim ve test olarak böler"""
    return train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# SMOTE ile veri dengeleme
def apply_smote(X_train, y_train):
    """SMOTE ile veri setini dengele"""
    smote = SMOTE(random_state=42)
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
    return X_train_resampled, y_train_resampled

# K-Fold Cross Validation fonksiyonu
def train_evaluate_models(X_train_resampled, y_train_resampled):
    """Modelleri eğitim ve test verisi ile eğitir ve değerlendirir"""
    svm_model = make_pipeline(StandardScaler(with_mean=False), SVC(kernel='linear', C=1, gamma='scale'))
    log_reg_model = make_pipeline(StandardScaler(with_mean=False), LogisticRegression(max_iter=1000, C=1, solver='liblinear'))

    svm_accuracies, log_reg_accuracies = [], []
    svm_f1_scores, log_reg_f1_scores = [], []
    svm_conf_matrix, log_reg_conf_matrix = [], []
    svm_predictions_all, log_reg_predictions_all, y_test_all = [], [], []

    svm_class_reports, log_reg_class_reports = [], []

    kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

```
    # K-Fold çapraz doğrulama ile her iki modelin eğitim ve test edilmesi
    for train_index, test_index in kf.split(X_train_resampled,
y_train_resampled):
        X_train_fold, X_test_fold = X_train_resampled[train_index],
X_train_resampled[test_index]
        y_train_fold, y_test_fold = y_train_resampled[train_index],
y_train_resampled[test_index]

        # SVM modelini eğitelim
        svm_model.fit(X_train_fold, y_train_fold)
        svm_predictions = svm_model.predict(X_test_fold)

        # Lojistik Regresyon modelini eğitelim
        log_reg_model.fit(X_train_fold, y_train_fold)
        log_reg_predictions = log_reg_model.predict(X_test_fold)

        # Performans metriklerini hesaplayalım
        svm_accuracies.append(accuracy_score(y_test_fold, svm_predictions))
        log_reg_accuracies.append(accuracy_score(y_test_fold,
log_reg_predictions))

        svm_f1_scores.append(f1_score(y_test_fold, svm_predictions,
average='macro'))
        log_reg_f1_scores.append(f1_score(y_test_fold, log_reg_predictions,
average='macro'))

        # Confusion matrix
        svm_conf_matrix.append(confusion_matrix(y_test_fold,
svm_predictions))
        log_reg_conf_matrix.append(confusion_matrix(y_test_fold,
log_reg_predictions))

        # Classification report
        svm_class_reports.append(classification_report(y_test_fold,
svm_predictions, output_dict=True))
        log_reg_class_reports.append(classification_report(y_test_fold,
log_reg_predictions, output_dict=True))

        # Tahminleri ve gerçek etiketleri saklayalım
        svm_predictions_all.extend(svm_predictions)
        log_reg_predictions_all.extend(log_reg_predictions)
        y_test_all.extend(y_test_fold)
```

```
    return svm_accuracies, log_reg_accuracies, svm_f1_scores,
log_reg_f1_scores, svm_conf_matrix, log_reg_conf_matrix,
svm_predictions_all, log_reg_predictions_all, y_test_all,
svm_class_reports, log_reg_class_reports

# Performans metriklerini yazdırma
def print_performance_metrics(svm_accuracies, log_reg_accuracies,
svm_f1_scores, log_reg_f1_scores, svm_predictions_all,
log_reg_predictions_all, y_test_all, svm_class_reports,
log_reg_class_reports):
    """Ortalama metrikleri ve classification report'ları yazdırır"""
    svm_avg_accuracy = sum(svm_accuracies)/len(svm_accuracies)
    log_reg_avg_accuracy = sum(log_reg_accuracies)/len(log_reg_accuracies)
    svm_avg_f1 = sum(svm_f1_scores)/len(svm_f1_scores)
    log_reg_avg_f1 = sum(log_reg_f1_scores)/len(log_reg_f1_scores)

    print(f"SVM Average Accuracy: {svm_avg_accuracy:.4f}")
    print(f"Logistic Regression Average Accuracy:
{log_reg_avg_accuracy:.4f}")
    print(f"SVM Average F1-Score: {svm_avg_f1:.4f}")
    print(f"Logistic Regression Average F1-Score: {log_reg_avg_f1:.4f}")

    print("\nSVM Classification Report:\n",
classification_report(y_test_all, svm_predictions_all))
    print("\nLogistic Regression Classification Report:\n",
classification_report(y_test_all, log_reg_predictions_all))

# Confusion matrix görselleştirme
def plot_confusion_matrix(cm, model_name, encoder):
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='g', cmap='Blues',
xticklabels=encoder.classes_, yticklabels=encoder.classes_)
    plt.title(f'Confusion Matrix - {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

# Ana fonksiyon
def main(file_path):
    data = load_data(file_path)
    X, y, vectorizer, encoder = preprocess_data(data)
    X_train, X_test, y_train, y_test = split_data(X, y)
    X_train_resampled, y_train_resampled = apply_smote(X_train, y_train)
```

```
    svm_accuracies, log_reg_accuracies, svm_f1_scores, log_reg_f1_scores,
svm_conf_matrix, log_reg_conf_matrix, svm_predictions_all,
log_reg_predictions_all, y_test_all, svm_class_reports,
log_reg_class_reports = train_evaluate_models(X_train_resampled,
y_train_resampled)

    print_performance_metrics(svm_accuracies, log_reg_accuracies,
svm_f1_scores, log_reg_f1_scores, svm_predictions_all,
log_reg_predictions_all, y_test_all, svm_class_reports,
log_reg_class_reports)

    svm_avg_conf_matrix = sum(svm_conf_matrix) / len(svm_conf_matrix)
    log_reg_avg_conf_matrix = sum(log_reg_conf_matrix) /
len(log_reg_conf_matrix)

    plot_confusion_matrix(svm_avg_conf_matrix, "SVM", encoder)
    plot_confusion_matrix(log_reg_avg_conf_matrix, "Logistic Regression",
encoder)

# Dosya yolunu burada belirtin
file_path = 'lemmatized_comments_with_labels.csv'
main(file_path)
```

