Sentiment Analysis of Movie Reviews using Naïve Bayes Classifier

Sentiment analysis, the process of extracting opinions and emotions from text, plays a crucial role in understanding public perception towards various products, services, or in this case, movies. In this report, we present the implementation and evaluation of a sentiment analysis system for movie reviews using the Naïve Bayes Classifier. The goal of this project is to classify movie reviews into positive and negative sentiments, providing valuable insights for filmmakers, critics, and moviegoers alike.

Implementation

1. Text Preprocessing:
The first step in our sentiment analysis pipeline involves text preprocessing to prepare the raw text data for analysis. We implemented several preprocessing techniques, including:

- Tokenization: Splitting the text into individual words.
- Stopword Removal: Eliminating common stopwords to reduce noise in the data.
- Lowercasing: Converting all words to lowercase to ensure consistency.
- Removing Non-Alphabetic Characters: Stripping non-alphabetic characters to focus on meaningful words.

2. Frequency Table Creation:
Once the text is preprocessed, we created frequency tables to capture the distribution of words across positive and negative reviews. This step involves counting the occurrences of each word in the reviews and organizing them by sentiment label.

3. Probability Calculation:
Using the frequency tables, we computed probabilities necessary for the Naïve Bayes Classifier, including:

- *P(positive): Probability of a review being positive.*
- *P(negative): Probability of a review being negative.*
- *P(word|positive): Probability of a word occurring in a positive review.*
- *P(word|negative): Probability of a word occurring in a negative review.*

4. Review Classification:

With the probabilities calculated, we implemented the Naïve Bayes Classifier to classify new reviews into positive or negative sentiments. The classifier utilizes Bayes' theorem to make predictions based on the probabilities of observed words in the review.

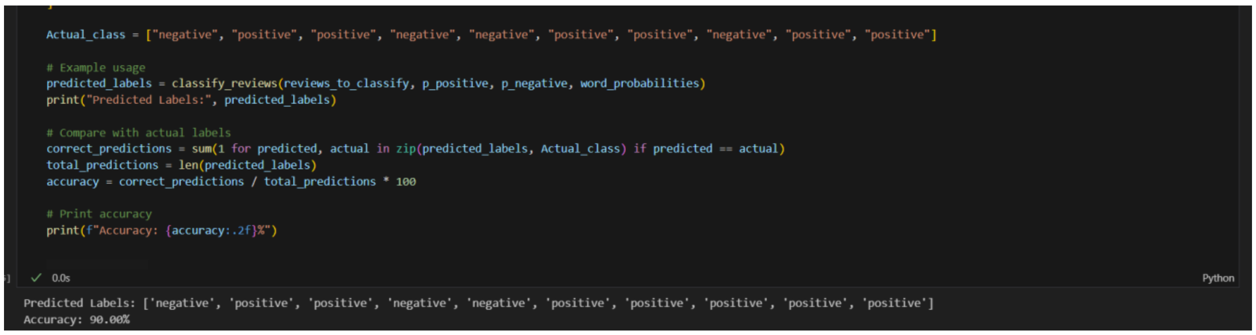
Training and Testing Data

For the training phase, a dataset consisting of 12,500 negative reviews and 12,500 positive reviews were used to train the model. This balanced dataset ensures that the classifier learns from an equal distribution of both positive and negative sentiments, contributing to robust model performance.

Additionally, a separate dataset containing 25,000 reviews was utilized to test the model's performance. The accuracy achieved on this test dataset was almost 65%, indicating the effectiveness of the trained model in classifying movie reviews.

Results

We used the model to test 10 IMDB movie reviews:



```python
Actual_class = ["negative", "positive", "positive", "negative", "negative", "positive", "positive", "negative", "positive", "positive"]

# Example usage
predicted_labels = classify_reviews(reviews_to_classify, p_positive, p_negative, word_probabilities)
print("Predicted Labels:", predicted_labels)

# Compare with actual labels
correct_predictions = sum(1 for predicted, actual in zip(predicted_labels, Actual_class) if predicted == actual)
total_predictions = len(predicted_labels)
accuracy = correct_predictions / total_predictions * 100

# Print accuracy
print(f"Accuracy: {accuracy:.2f}%")
```

```
Predicted Labels: ['negative', 'positive', 'positive', 'negative', 'negative', 'positive', 'positive', 'positive', 'positive', 'positive']
Accuracy: 90.00%
```

Accuracy was shown to be 90%.

