Md Sultanul Islam Ovi
G01444040
movi@gmu.edu

Report: Movie Review Classification

Miner Username: movi
Accuracy: 85%
Rank: 25

1 Introduction

The goal of this homework is to classify movie reviews as either positive or negative using a k-Nearest Neighbor (k-NN) classifier. I need to predict the sentiment of each review from the text data and implement my own k-NN classifier with cross-validation instead of relying on built-in libraries. In this report, I will describe how I prepared the data, implemented the custom k-NN algorithm, and selected the best features and settings for accurate predictions.

2 Data Handling Process

I created a custom `DataProcessor` class to process the movie review data from the `train.dat` and `test.dat` files into a format suitable for analysis.

Train Data Processing: For the training data, the `process_file` method read each review from `train.dat`, separated the sentiment label (+1 or -1), and cleaned the text by removing extra strings. The processed data was saved as `train.csv`, containing the review text and corresponding sentiment.

Test Data Processing: For the test data in `test.dat`, which lacks sentiment labels, I used the same method without removing sentiments. I also cleaned up unwanted strings like `#EOF` and HTML tags, and saved the processed data as `test.csv`.

Data Preview and Validation: After processing, I validated the data by loading the CSV files and previewing the first few rows. The training file contained both reviews and sentiment labels, while the test file contained only the reviews.

Distribution of Ratings: I plotted the count of positive and negative reviews from `train.csv` to visualize the sentiment distribution using Python's `matplotlib` library.

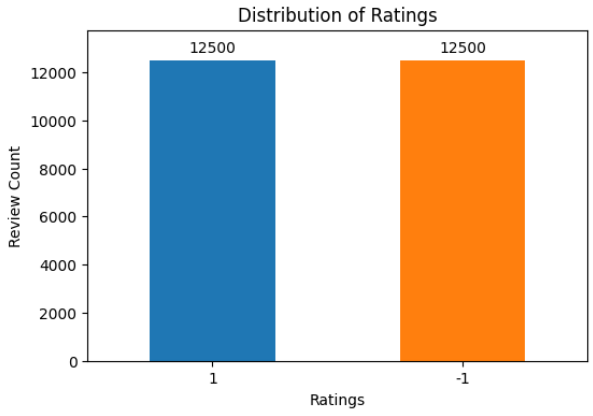
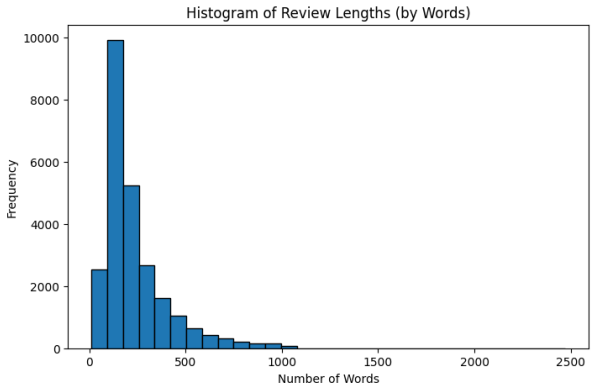


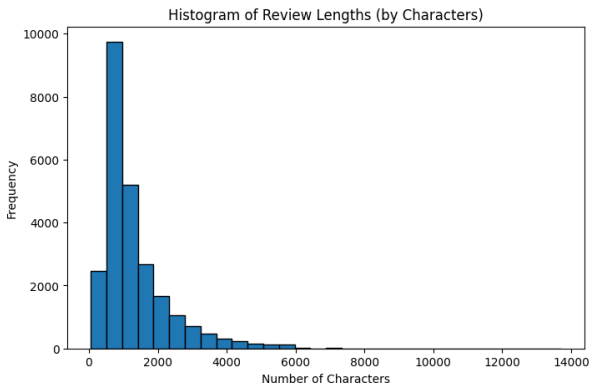
Figure 1: Distribution of Positive and Negative Ratings in the Training Data

Review Length Analysis: I generated two histograms to analyze the distribution of review lengths by words and characters. These histograms help understand the variability in review lengths, which may affect the classifier's performance.

