

EINIS - summary of the project

Bartosz Latosek

February 2024

Contents

1	Introduction	2
2	Data	2
2.1	Dataset selection	2
2.2	Dataset Preprocessing and Analysys	3
2.2.1	Preprocessing the data	3
2.2.2	Data analysys	3
3	Model training and comparison	6
3.1	Selected models	6
3.2	Models training	6
3.3	Models evaluation	6
3.3.1	Introduction to used metrics	6
3.3.2	Models comparison	7
3.3.3	Conclusion	7
4	User Interface	7

1 Introduction

The topic of the project was simply set as **"Email Spam Detector"**. Therefore, I was given a sort of blank canvas which means I had to make introductory assumptions and general plan of the project.

I set the goal of the project to create a simple interface for users to paste the email contents into the prompt field and receive a response determining whether the email is a **Spam** or **"Ham"**. (Ham is an actual term contradictory to spam used in the problem of classifying email contents, therefore It will be further used in this report).

I decided to divide the project into 3 main parts. First of them is oriented around the data - the process of choosing the right dataset, as well as adjusting and evaluating the selected option. The outcome of the second part of the project is training a couple of different models, evaluating their performance and choosing the best one. The third part of the project focuses on delivering an easy to use, user friendly interface on which the models can be accessed and predictions can be made.

2 Data

2.1 Dataset selection

The main source of datasets related to the task of email classification was *kaggle.com* website. In short - Kaggle is a platform that hosts data science competitions, provides datasets, and facilitates collaboration among data scientists and machine learning enthusiasts. From the website, four candidates for the dataset were selected and compared.

The whole analytical process can be seen in the *dataset_selection.ipynb* notebook in the project's source files. The summary of the analysis can be seen below:

Dataset Name	Size	Percentage of spam class
emails.csv	31.24 MB	29%
spam_assasin.csv	24.53 MB	32.71%
spam.csv	503.66 kB	13.41%
spam_ham_dataset.csv	5.5MB	28.99%

Figure 1: Summary table

Out of the four of them, the *spam.csv* dataset seems the worst. It has the smallest size and the data is unnormalised - there are some non-utf characters which need to be filtered. The class balance is also the poorest so we can safely eliminate this candidate. When it comes to the first dataset - *emails.csv* - the biggest advantage is already normalized data (which can also be a con - we don't have access to the original email text). The size is also the best and the class balance is decent. The second dataset - *spam_assasin.csv* seems to be a promising candidate. The snapshot of email's html gives us access to the original email content which allows our models to find hidden relations in - for example - the headers or picture banners. The class balance is also the best out of the three candidates, but it still needs to be balanced in training. However this dataset has a lot of useless information inside the email text column. Things like the sender email, IP address and date in combination with no structure allowing us to simply filter the contents sadly make this dataset unfitting for the problem. The last presented dataset - *spam_ham_dataset.csv* has a decent size, class balance and also gives us access to the email subject - which gives us additional data and gives the model another decisive point. Overall, the format of the data in combination with other aspects make this dataset the most fitting for the given problem.

2.2 Dataset Preprocessing and Analysys

The whole preprocess and analysys along with the code can be accessed in the source files of the project as *data_analysys_and_preprocess.ipynb*.

2.2.1 Preprocessing the data

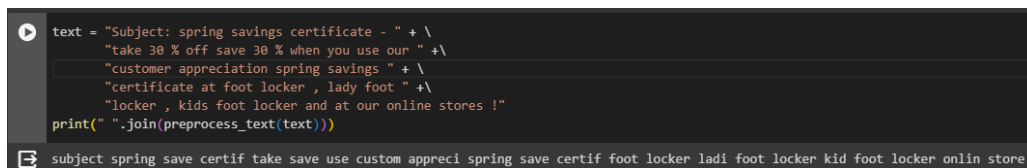
Before we move to the dataset analysys, we have to preprocess the data in order to increase it's value and make the possible relations easy to distinguish.

The whole preprocessing code can be seen in *data_analysys_and_preprocess.ipynb* notebook under the corresponding "Preprocess" subsection.

For data preprocessing I've decided to use the standardized approach in Natural Language Processing - word tokenization. Word tokenization is the process of breaking down a text into individual words or tokens. It involves separating the text into meaningful units, typically removing punctuation and splitting the text at spaces, to facilitate further analysis in natural language processing tasks.

