NLP HW#2

Student Name:
Mehdi Moosavi

SID:
810102264

April , 2024

Dept. of Computer Engineering

University of Tehran

Contents

1 Creating and Comparing Sentiment Analysis Representations

In this problem, our goal is to implement and compare various methods of representing text for sentiment analysis. We aim to explore the effectiveness of different techniques such as term frequency (TF), term frequency-inverse document frequency (TF-IDF), and positive pointwise mutual information (PPMI).

1.1 Preprocessing

In this section, I randomly selected 10,000 tweets, comprising 5,000 negative and 5,000 positive ones. Subsequently, I divided the tweets into 8,000 for training and 2,000 for testing. After data preparation, the next stage involved several steps as follows:

1. **Lowercasing:** The first step of preprocessing aimed to standardize the text by converting all words to lowercase. This transformation eliminates discrepancies in word representations caused by variations in capitalization. For instance, 'Good' and 'good' are treated as the same word after lowercasing, ensuring consistency in subsequent analyses. By enforcing uniformity in word casing, lowercasing enhances the accuracy and efficiency of downstream processes, such as tokenization and feature extraction.

2. **Tokenization:** Following lowercasing, the text underwent tokenization, which involved breaking down the tweets into individual words or tokens. Tokenization is essential for structuring the text data into meaningful units, enabling subsequent analyses at the word level. By segmenting the text into tokens, we create a structured representation that facilitates the extraction of features for model training and sentiment analysis. Additionally, tokenization helps capture the semantic meaning of words and their contextual relationships within sentences, thereby enriching the dataset for sentiment analysis tasks.

3. **Stemming:** The third step of preprocessing focused on stemming, a process aimed at reducing words to their root or base forms. While experimenting with different techniques, including lemmatization and a combination of stemming and lemmatization, it was found that stemming alone yielded the best results. Stemming reduces the dimensionality of the feature space by collapsing inflected or derived words to their common base form. This simplification enhances the efficiency of subsequent analyses and model training by reducing redundancy and noise in the data. By standardizing word forms, stemming improves the model's ability to generalize patterns and sentiments across different variations of words.

4. **Stopword Removal:** The final step of preprocessing involved removing words that appeared to have no significant impact on the sentiment of sentences. These words, known as stopwords, typically include common articles, prepositions, and conjunctions. By eliminating stopwords, we streamline the dataset, focusing on words with greater semantic value in determining sentiment. This refinement enhances the effectiveness of sentiment analysis by reducing noise and irrelevant information in the data, leading to more accurate and interpretable results.

1.2 Term Frequency

The term frequency (TF) method is a fundamental technique in natural language processing (NLP) used to quantify the importance of terms (words) within a document. It calculates the relative frequency of each term's occurrence in a document, incorporating a smoothing technique to address unseen words.

1.2.1 Explanation:

The TF method operates under the assumption that the more frequently a term appears in a document, the greater its significance to the overall meaning. This is because frequent terms are more likely to be directly relevant to the document's content. Consequently, the TF method assigns a weight to each term based on its frequency, with higher weights given to terms that appear more often.

For a document d, the term frequency ($TF_{t,d}$) of a term t is calculated as:

$$TF_{t,d} = \frac{n(t,d) + \text{Smoothing}}{N_d + \text{Smoothing} \cdot |V|} \tag{1}$$

where:

- $n(t, d)$: Number of times term t appears in document d
- *Smoothing*: Small constant value (e.g., 1) added to handle unseen words
- N_d: Total number of terms in document d
- $|V|$: Number of unique terms in the vocabulary (corpus)

The smoothing technique ensures all terms have a non-zero probability of occurrence, even if unseen in a specific document. It adds a small value (*Smoothing*) to both the numerator and denominator. This redistributes the probabilities, preventing zero values for unseen terms.

The TF method provides a basic but informative document representation by capturing term frequencies. The smoothing technique enhances robustness by handling unseen words. It serves as a cornerstone for more advanced NLP techniques like TF-IDF, text classification, and clustering.

