Report on H W1 - Movie Review Classification

User Information:

User Name: Rithvik

Rank & Accuracy:

Rank: 206

Accuracy: 0.81

Instructions to Run the Program:

1. Install the required Python libraries:

pip install numpy pandas nltk scikit-learn

2. Ensure the Train_data.txt and Test_data.txt files are available in the same directory as the Python script.

3. Run the Python script:

python HW1-G01501815.py

The program will:

- Preprocess the input text files.

- Train the KNN classifier using cross-validation.

- Output predictions to predictions.txt.

Approach:

1. Data Preprocessing:

The raw text data is preprocessed by removing unwanted characters (HTML tags, non-alphanumeric characters) and performing tokenization. The words are lemmatized, and stop words are removed to clean the text further.

2. Vectorization:

The cleaned text data is converted into numerical features using TfidfVectorizer, which creates a matrix representing the importance of each word in the context of all reviews.

3. Parameter Choices:

Number of Features: Feature selection is performed using SelectKBest with the chi-square (chi2) test, selecting the top 2000 features. This reduces the dimensionality of the dataset while preserving the most important features for classification.

Number of Neighbors (k): During hyperparameter tuning, various values of k were tested. After trying sorted values of [171, 173, 177, 167, 139, 181, 169, 175, 191, 185], the best performance was achieved with k=191, which provided the highest accuracy during cross-validation.

Cosine Similarity: Instead of traditional distance-based metrics, cosine similarity was used to measure the similarity between text vectors. This is particularly effective for high-dimensional, sparse data like text.

Cross-Validation:

A 6-fold cross-validation was performed to evaluate the model's performance across different splits of the data. This approach ensured that the model was not overfitting and that it generalized well to unseen data. The accuracy was computed for each fold, and the average accuracy was used to select the best parameters.

Results:

Fold	Accuracy
1	0.89
2	0.88
3	0.90
4	0.91
5	0.87
6	0.88

Average Accuracy: 0.88 (88%).

Efficiency and Runtime Optimization:

Initial Implementation: The KNN algorithm was initially implemented with loops, but this approach was slow due to the high dimensionality of the TF-IDF matrix. As a result, the cosine similarity between vectors was computed using NumPy's optimized matrix operations, significantly improving performance.

Dimensionality Reduction: By selecting the top 2000 features using SelectKBest, the dimensionality of the input data was reduced from its original size. This not only improved

the runtime but also enhanced the model's accuracy by focusing on the most relevant features.

Runtime Comparison:

Before Optimization (without feature selection): Processing time was approximately 120 seconds for the full training dataset with an accuracy of ~85%.

After Optimization (with feature selection and matrix operations): Processing time reduced to ~30 seconds, with improved accuracy of 88%.

Conclusion:

This project implemented a custom K-Nearest Neighbors classifier for sentiment analysis of text data. After hyperparameter tuning using values sorted as [171, 173, 177, 167, 139, 181, 169, 175, 191, 185], the optimal k was found to be 191, providing the best accuracy of 88%. By optimizing the model with cosine similarity, feature selection, and cross-validation, the final result is a fast, efficient, and accurate KNN classifier.