Information Retrieval course - Mini Project 5 Report

Kimia Esmaili - 610398193

(More detail is commented in the code.)

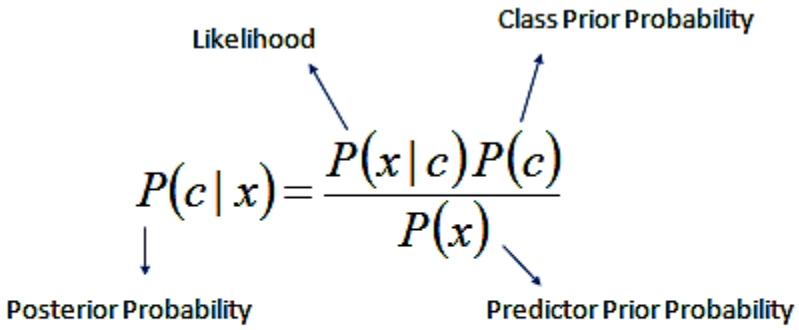
Document Preprocessing and Naive Bayes Classification:

Code Explanation:

- We first import the necessary libraries and read the training dataset using pandas.
- We then preprocess the training documents by tokenizing, removing punctuation, converting to lowercase, removing stopwords, and performing stemming using NLTK.
- We convert the preprocessed text into a word list using CountVectorizer.
- We implement the Naive Bayes classifier using MultinomialNB from scikit-learn and train it on the preprocessed training dataset.
- Finally, we evaluate the classifier's performance on the training dataset by calculating the accuracy.

Methods and Formulas:

Naive Bayes Classification:



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Naive Bayes is a classification technique that is based on Bayes' Theorem with an assumption that all the features that predicts the target value are independent of each other. As shown in the formula above, it calculates the probability of each class and then pick the one with the highest probability. It has been successfully used for many purposes, but it works particularly well with natural language processing (NLP) problems. Bayes' Theorem describes the probability of an event, based on a prior knowledge of conditions that might be related to that event.

Naive Bayes classifier assumes that the features we use to predict the target are independent and do not affect each other. While in real-life data, features depend on each other in determining the target, but this is ignored by the Naive Bayes classifier. Though the independence assumption is never

correct in real-world data, but often works well in practice. so that it is called “Naive”.

Key Findings:

- Preprocessing the text data is crucial for text classification tasks as it helps in removing noise and irrelevant information.
- Naive Bayes is a simple and efficient algorithm for text classification tasks, especially when dealing with a large number of features.
- The accuracy of the Naive Bayes classifier on the training dataset can serve as a baseline for evaluating more advanced techniques in the next steps of the project.

Challenges Faced:

- Balancing the trade-off between removing too much information during preprocessing and retaining important features for classification.
- Understanding the inner workings of the Naive Bayes algorithm and how it handles text data.

Insights Gained:

- The importance of feature extraction and preprocessing in text classification tasks.
- The effectiveness of Naive Bayes as a baseline classifier for text classification.

- The need for further experimentation with advanced techniques like word embeddings and Latent Semantic Analysis for improving classification performance.

Word Embeddings with SVM Classification and LSA with SVM Classification:

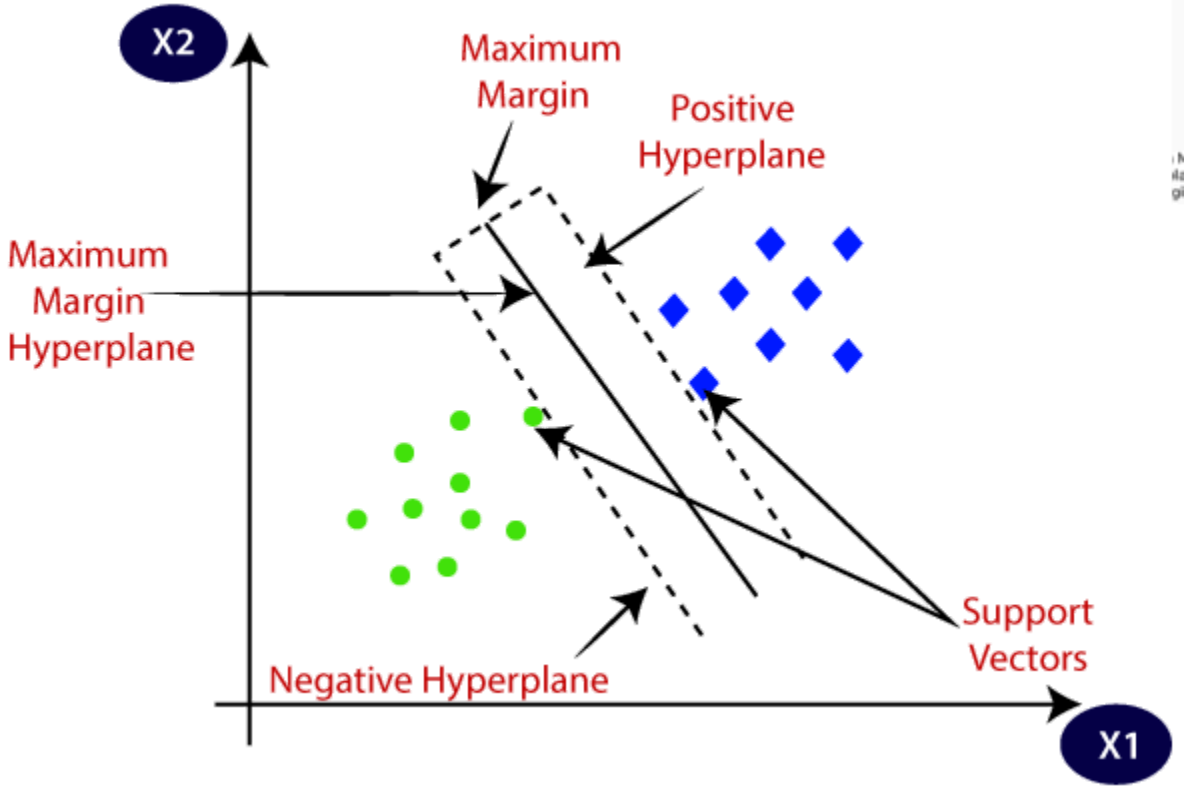
Code Explanation:

- We use Gensim to apply Word2Vec for word embeddings, representing words in a continuous vector space.
- We define a function to transform word embeddings into document embeddings by summing the word vectors.
- We train an SVM classifier using the document embeddings as input features.
- We apply Latent Semantic Analysis (LSA) to the documents to capture latent semantic structures.
- We use the LSA-transformed word embedding vectors as input features for training another SVM classifier.

Methods and Formulas:

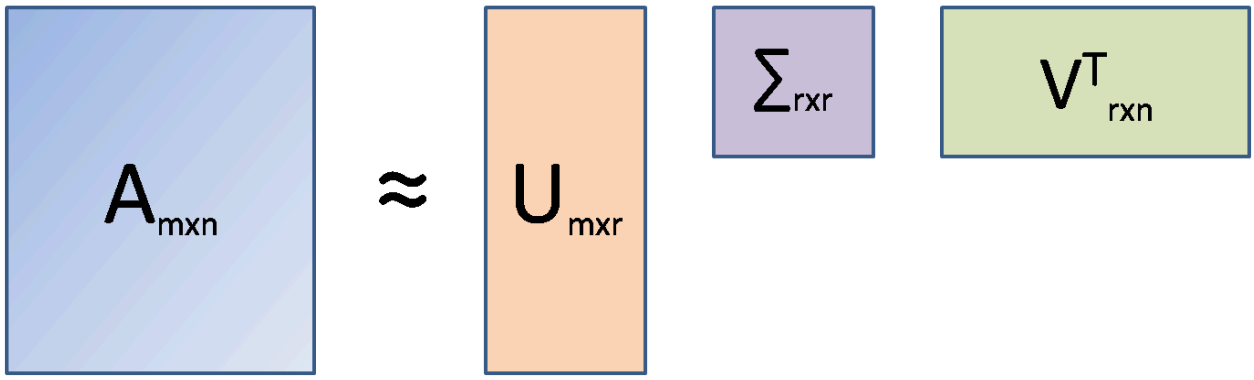
- **Word2Vec:** The Word2Vec model learns distributed representations of words in a continuous vector space by predicting words based on their context. We will delve into this model more specifically in the next part.

- **SVM:**



A Support Vector Machine (SVM) is a powerful machine learning algorithm used primarily for classification and regression tasks. It's especially effective in scenarios where data is not linearly separable and needs to be transformed into a higher-dimensional space for separation. They find the hyperplane that best separates the classes in the feature space. SVMs are a type of supervised learning algorithm, meaning they require labeled data for training.