Naïve Bayes Classifier for Color Image Classification

Image segmentation is a fundamental task in computer vision, aiming to partition an image into meaningful regions. In this approach, segmentation is achieved by assigning each pixel in the image to a particular class based on the probabilities derived from the RGB color space.

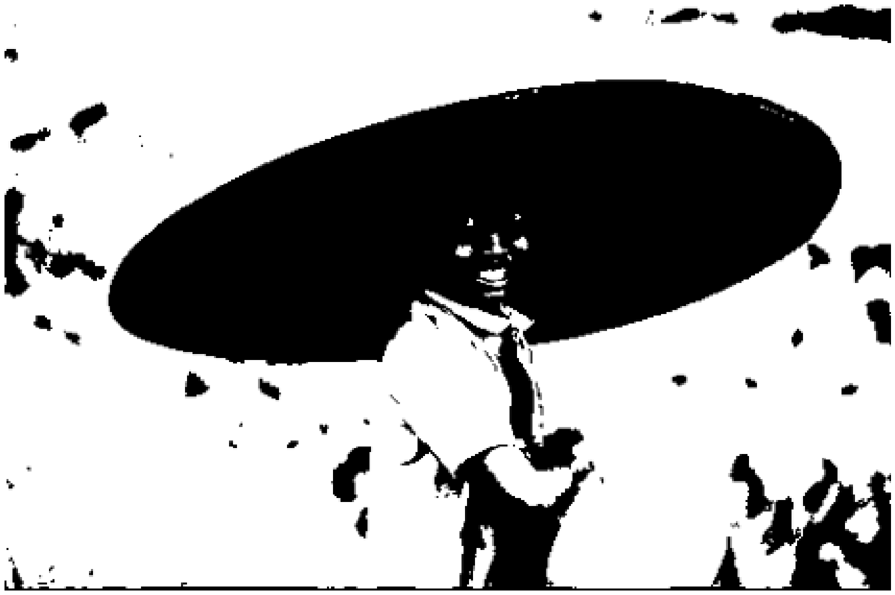
Implementation

1. Data Handling:
    - The code begins by mounting Google Drive to access image datasets stored in the cloud.
    - It processes both training and testing datasets, where images are loaded, converted to grayscale, and thresholded to create segmented images.
    - The code organizes processed image data into appropriate data structures for further analysis.

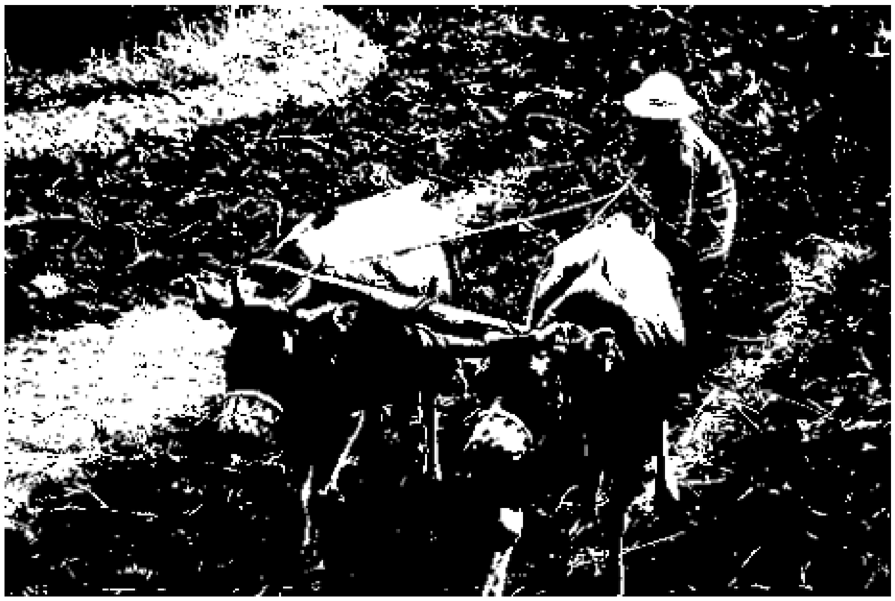
2. Image Processing:
    - Each image is converted to grayscale using the `rgb2gray` function from the `skimage.color` module.
    - Thresholding is applied to the grayscale image to segment it into binary regions, where pixels above a certain threshold are assigned one class (e.g., background) and pixels below the threshold are assigned another class (e.g., foreground).

