**I. Score**

At the time of writing this report, I received an accuracy of 77% and ranked 14th within the public leaderboard. My registered team name on the miner website is **asdf\_dsa**.

**II. Methodology**

**a. Research and Data Pipeline**

During the first lecture, I understood the concept and procedures of k-Nearest-Neighbors (kNN); however, I had many unanswered questions such as, “How do I represent a word within a meaningful graph?”, “What do I use to process all of the words?”, “Should I keep all of the words within the reviews?”, “What is the Bag-of-Words model?” Despite these questions, thorough research within these four topics encouraged the creation of a pipeline. In order to understand and organize the complexity of the project, a data mining pipeline was created. The pipeline involved 5 stages: pre-processing, feature extraction, feature representation, cross-validation, and kNN classification.

**b. Libraries**

Learning the preceding stages to kNN classification was essential. With no experience in data mining, the goal was to avoid re-inventing the wheel for the project. Many libraries and models were available for each stage of the pipeline. For the most part, the available libraries would help in pre-processing raw text data, extracting features given a list of words, and representing features in an n-dimensional vector space. A few Google searches resulted in several libraries such as NumPy, Scikit-learn, NLTK, Gensim, GloVe, and FastText that fulfilled all of the requirements. After discovering this, testing each of the libraries for performance and ease of API usability contributed towards increasing efficiency and reducing development time. The following list of libraries were chosen after testing and analysis of advantages and disadvantages: 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.

To avoid boilerplate code and encourage simplicity, Gensim’s implementation of Doc2Vec and Sklearn’s implementation of TF-IDF was used. 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 vector space. The authors of Doc2Vec, Mikolov and Le, mentioned the hyper-parameters they used in a Google forum about their original paper, *Distributed Representations of Sentences and Documents*. From the forum, there were two main parameters that contributed to either performance or quality: *dm* and *dbow\_words*. DM and DBOW are the two algorithms behind 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, DBOW was the algorithm used throughout the rest of the development. Although the number of dimensions is high, the small dataset allows for a 100-dimensional vector space to be computed in a short amount of time. Because of this, the Doc2Vec model was able to learn more of the context between words and output more accurate vector representations.

Other parameters such as *epochs=20* (number of iterations to train the model on), *workers=4* (number of cores to delegate tasks to), *negative=10* (number of words to update the weights in the neural network), *hs=0* (enables negative sampling), *min\_count=1* (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. During the CV process, the optimal values of k varied between 4, 5, and 7 depending on the values of the hyper-parameters. Also, there was an accuracy discrepancy between the training and test set. The smaller the dataset used to train the Doc2Vec model, the better the accuracy for the training set; however, this accuracy did not hold for the test set. For instance, the Doc2Vec trained on a smaller dataset (1500 documents) resulted in 88% accuracy during CV and 77% accuracy for the miner.

**d. kNN Classification**

For the kNN implementation, the Euclidean distances between the current test document and trained documents are 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 are found, each vote is weighted by the inverse of its associated distance. Thus, farther neighbors are weighted less and closer neighbors are weighted more. Each class of sentiment, ‘+1’ or ‘-1’, keeps track of the sum of the neighbors’ weights and the class with the highest sum is the predicted sentiment.

1. Describe how the metric Accuracy is computed. Which application will this be an unsuitable metric ?

**IV. Conclusion**