Texas A&M University - Corpus Christi

6300 Ocean Dr, Corpus Christi - 78412

School of Engineering & Computer Sciences

Spring 2024

Build an NLP System to Identify the Sentiment of a Document by Document Scoring Method

Project Report for COSC 6590 001

Natural Language Processing

ID	Name	Task
A04298487	Korlakunta, Reddy Bhuvan	Model Training
A04298775	Bandi, Om Preetham	Data Analysis
A04297618	Ullendula, Thriveen	Testing
A04299831	Virigineni, Sravya Sri	Documentation

Introduction

In the age of information, sentiment analysis has emerged as a critical tool for understanding the emotional undertones in vast tracts of text data. This project, titled "Objective: Build an NLP System to Identify the Sentiment of a Document by Document Scoring Method," is an ambitious venture into the realm of Natural Language Processing (NLP) with the purpose of designing a system capable of discerning the sentiment expressed within documents.

Our approach leverages the power of document scoring, employing a feature-rich model to parse through the nuanced linguistic elements that define sentiment. The cornerstone of our method is the Term Frequency-Inverse Document Frequency (tf-idf) statistic, which measures the importance of a word in the context of a document against its frequency across a corpus. This statistic allows for the distinguishing features of a text that are most indicative of sentiment to be quantified and utilized effectively.

This project unfolds in two primary stages: the design and implementation of text preprocessing techniques, and the construction and evaluation of a predictive model. In the initial stage, text preprocessing, we systematically clean and normalize the text data, stripping it of punctuation and adjusting letter cases. Verbs are lemmatized to their base forms, ensuring consistency across the dataset. Thereafter, a feature vector exceeding 200 elements is crafted to encapsulate the textual features significant for sentiment analysis.

The predictive model is built upon the robust foundation laid by the preprocessing stage. It calculates the tf-idf scores for the document features and averages them within the same sentiment category, creating a vector representation for positive, neutral, and negative classes. This model is trained with a dataset split into training and testing portions, and the efficiency of our model is rigorously evaluated through testing on unseen data, followed by accuracy measurements and a comprehensive contingency table.

With a meticulous blend of theoretical and practical techniques, this project aims not just to develop an NLP system but to push the boundaries of sentiment analysis, offering a granular understanding of document sentiments that can be instrumental for various real-world applications in fields ranging from market analysis to social media monitoring.

Project Description

1. Text Preprocessing Tasks and Discussions

Text preprocessing is a critical step in the pipeline of our NLP system. It involves preparing the raw text data for further analysis and processing. The preprocessing phase encompasses several tasks aimed at standardizing the text, reducing noise, and extracting meaningful features that are conducive to machine learning algorithms.

The following tasks are executed during the preprocessing phase:

- Converting all characters to lowercase to ensure that the algorithm treats words with the same characters equally, regardless of their case.
- Removing punctuation, as these characters are generally not informative for sentiment analysis and can introduce noise into the model.
- Tokenizing the text, which involves splitting the text into individual words or tokens.
- Lemmatizing the tokens, particularly verbs, to reduce them to their dictionary form, so that variations of a word are processed as a single item.

These preprocessing steps are encapsulated in a function, which is applied to our dataset before the feature extraction phase. Below is the snippet of code responsible for the text preprocessing:

```
# Convert to lowercase and remove punctuation
text = re.sub(r'[^a-zA-Z\s]', '', text.lower().strip())

# Tokenization and lemmatization
words = nltk.word_tokenize(text)
lemmatizer = nltk.stem.WordNetLemmatizer()
lemmatized_words = [lemmatizer.lemmatize(word, pos='v') for word in words]

return ' '.join(lemmatized_words)
```

Through trial and error, we determined that this combination of preprocessing techniques provides a clean and normalized text suitable for extracting meaningful patterns. Our experimentation has shown that these steps improve the accuracy of the subsequent sentiment analysis, as irrelevant variances in the text are minimized.

The preprocessing does not end with the text. The labels themselves—'positive', 'negative', and 'neutral'—are also converted into a numerical format that can be understood by the machine learning algorithms. This is a common practice in NLP tasks, where categorical data is often converted to a numerical format.