1.2.2 Results

The performance of the term frequency method was evaluated using two different classifiers: Multinomial Naive Bayes (MultinomialNB) and Gaussian Naive Bayes (GaussianNB). Table 4 summarizes the accuracy, precision, recall, and F1-score achieved by each classifier using term frequencies.

Table 1: Performance Metrics using Term Frequencies

Classifier	**Accuracy**	**Precision**	**Recall**	**F1-score**
MultinomialNB	0.7585	0.7681	0.7315	0.7494
GaussianNB	0.5585	0.6256	0.2624	0.3697

As shown in Table 4, MultinomialNB demonstrates respectable performance with an accuracy of 75.85%, precision of 76.81%, recall of 73.15%, and F1-score of 74.94% when utilizing term frequencies as features. While not achieving exceptionally high scores, these results indicate a solid performance in sentiment classification tasks. MultinomialNB effectively captures the nuances of sentiment within the dataset, maintaining a good balance between precision, recall, and overall accuracy.

1.3 TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF is a popular technique in natural language processing (NLP) used to quantify the importance of terms (words) within a document relative to a collection of documents (corpus). It combines term frequency (TF), which measures how frequently a term appears in a document, with inverse document frequency (IDF), which measures the rarity of a term across multiple documents.

1.3.1 Explanation

The TF-IDF method leverages the following components:

- **Term Frequency (TF)**: For a given document, TF calculates the relative frequency of each term's occurrence. In the provided code, the `_calculate_tf` method computes TF by dividing the frequency of each term by the total number of words in the document.
- **Inverse Document Frequency (IDF)**: IDF measures the rarity of a term across multiple documents in the corpus. Terms that occur frequently in a document but are rare across the corpus are assigned higher IDF weights. In the code, the `_calculate_idf` method computes IDF by considering the logarithmically scaled ratio of the total number of documents to the number of documents containing a specific term.

- **TF-IDF Calculation**: The TF-IDF score for each term in a document is obtained by multiplying its TF value by its IDF value. This computation gives higher weight to terms that are both frequent within the document and rare across the corpus, thus capturing their significance in representing the document's content. In the provided code, the `transform` method combines the TF and IDF values for each term to generate the TF-IDF representation for a document.
- **Additive Smoothing**: The code implements additive smoothing for both TF and IDF calculations to handle unseen words. Smoothing ensures that terms not present in the vocabulary or corpus are assigned a small, non-zero value to prevent division by zero errors and improve the robustness of the TF-IDF calculation.

The TF-IDF method provides a more nuanced representation of documents by considering both the local importance of terms within individual documents (TF) and their global significance across the entire corpus (IDF). This approach helps in capturing the distinctive characteristics of each document while mitigating the influence of common terms that occur frequently across the corpus.

1.3.2 Results

The performance of the TF-IDF (Term Frequency-Inverse Document Frequency) method was evaluated using two different classifiers: Multinomial Naive Bayes (MultinomialNB) and Gaussian Naive Bayes (GaussianNB). Table 2 summarizes the accuracy, precision, recall, and F1-score achieved by each classifier when utilizing TF-IDF as features.

Table 2: Performance Metrics using TF-IDF

Classifier	**Accuracy**	**Precision**	**Recall**	**F1-score**
MultinomialNB	0.751	0.747	0.749	0.748
GaussianNB	0.555	0.613	0.267	0.372

As shown in Table 2, both MultinomialNB and GaussianNB classifiers achieved varying levels of performance when utilizing TF-IDF as features. MultinomialNB attained an accuracy of 75.1%, precision of 74.7%, recall of 74.9%, and F1-score of 74.8%. On the other hand, GaussianNB yielded an accuracy of 55.5%, precision of 61.3%, recall of 26.7%, and F1-score of 37.2%. These results demonstrate the effectiveness of TF-IDF in capturing the importance of terms within documents.

1.4 PPMI (Positive Pointwise Mutual Information)

PPMI (Positive Pointwise Mutual Information) is a method used to quantify the strength of association between terms (words) within a document. Unlike traditional co-occurrence matrices that simply count co-occurrences, PPMI considers the frequency of co-occurrences relative to the expected frequency if the terms were independent. This helps in capturing meaningful associations between terms while mitigating the influence of common co-occurrences.

