Data science and Machine learning

Assignment - Designing a Spam Filter

Bachelor program - Mechatronics, Design and Innovation

5^{th} semester

Lecturer: Daniel T.McGuiness

Group: BA-MECH-22

Authors: Vella Nedelcheva

January 8, 2025

Contents

1 Introduction

The goal of the project is the development of spam detection system using machine learning techniques and the Apache Spam Assassin's public datasets [1] [2]. It involves extracting and preprocessing email content, building a predictive model, and integrating a graphical user interface (GUI) for user interaction. The emails are to be classified as either "Spam" or "Ham" based on their content by executing the **main_spam_classifier.py**.

2 Libraries

The following libraries were used for the efficient message recognition:

- `pandas`: for the data manipulation.
- `os`, `re` & `string`:
  - `os`: the os provides a way to interact with the operating system, enabling tasks such as file and directory management, path handling, and ensuring platform-independent code execution.
  - `re` & `string`: are for text preprocessing, such as cleaning and formatting textual data.
- `sklearn` modules:
  - `train_test_split`: splits datasets into training and testing subsets.
  - `TfidfVectorizer`: converts text data into numerical form using TF-IDF scores.
  - `LogisticRegression`: builds a logistic regression model.
  - `metrics`: evaluates the performance of the models (e.g., `accuracy_score`, `precision_score`).
  - `LabelEncoder`: converts categorical labels into numerical form.
  - `GridSearchCV`: performs hyperparameter optimization using grid search.
- `RandomForestClassifier`: implements a random forest classifier for predictive modeling.
- `roc_curve` and `auc (from sklearn.metrics)`:
  - `roc_curve`: calculates the Receiver Operating Characteristic curve for binary classification problems.
  - `auc`: computes the Area Under the Curve (AUC) from the ROC curve.
- `imblearn modules`:
  - `RandomOverSampler`: handles class imbalance by oversampling the minority class.
  - `Pipeline`: creates a robust pipeline combining sampling and modeling steps.
- `seaborn & matplotlib.pyplot`:
  - For visualization of confusion matrices and such.
- `tkinter`:
  - For creating graphical user interfaces (GUIs). `import tkinter as tk` initializes basic GUI components, while `from tkinter import ttk` provides access to themed widgets for improved aesthetics.
- `nltk`:
  - Used for the natural language processing tasks such as tokenization, lemmatization (via `WordNetLemmatizer`), and filtering out stopwords. The downloading of NLTK resources (`stopwords`, `punkt`, and `wordnet`) is important. If these resources are not already available, the program raises errors during execution.

3 Methodology

3.1 EXTRACTING FILES

The function **extract_tar_file**, defined in the **unzip_files.py**, extracts **.tar.bz2** archives into designated directories: **dataset\unzipped\easyham** and **dataset\unzipped\spam**. It first checks if the extraction directory exists and creates it if necessary. The function verifies the presence of the tar file, printing a message if the file is missing. If it is available, it uses the **tarfile** module to open the archive, extract its contents, and ensure proper closure upon completion. The main script sets the file paths for the **ham** and **spam** datasets and uses `extract_tar_file` to extract them for easier data handling.

3.2 PROCESSING EXTRACTED DATA

The **extract_data.py** processes email content by extracting the fields: *subject*, *date*, and *delivered-to*. The e-mails are loaded from the extracted folders, which are organized as **easy_ham** and **spam**.

The function `extract_subject_date_and_delivered_from_email` is applied to extract the required fields. To each email is assigned a class label (*ham* or *spam*) based on the folder from which it is retrieved. Entries with missing fields are filtered out, ensuring that only valid emails are processed.

The script processes up to 500 emails from each folder and combines them into one dataset as a pandas DataFrame. The dataset includes the extracted fields (*subject*, *date*, and *delivered-to*) and the class label (*ham* or *spam*). The file also checks that all entries have a subject, and any incomplete ones are removed. The final dataset is saved as a CSV file and then reduced to 400 per class via the **reduce_data.py**. Additionally, the code includes error handling to handle issues like missing folders or unreadable files, making it more reliable.

3.3 PREPROCESSING EMAIL

The dataset is initially loaded into a pandas DataFrame using the **main_spam_classifier.py**. The **preprocess_email** function is then applied to clean and standardize the email text, improving its suitability for the model. Frequent, unimportant terms such as 'the' and 'is' are removed using NLTK's predefined list of English stopwords, ensuring they don't interfere with the model's understanding. The text is then tokenized and lemmatized ('running' becomes 'run'), combining variations of similar terms and improving the model's ability to treat them as equivalents, thereby enhancing overall performance.