5. Zero-Shot Sentiment Analysis

5.1 Zero-Shot Learning

Zero-shot learning enables sentiment classification without the need for labeled data by using large language models. In this approach, a pre-trained model from the Hugging Face library (based on BERT) can be used to perform classifications for different classes. In this example, sentiment analysis was conducted using the **`nlptown/bert-base-multilingual-uncased-sentiment`** model with the labels "positive", "neutral", and "negative".
Code:

```
import torch
from transformers import pipeline
import pandas as pd
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

# Load the dataset
file_path = 'lemmatized_comments_with_labels.csv'
df = pd.read_csv(file_path)

# Use the Lemmatized_Comment for prediction
comments = df['Lemmatized_Comment'].tolist()
```

```
# Check if CUDA (GPU) is available and set the device accordingly
device = 0 if torch.cuda.is_available() else -1  # -1 means using CPU, 0
means using GPU

# Load a zero-shot classifier model (using Hugging Face's pipeline)
classifier = pipeline("zero-shot-classification", model="nlptown/bert-base-
multilingual-uncased-sentiment", device=device)  # Use GPU if available

# Define possible labels
labels = ['positive', 'neutral', 'negative']

# Run zero-shot classification on the comments
results = []
for comment in comments:
    result = classifier(comment, candidate_labels=labels)
    results.append(result['labels'][0])  # Get the label with the highest
score

# Add the predictions to the dataframe
df['Predicted_Label'] = results

# Calculate performance metrics
accuracy = accuracy_score(df['Label'], df['Predicted_Label'])
precision, recall, f1, _ = precision_recall_fscore_support(df['Label'],
df['Predicted_Label'], labels=['positive', 'neutral', 'negative'],
average='weighted')

# Save the dataframe with the predictions to a new CSV file
output_file = 'lemmatized_comments_with_predictions.csv'
df.to_csv(output_file, index=False)

# Print performance metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")

# Print the classification report
print("\nZero-Shot Classification Report:")
print(classification_report(df['Label'], df['Predicted_Label'],
labels=['positive', 'neutral', 'negative']))
# Provide the output file path
output_file
```

5.2 Performance Evaluation

The accuracy and F1-score of the zero-shot model were calculated as follows:

• **Accuracy:** 62.34%
• **Precision:** 0.5158
• **Recall:** 0.6234
• **F1-Score:** 0.4910

Overall, the model demonstrated successful performance in sentiment classification with average accuracy and F1-score.
Zero-shot learning enables sentiment analysis without requiring a labeled dataset, making such models useful in projects with limited data. Although the model's performance is at a good level, further development is needed to achieve higher accuracy and F1-score.

6. Result Comparison and Analysis

6.1 Classification Report and Confusion Matrix

The sentiment classification performance of different models was compared using metrics such as accuracy and F1-score. Below are the results for each model:

Model	Accuracy (%)	F1-Score
SVM	92.85	0.9268
Logistic Regression	93.72	0.9360
Zero-Shot	62.34	0.4910

SVM and logistic regression models have significantly higher accuracy and F1-score values compared to the zero-shot model, indicating that supervised learning methods perform stronger than the zero-shot model.

