

**AI Corse Final Project -   
Email Spam Detection with Machine Learning**

**Student:** Ron Keinan

**Lecturer:** Dr. Eyal Ben Isaac

Department of Data Mining

Lev Academic Center

February 2023 – אדר התשפ"ג

Contents

[1 Introduction 2](#_Toc128258527)

[2 Theoretical Review 3](#_Toc128258528)

[2.1 Text Classification 3](#_Toc128258531)

[2.2 Email Spam Detection 3](#_Toc128258532)

[2.3 Text Preprocessing 4](#_Toc128258533)

[2.4 Datasets Description 4](#_Toc128258534)

[2.5 Classifiers Description 7](#_Toc128258535)

[3 Methodology 9](#_Toc128258536)

[4 Program Verification 11](#_Toc128258537)

[5 Results 13](#_Toc128258538)

[6 Discussion 16](#_Toc128258539)

[7 Conclusions and Future Work 18](#_Toc128258540)

[8 References 19](#_Toc128258541)

[9 Appendices 21](#_Toc128258542)

A Examples of dev classifications with the GUI

B Results of all models scores………………………………………….34

# Introduction

This project is part of the assignments required in the "Artificial Intelligence" course as part of the master's degree studies in data mining at the Lev Academic Center.

In this project, I use the knowledge I gained during the course to perform a text classification task, using machine learning algorithms. The mission was described in the project requirements as follows:

***Email Segregation such as spam and phishing filters*** *– One of the latest trends in the cybersecurity market is email segregation using the AI for detecting, tracking, and analyzing the keywords in the emails to filter the emails for spam and phishing emails. The spam emails are dangerous because if they are not controlled, it occupies the space, delivers the malicious payload, and can cause security vulnerabilities. Similarly, the spear-phishing or the targeted phishing emails are malicious and can collect personal information about an individual for malicious intents such as stealing the data or stealing the money from the bank accounts, etext classification.*

The goal is to create a simple system that will receive from the user as input - the text of an email and classify the email as spam or not spam. The system will do this through preliminary training that will be performed on a data set of emails that are classified as spam or not. During the preliminary training stage, the system will compare the accuracy of 5 Machine Learning classifiers – Decision Tree, Random Forest, Support Vector Machine, Logistic Regression, K Nearest Neighbors, and another Deep Learning classifier - Multi-Layer Perceptron.

The features for the model will be the most significant words that appear in the emails dataset. Words will be determined as most significant using the TF-IDF rating. Also, during the training phase, there will be comparisons between different amounts of features to find the amount that will allow the most accurate prediction, as well as a comparison between types of features - will each feature be one, two, or three words - i.e. unigram, bigram or trigram.

I would like to take this opportunity to thank you for the significant learning in the course. This is a fascinating, modern, and rapidly developing field, and entering the world with the help of this course in particular and the data mining degree, in general, is significant in my professional development.

# Theoretical Review

2. 1. Text Classification

Automatic text classification aims to automatically categorize natural language text into predefined categories or classes. Text classification has a wide range of applications in fields such as information retrieval, content filtering, and email classification. One important area of text classification is topic classification, which involves assigning a topic or theme to a piece of text. Another area of text classification is document classification, which involves categorizing documents based on their content. Many studies in text classification have addressed tasks such as topic identification in news articles, classification of research articles into domains, and email filtering

In text classification, a main technique has been proposed: machine learning (ML). The ML approach is composed of two general steps: (1) learn the model from a training corpus, and (2) classify a test corpus based on the trained model (Pang et al. 2002 and Jeonghee et al. 2003). Various ML methods have been applied for sentiment classification. For instance, Pang and Lee (2005) applied three ML methods: Naive Bayes (NB), Maximum Entropy (ME), and Support Vector Machines (SVM), and combined SVM and regression (SVR) modes, with metric labeling.

* 1. Email Spam Detection

Email spam detection can be approached as a text classification problem, where the goal is to classify emails into two categories: spam and non-spam (also known as "ham"). The classification model is trained on a dataset of labeled emails, where each email is labeled as either spam or ham. The model then learns to associate certain words or patterns of words with either spam or ham and uses this knowledge to predict the label of new emails.