   Example of used images:

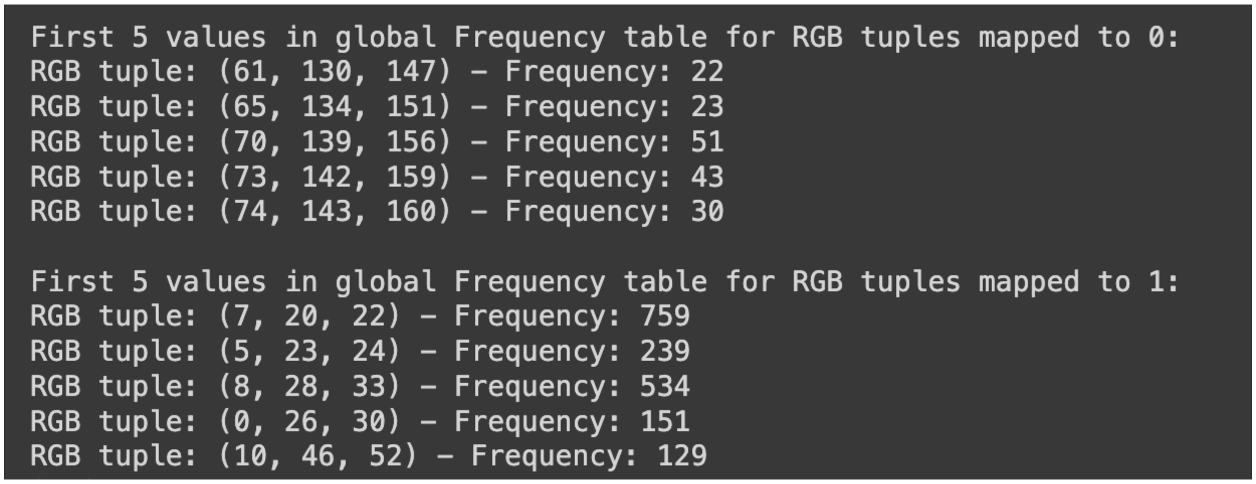
Segmented Image: 189011.jpg

- 

Segmented Image: 202012.jpg

- 

3. Probability Calculation:
   - The code calculates probabilities for each RGB tuple being mapped to either class (0 or 1) based on the segmented images.
   - It creates a global frequency table for RGB tuples mapped to 0 and 1, providing insights into the distribution of colors across different classes.
   - Probabilities are calculated based on the frequency of each RGB tuple occurrence in the segmented images.



```
First 5 values in global Frequency table for RGB tuples mapped to 0:
RGB tuple: (61, 130, 147) - Frequency: 22
RGB tuple: (65, 134, 151) - Frequency: 23
RGB tuple: (70, 139, 156) - Frequency: 51
RGB tuple: (73, 142, 159) - Frequency: 43
RGB tuple: (74, 143, 160) - Frequency: 30

First 5 values in global Frequency table for RGB tuples mapped to 1:
RGB tuple: (7, 20, 22) - Frequency: 759
RGB tuple: (5, 23, 24) - Frequency: 239
RGB tuple: (8, 28, 33) - Frequency: 534
RGB tuple: (0, 26, 30) - Frequency: 151
RGB tuple: (10, 46, 52) - Frequency: 129
```

This Screenshot shows the frequency of some RGB tuples.