Before the tokenization process, I've also transformed the email text to lowercase and removed words considered as stopwords. These are common words, such as "the," "and," and "is," that are often removed during natural language processing to focus on the more meaningful words. They are considered to be of low informational value as they appear frequently across various texts in both classes.

The example of text preprocessing can be seen on the code snippet below:



```
text = "Subject: spring savings certificate - " + \
      "take 30 % off save 30 % when you use our " + \
      "customer appreciation spring savings " + \
      "certificate at foot locker , lady foot " + \
      "locker , kids foot locker and at our online stores !"
print(" ".join(preprocess_text(text)))
```

subject spring save certif take save use custom appreci spring save certif foot locker ladi foot locker kid foot locker onlin store

Figure 2: Text preprocess example

We can see that the stopwords were stripped away from the text and only the most meaningful parts of certain words were left - *Appreciation* = *Appreci*. This proves, that the tokenization process makes the training data more general.

The last part of text preprocessing is to vectorize it. It is hard for model to make sense out of string sequences, therefore a TF-IDF vectorization is performed on the data text before feeding the datasets into the models. The TF-IDF vectorization process involves assigning weights to words in a document sequence. A high TF-IDF weight is assigned to a word that appears frequently in a specific document but infrequently across the entire collection, indicating its importance in that particular document. This method helps capture the significance of words in representing the content of documents while downweighting common terms. The result is a numerical vector representation for each document, reflecting the importance of individual words in the context of the entire collection.

2.2.2 Data analysys

Comparing the word size for both classes we can see, that the average value for spam emails is slightly bigger than that of the non-spam emails.

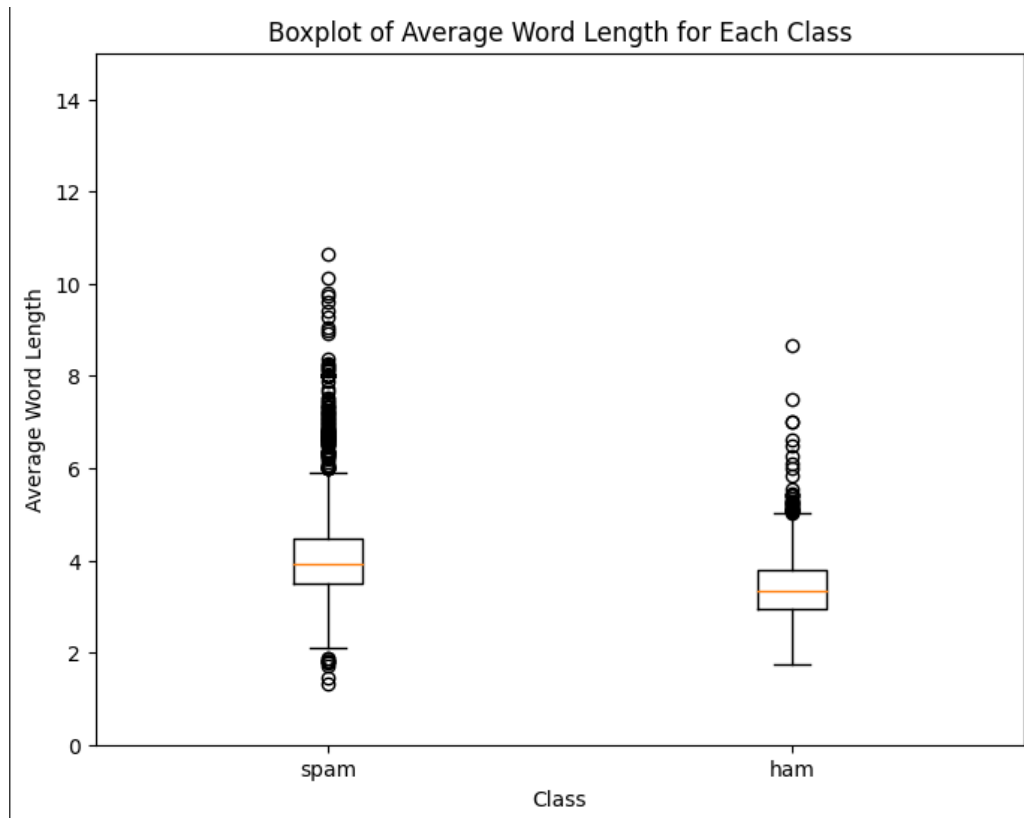


Figure 3: Average word size for both classes

Common subject related to Natural Language Processing tasks is Latent Dirichlet Allocation. LDA is a probabilistic model used for topic modeling in natural language processing. It assumes that documents are mixtures of topics, and each topic is a distribution of words. LDA's goal is to uncover these latent topics from a collection of documents by iteratively assigning words to topics and documents to topic mixtures in a way that reflects the observed word co-occurrence patterns. It is widely employed for discovering the underlying thematic structure in a large set of texts, aiding in tasks such as document categorization and content analysis.