The following code snippet demonstrates how the labels are mapped to numbers:

```
label_mapping = {
    'positive': 0,
    'negative': 1,
    'neutral': 2
}
```

This mapping is essential as it transforms the label data into a format that can be easily used in the computation of model accuracy and the confusion matrix during the evaluation phase. By assigning a unique integer to each possible class label, we can perform mathematical operations on the label data and leverage the full suite of functions provided by libraries such as Scikit-learn for model training and evaluation.

2. The Creation of the Feature Vector

The transformation of textual data into a format suitable for machine learning involves the creation of a feature vector. A feature vector is a numerical representation of the text, where each dimension corresponds to a specific feature extracted from the text, such as the presence or frequency of words or n-grams.

In our project, we utilize the 'CountVectorizer' provided by the scikit-learn library to tokenize the text and create a bag-of-words model, where the vocabulary size directly influences the dimensionality of our feature vector. The following code is used to perform this operation:

```
# Vectorize documents
vectorizer = CountVectorizer(tokenizer=nltk.word_tokenize, token_pattern=None)
count_matrix = vectorizer.fit_transform(documents)

# Check if the number of features is at least 200
if len(vectorizer.get_feature_names_out()) < 200:
    raise ValueError("The number of features is less than 200.
```

```
Consider reducing text preprocessing or using a larger corpus.")
```

The dimension of the feature vector is determined by the size of the vocabulary, which is the set of unique tokens obtained after tokenizing the preprocessed text. Given the variability of natural language, this can lead to a high-dimensional space, often referred to as high-dimensional sparsity. Our model ensures that each feature vector has a dimensionality greater than 200, allowing for a nuanced representation of the text.

By transforming the raw text into a structured, numerical format, we lay the groundwork for the machine learning algorithms to discern patterns and relationships indicative of sentiment within the text. This feature vector becomes the input for our predictive model, driving the classification process.

3. The Detailed Predictive Model

Central to our NLP system is the predictive model, which utilizes the term frequency-inverse document frequency (TF-IDF) scores of the features to assess the sentiment of documents. The TF-IDF score is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

For the predictive model, we employ a methodical approach to calculate these scores and construct a vector space model where each document is represented by its TF-IDF scores across the entire vocabulary. The following snippet from our codebase outlines the computation of the TF-IDF matrix:

```
    # Compute TF and IDF
    tf = count_matrix.astype(float)
    tf.data = np.log10(tf.data + 1)

    df = np.log10(tf.shape[0] / (count_matrix.getnnz(axis=0) + 1))
    idf = np.array(df).flatten()

    # Compute TF-IDF
    tfidf = tf.multiply(idf)
    return csr_matrix(tfidf), vectorizer
```

The trained parameters of our model are essentially the TF-IDF scores for each word feature, which are used to evaluate new, unseen documents. By comparing the TF-IDF vectors of new documents to the average vectors of documents within each category, our model can predict the category to which a new document most likely belongs.

The predictive model is encapsulated in a function that takes in the preprocessed training documents and their associated labels to form the TF-IDF vectors. It also includes an averaging step where the TF-IDF vectors of all documents within the same category are averaged to create a centroid vector for that category. This centroid represents the "average" document for a category, against which new documents can be compared.

The training process of our model consists of two stages: calculating the TF-IDF scores and then averaging these scores for documents within the same category. This approach to training allows for a more nuanced understanding of each category's linguistic characteristics, leading to more accurate predictions when new documents are evaluated.

```
# Training the TF-IDF model
    tfidf_matrix, vectorizer = calculate_tfidf(train_documents)
# Preprocess labels
    train_labels = preprocess_labels(train_labels)
    test_labels = preprocess_labels(test_labels)

    # Combine training and test labels, then fit the LabelEncoder
    all_labels = np.concatenate([train_labels, test_labels])
    label_encoder = LabelEncoder()
    encoded_labels = label_encoder.fit_transform(all_labels)

    # Get indices for training labels
    train_label_indices = encoded_labels[:len(train_labels)]

    # Compute average TF-IDF vector for each category
    tfidf_means = []
    for category in np.unique(train_label_indices):
        category_indices = (train_label_indices == category)
        category_tfidf = tfidf_matrix[category_indices].mean(axis=0)
        tfidf_means.append(category_tfidf)  # Keep as sparse matrix
```

Once trained, the predictive model captures the essence of each sentiment class and becomes a powerful tool for classifying the sentiment of new documents. Its performance is a testament to the careful consideration of linguistic features and the mathematical rigor applied during its construction.