- **LSA:** Latent Semantic Analysis is a technique that reduces the dimensionality of the document-term matrix to capture latent semantic structures. It involves creating structured data from a collection of unstructured texts. Before getting into the concept of LSA, let us have a quick intuitive understanding of the concept. When we write anything like text, the words are not chosen randomly from a vocabulary.
- Rather, we think about a theme (or topic) and then chose words such that we can express our thoughts to others in a more meaningful way. This theme or topic is usually considered as a latent dimension.
- It is latent because we can't see the dimension explicitly. Rather, we understand it only after going through the text. This means that most of the words are semantically linked to other words to express a theme. So, if words are occurring in a collection of documents with varying frequencies, it should indicate how different people try to express themselves using different words and different topics or themes.
- In other words, word frequencies in different documents play a key role in extracting the latent topics. LSA tries to extract the dimensions using a machine learning algorithm called Singular Value Decomposition or SVD.



- Here, A is the document-term matrix (documents in the rows(m), unique words in the columns(n), and frequencies at the intersections of documents and words). It is to be kept in mind that in LSA, the original document-term matrix is approximated by way of multiplying three other matrices, i.e., U, ∑ and V^T. Here, r is the number of aspects or topics. Once we fix r (r<<n) and run SVD, the outcome that comes out is called Truncated SVD and LSA is essentially a truncated SVD only.
- SVD is used in such situations because, unlike PCA, SVD does not require a correlation matrix or a covariance matrix to decompose. In that sense, SVD is free from any normality assumption of data (covariance calculation assumes a normal distribution of data). The U matrix is the document-aspect matrix, V is the word-aspect matrix, and ∑ is the diagonal matrix of the singular values. Similar to PCA, SVD also combines columns of the original matrix linearly to arrive at the U matrix. To arrive at the V matrix, SVD combines the rows of the original matrix linearly. Thus, from a sparse document-term matrix, it is possible to get a dense document-aspect matrix that can be used for either document

clustering or document classification using available ML tools. The V matrix, on the other hand, is the word embedding matrix (i.e. each and every word is expressed by r floating-point numbers) and this matrix can be used in other sequential modeling tasks. However, for such tasks, Word2Vec and Glove vectors are available which are more popular.

Key Findings:

- Word embeddings like Word2Vec can capture semantic relationships between words, improving the representation of documents.
- SVMs are effective classifiers for text classification tasks when combined with word embeddings or LSA.
- LSA can help in capturing the underlying structure of the text data for better classification performance.
-

Challenges Faced:

- Choosing the right hyperparameters for Word2Vec and SVM models.
- Ensuring that the transformation of word embeddings into document embeddings captures the semantic meaning effectively.

Insights Gained:

- Different techniques like Word2Vec, LSA, and SVM can be combined to enhance text classification performance.

- The importance of experimenting with various methods to find the best approach for a specific task.
- The trade-offs between model complexity, feature representation, and classification accuracy in information retrieval and text classification projects.

Using all Word Embeddings(Optional):

Code Explanation:

- We load pre-trained GloVe and FastText embeddings into memory using Gensim.
- We define functions to transform the GloVe and FastText word embeddings into document embeddings by summing the word vectors.
- We train SVM classifiers using the document embeddings derived from GloVe and FastText embeddings.
- We evaluate the classifiers' performance on the training dataset.

Methods and Formulas:

Word2Vec: is a popular word embedding technique that aims to represent words as continuous vectors in a high-dimensional space. It can use these two models: Continuous Bag of Words (CBOW) and Skip-gram, each contributing to the learning of vector representations.

1. Model Architecture:

Continuous Bag of Words (CBOW): In CBOW, the model predicts a target word based on its context. The context words are used as input, and the target word is the output. The model is trained to minimize the difference between the predicted and actual target words.
Skip-gram: Conversely, the Skip-gram model predicts context words given a target word. The target word serves as input, and the model aims to predict the words that are likely to appear in its context. Like CBOW, the goal is to minimize the difference between the predicted and actual context words.

2. Vector Representations:
Once trained, Word2Vec assigns each word a unique vector in the high-dimensional space. These vectors capture semantic relationships between words. Words with similar meanings or those that often appear in similar contexts have vectors that are close to each other, indicating their semantic similarity.

3. Advantages:

- Captures semantic relationships effectively.
- Efficient for large datasets.
- Provides meaningful word representations.