6.1.1 Classification Report

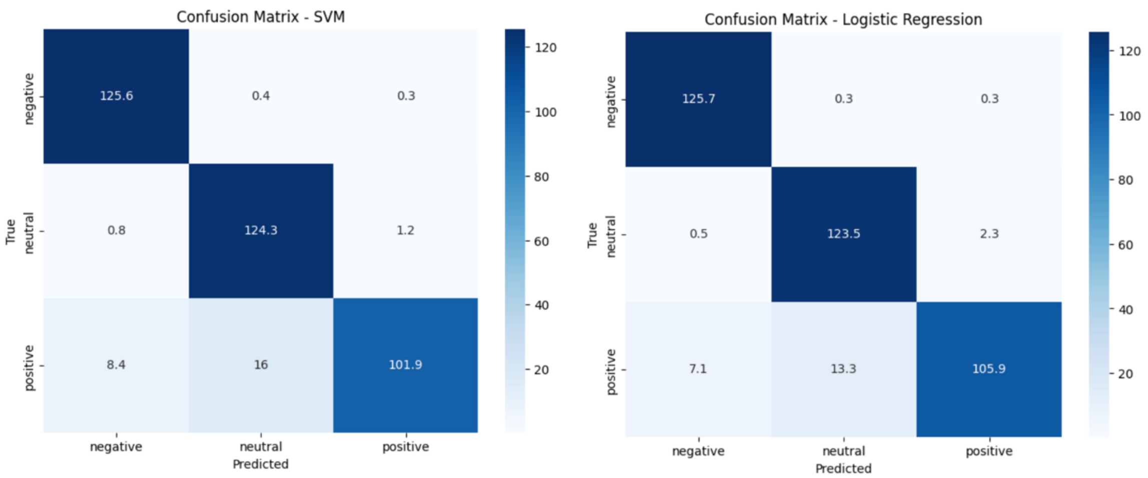
The classification report provides a detailed performance overview of the model for each class (negative, neutral, positive). This report includes metrics such as precision, recall, and F1-score for each class.

```
Accuracy: 0.6234
Precision: 0.5158
Recall: 0.6234
F1-score: 0.4910

Zero-Shot Classification Report:
              precision    recall  f1-score   support

    positive       0.63      0.99      0.77      1579
     neutral       0.19      0.01      0.01       584
    negative       0.56      0.04      0.07       370

    accuracy                           0.62      2533
   macro avg       0.46      0.34      0.28      2533
weighted avg       0.52      0.62      0.49      2533
```

```
Using device: cuda
SVM Average Accuracy: 0.9285
Logistic Regression Average Accuracy: 0.9372
SVM Average F1-Score: 0.9268
Logistic Regression Average F1-Score: 0.9360

SVM Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.99      0.96      1263
           1       0.88      0.98      0.93      1263
           2       0.99      0.81      0.89      1263

    accuracy                           0.93      3789
   macro avg       0.93      0.93      0.93      3789
weighted avg       0.93      0.93      0.93      3789


Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      1263
           1       0.90      0.98      0.94      1263
           2       0.98      0.84      0.90      1263

    accuracy                           0.94      3789
   macro avg       0.94      0.94      0.94      3789
weighted avg       0.94      0.94      0.94      3789
```