Md Sultanul Islam Ovi
G01444040
movi@gmu.edu



(a) Histogram of Review Lengths (by Words)



(b) Histogram of Review Lengths (by Characters)

Figure 2: Distribution of Review Lengths in the Training Data

3 Data Preprocessing

To prepare the movie reviews for classification, I applied several preprocessing steps to clean and standardize the text data. The preprocessing steps are combined into a single function that performs all the necessary tasks to clean the reviews. Here is a breakdown of the steps involved:

Handling Stopwords and Negations: I started by setting up a list of stopwords using the NLTK library. Stopwords are common words like "the" and "is" that don't contribute much meaning to the sentiment of the text. I kept important negation words such as "no," "nor," and "not," because they can reverse the meaning of a sentence. I also added 'eof' to the stopwords list, as it appeared in the reviews but was irrelevant for sentiment classification.

Removing HTML Tags and Special Characters: The reviews contained HTML tags and special characters that were unnecessary for classification. I used regular expressions to remove these elements and keep only the important parts of the review.

Expanding Contractions and Lowercasing Text: Next, I expanded contractions like "don't" to "do not" using the `contractions` library. This step ensures that all words are properly represented in their full form. After that, I converted all the text to lowercase to avoid treating words like "Good" and "good" as different.

Part-of-Speech Tagging: To improve the quality of the text, I used part-of-speech (POS) tagging to filter out words that are not relevant to sentiment. I kept adjectives, adverbs, verbs, and negations because these types of words often carry the most sentiment-related information.

Finding and Replacing Common Bigrams and Trigrams: To improve the quality of the text for classification, I identified the most common bigram (two-word) and trigram (three-word) combinations from both positive and negative reviews. I used the NLTK library to tokenize the reviews and extract frequent combinations. Once the most common bigrams and trigrams were found, I replaced them with single-word alternatives to simplify the text and make the meaning more concise. For example, phrases like "not good" were replaced with "bad," and "well worth watching" was replaced with "recommended."

Combining All Preprocessing Steps: Finally, I combined all the above steps into a single function that processes each review. This function removes HTML tags, expands contractions, lowercases the text, removes special characters, applies POS tagging, and replaces bigrams and trigrams with simpler terms. The processed reviews were stored in a new column for further analysis.

Word Cloud for Positive and Negative Reviews: To visualize the most frequent words in both positive and negative reviews, I generated word clouds. Word clouds highlight the most common words in the reviews, with larger words representing higher frequencies. I created separate word clouds for positive and negative reviews, using a blue color scheme for positive words and a red color scheme for negative words.

Md Sultanul Islam Ovi
G01444040
movi@gmu.edu

(a) Word Cloud for Positive Reviews

(b) Word Cloud for Negative Reviews

Figure 3: Word Clouds for Positive and Negative Reviews

Handling Missing Values: I filled any missing values in the `Reviews` column with empty strings and removed leading or trailing whitespace to avoid issues during feature extraction.

Feature Extraction: Bag-of-Words vs. TF-IDF: To convert the cleaned text data into numerical features, I tried both Bag-of-Words (BoW) and TF-IDF. For BoW, I experimented with both binary and raw frequency counts. For TF-IDF, I applied term frequency-inverse document frequency to weigh words based on their importance across the reviews, which helped reduce the influence of commonly occurring words. After comparing both approaches, I found that TF-IDF produced better results, especially when combined with feature selection techniques.

Choosing Distance/Similarity Measures: I tested several distance measures, including Euclidean distance, Manhattan distance, and Cosine similarity. After experimentation, I decided to use Cosine similarity as it proved to be the most effective for handling text data, particularly when working with vectors.