Text classification for email spam detection typically involves the following steps:

1. Data preprocessing: The emails are preprocessed to remove stop words (common words such as "the" and "and" that do not carry much meaning), punctuation, and other noise. The remaining words are then transformed into a numerical representation, such as bag-of-words or term frequency-inverse document frequency (TF-IDF), which can be used as input features for the classification model.
2. Feature selection: Not all words are equally informative for spam detection, so feature selection is used to identify the most relevant words or n-grams (sequences of words) for the classification model.
3. Model training: A classification model is trained on the labeled dataset using a supervised learning algorithm such as Naive Bayes, Support Vector Machines, or Logistic Regression. The model is trained to predict the label (spam or ham) of new emails based on the input features.
4. Model evaluation: The performance of the classification model is evaluated using metrics such as accuracy, precision, recall, and F1-score. The model may also be evaluated on a separate test set to assess its generalization ability.

Several studies were done along the second in favor of identifying spam in emails with the help of ML. Olatunji (2017) created a model based on SVM to classify spam emails and reached an accuracy level of 94%. Another study (Agarwal & Kumar, 2018) built a model based on Naive Bayes and reached an accuracy level of 95.5%. Kumar & Sanket (2020) compared Various methods of machine learning, such as SVM, KNN, RF, DT, NB, and more, and achieved a high accuracy of 98% with the help of Naive Bayes.

* 1. Text Preprocessing

Text preprocessing is crucial in NLP fields such as ML, sentiment analysis, text mining, and text classification. In both general and social text documents, noise such as typos, emojis, slang, HTML tags, spelling mistakes, and repetitive letters often appear. Improperly preprocessed text can result in an incorrect analysis of text classification. In some cases preprocessing methods are considered effective for text classification tasks. For instance, HaCohen-Kerner et al. (2008) demonstrated that using word unigrams including stop words leads to improved results compared to the results obtained using word unigrams excluding stop words.

HaCohen-Kerner et al. (2019) investigated the impact of all possible combinations of six preprocessing methods (punctuation mark removal, reduction of repeated characters, spelling correction, HTML tag removal, converting uppercase letters into lowercase letters, and stopword removal) on text classification in three datasets. In another study, HaCohen-Kerner et al. (2020) explored the influence of combinations of the sentiment analysis of six basic preprocessing methods mentioned in the previous paragraph on text classification in four general benchmark text corpora using a bag-of-words representation. The main conclusion recommended is always to perform an extensive and systematic variety of preprocessing methods, combined with many ML methods to improve the accuracy of text classification.