4. The Evaluation Metrics

Evaluating the efficacy of an NLP system is as crucial as the system's construction itself. For this project, we utilized a variety of metrics to assess the performance of our sentiment analysis model thoroughly. These metrics provide a comprehensive picture of the model's ability to accurately classify the sentiment of documents.

The primary metrics used were:

- **Accuracy**: The proportion of total predictions that were correct.
- **Precision**: The ratio of correctly predicted positive observations to the total predicted positives for each class.
- **Recall** (or Sensitivity): The ratio of correctly predicted positive observations to all actual positives for each class.
- **F1-Score**: The weighted harmonic mean of precision and recall. The best value is 1, and the worst is 0. It is particularly useful when the class distribution is uneven.
- **Support**: The number of actual occurrences of the class in the specified dataset.

These metrics were derived from the model's confusion matrix, which is a specific table layout that allows visualization of the performance of an algorithm. It is especially useful for seeing whether the system is confusing two classes (i.e., commonly mislabeling one as another).

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Positive	0.72	0.34	0.46	408
Negative	0.59	0.45	0.51	181
Neutral	0.78	0.89	0.83	864
Average/Total	0.74	0.68	0.68	1453

Table 1: Classification Report

The classification report generated by the scikit-learn library provides an easy-to-understand visualization of the main classification metrics. By examining these metrics, we gain insights

into not just the overall accuracy of the model, but also how well it performs for each individual sentiment class. This is particularly important in applications where the cost of misclassification varies by class, or where classes are imbalanced.

5. The Contingency Table and Discussions

A contingency table, commonly known as a confusion matrix in the context of classification tasks, is a pivotal tool for evaluating the performance of a predictive model. It provides a detailed breakdown of the model's predictions, offering insights into the true positive, false positive, false negative, and true negative predictions across the different classes.

In the context of our NLP system, the contingency table allows us to assess the model's ability to correctly classify documents into the sentiment categories of positive, negative, and neutral. It is particularly useful for visualizing the performance of our model where accuracy alone can be misleading, especially if the classes are imbalanced.

The following table displays the contingency table for our model's predictions:

	Predicted Label		
True Label	Positive	Negative	Neutral
Positive	137	54	217
Negative	18	82	81
Neutral	68	30	766

Table 2: Contingency Table for Model Predictions

Each cell in the table reveals how the documents of a true sentiment label (row) were classified as by the model (column). For instance, the cell at the intersection of the "Positive" row and "Positive" column indicates the number of documents that were correctly identified as positive (true positives), while the cell at the intersection of the "Positive" row and "Neutral" column indicates the number of positive documents that were incorrectly labeled as neutral (false negatives for the positive class).

An analysis of the contingency table can shed light on the model's strengths and weaknesses. A high number on the diagonal from top left to bottom right indicates a strong alignment between the true labels and the model's predictions, while significant numbers off the diagonal may suggest areas where the model is prone to misclassification.

From the table, we observe that the model performs exceptionally well in identifying neutral sentiments, as indicated by the high number in the "Neutral" row and "Neutral" column. Conversely, the model shows a tendency to misclassify positive documents as neutral,

suggesting a potential area of improvement, possibly through further tuning of the model or rebalancing the training data.

6. The System Accuracy Discussions

The accuracy of our NLP system is a fundamental metric, providing a clear and intuitive measure of our model's overall performance. Accuracy is defined as the proportion of true results—both true positives and true negatives—among the total number of cases examined. To compute accuracy, we use the formula:

$$Accuracy = \frac{NumberofCorrectPredictions}{TotalNumberofPredictions}$$

For a multi-class classification problem such as ours, accuracy tells us the fraction of the entire set of documents that were correctly classified.

Here is the code that was used to compute the model's accuracy:

```
# Calculate the accuracy
accuracy = np.mean([y_hat == y_true for y_hat, y_true in
                    zip(predicted_labels, true_labels)])
print(f'Accuracy: {accuracy}')
```

In our project, the system achieved an accuracy of approximately **67.79%**. This indicates that out of all documents in the test set, around two-thirds were correctly categorized by our model. While a high accuracy rate is desirable, it is important to consider it within the context of the distribution of classes within our data. If the classes are significantly imbalanced, the accuracy might be skewed, and additional metrics such as precision, recall, and the F1-score should also be considered to gain a comprehensive understanding of the model's performance.

