Sentiment Analysis of Restaurant Reviews Using Naive Bayes Classification

Ye-eun Kim 22098692
dept. School of Physics, Engineering and Computer Science
University of Hertfordshire
Hatfield, Hertfordshire
july215215@gmail.com

I. Introduction

This project explores Natural Language Processing (NLP) by developing a Naive Bayes classifier from scratch for sentiment analysis of restaurant reviews. Sourced from Kaggle, the dataset contains 1,000 reviews, each labeled as positive or negative. (https://www.kaggle.com/datasets/nanuprasad/restaurant-reviews) The aim is to categorize these reviews, enhancing customer experience analysis. The approach involves comprehensive data preprocessing using NLTK for tokenization, stopwords removal, and stemming, followed by feature extraction through TF-IDF vectorization. This process, building upon the Bag of Words model, prepares the text for a custom-built Naive Bayes classifier that incorporates Laplace smoothing and log likelihood calculations to predict sentiments

II. Methodology

This project employs a practical approach to NLP by constructing a Naive Bayes classifier from the ground up, specifically for sentiment analysis of restaurant reviews. The methodology starts with data preprocessing, where the Bag of Words model is implicitly utilized as a foundation for feature extraction through TF-IDF vectorization.

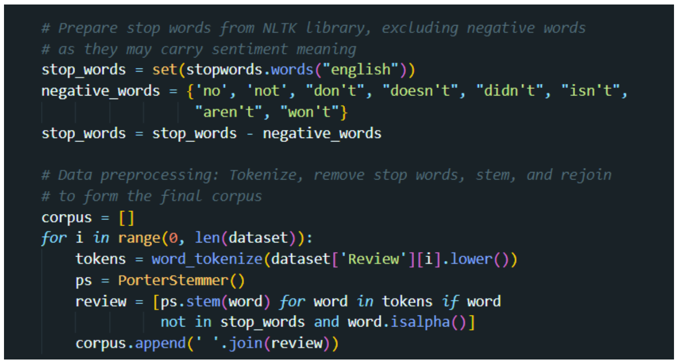
A. Data Preprocessing

The dataset, comprising 1,000 restaurant reviews, underwent a series of preprocessing steps designed to standardize the text and reduce noise that could potentially skew the analysis. The first step involved tokenization, where the text was split into individual words, or "tokens," using the `word_tokenize` function from the NLTK library, which is proficient in handling the nuances of natural language.

Subsequently, all tokens were converted to lowercase to maintain consistency, as text data is case-sensitive, and the presence of both uppercase and lowercase versions of the same word would be treated as distinct entities otherwise. The process continued with the removal of non-alphabetic characters to exclude any numbers and symbols that do not contribute to sentiment analysis.

A critical aspect of the preprocessing phase was the exclusion of stopwords, which are commonly used words in a language that carry minimal informative weight, such as "the," "is," and "in." A curated list of stopwords from the NLTK corpus was utilized, with the exception of negations like "not," "don't," and "isn't," as these can significantly alter the sentiment of a phrase. The retention of these negative words is essential in sentiment analysis, as they can invert the sentiment expressed in a sentence.

Following the removal of stopwords, the Porter Stemming algorithm, also from NLTK, was applied to reduce words to their root form, or "stem," thereby consolidating different grammatical variations of a word into a single representation.



```python
# Prepare stop words from NLTK library, excluding negative words
# as they may carry sentiment meaning
stop_words = set(stopwords.words("english"))
negative_words = {'no', 'not', "don't", "doesn't", "didn't", "isn't",
                  "aren't", "won't"}
stop_words = stop_words - negative_words

# Data preprocessing: Tokenize, remove stop words, stem, and rejoin
# to form the final corpus
corpus = []
for i in range(0, len(dataset)):
    tokens = word_tokenize(dataset['Review'][i].lower())
    ps = PorterStemmer()
    review = [ps.stem(word) for word in tokens if word
              not in stop_words and word.isalpha()]
    corpus.append(' '.join(review))
```