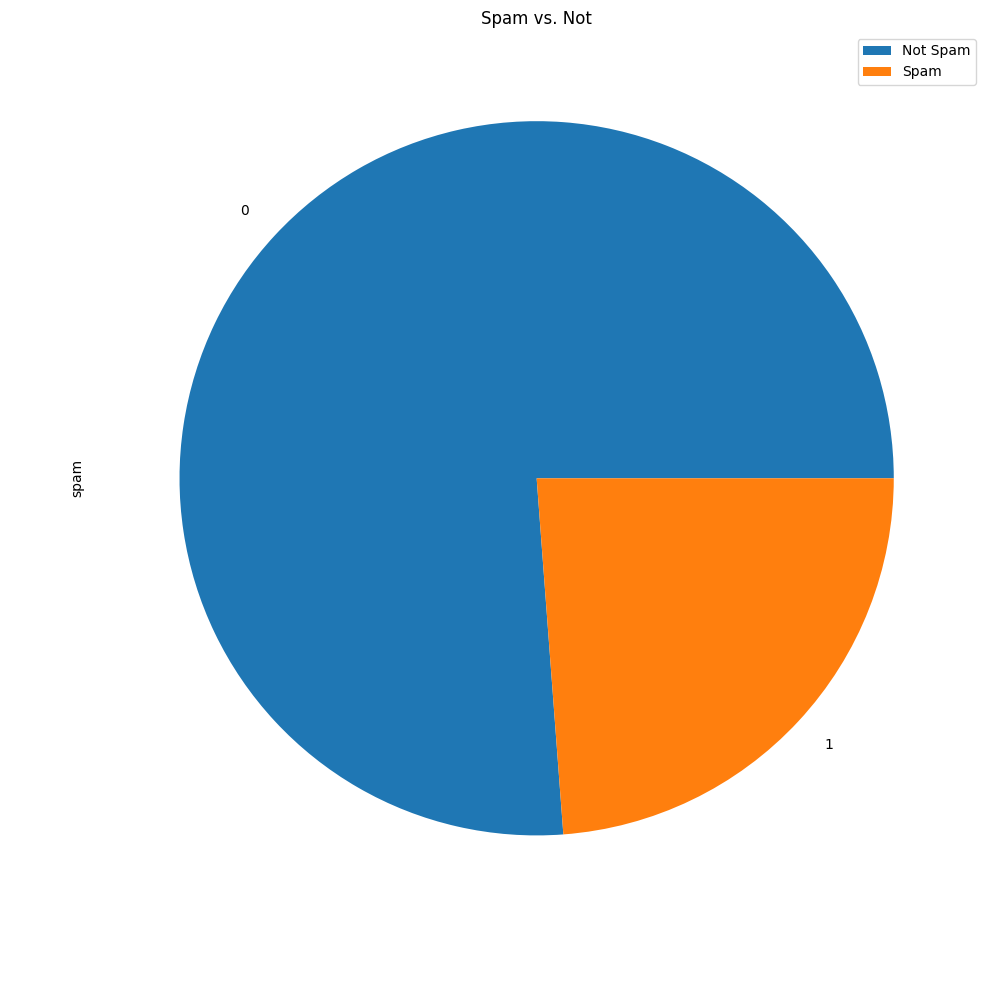
* 1. Datasets Description

The system is based on a collection of 2 Email datasets in English. The dataset involves texts labeled with two classes (1 for spam, and 0 for not spam).

The first dataset contains 5708 emails and was retrieved from: <https://www.kaggle.com/datasets/ozlerhakan/spam-or-not-spam-dataset?resource=download>

Some statistics regarding the dataset:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **spam** | **email\_length** | **email\_words** | **email\_unique\_words** |
| **count** | 5708 | 5708 | 5708 | 5708 |
| **mean** | 0.238086896 | 1541.859145 | 323.9922915 | 137.1135249 |
| **std** | 0.425949893 | 1884.805797 | 390.9019613 | 107.5285745 |
| **min** | 0 | 13 | 2 | 2 |
| **25%** | 0 | 508 | 101 | 70 |
| **50%** | 0 | 979 | 210 | 109 |
| **75%** | 0 | 1892.5 | 401 | 174 |
| **max** | 1 | 31055 | 6348 | 1357 |



The second dataset contains 2984 emails and was retrieved from:

<https://github.com/amankharwal/Email-spam-detection>.

Some statistics regarding the dataset:

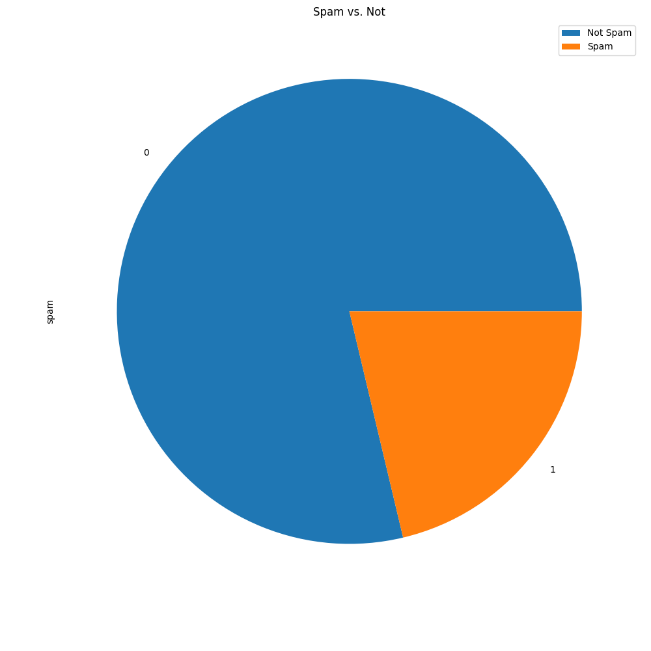
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **spam** | **email\_length** | **email\_words** | **email\_unique\_words** |
| **count** | 2983 | 2983 | 2983 | 2983 |
| **mean** | 0.1645994 | 1240.067382 | 216.0831378 | 115.1830372 |
| **std** | 0.37088077 | 1897.926253 | 324.5735774 | 112.4566271 |
| **min** | 0 | 1 | 0 | 0 |
| **25%** | 0 | 387 | 67 | 51 |
| **50%** | 0 | 756 | 133 | 89 |
| **75%** | 0 | 1347 | 235 | 140 |
| **max** | 1 | 22067 | 3489 | 1039 |



I Built a third dataset as a union of the first 2 datasets.