Once they are removed and the text is lemmatized, additional preprocessing steps are applied. Email headers, such as content between **"Subject:"** and **"Date:"**, are removed as they are irrelevant to the analysis. All numbers in the text are then replaced with the placeholder **"NUMBER"** to generalize numeric data and avoid bias toward specific numbers. URLs are standardized by replacing them with the placeholder **"URL"**.

Finally, the text is converted to lowercase to ensure consistency and eliminate any case sensitivity issues, and punctuation marks are removed for a clean, standardized representation of the email content, ready for model training.

3.4 LABEL ENCODING

The target variable, **y**, which represents whether an email is "Spam" or "Ham," is encoded into numerical values using the `LabelEncoder` from scikit-learn. Spam emails are assigned a label class of 1, and Ham emails are assigned 0.

3.5 DATA SPLITTING

The dataset is divided into training and testing sets using an 80-20 split. The training data is used to train the model, while the testing data is reserved for evaluating the model's performance.

3.6 MODELS FOR TRAINING AND TESTING THE DATA

The three models Logistic Regression, Random Forest Classifier, and SVC are compared for spam classification. Logistic Regression, is a linear model, and is computationally efficient and effective for linearly separable data. Random Forest Classifier, using decision trees, captures non-linear patterns but risks overfitting. SVC, with robust non-linear decision boundaries, is computationally intensive.

3.7 MODEL BUILDING

The pipelines streamline the process of feature extraction, data resampling, and model training for spam email classification. Similar to [3], [4], and [5], each pipeline—`pipeline_lr`, `pipeline_rf`, and `pipeline_svc`—utilizes `TfidfVectorizer` for feature extraction, `RandomOverSampler` to address class imbalance, and a classifier. This ensures consistent preprocessing during both training and evaluation, minimizing errors and enabling accurate performance comparisons.

Although the use of bigrams and trigrams can increase dimensionality by generating more features, the `max_features=10000` parameter ensures that only the top 10,000 features, based on their relevance (e.g., TF-IDF scores), are retained. This prevents the feature space from growing excessively large and helps balance computational complexity with model performance.

The `TfidfVectorizer` is configured with `binary=True`, meaning it represents terms as either present or absent, disregarding term frequency. If a word appears in a document, its value is set to 1, regardless of how often it appears; otherwise, it's set to 0.

To address class imbalance, both `RandomOverSampler` (which duplicates samples from the minority class when needed) and `class_weight='balanced'` (which adjusts class weights based on frequencies) are employed. These techniques ensure equal treatment of both spam and ham emails during training, improving the model's ability to handle class imbalance effectively.

The main difference between the pipelines lies in the classifiers applied, whereby `pipeline_lr` employs `LogisticRegression` with the efficient `liblinear` solver. The random forest pipeline `pipeline_rf` uses `RandomForestClassifier`, an ensemble method that combines multiple decision trees to enhance accuracy and reduce overfitting.

In contrast, the support vector classifier pipeline, `pipeline_svc`, uses `SVC` with `probability=True`, enabling the `predict_proba` method. This method provides probability estimates for predictions, which are essential for evaluating model performance using metrics like ROC curves and AUC scores. Without this configuration, the `SVC` model would not support probability-based evaluations.

By combining the classifiers into pipelines, the framework provides an efficient and robust solution for spam email classification. It ensures that preprocessing is consistent across all models, enabling smooth comparison of logistic regression, random forest, and support vector classifiers.

3.8 MODEL TUNNING

Model tuning is a crucial step in optimizing the performance of machine learning models and preventing overfitting. For this purpose, `GridSearchCV` is used to search for the best combination of hyperparameters. Similar to [3], the parameter grid evaluates the model using cross-validation to find the optimal settings. The vectorizer is tuned with parameters that include TF-IDF settings such as `max_df` (maximum document frequency), `min_df` (minimum document frequency), and `ngram_range` (the range of n-grams used). The `max_df` values are tested with 0.75, 0.85, and 0.95 to exclude overly common terms, while `min_df` is set to 1 and 2 to remove rare terms. The `ngram_range` is tested using the values (1,1), (1,2), and (1,3), which correspond to unigrams, bigrams, and trigrams, respectively. This approach allows the model to capture different levels of context and semantic meaning.

Additionally, adjusting `max_df` and `min_df` helps control the size of the feature set, which can impact both the model's performance and computational cost.