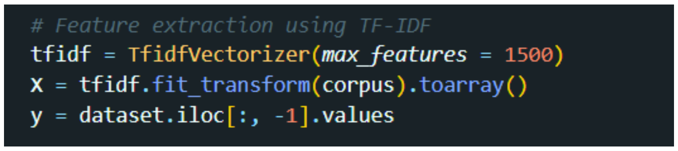
Fig. 1. Data Preprocessing

B. Feature Extraction

The preprocessed data undergoes feature extraction through TF-IDF vectorization, a process that initially aligns with the Bag of Words (BoW) model. This method, executed by the `TfidfVectorizer`, begins by counting the occurrence of each word within documents, akin to the BoW approach. It then extends this model by applying the Term Frequency-Inverse Document Frequency (TF-IDF) measure, which quantifies text by emphasizing words that uniquely characterize each document. This dual approach captures both the frequency of words (BoW) and their relative importance (TF-IDF), thus creating a more nuanced feature set for subsequent sentiment analysis.

C. Custom Naive Bayes Classifier

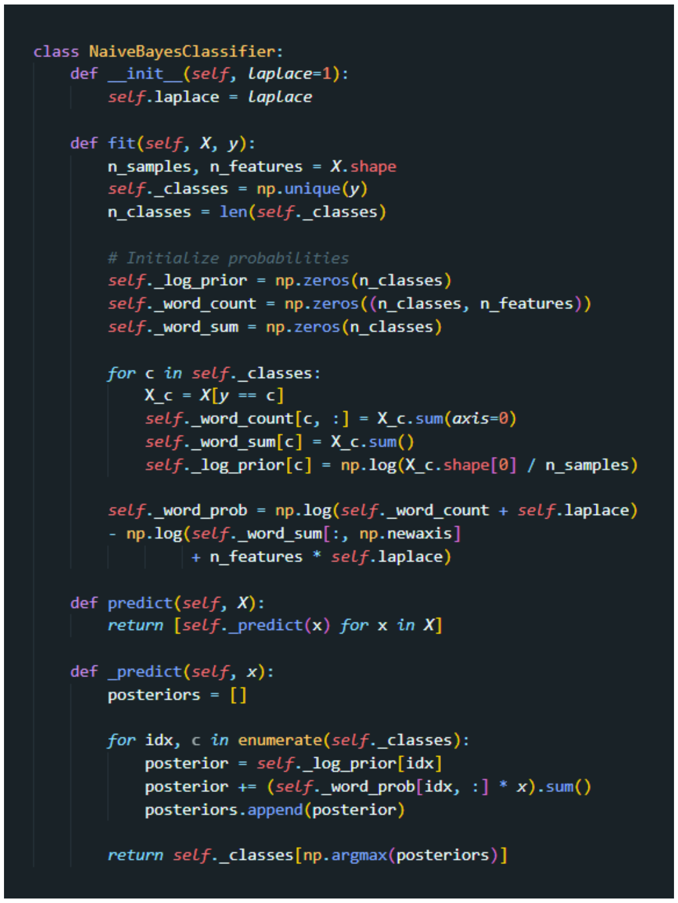
In this project's Custom Naive Bayes Classifier, the class embodies the fundamentals of the Naive Bayes algorithm. It begins by calculating conditional probabilities of each word



```
# Feature extraction using TF-IDF
tfidf = TfidfVectorizer(max_features = 1500)
X = tfidf.fit_transform(corpus).toarray()
y = dataset.iloc[:, -1].values
```

Fig. 2. Feature Extraction

given a class, using the formula P(word—class) = (frequency of the word in the class + 1) / (total count of all words in the class + size of the vocabulary). This approach, known as Laplace smoothing, addresses the zero-frequency problem by adding 1 to each frequency count. During fitting, these probabilities are converted to log scale to simplify computations, effectively managing the high-dimensional nature of text data. The classifier then aggregates these log probabilities for each class, and prediction is made by selecting the class with the highest posterior probability. This methodology enhances the accuracy and reliability of sentiment predictions in our text data.