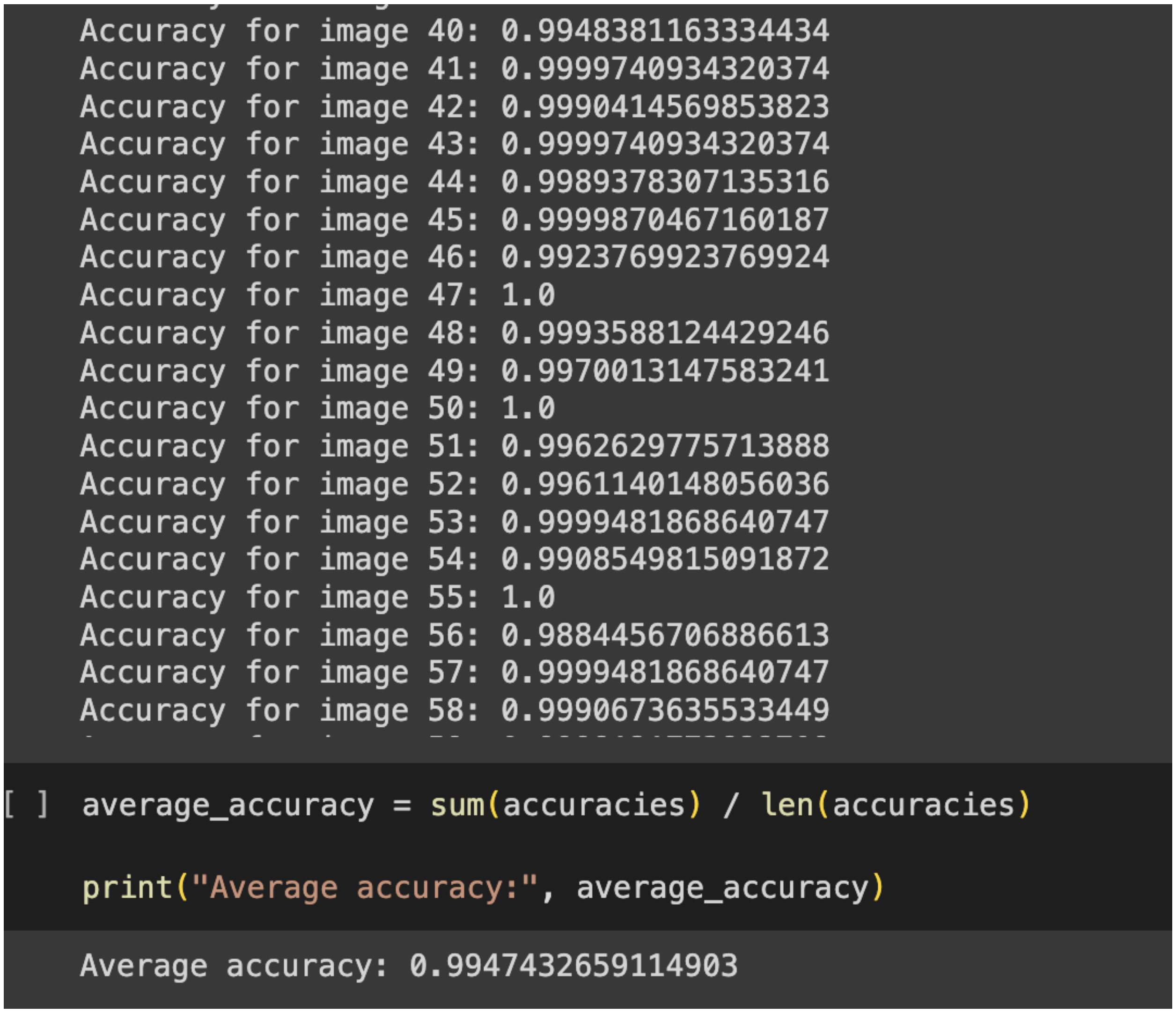
4. Image Classification:
    - For each test image, the code replaces pixel values based on the calculated probabilities.
    - Pixels are classified into either class 0 or 1 depending on the probability of their RGB tuple belonging to each class.
    - The classification process results in new segmented images with pixel values replaced according to the calculated probabilities.

5. Evaluation:
    - The code evaluates the accuracy of the segmentation by comparing the results with ground truth images.
    - Accuracy is calculated as the ratio of correctly classified pixels to the total number of pixels in the ground truth images.
    - The accuracy metric provides a quantitative measure of how well the segmentation method performs.

Training and Testing Data

The training dataset comprised 200 images, while the testing dataset also consisted of 200 images. These datasets serve as valuable resources for training and evaluating the performance of the segmentation method. By utilizing a substantial number of images for both training and testing, the model can learn robust features and generalize well to unseen data.

In Conclusion, The Accurracy result after testing the 200 images data was close to 99%



```
Accuracy for image 40: 0.9948381163334434
Accuracy for image 41: 0.9999740934320374
Accuracy for image 42: 0.9990414569853823
Accuracy for image 43: 0.9999740934320374
Accuracy for image 44: 0.9989378307135316
Accuracy for image 45: 0.9999870467160187
Accuracy for image 46: 0.9923769923769924
Accuracy for image 47: 1.0
Accuracy for image 48: 0.9993588124429246
Accuracy for image 49: 0.9970013147583241
Accuracy for image 50: 1.0
Accuracy for image 51: 0.9962629775713888
Accuracy for image 52: 0.9961140148056036
Accuracy for image 53: 0.9999481868640747
Accuracy for image 54: 0.9908549815091872
Accuracy for image 55: 1.0
Accuracy for image 56: 0.9884456706886613
Accuracy for image 57: 0.9999481868640747
Accuracy for image 58: 0.9990673635533449
```

```
[ ]  average_accuracy = sum(accuracies) / len(accuracies)

     print("Average accuracy:", average_accuracy)
```

```
Average accuracy: 0.9947432659114903
```

Some Accurracies of image and the final accuracy average over 200 images.