6.1.2 Confusion Matrix

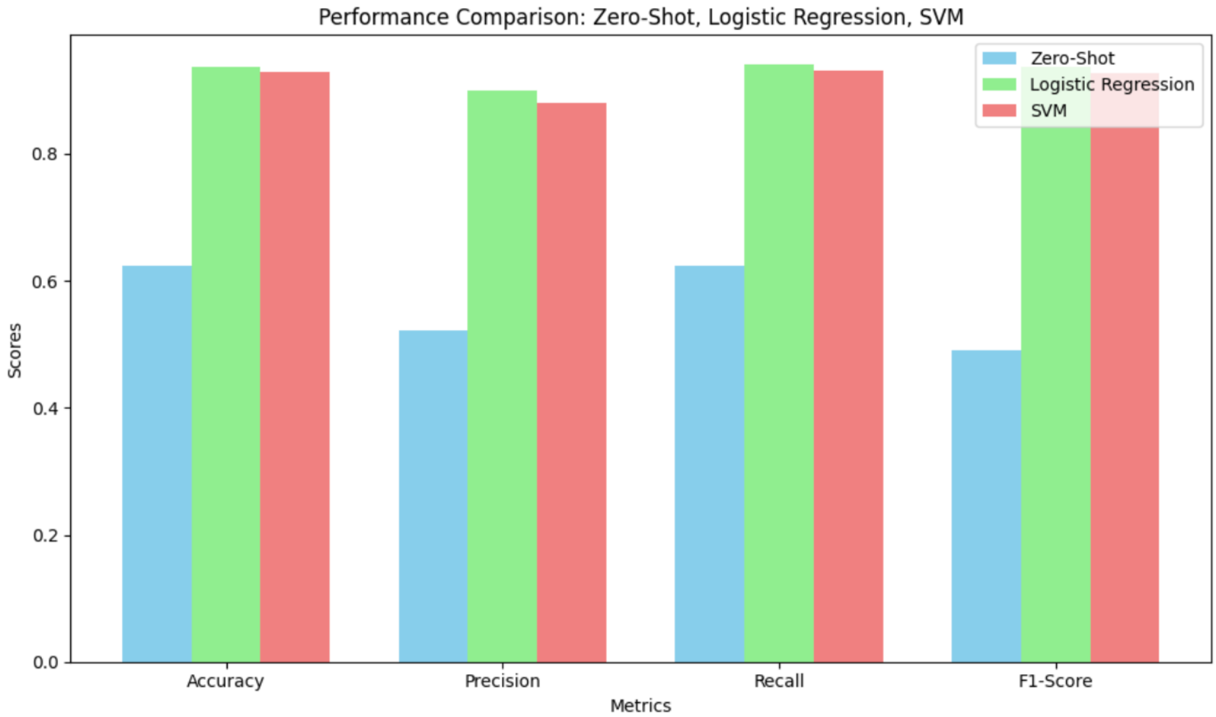
Confusion matrices show how accurately the models predict each class. These matrices are created for each fold during cross-validation and averaged by dividing by the number of folds.



6.2 Comparison of Supervised vs Zero-Shot Models

The comparison between supervised learning (SVM and logistic regression) and zero-shot models has highlighted the advantages and limitations of each model type. Supervised models demonstrate better performance with higher accuracy and F1-scores, achieving strong results based on training with labeled datasets.

On the other hand, while the zero-shot learning model has lower accuracy and F1-score, its advantage lies in its ability to provide quick and flexible solutions without the need for labeled data. The zero-shot approach offers a significant advantage, especially in situations where working with limited data and the data labeling process is challenging.




```
SVM Average Accuracy: 0.9285
Logistic Regression Average Accuracy: 0.9372
Zero-Shot Accuracy: 0.6234
----------------------------------------
SVM Average F1-Score: 0.9268
Logistic Regression Average F1-Score: 0.9360
Zero-Shot F1-Score: 0.4910
```

6.3 Findings

Although the accuracy and F1-score of the zero-shot model lag behind those of the supervised models, the zero-shot learning method provides a significant advantage with its ability to perform sentiment classification without the need for a labeled dataset. This is ideal for rapid prototyping and flexible applications. However, for applications that require higher accuracy and F1-scores, supervised learning methods should be preferred.

7. Discussion and Conclusion

7.1 Interpretation of Results

The results demonstrate the different advantages of supervised learning models and the zero-shot model. Supervised models (SVM and Logistic Regression) have performed successfully with higher accuracy and F1-scores, showcasing strong generalization capabilities through training with labeled data. The zero-shot model, on the other hand, offers flexibility by being able to perform sentiment analysis without labeled data, but it has lower accuracy and F1-scores.

7.2 Conclusion

This study compared supervised learning and zero-shot learning methods for sentiment analysis tasks. While supervised learning provides high accuracy and reliability, despite the need for labeled data, zero-shot learning offers more flexible and faster solutions. In cases where labeled data is unavailable, the zero-shot model provides a significant advantage. In the future, performance could be improved by combining both approaches in hybrid models.

References

Chawla, N. V., et al. (2002). *SMOTE: Synthetic Minority Over-sampling Technique*. Journal of Artificial Intelligence Research.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv:1810.04805.

Hugging Face. (n.d.). *Transformers Documentation*. Hugging Face. URL: https://huggingface.co/docs/transformers/index

Selenium. (n.d.). *Selenium Documentation*. Selenium. URL: https://www.selenium.dev/documentation/en/

Scikit-learn. (n.d.). *Scikit-learn Documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/

NLTK. (n.d.). *NLTK Documentation*. Natural Language Toolkit. URL: https://www.nltk.org/