Some statistics:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **spam** | **email\_length** | **email\_words** | **email\_unique\_words** |
| **count** | 8691 | 8691 | 8691 | 8691 |
| **mean** | 0.212863882 | 1438.275572 | 286.9547808 | 129.5863537 |
| **std** | 0.409355751 | 1894.636813 | 372.997566 | 109.7337881 |
| **min** | 0 | 1 | 0 | 0 |
| **25%** | 0 | 467 | 89 | 63 |
| **50%** | 0 | 896 | 176 | 101 |
| **75%** | 0 | 1690.5 | 346 | 162 |
| **max** | 1 | 31055 | 6348 | 1357 |



* 1. Classifiers Description

I applied 6 supervised ML methods on the training datasets' feature matrices: Random Forest (RF), Support Vector Classifier (SVC), Logistic regression (LR), Decision Tree (DT), K Nearest Neighbors (KNN), and Multinomial Naive Bayes (MNB). I also applied a supervised DL method: Multi-layer Perceptron (MLP).

RF is an ensemble learning algorithm that is used for classification and regression problems. (Breiman, 2001). Ensemble methods use multiple learning algorithms to obtain improved predictive performance compared to what can be obtained from any of the constituent learning algorithms. RF combines multiple decision trees to form a forest of trees, and the final prediction is made by taking a majority vote of the trees. RF combines Breiman's “[bagging](https://en.wikipedia.org/wiki/Bootstrap_aggregating)” (Bootstrap aggregating) idea in Breiman (1996) and a random selection of features introduced by Ho (1995) to construct a forest of decision trees

SVC is a variant of the SVM ML method (Cortes and Vapnik, 1995) implemented in SkLearn. SVC uses LibSVM (Chang & Lin, 2011), which is a fast implementation of the SVM method. SVM is a supervised learning algorithm that is used for classification and regression analysis. It works by finding the hyperplane that maximally separates the data into classes. SVM is known for its good generalization ability and its ability to handle non-linearly separable data using the kernel trick.

LR (Cox, 1958; Hosmer et al., 2013)is a supervised learning algorithm that is used for binary and multiclass classification problems. It models the relationship between the dependent variable and the independent variables using a logistic function. It is known also as maximum entropy regression (MaxEnt), logit regression, and the log-linear classifier.

Decision Tree DT (Song and Ying, 2015) is a flowchart-like structure method in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes).

KNN stands for K-Nearest Neighbors and is a supervised learning algorithm used for classification and regression analysis (Guo et al., 2003). It works by finding the K closest points (i.e. neighbors) to a given data point in the training set and then classifying the data point based on the class labels of its nearest neighbors. The value of K in KNN is usually chosen by the user and determines how many neighbors to consider when classifying a data point. A smaller value of K (e.g. K=1) can result in a more flexible and potentially more accurate model, but can also be more prone to overfitting. Conversely, a larger value of K can result in a more stable and robust model, but may not be as accurate. KNN is based on the assumption that data points that are close to each other in the feature space are likely to belong to the same class. This makes KNN a useful algorithm for classification tasks where the decision boundary between classes is highly non-linear or difficult to model using other algorithms.

Multinomial Naive Bayes (MNB) is a statistical machine learning algorithm based on the Bayes theorem (Kim et al., 2006). MNB assumes that features are conditionally independent given the target class, estimates the probabilities of each class and the probabilities of each feature given the class, and uses these probabilities to make predictions.

A Multilayer Perceptron (MLP) is a type of artificial neural network (ANN) that is used for a variety of tasks, including classification and regression. An MLP consists of an input layer, one or more hidden layers, and an output layer. The input layer receives the inputs to the network, which are then processed and transformed by the hidden layers. The output layer produces the final output of the network (Hassan et al. 2016).

These ML methods were applied using the following tools and information sources:

* The Python 3.8 programming language. (Van Rossum & Drake, 2009).
* Sklearn – a Python library for ML methods. (Buitinck et al., 2013).
* Pandas – a Python library for data analysis. It provides data structures for efficiently storing large datasets and tools for working with them (McKinney, 2010).
* Thinker – a library to create a simple python GUI application.

I created feature matrices from the datasets, while the features were terms from the dataset with the highest TF-IDF grade.

TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure used in information retrieval and natural language processing (Ramos, 2003). It reflects the importance of a word in a document within a corpus of documents. The main idea behind TF-IDF is that a word that occurs frequently in a document but not in many other documents across the corpus is likely to be more important to that document than a word that occurs frequently across many documents. The term frequency (TF) of a word in a document is the number of times it appears in the document. The inverse document frequency (IDF) of a word is the logarithm of the number of documents in the corpus divided by the number of documents in which the word appears. The TF-IDF score for a word in a document is the product of its term frequency and inverse document frequency. I calculated TF-IDF values for word n-grams and char Ngrams in ranges 1-3.

# Methodology

My way of working was based on the 3 datasets I described. The goal was to train different models on the training dataset and select the best models according to the accuracy score.

For classic ML models, I worked as follows:

* In the first step, for each language, I created a TF-IDF table for all the n-grams in the language and tried to identify how many grams should be selected as model features. I chose the [1,2,3] word n-grams.
* I ran 7 classic classifiers - LR, MNB, KNN, RF, SVM, DT, MLP with varying amounts of features - 500, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000 and saw which amount of features gets the highest score for each model.
* I applied different pre-processing methods to the data - each method separately as well as the combinations of the methods and examined the level of improvement in each model and the effect of each pre-processing method. The best combination I found was lowercase the text, removing punctuation marks, and removing the phrase "subject: "at beginning of every mail in the first dataset. Removing the stopwords lead to lower accuracy. Lemmatization and stemming took a lot of time so I didn't check it for the whole dataset.
* The best models for each dataset were saved to the file system.
* I created a basic GUI for the system that allows entering an email, choosing a classifier and the model on which it is trained, and trying to classify the email using the trained model that will predict its labeling as spam or not. The GUI allows you to see the email after preprocessing as well as the score of each model to choose the desired one.

The project is built from the following files:

* data\_exploration.py - code that goes through the datasets and produces various charts to describe the information.
* data\_preperation.py - code that goes over the raw data, and performs various pre-processing functions according to the requirement and more DataFrame is ready for learning.
* train\_models.py - the code that runs all the models and trains them, and saves the results to the file system.
* gui.py - this code must be run. Creates the system's GUI and goes behind the scenes to the various functions to pull trained models and try and classify an email entered as input.
* verification.py – the code that runs all models to classify the dev set spam classification and then prints the accuracy of each dataset model.
* data - a folder that contains all the different datasets. There are 2 datasets in Basis numbered 1,2, and there is a combined dataset of both called 1+2. Also, I removed from each dataset about 20 emails that Model did not train on, which can be used to try the system.
* data\_exploration - a folder that contains all the figures and tables describing the different datasets
* trained\_models - a folder that contains all the trained models, the feature matrices built based on tf-idf, and CSV files containing the results of all the models as well as the outstanding models with accuracy, precision, recall, f1 indicators.

**It is important to note that all the files are documented in a very thorough way, you can understand what the role of each function is and what the main lines do in each function.**

# Program Verification

As explained above, I used 2 basic datasets and a third united dataset. Before I started training the models, I created 2 small datasets called "dev" that contain some emails from the original datasets that the models were not trained on, and in this way, I can check if the models were able to learn.

To validate the model, I created an automated code that pulls the best models from each dataset type and size, i.e. emails1 in trained sizes - 100, 500, and 1000 words in the trained model.

For each word in the dev database, I found its classification by each of the outstanding models and chose the most common classification among all, and compared the original classification of the word against the classification it received in each of the databases - emails1, emails2, emails1+2