The obtained accuracy provides us with a benchmark for the current iteration of our NLP system. As we seek to improve the system, we will aim to increase this metric by refining our preprocessing steps, feature engineering, and potentially exploring alternative modeling techniques that may yield better results. Enhancements such as feature selection, parameter optimization, and class balancing could lead to a more accurate and robust model.

Experiments and Results

In the pursuit of developing an effective NLP system for sentiment analysis, we conducted a series of experiments to evaluate our model's performance. The experiments were designed to test the robustness of our text preprocessing techniques, the suitability of our feature vector representation, and the accuracy of our predictive model.

Our experimental setup involved a split of the data into training and testing sets, with the training set used to compute the TF-IDF scores and train the model, while the testing set served to evaluate the model's predictive power. The experiments aimed to answer key questions about the preprocessing steps' impact on model performance, the optimal size of the feature vector, and the model's ability to generalize from training to unseen data.

Results Interpretation

The results of our experiments are summarized in the performance metrics previously discussed and are further illuminated by the system accuracy and contingency table.

System Accuracy: The model achieved an overall accuracy of approximately **67.79%**. This metric indicates that our model correctly predicted the sentiment of two-thirds of the documents in the testing set. This level of accuracy, while indicative of a reasonably performing model, also highlights the potential for improvement.

Contingency Table: The contingency table provided deeper insights into the model's classification behavior.

The high number of true negatives in the 'Neutral' category is encouraging, as it shows that the model is particularly adept at identifying neutral sentiments. However, the model appears to struggle with distinguishing between positive and neutral sentiments, as seen by the substantial number of positive documents misclassified as neutral.

Classification Report: The classification report revealed the model's precision, recall, and F1-scores for each sentiment category. The model showed a higher precision in identifying 'Neutral' sentiments, suggesting that when it predicts a document as 'Neutral', it is likely to be correct. However, the recall for 'Positive' sentiments was comparatively low, indicating a tendency of the model to overlook positive sentiments.

Discussion of Findings

The experiments and their corresponding results offer valuable lessons. Firstly, the effectiveness of our text preprocessing suggests that our approach can significantly filter out noise

and reduce the data to its most salient features. Secondly, the dimensionality of our feature vector appears to be sufficient, given the number of features exceeds 200, providing a detailed representation of the text data.

Nonetheless, the results also highlight areas for potential enhancement. The misclassification of positive sentiments suggests that our feature selection may be disproportionately influenced by features more commonly found in neutral or negative sentiments. Future work could explore feature selection methods to balance the representation of all sentiment categories better.

Further, the experiments underline the importance of a balanced training set. Given the discrepancy in classification performance across categories, it may be beneficial to investigate techniques such as oversampling the minority class or undersampling the majority class to achieve a more balanced distribution of training data.

In conclusion, the results from these experiments provide a solid foundation for ongoing improvement and refinement of our NLP system. The insights gained from analyzing the model's performance underscore the importance of continuous iteration and optimization in the development of machine learning models.

Further Improvements

While our NLP system has demonstrated a promising ability to classify sentiment accurately, there remains room for improvement. The iterative nature of machine learning and NLP projects means that enhancements are both expected and necessary to achieve higher levels of performance. The following are potential strategies for refining our sentiment analysis system:

- **Feature Engineering:** Expanding or pruning the feature set could lead to significant improvements. Incorporating semantic analysis, such as word embeddings, could provide a more nuanced understanding of language.
- **Algorithm Optimization:** Exploring alternative algorithms or fine-tuning the hyperparameters of the current model may yield better results. Ensemble methods or deep learning architectures like LSTM networks might capture complex patterns in the data more effectively.
- **Handling Class Imbalance:** The current model tends to misclassify positive sentiments as neutral. Techniques such as SMOTE for oversampling the minority class or

adjusting class weights in the model training process could mitigate this imbalance.

- **Data Augmentation:** Enriching the training dataset with more varied examples, especially for underrepresented classes, could improve the model's ability to generalize.
- **Cross-validation:** Employing cross-validation techniques would provide a more robust estimate of the model's performance and its expected accuracy on unseen data.
- **Error Analysis:** A deeper dive into the misclassifications could reveal specific patterns or types of text where the model is underperforming, leading to targeted improvements in data preprocessing or feature engineering.

Future work will incorporate these improvements and continuously monitor the model's performance, ensuring that each iteration brings us closer to a more accurate and reliable sentiment analysis system.