Dimensionality Reduction and Feature Selection: To further improve model performance and efficiency, I explored dimensionality reduction techniques, including Principal Component Analysis (PCA), which reduces the feature space by finding uncorrelated components. I also tested SelectKBest with the chi-square test, which selects the top features most relevant to predicting sentiment. After evaluating both methods, I chose SelectKBest with chi-square, as it worked well with TF-IDF features and provided a good balance between dimensionality reduction and retaining the important information needed for classification. PCA, while useful, did not perform as well in this context due to the sparse nature of the TF-IDF features.

4 k-NN Classifier Implementation

The k-Nearest Neighbor (k-NN) algorithm was implemented from scratch to classify the sentiment of movie reviews. First, the text data was transformed into numerical features using TF-IDF vectorization with a range of n-grams, filtering terms based on their document frequency. After feature extraction, dimensionality reduction was applied using the Chi-Square test to select the most relevant features for classification.

For the classification itself, I developed a custom k-NN classifier that computes distances between a test review and each review in the training set. The algorithm uses a specified number of neighbors and a chosen distance metric, such as Cosine similarity, to determine the predicted label. Cross-validation was performed to evaluate different configurations of the classifier, adjusting the number of neighbors and testing various distance metrics. This process allowed me to identify the optimal configuration, which was then used in the final model.

5 Experiments and Results

I conducted multiple experiments by adjusting preprocessing steps and tuning the parameters of the classifier to find the best-performing configuration. Initially, I used the built-in k-NN classifier from the scikit-learn library to

test different parameter ranges, such as the number of neighbors and distance metrics. This allowed me to identify a reasonable parameter base for the model. Once the optimal range of parameters was established, I transitioned to a custom k-NN implementation.

To achieve optimal results, I applied cross-validation to evaluate different combinations of feature selection and the number of neighbors. To improve the efficiency of the algorithm, I implemented dimensionality reduction using the Chi-Square test to select the most relevant features. This significantly reduced the feature space and sped up the training process. Additionally, I parallelized the k-NN prediction process across multiple CPUs, running the program on the Lightning.ai platform with 32 CPUs. This enabled faster execution, as running the program on a single CPU was too slow for practical use.

The final set of experiments focused on identifying the best values for feature selection and k-NN neighbors from the following parameter grid:

```
param_grid = {
    'feature_selection__k': list(range(45000, 60001, 1000)),
    'knn__n_neighbors': list(range(450, 551, 1)),
    'knn__metric': ['cosine']
}
```

Using 5-fold cross-validation, I tested 1616 different combinations of parameters, resulting in a total of 8080 fits. The table below presents the top 5 parameter combinations based on cross-validated accuracy.

Table 1: Top 5 Models Based on Cross-validated Accuracy

Model	Distance Metric	Feature Selection (k)	k-NN Neighbors	Accuracy (%)
Model 989	Cosine	54000	529	85.28
Model 1084	Cosine	55000	523	85.27
Model 987	Cosine	54000	527	85.26
Model 896	Cosine	53000	537	85.25
Model 1080	Cosine	55000	519	85.25

6 Result Summary

The final model achieved a training accuracy of 85.28%, while the highest accuracy on the public leaderboard was 85%. Multiple submissions were made, and the best result, reported here, ranked 25th on the public leaderboard. The ranking could potentially have been higher if only unique submissions were considered, as many participants submitted multiple results. The k-Nearest Neighbor (k-NN) classifier performed best with $k = 529$ using Cosine similarity. For feature extraction, TF-IDF vectorization was applied with an n-gram range of 1 to 4, `min_df = 6`, and `max_df = 0.8`. Feature selection was done using SelectKBest with the chi-square test, retaining the top 54,000 features relevant to sentiment classification. These parameters and techniques resulted in the best performance for my classifier.

7 Instructions for Running the Code

To reproduce the results, follow these steps:

1. Install the necessary libraries.
2. Run the script `knn_classifier.py`.
3. Ensure that the dataset files `train.dat` and `test.dat` are in the same directory as the script.