Identifier: jpenaarc

Highest score: 0.74

Yelp Reviews Classifier

In this report, we undertake the task of solving a classification problem centered around Yelp reviews. The goal is to develop a model that accurately predicts whether a given Yelp review is favorable or unfavorable. To tackle this task, we employ a popular algorithm in supervised learning known as K-Nearest Neighbors (KNN).

KNN is a simple yet powerful algorithm that classifies data points based on the majority class of their nearest neighbors in feature space. Parameter tuning lies at the heart of achieving optimal performance with KNN. Key parameters include the number of neighbors (K) and the distance metric used to measure similarity between data points. Finding the right combination of these parameters is crucial for maximizing the model's predictive accuracy.

Sections in the Report:

1. **Method**: In this section, we outline our approach to solving the classification problem. We detail the steps taken, including data preprocessing, feature extraction, model training, and evaluation. We specify which parts of the code were developed from scratch and which libraries were utilized.
2. **Results**: This section presents the outcomes of our experiments. We include tables, graphs, and plots to showcase performance metrics, such as accuracy, precision, recall, and F1 score. Each plot and graph is labeled appropriately, and we provide a brief explanation of the methodologies used to generate them.
3. **Conclusions**: Here, we summarize the findings from our analysis. We discuss what the results convey about the effectiveness of our approach, any insights gained from the experimentation process, and potential areas for future research or improvement.

Method

Preprocessing

The initial step in implementing our model involved processing the text data extracted from the csv file containing the Yelp reviews. The primary objective was to create a structured representation (in our case a vector) of the textual content in each review so that it could be utilized by our model. To achieve this, we began by constructing a vocabulary from the training data, which would then be converted into a numerical format containing TF-IDF (Term Frequency-Inverse Document Frequency) values. To streamline this process and improve the quality of our features, we employed several preprocessing techniques. Firstly, we removed unnecessary words known as stop words, which are common words that occur frequently in language but typically contain little semantic meaning (e.g., "the", "and", "is"). Additionally, we eliminated all punctuation marks to ensure a more uniform text corpus. Following this, we applied lemmatization to the words, a process that involves reducing

them to their base or dictionary form (e.g., "running" becomes "run"). Finally, we employed stemming, which is the process of reducing words to their root or stem form (e.g., " happiness" becomes "happi"), to further normalize the text data. These preprocessing steps collectively serve to refine the text in each review, reducing the vocabulary to which new entries will be compared, and thereby enhancing the effectiveness of our classification model.

Libraries used

Removing punctuation: Re.sub
Stemming: nltk.stem
Removing stop words: nltk.corpus
Lemmatizing words: nltk.stem

TF-IDF

After preprocessing all the Yelp reviews, we convert them into a numerical format using a technique called TF-IDF (Term Frequency-Inverse Document Frequency). TF-IDF is a statistical measure that evaluates the importance of a word in a document relative to a collection of documents. In our case each document represents a review. TF-IDF considers how often a word appears in a specific document (Term Frequency) and balances it against how often the word appears in the entire collection of documents (Inverse Document Frequency). This helps in highlighting words that are unique or important to a particular document while downplaying common words that appear frequently across all documents. The TF-IDF score for a term in a document is calculated as follows:

$$TF = \frac{\text{number of times the term appears in the document}}{\text{total number of terms in the document}}$$

$$IDF = log(\frac{\text{number of the documents in the corpus}}{\text{number of documents in the corpus contain the term}})$$

$$TF\text{-}IDF = TF * IDF$$

Once we've computed the TF-IDF scores for all terms in the documents, we use these scores to represent each document as a numerical array. The sklearn library provides efficient tools for this vectorization process, allowing us to transform the preprocessed Yelp reviews into TF-IDF arrays, where each element of the array corresponds to the TF-IDF score of a particular term in the vocabulary. These TF-IDF arrays serve as input data for training our classification model, enabling us to leverage the textual content of the reviews while preserving the statistical significance of each word within the document collection.

KNN Algorithm

Our implementation of the KNN (K-Nearest Neighbors) algorithm consists of a set of functions designed to carry out different steps of the classification process. Firstly, we have a function called `cosine_distance(vec1, vec2)` which computes the cosine distance between two vectors, representing the similarity between them. This function utilizes the `cosine_distances` method from the sklearn library to calculate the cosine distance between two vectors. The next function, `get_neighbors(train,train_y, test_row, num_neighbors)`, is responsible for identifying the K nearest neighbors of a given test sample within the training data. It iterates through each row in the training data, computes the cosine distance between the test sample and each training sample, and stores the distances along with their corresponding indices. The distances are then sorted, and the indices of the K nearest neighbors are returned.The `predict(train,train_y, test_row, num_neighbors)` function predicts the class label for a given test sample based on the class labels of its nearest neighbors. It retrieves the class labels of the K nearest neighbors from the training data and selects the most common class label among them as the predicted class label for the test sample. Finally, the `k_nearest_neighbors(train,train_y, test, num_neighbors)` function implements the entire KNN algorithm by iterating through each test sample, predicting its class label using the `predict` function, and storing the predictions in a list. This function returns a list of predicted class labels for all test samples.