1.4.1 Explanation

The PPMI method leverages the following steps:

- **Building Co-occurrence Matrix:** The first step involves constructing a co-occurrence matrix from the input documents. For each term, the co-occurrence matrix records the frequency of co-occurrences with other terms within a specified window size. This matrix captures the local context of terms within the documents.
- **Calculating PPMI:** PPMI is then calculated based on the co-occurrence matrix. It builds upon Pointwise Mutual Information (PMI), a measure of the mutual dependence between two terms. PMI captures how much the presence of one term informs us about the presence of another, compared to if they were independent. However, PMI can result in negative values, which are difficult to interpret.

  PPMI addresses this issue by focusing solely on positive co-occurrences, which represent stronger relationships between terms. It computes PPMI by comparing observed co-occurrences to what would be expected by chance, without smoothing. This emphasizes rare co-occurrences and plays down common ones, highlighting meaningful term associations.

- **Transformation:** Lastly, the PPMI matrix is converted into PPMI vectors for each document. These vectors indicate how strongly each term in the document relates to the entire vocabulary. Higher PPMI values signal stronger term associations within the document.

1.4.2 Results

The effectiveness of the PPMI method as before was evaluated using two different classifiers: Multinomial Naive Bayes (MultinomialNB) and Gaussian Naive Bayes (GaussianNB). Table 3 provides an overview of the accuracy, precision, recall, and F1-score achieved by each classifier utilizing the PPMI method.

Table 3: Performance Metrics using PPMI Method

Classifier	**Accuracy**	**Precision**	**Recall**	**F1-score**
MultinomialNB	0.639	0.642	0.608	0.624
GaussianNB	0.653	0.689	0.541	0.606

As illustrated in Table 3, both MultinomialNB and GaussianNB classifiers exhibit competitive performance with the PPMI method. MultinomialNB achieved an accuracy of 63.9%, with precision, recall, and F1-score of 64.2%, 60.8%, and 62.4%, respectively. On the other hand, GaussianNB attained an accuracy of 65.3%, with precision, recall, and F1-score of 68.9%, 54.1%, and 60.6%, respectively.

These results suggest that both classifiers effectively leverage the PPMI method for sentiment analysis, demonstrating comparable performance to the term frequency method. GaussianNB shows a slightly higher accuracy and precision, while MultinomialNB exhibits slightly better recall and F1-score.

1.5 Analysis of Results

The performance of three different text representation methods, namely Term Frequencies (TF), Term Frequency-Inverse Document Frequency (TF-IDF), and Positive Pointwise Mutual Information (PPMI), was evaluated on a small dataset containing 10,000 rows, with 2,000 rows used for testing.

As shown in Tables 4 and 3, the TF method achieved the highest performance metrics across all categories, with an accuracy of 75.85%, precision of 76.81%, recall of 73.15%, and F1-score of 74.94%. The TF-IDF method showed slightly lower performance, with an accuracy of 75.1%, precision of 74.72%, recall of 74.87%, and F1-score of 74.80%.

The PPMI method, on the other hand, showed lower performance metrics, with an accuracy of 63.9%, precision of 64.2%, recall of 60.8%, and F1-score of 62.4%. This could be due to the fact that PPMI, unlike TF and TF-IDF, takes into account the co-occurrence of words, which might not be as effective for this particular dataset.

Given the small size of the dataset, these results should be interpreted with caution. The TF and TF-IDF methods might have performed better due to their simplicity and the fact that they rely solely on term frequencies, which could be more reliable in a smaller dataset. On the other hand, the PPMI method might perform better on a larger and more diverse dataset, where word co-occurrences are more varied and informative.

2 GLOVEs for Sarcasm Detection

In the second problem our objective is to employ GloVe embeddings coupled with logistic regression for sarcasm detection in textual data. Sarcasm poses a nuanced challenge, demanding sophisticated representation techniques. By leveraging GloVe embeddings and logistic regression, we aim to capture the intricate semantic nuances indicative of sarcasm.