Given the above, we can implement a simple LDA model to inspect the relations between topics present in the text data. Spam detection is a simple two-class classification problem, therefore it is safe to assume that the number of topic should be 2 - spam oriented and non-spam oriented.

The outcome of this analysis can be seen on the diagrams below:

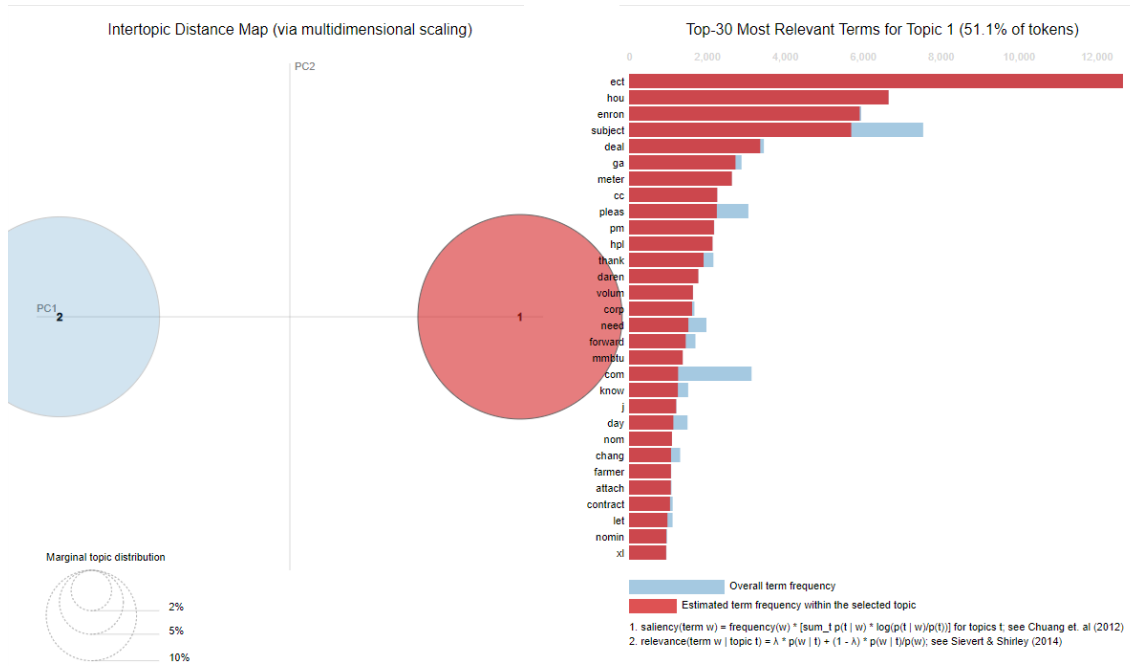


Figure 4: Topic 1 term distribution.