The results I received can be seen in the next table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **text id** | **spam** | **source** | **emails1 classification** | **emails2 classification** | **emails12 classification** |
| 1 | 0 | emails1 | 0 | 0 | 0 |
| 2 | 1 | emails1 | 1 | 1 | 1 |
| 3 | 1 | emails1 | 0 | 1 | 1 |
| 4 | 1 | emails1 | 1 | 1 | 1 |
| 5 | 1 | emails1 | 1 | 0 | 1 |
| 6 | 0 | emails1 | 0 | 0 | 0 |
| 7 | 0 | emails1 | 0 | 1 | 0 |
| 8 | 0 | emails1 | 0 | 1 | 0 |
| 9 | 0 | emails1 | 0 | 1 | 0 |
| 10 | 1 | emails1 | 1 | 1 | 1 |
| 11 | 1 | emails1 | 1 | 1 | 1 |
| 12 | 1 | emails1 | 1 | 1 | 1 |
| 13 | 1 | emails1 | 1 | 1 | 1 |
| 14 | 1 | emails1 | 1 | 1 | 1 |
| 15 | 0 | emails1 | 0 | 1 | 0 |
| 16 | 0 | emails1 | 0 | 0 | 0 |
| 17 | 0 | emails1 | 0 | 0 | 0 |
| 18 | 0 | emails1 | 0 | 0 | 0 |
| 19 | 0 | emails2 | 1 | 0 | 0 |
| 20 | 0 | emails2 | 0 | 0 | 0 |
| 21 | 1 | emails2 | 1 | 1 | 1 |
| 22 | 1 | emails2 | 1 | 1 | 1 |
| 23 | 1 | emails2 | 0 | 0 | 0 |
| 24 | 0 | emails2 | 0 | 0 | 0 |
| 25 | 1 | emails2 | 1 | 1 | 1 |
| 26 | 1 | emails2 | 1 | 1 | 1 |
| 27 | 0 | emails2 | 1 | 0 | 0 |
| 28 | 0 | emails2 | 1 | 0 | 0 |
| 29 | 1 | emails2 | 1 | 1 | 1 |
| 30 | 0 | emails2 | 0 | 0 | 0 |
| 31 | 0 | emails2 | 1 | 0 | 0 |
| 32 | 0 | emails2 | 1 | 0 | 0 |
| 33 | 1 | emails2 | 1 | 1 | 1 |
| 34 | 1 | emails2 | 1 | 1 | 1 |

The accuracy reflected in the table is as follows:

* emails1 accuracy:  0.7941176470588235
* emails2 accuracy:  0.8235294117647058
* emails12 accuracy:  0.9705882352941176

As can be seen from these results, models trained on dataset emails1 classified spam emails that came from datasets 1 and 2, with a success rate of about 79%. Dataset emails2 was slightly better and the models trained on it classified the same emails with 82% success. It can be seen that the majority of the errors of the models trained on dataset #1 are on emails that came from dataset #2 and vice versa. That means clearly, each database excels at classifying emails that are similar to the emails it was trained on. Therefore, the significant improvement at the level of 97% comes precisely in the models that were adjusted on the 2 reservoirs together. This is a very significant success rate. This illustrates the power and needs for diverse repositories that enable a more general model.

**Examples of email classification from the dev database by the system can be seen in detail in Appendix A at the end of this report.**

# Results

An example of classification results can be seen in the previous chapter. In this chapter, I will present the results of the best models in each dataset and the score they received. For each model I trained, I took 75% of the dataset as training data, and 25% as test data with which I evaluated the accuracy of the model.

* The results of the best models of each classifier, for a dataset based on the emails1 pool of 100 emails:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **model** | **feature\_matrix** | **accuracy** | **precision** | **recall** | **f1\_score** |
| Decision Tree | word\_1\_2000 | 0.94 | 0.91428571 | 1 | 0.95522388 |
| KNN | word\_1\_1000 | 0.96 | 1 | 0.9375 | 0.96774194 |
| Logistic Regression | word\_1\_500 | 0.94 | 1 | 0.90625 | 0.95081967 |
| Multi-layer Perceptron | word\_1\_2000 | 0.98 | 1 | 0.96875 | 0.98412698 |
| Multinomial Naive Bayes | word\_1\_500 | 0.96 | 1 | 0.9375 | 0.96774194 |
| Random Forest | word\_1\_5000 | 0.98 | 0.96969697 | 1 | 0.98461538 |
| Support Vector Machine | word\_1\_500 | 0.96 | 1 | 0.9375 | 0.96774194 |

* The results of the best models of each classifier, for a dataset based on the emails2 pool of 100 emails:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **model** | **feature\_matrix** | **accuracy** | **precision** | **recall** | **f1\_score** |
| Decision Tree | word\_2\_100 | 0.9 | 0.95454545 | 0.84 | 0.89361702 |
| KNN | word\_1\_100 | 0.9 | 0.85714286 | 0.96 | 0.90566038 |
| Logistic Regression | word\_1\_1000 | 0.98 | 1 | 0.96 | 0.97959184 |
| Multi-layer Perceptron | word\_1\_500 | 0.98 | 1 | 0.96 | 0.97959184 |
| Multinomial Naive Bayes | word\_1\_500 | 0.98 | 1 | 0.96 | 0.97959184 |
| Random Forest | word\_1\_500 | 0.98 | 1 | 0.96 | 0.97959184 |
| Support Vector Machine | word\_1\_100 | 0.98 | 1 | 0.96 | 0.97959184 |

