Assignment - Designing a Spam Filter

Machine Learning & Data Science II (BA-MECH-22-4A)

BSc - Mechatronics, Design & Innovation

5th Semester

Lector: PhD Daniel McGuiness

Class: BA-MECH-22

Author: Maximilian Nachbaur

December 27, 2024

Contents

1 Introduction

The goal of this project was to implement an email spam filter in Python, using the methods learned during the lecture. As part of the project, a pipeline was developed to process the data, a vectorization method was implemented to convert the emails into vectors and a word frequency analysis was incorporated to further prepare the data for the model.

Several algorithms, including Naive Bayes, Logistic Regression, Random Forest, Support Vector Machines, and Stochastic Gradient Descent, were tested and compared. Finally, a confusion matrix was created for each model to illustrate the results.

This report outlines the key steps of data processing, modeling and evaluation of the spam filter models and explains why the used methods were chosen.

2 Libraries

For the implementation of the spam filter project, various Python libraries were used, providing essential functions for data processing, machine learning, and visualization. In the following a brief description of the libraries is given:

- **os**: Allows interaction with the operating system and is used to read the filepath of the emails. It allows access to file paths, navigation through directories, and reading files.
- **re**: Is used for working with regular expressions and is applied in the preprocessing of emails. It enables the removal of unwanted elements such as URLs, email addresses, numbers and special characters.
- **pandas**: Is used for data manipulation and analysis. In the project, it is used to present the results of the word frequency analysis in a clear tabular format.
- **scikit-learn**: Is the core library for machine learning in this project. It is used for vectorization, modeling and evaluation, as well as for shuffling and splitting the data.
- **matplotlib & seaborn**: Libraries for data visualization that are used to graphically represent the results.
- **math**: Is used for mathematical operations.

3 Implementation

This chapter provides a detailed description of the implementation of the spam filter. The individual functions and sections of the code are explained, including the chosen methods and the reasons for their use. The goal is to clearly the project's architecture and justify the methodological decisions. Since the code is well documented in the python script itself, a detailed explanation of all functions and commands is not the main focus here in the report.

3.1 LOADING AND PREPARING THE DATA

read_email_from_folder

This function reads all emails from a specified folder and stores them in a list. Accessing the raw data in the various directories is the first step in feeding the data into the system. The use of the os library enables efficient handling of files and directories, which is the main reason for choosing this library.

preprocess_emails

The function cleans the content of the emails by removing unwanted elements such as URLs, email addresses, numbers, and special characters. Additionally, the text is converted to lowercase. To train the models with consistent and relevant data, the re library was used to remove unnecessary information.

process_emails

This function combines the two steps mentioned above by reading and preprocessing emails directly. Merging these steps reduces code complexity and ensures efficient data handling.

3.2 VECTORIZING THE DATA

vectorizing_emails

The TfidfVectorizer is used to transform emails into numerical vectors. Each email is represented as a combination of words, with the relevance of each word determined by its Tf-idf values.

```python
    # Vectorizes the emails using the TfidfVectorizer
    vectorizer = TfidfVectorizer(
        max_features=10000, max_df=0.5, min_df=1, stop_words="english")
    vectors = vectorizer.fit_transform(emails)
```

Listing 1: Snippet of vectorizer function

The parameters refine the vectorization by limiting features to the top 10,000 (max_features=10000), excluding overly common terms (max_df=0.5), including terms present in at least one document (min_df=1), and removing common English stopwords (stop_words="english"). This ensures the matrix focuses on meaningful, relevant terms 1.

Another method could have been the CountVectorizer. However, the TfidfVectorizer was chosen because it produced better results by emphasizing rare but relevant words while down-weighting frequently occurring, less informative words.