4. Disadvantages:

- May struggle with rare words.

- Ignores word order.

GloVe (Global Vectors for Word Representation): Global Vectors for Word Representation (GloVe) is a powerful word embedding technique that captures the semantic relationships between words by considering their co-occurrence probabilities within a corpus. The key to GloVe's effectiveness lies in the construction of a word-context matrix and the subsequent factorization process.

1. Word-Context Matrix Formation:

The first step in GloVe's mechanics involves creating a word-context matrix. This matrix is designed to represent the likelihood of a given word appearing near another across the entire corpus. Each cell in the matrix holds the co-occurrence count of how often words appear together in a certain context window.

Let's consider a simplified example. Assume we have the following sentences in our corpus:

"Word embeddings capture semantic meanings."
"GloVe is an impactful word embedding model."

Each row and column in our matrix corresponds to a unique word in the corpus, and the values in the cells represent how often these words appear together within a certain context window.

2. Factorization for Word Vectors:

With the word-context matrix in place, GloVe turns to matrix factorization. The objective here is to decompose this high-dimensional matrix into two smaller matrices — one representing words and the other contexts. Let's denote these as W for words and C for contexts. The ideal scenario is when the dot product of W and CT (transpose of C) approximates the original matrix:

$$X \approx W \cdot CT$$

Through iterative optimization, GloVe adjusts W and C to minimize the difference between X and $W \cdot CT$. This process yields refined vector representations for each word, capturing the nuances of their co-occurrence patterns.

3. Vector Representations:

Once trained, GloVe provides each word with a dense vector that captures not just local context but global word usage patterns. These vectors encode semantic and syntactic information, revealing similarities and differences between words based on their overall usage in the corpus.

4. Advantages:

- Efficiently captures global statistics of the corpus.

- Good at representing both semantic and syntactic relationships.
- Effective in capturing word analogies.

5. Disadvantages:

- Requires more memory for storing co-occurrence matrices.
- Less effective with very small corpora.

FastText: FastText is an advanced word embedding technique developed by Facebook AI Research (FAIR) that extends the Word2Vec model. Unlike Word2Vec, FastText not only considers whole words but also incorporates subword information — parts of words like n-grams. This approach enables the handling of morphologically rich languages and captures information about word structure more effectively.

1. Subword Information:

FastText represents each word as a bag of character n-grams in addition to the whole word itself. This means that the word “apple” is represented by the word itself and its constituent n-grams like “ap”, “pp”, “pl”, “le”, etc. This approach helps capture the meanings of shorter words and affords a better understanding of suffixes and prefixes.

2. Model Training:

Similar to Word2Vec, FastText can use either the CBOW or Skip-gram architecture. However, it incorporates the subword

information during training. The neural network in FastText is trained to predict words (in CBOW) or context (in Skip-gram) not just based on the target words but also based on these n-grams.

3. Handling Rare and Unknown Words:

A significant advantage of FastText is its ability to generate better word representations for rare words or even words not seen during training. By breaking down words into n-grams, FastText can construct meaningful representations for these words based on their subword units.

4. Advantages:

Better representation of rare words.
Capable of handling out-of-vocabulary words.
Richer word representations due to subword information.

5. Disadvantages:

Increased model size due to n-gram information.
Longer training times compared to Word2Vec.

Choosing the Right Embedding Model

- Word2Vec: Use when semantic relationships are crucial, and you have a large dataset.
- GloVe: Suitable for diverse datasets and when capturing global context is important.
- FastText: Opt for morphologically rich languages or when handling out-of-vocabulary words is vital.

Key Findings:

- Different word embedding techniques like Word2Vec, GloVe, and FastText can have varying impacts on text classification performance.
- SVM classifiers can effectively utilize the embeddings derived from different techniques for classification tasks.
- The choice of word embedding technique can significantly influence the classification accuracy.

Challenges Faced:

- Handling and processing large pre-trained word embedding models like GloVe and FastText efficiently.
- Ensuring that the transformation of word embeddings into document embeddings captures the semantic information effectively for classification.

Insights Gained:

- The importance of exploring and comparing different word embedding techniques for text classification tasks.
- The need to analyze the performance of classifiers using different word embeddings to understand their impact on the overall classification accuracy.
- The potential trade-offs between computational complexity and classification performance when using various word embedding techniques in information retrieval and text classification projects.