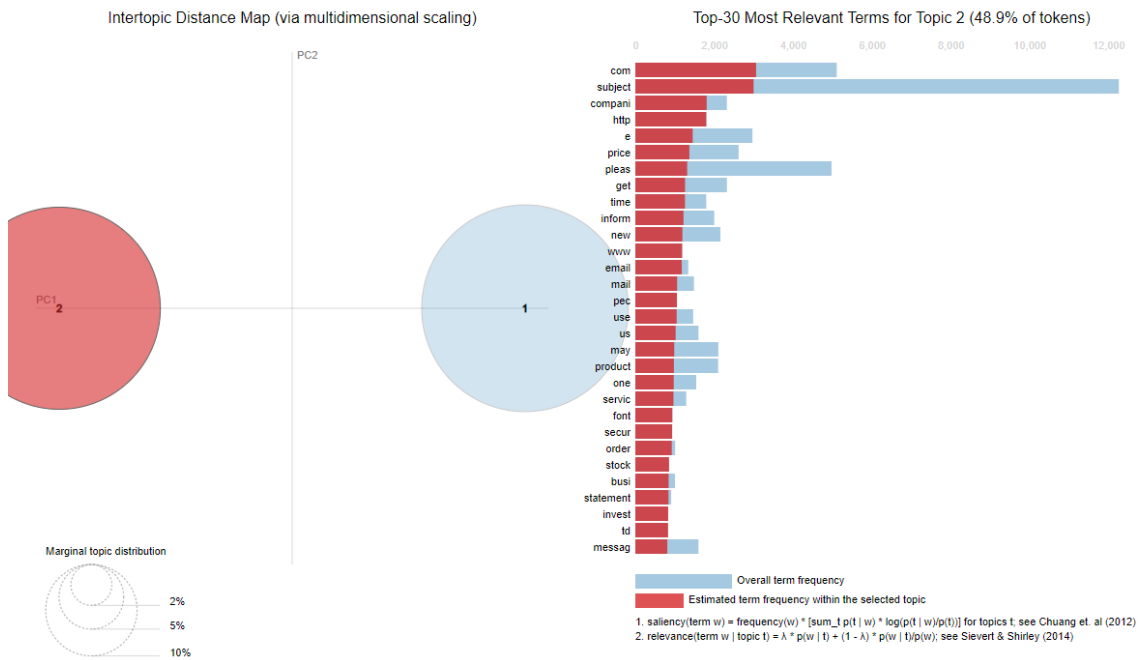


Figure 5: Topic 2 term distribution.

On both the above graphs we can see that the two topics are separable and described by unrelated terms - which should be helpful in the final classification task.

3 Model training and comparison

3.1 Selected models

For the sake of this project - four different classification models were selected to be trained on the test data and evaluated in order to choose the best one.

First chosen model is **LogisticRegression**. Logistic Regression is a statistical model used for binary classification, predicting the probability of an instance belonging to a particular class. It employs the logistic function to map a linear combination of input features to a probability score, making it widely used in machine learning for tasks such as spam detection or medical diagnosis. For the sake of this project - an implementation from *sklearn* library was selected.

Gaussian Naive Bayes was chosen as second contender. Naive Bayes is a probabilistic machine learning algorithm based on Bayes' theorem that assumes independence between features given the class label. It is commonly used for classification tasks and is particularly efficient with high-dimensional data, making it suitable for applications such as text categorization and spam filtering. Gaussian Naive Bayes is a variant of the Naive Bayes algorithm specifically designed for continuous data. It assumes that the features follow a Gaussian (normal) distribution and estimates the mean and variance for each class, making it suitable for problems where the features are real-valued and can be modeled as continuous random variables.

Next evaluated model is **Random Forest**, which implementation was also borrowed from the *sklearn* library. Random Forest is an ensemble machine learning algorithm that constructs multiple decision trees during training and outputs the mode of the classes (classification) or the average prediction (regression) of the individual trees. It enhances predictive accuracy and generalization by combining the results of diverse and independently trained trees.

The last model is a current State of The Art solution in classifier category - **XGBoost**. XGBoost (Extreme Gradient Boosting) is a powerful and efficient gradient boosting algorithm that sequentially builds a series of decision trees to minimize a predefined loss function, providing high predictive performance. It incorporates regularization techniques and parallel computing to handle large datasets, making it widely used for both classification and regression tasks in machine learning.

For comparison purposes - naive model was implemented and introduced among other described models, but it does not count as a valuable contender - more like a baseline on which sets the bottom margin of performance for other models.

3.2 Models training

Each of the models mentioned above (except the naive model) was trained on the same train dataset, created from the original processed data and split using the *train_test_split* function of *sklearn* library. The models were then evaluated on the same test dataset, obtained the same way.

3.3 Models evaluation

3.3.1 Introduction to used metrics

In the realm of machine learning and statistics, various metrics are employed to assess the performance of models across different tasks. Three commonly used metrics are R^2 score, **accuracy**, and **F1 score**.

The R^2 score, also known as the coefficient of determination, measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It ranges from 0 to 1, with higher values

indicating a better fit of the model.

Accuracy, on the other hand, is a metric frequently used in classification tasks. It represents the ratio of correctly predicted instances to the total number of instances, providing an overall measure of a model's correctness.

