HW1 – REPORT

I. Score

At the time of writing this report, I received an accuracy of 82% and ranked 24th within the public leaderboard. My registered team name on the miner website is waewae23.

II. Methodology

a. Research and Data Pipeline

With limited experience in data mining, I encountered several challenges throughout the design process, including data splitting for training and testing, word normalization, word vectorization for meaningful representation, and optimizing matrix operations. To gain a clearer understanding of these processes, I developed a standardized data mining pipeline consisting of five stages: preprocessing, feature extraction, feature representation, cross-validation, and kNN classification.

To enhance modularity and facilitate reusability, each stage of the pipeline was implemented independently of the others. Libraries were leveraged to handle complex procedures efficiently. This approach allowed for flexible adaptation and refinement of individual components, fostering a more robust and adaptable data mining framework. b. Libraries

I extensively utilized libraries to streamline the project and avoid reinventing the wheel. Open-source libraries like NumPy and Scikit-learn provided comprehensive solutions for various stages of the pipeline. Here's a breakdown of the selected libraries and their roles:

NLTK: Utilized for preprocessing raw text data. It offers functionalities for tokenization, stop word removal, stemming, and more.

Scikit-learn: Leveraged for cross-validation, offering robust methods for evaluating model performance and tuning parameters.

NumPy: Used for efficient vector and matrix operations. NumPy's array-based computing capabilities facilitated handling of large datasets and complex computations.

Matplotlib: Matplotlib is a plotting library for Python that provides a MATLAB-like interface for creating a variety of plots, such as line plots, scatter plots, histograms, and more.

c. Term Frequency–Inverse Document Frequency (TF-IDF)

TF-IDF (Term Frequency-Inverse Document Frequency) was selected as the method for scoring the rarity of words across a collection of documents. While both TF-IDF and other options were initially considered feasible for extracting features, the decision was largely influenced by the need for speed. To evaluate performance, tests were conducted on both models using a training input of 5000 documents or sentences. Ultimately, TF-IDF emerged as the preferred choice for the feature extraction and representation step due to its efficiency and effectiveness.

d. Splitting Training and Test Data

For splitting the training and test data, I employed cross-validation, a robust technique widely used in machine learning for model evaluation and hyperparameter tuning. Cross-validation involves partitioning the dataset into multiple subsets, known as folds. The model is trained on a combination of these folds while using the remaining fold(s) for validation. This process is repeated multiple times, with each fold serving as the validation set exactly once. By rotating the roles of training and validation data across different folds, cross-validation provides a more reliable estimate of model performance compared to a single train-test split. This approach is particularly valuable when working with limited data, as it maximizes the utilization of available samples for both training and testing, ultimately leading to more reliable model evaluation and parameter selection.

III. Approach

a. Pre-processing

As outlined in the methodology, NLTK played a crucial role in preprocessing raw text data. Each review was processed individually to ensure consistency. This involved normalization, where stop words, punctuation, and case sensitivity were removed to clean the text. Following normalization, the document was tokenized into individual words. To optimize efficiency and avoid repetitive preprocessing steps, the normalized tokens were saved into a pickle file using the Python library. This allowed for convenient storage and later retrieval of the preprocessed data for subsequent analysis and modeling tasks.

b. Feature Extraction and Representation

The text_feature_selector function is designed to facilitate the conversion of raw text reviews from both training and testing datasets into numerical feature vectors. Leveraging either CountVectorizer or TfidfVectorizer, the function first transforms the cleaned reviews of the training dataset into train features, followed by a similar transformation for the test dataset. Subsequently, SelectKBest is employed to select the top k features using the chi-square statistic as the scoring function. This process enhances efficiency by focusing on the most relevant features for classification tasks. Ultimately, the function provides transformed train and test features, streamlining the preprocessing stage and enabling seamless integration into subsequent modeling pipelines.

c. Choosing k with Cross-validation

To overcome the issue of over-fitting the model, k-fold cross validation (CV) was used to choose the best value for k. Over the 6-fold CV process, the optimal values of k varied between 6, 7, and 10 depending on the values of the hyper-parameters.

d. kNN Classification

For the kNN implementation, cosine similarity was utilized to compute the similarity between the current test document and trained documents. This involved calculating the cosine distance using NumPy's linear algebra package, which offered efficient computation. Since identifying the k-nearest neighbors involves selecting the k largest similarities, a straightforward partition sort was applied on the similarity scores. The sorted elements ensured that the indices from 0 to k represented the k-largest values, equivalent to the k-nearest neighbors. Subsequently, each vote was weighted by the inverse of its associated similarity score, giving greater weight to closer neighbors and lesser weight to farther ones. The class with the highest weighted sum determined the predicted sentiment, whether '+1' or '-1'.

