**I. Score**

At the time of writing this report, I received an accuracy of 78% and ranked 22nd within the public leaderboard. My registered team name on the miner website is **asdf\_dsa**.

**II. Methodology**

**a. Research and Data Pipeline**

With no experience in data mining, there were many obstacles during the design process such as splitting training/test data, normalizing words, representing words as meaningful vectors, and efficient matrix operations. To understand these processes from a clearer perspective, a standard data mining pipeline was created. The pipeline involved 5 stages: pre-processing, feature extraction, feature representation, cross-validation, and kNN classification. For design and re-use purposes, implementation of each stage was independent from the other stages and held libraries to accommodate for complex processes.

**b. Libraries**

Since the goal was to avoid re-inventing the wheel for the project, libraries were extensively utilized. Many open-source libraries such as NumPy, Scikit-learn, NLTK, Gensim, GloVe, and FastText were available to address this issue. The listed libraries fulfilled all requirements for each stage of the pipeline. The following list of libraries was selected: NLTK for preprocessing raw text data, Gensim’s Doc2Vec implementation for feature extraction/representation, Scikit-learn for cross-validation, and NumPy for efficient vector/matrix operations as my main libraries.

**c. Doc2Vec vs. Term Frequency–Inverse Document Frequency (TF-IDF)**

Doc2Vec is a concept introduced by Tomas Mikolov and Quoc Le, which represents documents as vectors through the Distributed Memory (DM) and Distributed Bag of Words (DBOW) model. In short, DM is responsible for remembering the context (ordering of words) in a document while DBOW is responsible for predicting the context of a document. On the other hand, TF-IDF was a method of scoring the rarity of words across a collection of documents. At face-value, both options were feasible solutions for extracting features. Since speed was a strong deciding factor, tests were performed on both models which involved a training input of 5000 documents/sentences. The results showed that Doc2Vec took ~5 seconds to train while TF-IDF took ~10 seconds to train. Thus, Doc2Vec was chosen for the feature extraction and representation step.

**d. Splitting Training and Test Data**

Originally, a combination of the training and test data was the input into the Doc2Vec model; however, this was not the best approach for a small dataset with only 18560 records. To prevent using the test set for hyper-parameter tuning, k-fold cross-validation was the most suitable approach with the small dataset.

**III. Approach**

**a. Pre-processing**

As mentioned in the above methodology, NLTK was used to clean and filter raw text data. Each review is read and normalized as a single document. Normalizing a document means to remove stop words, punctuation, and case sensitivity. Once a document is filtered, it is tokenized into individual words. Since the data remains the same and repeating the pre-processing step is unnecessary, the normalized tokens are saved into a pickle (Python library) file and later loaded for subsequent uses.

**b. Feature Extraction and Representation**

Gensim’s Doc2Vec implementation was used to extract and represent features in a 100-dimensional sparse matrix. In section II(c), DBOW and DM are the two main hyper-parameters to consider for Doc2Vec. By trial-and-error for this small dataset, using DM decreased the training time and accuracy. DBOW increased the training time and accuracy. Having noted this down, pure DBOW was the algorithm used to learn the context between words (dm=0, dbow=0).

Other parameters such as *epochs=20* (number of iterations to train the model on), *workers=4* (number of cores to delegate tasks to), *negative=5* (number of words to update the weights in the neural network), *hs=0* (enables negative sampling), *min\_count=3* (ignores words less than this frequency), and *window=10* (length of the sliding text window for DBOW) were set from trial-and-error as well.

**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 3-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, the Euclidean distances between the current test document and trained documents were computed through the vector norms. These distances were calculated using NumPy’s linear algebra package, which proved to be efficient and quick. Since finding the k-nearest neighbors is a k-smallest-values problem and order does not matter, a simple partition sort was applied on the distances. The sorted elements were ordered such that the elements from index 0 to k were the k-smallest values—similarly, the k-nearest neighbors. Once the k-nearest neighbors were found, each vote was weighted by the inverse of its associated distance. Thus, farther neighbors were weighted less and closer neighbors were weighted more. Each class of sentiment, ‘+1’ or ‘-1’, kept track of the sum of the neighbors’ weights and the class with the highest sum was the predicted sentiment.

**e. Accuracy Metric**

During the CV process, the standard accuracy metric was computed by taking the proportion of correct predictions to the total size of test set in each fold. The optimal value of k was determined by the highest accuracy out of all 3-folds. The accuracy metric is as follows:

where **N = size of each 3-fold test set** and **k = value of k for kNN**.

This metric would be unsuitable in situations where the number of positive sentiments for the whole training set was skewed. For instance, if 80% of the sentiments were positive, then the classifier would be skewed and appear more accurate than it should be. A confusion matrix can help reveal this issue by displaying the totals in the actual and predicted class.

**IV. Conclusion**

Overall with k=10 and the Doc2Vec model trained on 3-folds, the final accuracy outputted by the CV process resulted in 93%. However, the real test set resulted in 78%. 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. Future improvements will involve tuning the Doc2Vec hyper-parameters, training on a much larger dataset, and other classification techniques.