2.1 Preprocessing

In this phase, several steps are performed to clean and structure the text data:

- **Lowercasing**: All text is converted to lowercase to ensure uniformity and prevent redundancy in the vocabulary. This step helps in treating words with different cases as identical.
- **Tokenization**: Text is split into individual tokens, typically words or phrases. This step facilitates further processing by breaking down the text into meaningful units.
- **Removing punctuation**: Punctuation marks such as commas, periods, and exclamation marks are removed from the text. This helps in focusing on the actual content of the text and eliminates noise.
- **Removing stopwords**: Stopwords, which are common words that do not carry significant meaning (e.g., "is", "the", "and"), are removed from the text. This step reduces the dimensionality of the data and removes irrelevant information.
- **Lemmatization**: Words are reduced to their base or root form. This process helps in standardizing words with similar meanings and reduces the complexity of the vocabulary.

After preprocessing, the data is split into features (X) and target (y) variables, where X represents the preprocessed text headlines, and y indicates whether each headline is sarcastic or not. Finally, the data is divided into training and testing sets using an 80/20 ratio to facilitate model training and evaluation.

2.2 Word Representations using GloVe Embeddings

To represent words in the text data using pre-trained GloVe embeddings, the following steps are performed:

1. **Loading GloVe Vectors**: Pre-trained GloVe vectors are loaded from a file containing word embeddings. Each word in the vocabulary is associated with a 300-dimensional vector representation capturing its semantic meaning.
2. **Creating Headline Representations**: For each headline in the dataset, a fixed-length representation is created by averaging the GloVe embeddings of the constituent words. This involves iterating over each word in the headline, adding its embedding vector to the headline representation if available, and then dividing the resulting vector by the number of words to obtain the average embedding.
3. **Applying to Training and Testing Data**: The headline representations are created for both the training and testing datasets using the GloVe embeddings.

By utilizing GloVe embeddings, each headline is transformed into a dense 300-dimensional numerical vector representation capturing its semantic content. These representations serve as input features for training the logistic regression model for sarcasm detection.

2.3 Training Model and Evaluation

The logistic regression model, a popular classification algorithm, is employed for sarcasm detection using GloVe embeddings. In this approach, the model is initialized with a maximum of 500 iterations and trained on the training data using headline representations derived from GloVe embeddings. Subsequently, predictions are made on the test set using the trained model, and performance metrics including accuracy, precision, recall, and F1-score are computed to evaluate the model's effectiveness in sarcasm detection.

2.4 Results

Table 4 summarizes the performance metrics obtained from training the logistic regression model on GloVe embeddings.

Metric	**Value**
Accuracy	0.740
Precision	0.724
Recall	0.723
F1-score	0.724

Table 4: Performance Metrics of the Logistic Regression Model

These results indicate that the logistic regression model trained on GloVe embeddings achieves competitive performance in sarcasm detection.

3 Creating Vector Semantics Using Skipgram

In Natural Language Processing (NLP), capturing the semantic relationships between words is crucial for various tasks like machine translation, sentiment analysis, and information retrieval. One effective technique for achieving this goal is through word embeddings, which represent words as vectors in a high-dimensional space. This section explores the Skipgram model, a widely used approach for generating word embeddings. In this section, we will implement the Skipgram model to create word embeddings and evaluate their effectiveness in capturing semantic relationships.

3.1 Preprocessing

The textual data was prepared for the Skipgram model by first reading the content of the given file, tokenizing the text into individual words, and converting all letters to lowercase. Subsequently, a word-to-index mapping (word2id) and its inverse (id2word) were created. Vocabulary size was determined and the embedding size was then set to 100 to specify the dimensionality of word vectors. Finally, each document in the corpus was converted into a sequence of word IDs. These sequential steps ensure that the data is appropriately formatted and ready for utilization in the Skipgram model.

Creating pairs for Skipgram involves selecting a target word and its context words within a specified window size (`window_size`), essentially training the model to predict surrounding words given a central word. To improve computational efficiency and address the issue of frequent words dominating the training data (imbalanced samples), negative sampling is employed. This technique involves randomly selecting "negative" context words that do not co-occur with the target word within the window. This process helps the model learn to distinguish between true context words and irrelevant words, improving the overall quality of the learned embeddings. The parameters used here are `window_size=2` and `negative_samples=4`, controlling the size of the context window and the number of negative samples generated for each target word, respectively.