3.3 ANALYZING THE WORD FREQUENCY

analyze_word_frequencies

The function calculates the frequencies of words in spam and ham emails and generates a sorted list based on the difference in frequencies. Analyzing word frequencies helps identify the key features (relevant words) that contribute to distinguish between spam and ham. This enhances the overall efficiency of the model.

```
# Analyzes the word frequencies in the emails to identify the most common words
word_frequencies = analyze_word_frequencies(vectors, all_labels, vocab)

# Words with the highest difference in frequency between spam and ham emails
word_frequencies["Difference"] = abs(
    word_frequencies["Spam"] - word_frequencies["Ham"])
relevant_words = word_frequencies.sort_values(
    by="Difference", ascending=False).index[:10000]
```

Listing 2: Snippet of word frequency analysis section

The code in 2 analyzes word frequencies in emails to identify the most distinguishing words between spam and ham. The words are then sorted by their frequency difference in descending order, and the top 10,000 most relevant words are selected for further use. The number of relevant words should not be set too low since this would affect the performance of the model.

3.4 SHUFFLING AND SPLITTING THE DATA

The vectors and labels are randomly shuffled before being split into training and test data. Shuffling ensures that no systematic bias arises from the order of the data. Splitting into training and test sets ensures that the models are evaluated fairly and do not become overfitted.

3.5 MODELS, TRAINING & EVALUATION

train_and_evaluate_model

This function trains the models on the training data and evaluates them on the test data using metrics such as accuracy, precision, recall, and F1-score. The use of multiple evaluation metrics provides a comprehensive assessment of model performance. This process is encapsulated in a function to test and evaluate multiple models simultaneously.

plot_confusion_matrix

The confusion matrix is graphically displayed for each model to visualize the distribution of classifications (True Positives, True Negatives, False Positives, False Negatives). Visualizing the confusion matrix for each model provides much deeper insights into the model's performance.

Models

```python
# Defines the models to train and evaluate
models = [
    ("Naive Bayes", MultinomialNB(alpha=0.001)),
    ("Logistic Regression", LogisticRegression(
        max_iter=1000, class_weight="balanced", random_state=42)),
    ("Random Forest", RandomForestClassifier(
        n_estimators=100, class_weight="balanced", random_state=42)),
    ("Linear SVM", LinearSVC(C=1, class_weight="balanced", random_state=42)),
    ("SGD Classifier", SGDClassifier(alpha=0.001,
     class_weight="balanced", random_state=42))
]
```

Listing 3: Snippet of training and evaluation section

Naive Bayes: A probabilistic approach based on conditional probability, commonly used for text classification. The alpha parameter in MultinomialNB controls Laplace smoothing. It prevents probabilities for words that do not appear in a class from becoming zero by adding a small value to each word's frequency. In this case, alpha=0.001 ensures minimal smoothing, which is reasonable because sufficient training data is available.

Logistic Regression: A robust linear model well-suited for binary classification tasks. max_iter=1000 ensures that the optimization algorithm has sufficient iterations to converge, especially for complex or large datasets, preventing too early termination of training. class_weight="balanced" compensates for imbalanced class distributions (e.g., significantly more ham than spam) to prevent the model from favoring the majority class, to ensure fair classification. random_state=42 guarantees consistent results across runs, supporting reproducibility.

Random Forest: An ensemble learning model that combines multiple decision trees, enhancing robustness and accuracy. n_estimators=100 is the number of decision trees in the forest. More trees can improve model performance but also increase computation time. class_weight and random_state have the same effect as in the logistic regression model.

Linear SVM: A powerful linear model that maximizes the decision boundary between classes. The regularization parameter C controls the complexity of the model. A lower value implies stronger regularization, while higher values make the model more flexible. C=1 is a balanced default value and works very well for the spam filter task. class_weight and random_state again have the same effect as in the logistic regression model.

SGD Classifier: An incremental algorithm particularly well-suited for large datasets. alpha=0.001 supports flexible model fitting by ensuring the regularization is not too strong, which is appropriate in this case since sufficient data is available. class_weight and random_state again have the same effect as in the models before.