Results

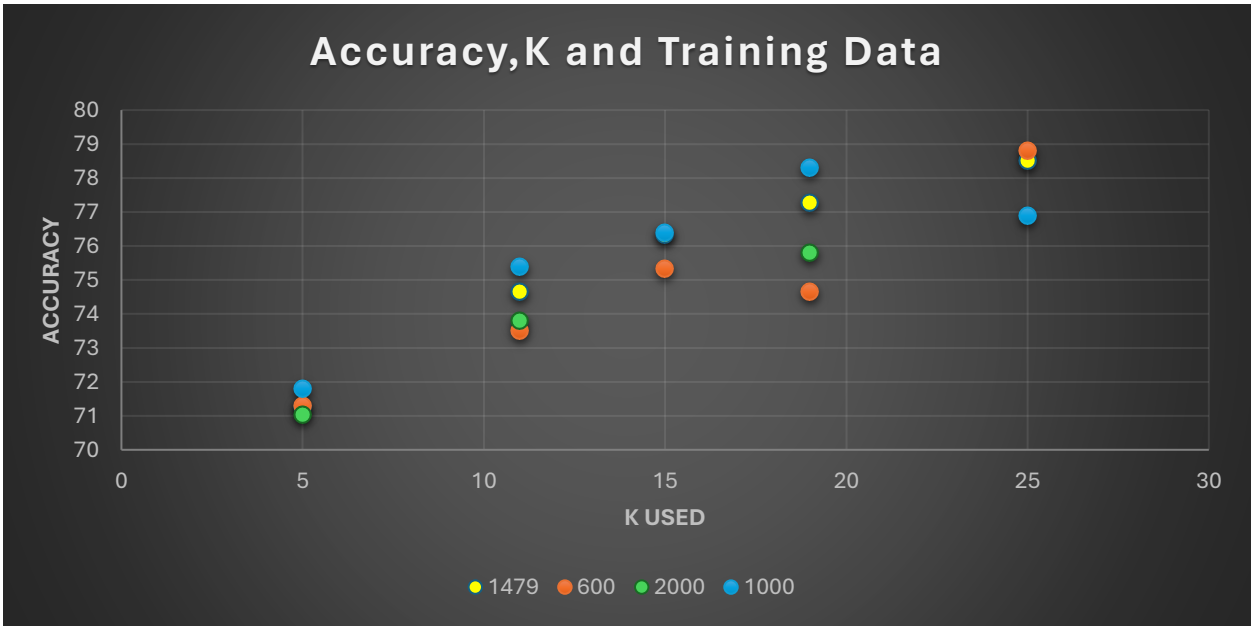
Cross Validation

To test the overall accuracy of our model, we employed the K-Fold cross-validation method, using a functionality provided by the sklearn library. We utilized a K value of 10, dividing our dataset into 10 sections or "folds." During each iteration of the cross-validation process, 9 of these folds were utilized for training the model, while the remaining fold was reserved for testing. This approach ensured that every data point had the opportunity to be both in the training and testing sets.

For each iteration, we recorded the accuracy of the model on the test fold. By repeating this process 10 times, with each fold serving as the test set once, we obtained a collection of accuracy scores. Finally, to obtain a single representative measure of the model's performance, we calculated the average accuracy across all iterations. This methodology allowed us to obtain a robust estimate of the model's accuracy, mitigating the potential impact of random variations in the training and testing data splits.

Data Collected

Accuracy	Traing Data(Docs)	K	Time (predictions per minute)	vocabulary size
71.26	1479	5	93	8237
76.33	1479	15	93	8237
77.28	1479	19	93	8237
78.5	1479	25	93	8237
74.64	1479	11	93	8237
73.5	600	11	349	5311
71.3	600	5	349	5311
74.66	600	19	349	5311
78.8	600	25	349	5311
75.33	600	15	349	5311
71.05	2000	5	76	9453
75.8	2000	19	76	9453
73.8	2000	11	76	9453
75.4	1000	11	130	6747
71.8	1000	5	130	6747
76.4	1000	15	130	6747
78.3	1000	19	130	6747
76.9	1000	25	130	6747



Conclusions

Our results show that the accuracy of the k-Nearest Neighbors (kNN) algorithm heavily relies on parameter tuning, specifically the choice of the parameter 'k' and the size of the training data. Our tests revealed significant insights into optimizing these parameters for improved performance.Through experimentation, we found that the optimal amount of training data was 1000 reviews, suggesting a balance between having enough data to capture the underlying patterns in the dataset while avoiding excessive computational complexity. Additionally, we determined that the ideal range for the 'k' parameter fell between 15 and 19. Our tests demonstrated that with a 'k' value of 15, our algorithm achieved a leaderboard accuracy of 74%. However, further analysis indicated that we could potentially enhance this accuracy by increasing the 'k' value to 19. Moreover, we observed diminishing returns in accuracy improvement beyond 'k' = 19. This phenomenon suggests that after a certain point, increasing the size of the neighborhood for classification yields minimal benefits. Additionally, escalating the training data beyond the optimal threshold of 1000 reviews led to a significant expansion of the vocabulary (vectors), resulting in diminishing returns and computational overhead.