3.2 Model and Training

The Skipgram model, a popular architecture in word embedding techniques, is utilized to predict context words given a target word. It employs two input layers for word and context indices, followed by embedding layers to convert indices into dense vectors. The model utilizes batch normalization for stabilization and the dot product operation to calculate word-context similarities. Training involves binary cross-entropy loss optimization using the Adam optimizer with an EMA momentum of 0.95. The architecture is depicted in Figure 1.

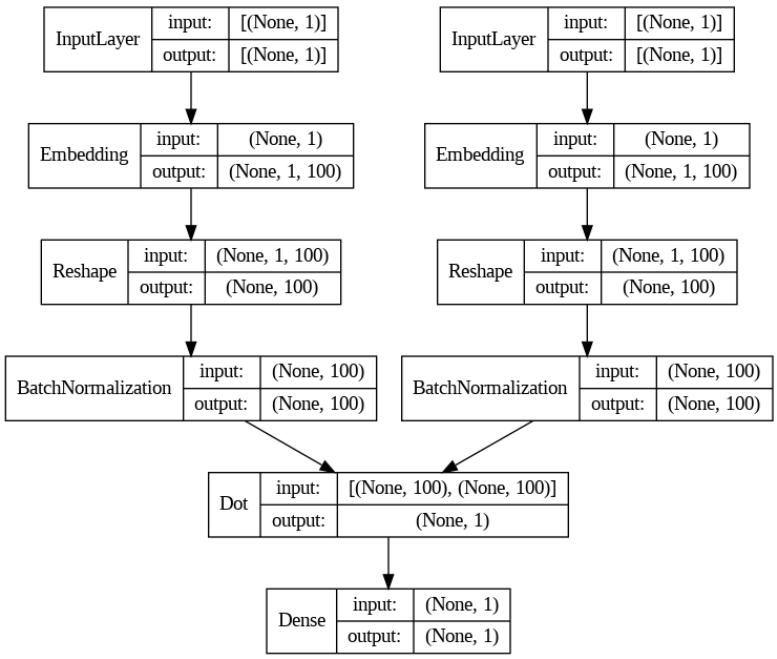

Figure 1: Architecture of the Skipgram Model

The training loop for the Skipgram model involves several key steps:

- **Dataset Preparation**: We set the batch size to 256, determining the number of training examples processed in each iteration. We then create a TensorFlow Dataset from the pairs of word indices and their corresponding labels, indicating whether the context word is a true or negative sample. This dataset is shuffled and batched for efficient training. Finally, we define the number of training epochs as 10, specifying how many times the entire dataset will be iterated over during training.
- **Training Loop**: Within each epoch, we iterate over batches of training examples. For each batch:
  - Separate the target and context word indices.
  - Define the input and output for training.
  - Perform a single update (training step) using the current batch of data and the binary cross-entropy loss function.
  - Accumulate the loss over all batches in the epoch.

  This training loop allows the Skipgram model to iteratively learn and update its parameters using batches of training examples, gradually improving its ability to predict context words given a target word. Through this iterative process, the Skipgram model progressively refines its word embeddings, capturing the semantic relationships between words in the text corpus.

3.3 Results

In this section, we assess the quality of the learned word embeddings by evaluating their ability to capture semantic relationships between words. We performed the following steps:

- We obtained the vector representations for the words "king," "man," and "woman" from the learned embeddings.
- We applied a semantic operation by subtracting the vector representation of "man" from "king" and adding the vector representation of "woman." This operation, known as word analogy, aims to infer a word that might be analogous to "king" in a similar context but with a gender-neutral connotation (e.g., monarch).
- We computed the similarity (using cosine similarity) between the resulting vector and the vector representation of the word "queen," which is expected to be semantically related to the inferred word.