Finally, the $F1$ score is a metric that balances precision and recall, making it particularly useful when dealing with imbalanced datasets. It combines these two measures into a single value, providing a comprehensive evaluation of a model's performance, especially in binary classification tasks.

3.3.2 Models comparison

Table 1: Performance Metrics for Different Models			
Model	R2 Score	Accuracy	F1 Score
Logistic Regression	0.914	0.983	0.970
Gaussian Naive Bayes	0.770	0.954	0.918
Random Forest Classifier	0.918	0.984	0.971
XGBoost Classifier	0.870	0.974	0.954
Naive Model	-1.458	0.505	0.380

Comments

- **Logistic Regression:** Achieves high R^2 score, accuracy, and F1 score, indicating a strong performance across regression and classification metrics.
- **Gaussian Naive Bayes:** Shows good performance but with slightly lower scores compared to Logistic Regression, particularly in R^2 and F1 score.
- **Random Forest Classifier:** Performs exceptionally well with high scores across all metrics, suggesting a good solution for the problem.
- **XGBoost Classifier:** Demonstrates a solid performance, falling slightly below Random Forest in terms of R^2 score, accuracy and F1 score.
- **Naive Model:** Exhibits poor performance with a negative R^2 score, low accuracy, and F1 score, emphasizing the importance of using more sophisticated models.

3.3.3 Conclusion

After evaluating the performance of various models using R^2 score, accuracy, and F1 score, it is evident that the Random Forest Classifier emerges as the best-performing model. With an impressive R^2 score of 0.918, an accuracy of 0.984, and an F1 score of 0.971, the Random Forest Classifier consistently outperforms other models, therefore the final interface will use it as the core tool of prediction making.

4 User Interface

For the **User Interface** I've decided to use the *Streamlit* framework. *Streamlit* is an open-source Python library that simplifies the process of creating web applications for data science and machine learning by allowing developers to turn data scripts into interactive, shareable apps with minimal code. It provides a straightforward way to build user interfaces and dashboards directly from Python scripts.

The script of the app is available in *main_app.py* file in the source folder. To run the app, type the following command:

```
python3 -m streamlit run srcmain_app.py
```

The example screenshot from the app, can be seen on the picture below:

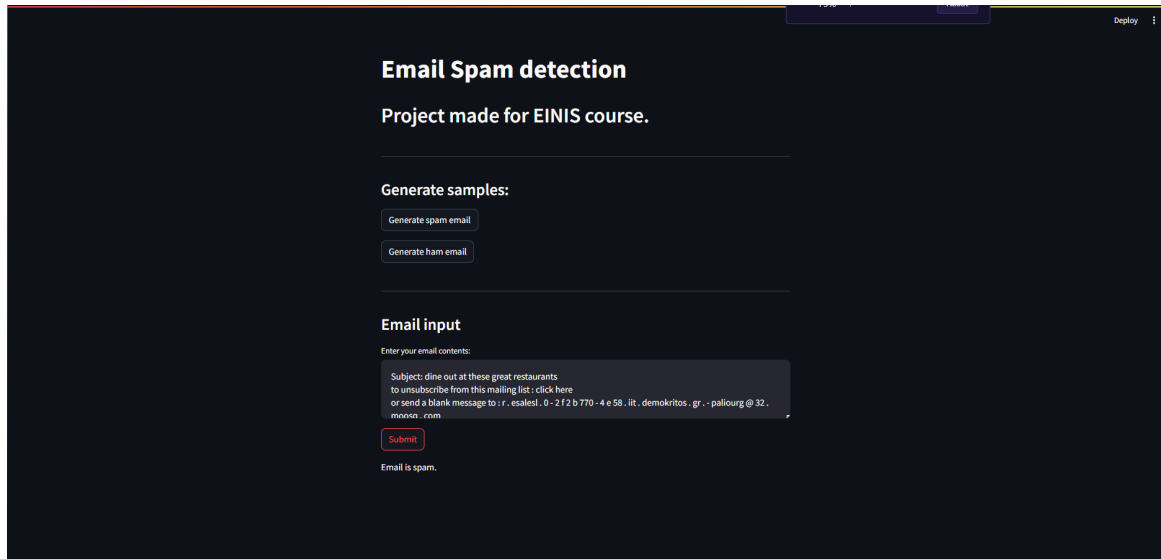


Figure 6: Example application screenshot.

You can type in your own email contents or generate sample data from the selected dataset. The output of the predictor is given as a response, below the input textbox.