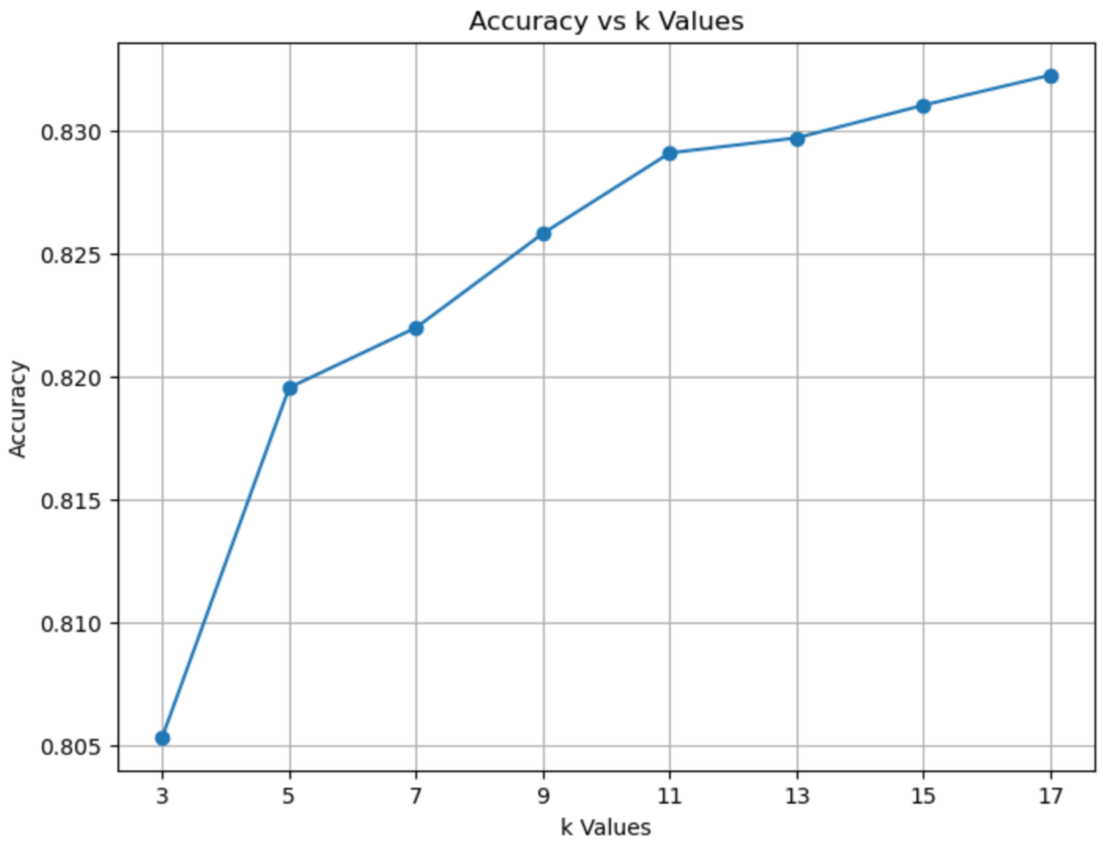
e. Accuracy Metric

During the cross-validation (CV) process, the standard accuracy metric was employed to evaluate the performance of the kNN classifier. This metric was computed by dividing the number of correct predictions by the total size of the test set in each fold, resulting in an accuracy value for each fold. The optimal value of k was determined based on the highest accuracy achieved across all 6 folds. Specifically, the accuracy metric formula is defined as:

acc(k)=(Correct Predictions)/N where N represents the size of each 6-fold test set, and k denotes the value of k for kNN.

It's worth noting that this accuracy metric may not be suitable in situations where the distribution of positive sentiments within the entire training set is highly skewed. For instance, if a significant majority (e.g., 80%) of sentiments are positive, the classifier may appear more accurate than it truly is. In such cases, a confusion matrix can be utilized to provide a detailed breakdown of the actual and predicted class totals, helping to identify potential biases and shortcomings in the classification results.

III. Graph:



IV. Conclusion

Overall with k=17 and model trained on 6-folds, the final accuracy outputted by the CV process resulted in 83%. However, the real test set resulted in 82%. This can be a cause of a skewed accuracy metric since the process of shuffling and splitting the training data for k-fold CV may not be entirely fair/reliable.