The computed cosine similarity score of 0.77 provides valuable insights into the effectiveness of the learned word embeddings in capturing semantic relationships within the text corpus. A cosine similarity score of 0.77 is considered relatively high in this context, where scores typically range between 0 (no similarity) and 1 (perfect similarity). This indicates that the embeddings successfully capture the analogous relationship between "king" and "queen." Such results demonstrate the potential of these embeddings for various downstream NLP tasks, such as word sense disambiguation or analogy completion.

In continuation of our evaluation, we further explored the semantic relationships captured by the learned word embeddings through visualization in a 2D space. We obtained the vector representations for specific words of interest, including "brother," "sister," "uncle," and "aunt," from the learned embeddings. Principal Component Analysis (PCA) was applied to reduce the dimensionality of the embeddings to 2 dimensions, facilitating visualization while preserving variance. The 2D embeddings for these words were extracted and plotted in a scatter plot, allowing us to visualize their positions in the reduced space. As expected, the visualization shows that the embeddings capture semantic similarities between related words, with "brother" and "sister" positioned close together, and "uncle" and "aunt" appearing nearby. Lines were drawn between these related word pairs to illustrate these semantic relationships. The visualization of 2D embeddings is shown in Figure 2.

This visualization offers valuable insights into the semantic relationships captured by the learned word embeddings. The parallel lines connecting related word pairs (e.g., "brother" and "sister," "uncle" and "aunt") highlight consistent semantic associations between these words. This consistency strengthens the assessment of the embeddings' quality and effectiveness in capturing semantic nuances and relationships within the text corpus.

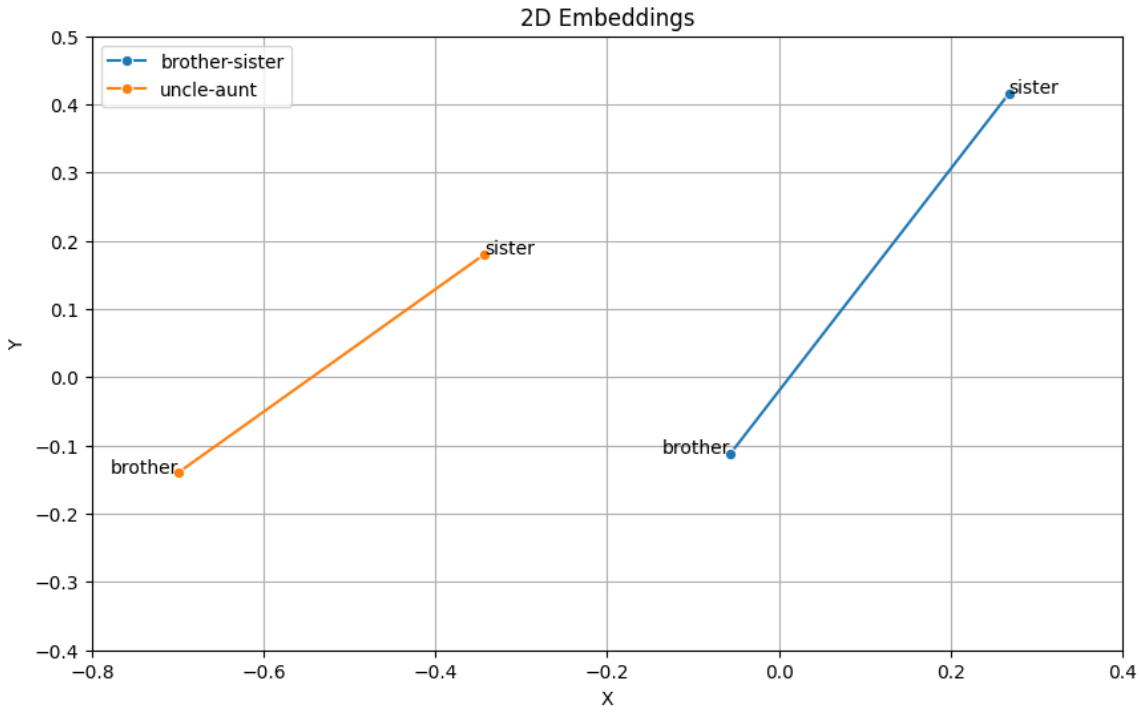

Figure 2: Visualization of 2D Embeddings