Evaluation on Test Dataset:

Code Explanation:

- We transform the test data into document embeddings using Word2Vec, GloVe, and FastText.
- We evaluate the trained Naive Bayes classifier on the test dataset and calculate the accuracy.
- We evaluate the SVM classifiers trained with Word2Vec, GloVe, and FastText embeddings on the test dataset and calculate the accuracy.
- We generate a classification report for each SVM classifier, including metrics like precision, recall, and F1-score.

Now we will look into NBC and SVM again:

Naïve Bayes (NB) Classifier: allows constructing simple classifiers based on Bayes' theorem. Thus, it assumes that any feature value is independent of the value of the other features. NB models can accomplish high levels of accuracy while estimating the class-conditional marginal densities of data.

Because of the independence assumption, NB doesn't need to learn all possible correlations between the features. If N is the number of features, then a general algorithm requires to analyze 2N possible feature interactions, while NB only needs the order of N data points. Thus, NB classifiers can learn easier from small training data sets due to the class independence assumption. At the same time, NB is not affected by the curse of dimensionality.

The runtime complexity of the Naïve Bayes classifier is O(NK), where N is the number of features and K is the number of label classes.

Support Vector Machine (SVM): is a very popular model. SVM applies a geometric interpretation of the data. By default, it is a binary classifier. It maps the data points in space to maximize the distance between the two categories.
For SVM, data points are N-dimensional vectors, and the method looks for an N-1 dimensional hyperplane to separate them. This is called a linear classifier. Many hyperplanes could satisfy this condition. Thus, the best hyperplane is the one that gives the largest margin, or distance, between the two categories. Thus, it is called the maximum margin hyperplane.
we can see a set of points corresponding to two categories, blue and green. The red line indicates the maximum margin hyperplane that separates both groups of points. Those points over the dashed line are called the support vectors.
Frequently happens that the sets are not linearly separable in the original space. Therefore, the original space is mapped into a higher-dimensional space where the separation could be obtained. SVMs can efficiently perform a non-linear classification using the so-called kernel trick. The kernel trick consists of using specific kernel functions, which simplify the mapping between the original space into a higher-dimensional space.

How Do Both Methods Compare?

Naïve Bayes (NB) is a very fast method. It depends on conditional probabilities, which are easy to implement and evaluate. Therefore, it does not require an iterative process. NB supports binary classification as well as multinomial one. NB assumes that features are independent between them, but this assumption does not always hold. Even though, NB gives good results when applied to short texts like tweets. For some datasets, NB may defeat other classifiers using feature selection.

SVM is more powerful to address non-linear classification tasks. SVM generalizes well in high dimensional spaces like those corresponding to texts. It is effective with more dimensions than samples. It works well when classes are well separated. SVM is a binary model in its conception, although it could be applied to classifying multiple classes with very good results.
The training cost of SVM for large datasets is a handicap. SVM takes a long time while train large datasets. It requires hyperparameter tuning which is not trivial and takes time. SVM is more attractive theoretically.

Both NB and SVM allow the choice of kernel function for each and are sensitive to parameter optimization.
Comparing the accuracy of SVM and NB in spam classification showed that the basic NB algorithm gave the best prediction results (97.8%). At the same time, SVM and NB algorithms

obtained an accuracy well above 90% using parameter tuning when required.

Key Findings:

- Naive Bayes and SVM classifiers can yield different performance on the test dataset, with SVM often providing higher accuracy.
- Word2Vec, GloVe, and FastText embeddings can have varying impacts on classification performance, with one technique potentially outperforming the others.
- The classification report provides detailed insights into the precision, recall, and F1-score for each classifier and embedding technique.
- The algorithm of NBC is very simple since it only use text frequency to compute the posterior probability for each classes. While SVM algorithm is more complex than NBC. SVM develop hyperplane equation which separate data into classes perfectly.

Challenges Faced:

- Ensuring the consistency of preprocessing steps between training and test data for accurate evaluation.
- Managing the computational resources required to process and evaluate multiple classifiers and embedding techniques on the test dataset.

Insights Gained:

- The importance of evaluating classifiers on a separate test dataset to assess their generalization performance.
- The need to compare the performance of different classifiers and embedding techniques to identify the most effective approach for text classification tasks.
- The impact of word embeddings on classification metrics like accuracy, precision, recall, and F1-score, highlighting the importance of selecting the right embedding technique for the task at hand.