In logistic regression, the regularization strength, controlled by the `classifier_C` parameter, is tuned to balance model fitting and simplicity. This parameter is tested with the values $\{0.01, 0.1, 1, 10, 100\}$, helping to control the complexity and prevent overfitting [6] and [7], while finding a balance between fitting the data and maintaining simplicity. Larger values of `C` correspond to weaker regularization, allowing the model to fit the data more closely, but increasing the risk of overfitting. Smaller values of `C` apply stronger regularization, simplifying the model at the cost of potentially higher bias.

For the random forest, similar to [8], the parameters are set as follows: the classifier_n_estimators, representing the number of trees, is tested with the values $\{100, 200, 500\}$, and the classifier_max_depth parameter, which limits the depth of the trees, is tested with $\{10, 20, 50\}$ to prevent overfitting. Since the data is complex and large enough, increasing the number of trees (`n_estimators`) in this case will likely improve the performance by reducing variance, though at the cost of slower execution. Shallower trees (smaller values of `max_depth`) can may recude overfitting by limiting the model's capacity to fit noise in the data. However, overly shallow trees might fail to capture important patterns, leading to underfitting.

For the SVC model, the `classifier_C` parameter is tested with the same values to balance bias and variance effectively.

The application of GridSearchCV, similar to [9], utilizes the parameter grid for each model and its pipeline. The search is done via `cv=3`, which represents the 3-fold cross-validation. The **`verbose=1`** ensures that progress updates, such as the number of iterations or status messages, are displayed during execution. The parameter **`n_jobs=-1`** allows the algorithm to utilize all available CPU cores, maximizing parallel processing and significantly improving computation speed.

3.9 VISUALIZING TF-IDF VECTORS

The `visualize_vectors` function processes a list of input to generate and display its vectorized form as a string. It first converts the data into a sparse matrix using the TF-IDF vectorizer, then transforms it into a standard format for easier manipulation. The feature names, representing the terms identified by the vectorizer, are retrieved, as shown in image 3.1.

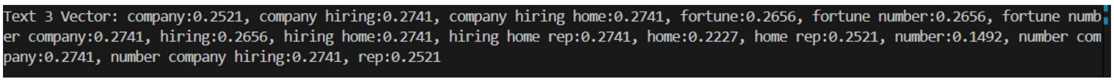

Figure 3.1: **`Vectorized words:`** An example of the TF-IDF vectorization process

3.10 RESULTS

```
Classification results for Logistic Regression:
              precision    recall  f1-score   support

         ham       0.95      0.93      0.94        80
        spam       0.93      0.95      0.94        80

    accuracy                           0.94       160
   macro avg       0.94      0.94      0.94       160
weighted avg       0.94      0.94      0.94       160


Classification results for Random Forest:
              precision    recall  f1-score   support

         ham       0.92      0.76      0.84        80
        spam       0.80      0.94      0.86        80

    accuracy                           0.85       160
   macro avg       0.86      0.85      0.85       160
weighted avg       0.86      0.85      0.85       160


Classification results for SVC:
              precision    recall  f1-score   support

         ham       0.96      0.91      0.94        80
        spam       0.92      0.96      0.94        80

    accuracy                           0.94       160
   macro avg       0.94      0.94      0.94       160
weighted avg       0.94      0.94      0.94       160
```

Figure 3.2: Logistic Regression, Random Forest and Support Vector Classifier

The classification results in 3.2 provide a breakdown of the precision, recall, and F1 scores for both classes (Spam and Ham).

The Logistic Regression provides the highest overall accuracy (0.94) with a balanced precision and recall for both ham and spam. It offers better precision and recall balance than SVC (which has a higher recall for spam and lower recall for ham) and also outperforms the Random Forest Classifier, which has a significantly lower recall for ham (0.76). SVC is still a solid option, performing similarly to Logistic Regression with 0.94 accuracy and strong F1-scores, but less consistent for ham.

The Random Forest Classifier, while a strong ensemble method, does not perform as well on ham classification. The recall for `Ham` (0.76) is notably lower than for `Spam` (0.94), indicating that the model is better at detecting spam but struggles with ham classification. Accuracy is `0.85`, which is decent. The `F1-scores` for both classes are as follows: `Ham` 0.84 and `Spam` 0.86. These F1-scores show that the model is performing reasonably well overall, but there is some imbalance in performance between the two classes.

The `classification results for SVC` show that the model achieves high precision, recall, and F1-score for both ham and spam. For `Ham`, the precision is 0.96, recall is 0.91, and F1-score is 0.94. For `Spam`, the precision is 0.92, recall is 0.96, and F1-score is 0.94. The model achieves an accuracy of `0.94`, with macro and weighted averages also showing high values of `0.94` for precision, recall, and F1-score. This indicates a well-balanced model performance across both classes, but it is not as good as the Logistic Regression.