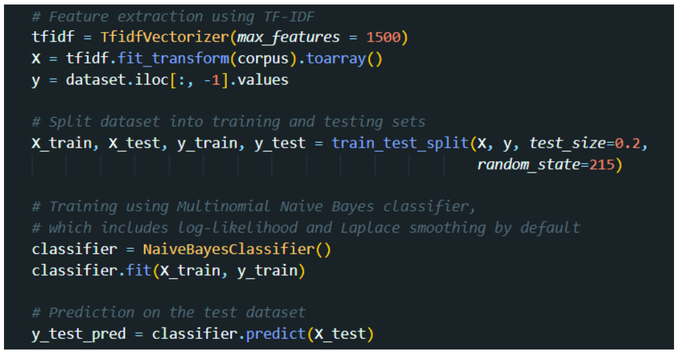


```
class NaiveBayesClassifier:
    def __init__(self, laplace=1):
        self.laplace = laplace

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self._classes = np.unique(y)
        n_classes = len(self._classes)

        # Initialize probabilities
        self._log_prior = np.zeros(n_classes)
        self._word_count = np.zeros((n_classes, n_features))
        self._word_sum = np.zeros(n_classes)

        for c in self._classes:
            X_c = X[y == c]
            self._word_count[c, :] = X_c.sum(axis=0)
            self._word_sum[c] = X_c.sum()
            self._log_prior[c] = np.log(X_c.shape[0] / n_samples)

        self._word_prob = np.log(self._word_count + self.laplace)
        - np.log(self._word_sum[:, np.newaxis]
                  + n_features * self.laplace)

    def predict(self, X):
        return [self._predict(x) for x in X]

    def _predict(self, x):
        posteriors = []

        for idx, c in enumerate(self._classes):
            posterior = self._log_prior[idx]
            posterior += (self._word_prob[idx, :] * x).sum()
            posteriors.append(posterior)

        return self._classes[np.argmax(posteriors)]
```

Fig. 3. Custom Naive Bayes Classifier

D. Model Training and Prediction

The dataset was initially partitioned into training and testing subsets, with 80% allocated for training and the remaining 20% designated for evaluation. This division ensures a comprehensive approach in both the learning phase and subsequent validation of the model's efficacy. Post-training, the model's performance was meticulously gauged using the test set. Comparisons of predictions against actual sentiments were conducted, leading to the construction of a confusion matrix. This matrix was instrumental in deriving various performance metrics, such as accuracy, precision, recall, and the F1 score. These metrics collectively offer insights into the classifier's proficiency in sentiment analysis.



```
# Feature extraction using TF-IDF
tfidf = TfidfVectorizer(max_features = 1500)
X = tfidf.fit_transform(corpus).toarray()
y = dataset.iloc[:, -1].values

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=215)

# Training using Multinomial Naive Bayes classifier,
# which includes log-likelihood and Laplace smoothing by default
classifier = NaiveBayesClassifier()
classifier.fit(X_train, y_train)

# Prediction on the test dataset
y_test_pred = classifier.predict(X_test)
```

Fig. 4. Model Training and Prediction

III. Results and Discussion

The performance of the custom Naive Bayes classifier was evaluated using a set of metrics derived from the confusion matrix. The classifier achieved an accuracy of 78%, with a precision of 76%, recall of 82%, and an F1 score of 79%. The confusion matrix is presented as follows:

$$\begin{bmatrix}75 & 26\\18 & 81\end{bmatrix}$$

The matrix indicates that while the classifier performs reasonably well in identifying both positive and negative sentiments, there is a notable incidence of false positives and false negatives. The precision metric suggests that when the classifier predicts a review to be positive, it is correct 76% of the time. The recall metric shows that the classifier successfully identifies 82% of all actual positive reviews. The F1 score, which combines precision and recall, indicates a balanced performance.

IV. Conclusion

In conclusion, the custom Naive Bayes classifier developed for this project demonstrates considerable efficacy in the sentiment analysis of restaurant reviews. Despite the challenges inherent in text classification, the classifier exhibits a commendable balance of precision and recall, as reflected in the F1 score. The insights garnered from the performance metrics underscore the potential of such tailored classifiers in NLP applications. While there is room for enhancement, particularly in reducing false positives, the foundation established by this study provides a robust basis for future explorations. Further research could explore more sophisticated natural language processing techniques and the integration of additional contextual factors to refine the classifier's performance.