3.11 CONFUSION MATRIX

Additionally, a confusion matrix is generated to show the performance of the models. The image 3.3 is a representation of the good performance of the LogisticRegression model, indicating a high number of true positives and a low number of false positives.

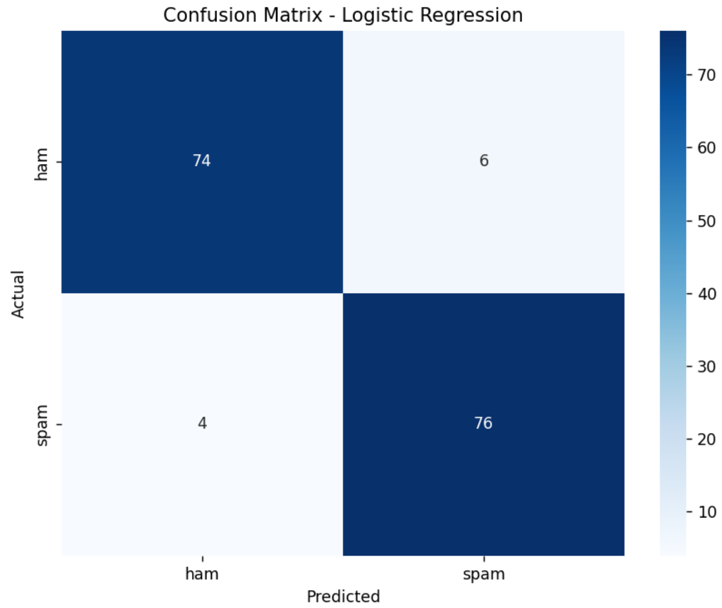

Figure 3.3: Confusion Matrix of the Logistic Regression

3.12 ROC CURVE

As described in [10], the Receiver Operating Characteristic (ROC) curve shows the performance of each model by plotting the True Positive Rate (TPR). The Area Under the Curve (AUC) summarizes this performance, an AUC of 1.0 indicates perfect classification, while 0.5 reflects random guessing.

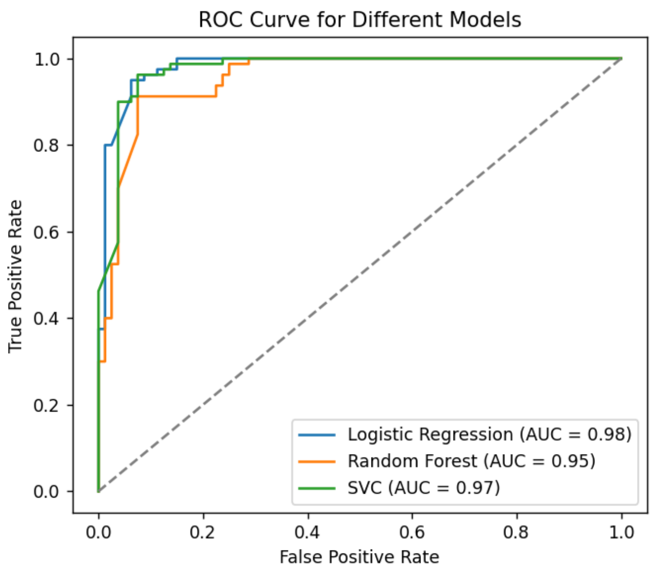

Figure 3.4: ROC curve

3.13 GUI FOR SPAM DETECTION

A GUI, shown in 3.6 and 3.5 is developed using Tkinter to allow users to input messages from the dataset '`for_testing_messages_spam_ham.csv`' for manual model testing. This dataset is extracted with '`unzip_files.py`' from '`20021010_easy_ham.tar.bz2`' and '`20021010_spam.tar.bz2`', then processed sequentially through '`extract_data.py`' and '`reduce_data.py`'

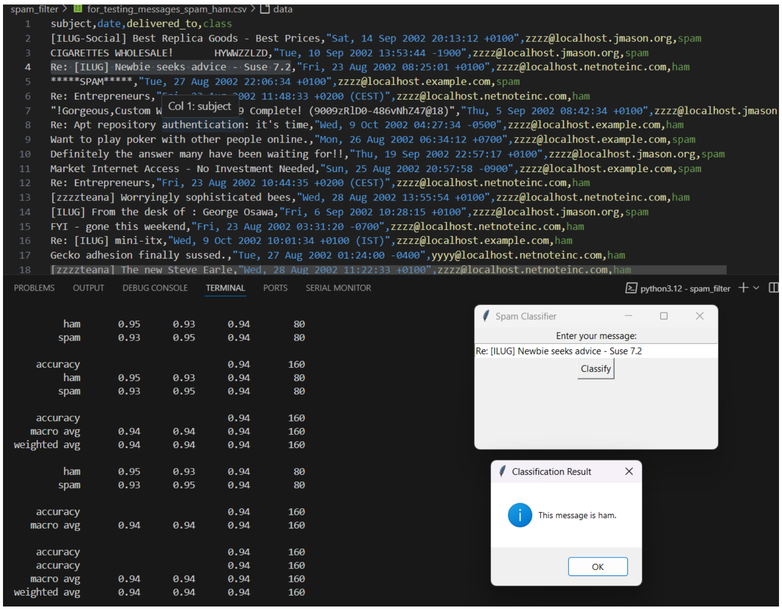

Figure 3.5: User interface spam filter test with ham message

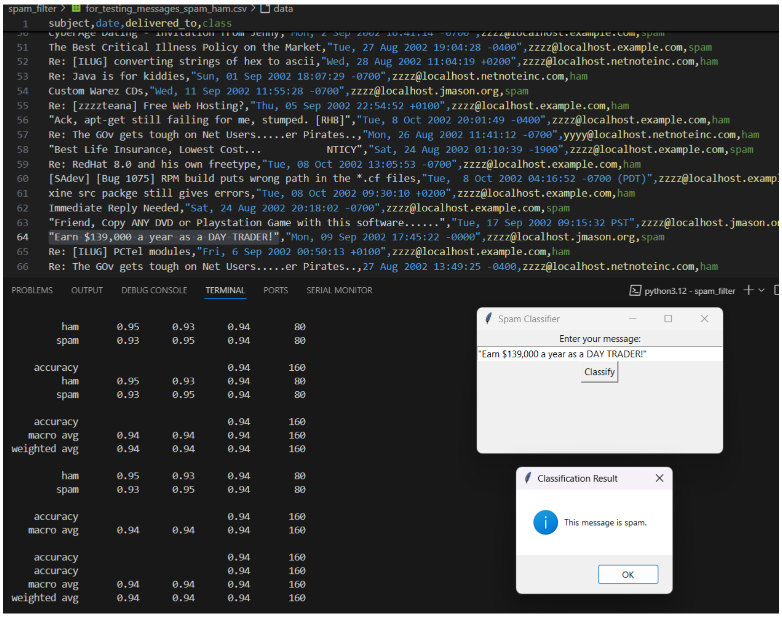

Figure 3.6: User interface spam filter test with spam message

4 Conclusion

Both Logistic Regression and SVC perform similarly with an overall accuracy of 0.94. However, SVC shows a slight advantage in precision for ham (0.96) and recall for spam (0.96), while Logistic Regression performs slightly better in recall for ham (0.93) and precision for spam (0.93). The F1-scores for both models are the same across both classes, suggesting that both models are well-suited for the spam filter. However, Logistic Regression shows more consistent results, with a more balanced approach between precision and recall, making it the better choice for the task when aiming for overall performance.

List of Figures

References

[1] "Apache Spam Assassin's public datasets," Available online at https://spamassassin.apache.org/old/publiccorpus/, accessed on: December 5, 2024.

[2] https://www.deepl.com/en/translator, accessed on: January 2, 2025.

[3] https://stackoverflow.com/questions/50285973/pipeline-multiple-classifiers, accessed on: December 5, 2024.

[4] https://stackoverflow.com/questions/50285973/pipeline-multiple-classifiers, accessed on: December 5, 2024.

[5] https://www.geeksforgeeks.org/shap-with-a-linear-svc-model-from-sklearn-using-pipeline/, accessed on: December 5, 2024.

[6] https://pythonsimplified.com/how-to-use-k-fold-cv-and-gridsearchcv-with-sklearn-pipeline/?utm_content=cmp-true, accessed on: December 5, 2024.

[7] https://www.deeplearning.lipingyang.org/wp-content/uploads/2018/07/SVM-Parameter-Tuning-in-Scikit-Learn-using-GridSearchCV.pdf#:~:text=You%20just%20need%20to%20import%20GridSearchCV%20from%20sklearn.grid_search%2C,number%20of%20cross%20validations%20to%20the%20GridSearchCV%20method., accessed on: December 5, 2024.

[8] https://mljourney.com/step-by-step-guide-to-random-forest-in-sklearn/, accessed on: December 5, 2024.

[9] https://www.geeksforgeeks.org/how-to-optimize-logistic-regression-performance/, accessed on: December 5, 2024.

[10] https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc, accessed on: December 5, 2024.