* The results of the best models of each classifier, for a dataset based on the emails1+2 pool of 100 emails:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **model** | **feature\_matrix** | **accuracy** | **precision** | **recall** | **f1\_score** |
| Decision Tree | word\_1\_8000 | 0.96 | 0.96551724 | 0.96551724 | 0.96551724 |
| KNN | word\_1\_500 | 0.94 | 0.96428571 | 0.93103448 | 0.94736842 |
| Logistic Regression | word\_1\_500 | 0.94 | 0.96428571 | 0.93103448 | 0.94736842 |
| Multi-layer Perceptron | word\_1\_3000 | 0.96 | 0.96551724 | 0.96551724 | 0.96551724 |
| Multinomial Naive Bayes | word\_1\_2000 | 0.98 | 1 | 0.96551724 | 0.98245614 |
| Random Forest | word\_1\_8000 | 0.9 | 0.9 | 0.93103448 | 0.91525424 |
| Support Vector Machine | word\_1\_2000 | 0.92 | 0.96296296 | 0.89655172 | 0.92857142 |

* The results of the best models of each classifier, for a dataset based on the emails1 pool of 500 emails:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **model** | **feature\_matrix** | **accuracy** | **precision** | **recall** | **f1\_score** |
| Decision Tree | word\_1\_500 | 0.948 | 0.92857143 | 0.96694215 | 0.94736842 |
| KNN | word\_1\_1000 | 0.92 | 0.88549618 | 0.95867769 | 0.92063492 |
| Logistic Regression | word\_1\_1000 | 0.976 | 0.96747967 | 0.98347107 | 0.97540983 |
| Multi-layer Perceptron | word\_1\_2000 | 0.988 | 0.99166667 | 0.98347107 | 0.98755187 |
| Multinomial Naive Bayes | word\_1\_2000 | 0.98 | 0.98333333 | 0.97520661 | 0.97925311 |
| Random Forest | word\_1\_500 | 0.968 | 0.94488189 | 0.99173554 | 0.96774194 |
| Support Vector Machine | word\_1\_500 | 0.984 | 0.97560976 | 0.99173554 | 0.98360656 |

* The results of the best models of each classifier, for a dataset based on the emails2 pool of 500 emails:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **model** | **feature\_matrix** | **accuracy** | **precision** | **recall** | **f1\_score** |
| Decision Tree | word\_1\_500 | 0.968 | 0.94890511 | 0.99236641 | 0.97014925 |
| KNN | word\_1\_500 | 0.932 | 0.91304348 | 0.96183206 | 0.93680297 |
| Logistic Regression | word\_1\_2000 | 0.984 | 0.99224806 | 0.97709924 | 0.98461539 |
| Multi-layer Perceptron | word\_1\_3000 | 0.996 | 0.99242424 | 1 | 0.99619772 |
| Multinomial Naive Bayes | word\_1\_2000 | 0.988 | 1 | 0.97709924 | 0.98841699 |
| Random Forest | word\_1\_500 | 0.988 | 0.99230769 | 0.98473282 | 0.98850574 |
| Support Vector Machine | word\_1\_1000 | 0.992 | 1 | 0.98473282 | 0.99230769 |

* The results of the best models of each classifier, for a dataset based on the emails1+2 pool of 500 emails:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **model** | **feature\_matrix** | **accuracy** | **precision** | **recall** | **f1\_score** |
| Decision Tree | word\_1\_3000 | 0.94 | 0.93548387 | 0.94308943 | 0.93927125 |
| KNN | word\_1\_1000 | 0.94 | 0.921875 | 0.95934959 | 0.94023904 |
| Logistic Regression | word\_1\_2000 | 0.984 | 0.976 | 0.99186992 | 0.98387097 |
| Multi-layer Perceptron | word\_2\_10000 | 0.992 | 0.984 | 1 | 0.99193548 |
| Multinomial Naive Bayes | word\_1\_4000 | 0.988 | 0.97619048 | 1 | 0.98795181 |
| Random Forest | word\_1\_6000 | 0.98 | 0.9609375 | 1 | 0.98007968 |
| Support Vector Machine | word\_1\_1000 | 0.988 | 0.99180328 | 0.98373984 | 0.9877551 |

* The results of the best models of each classifier, for a dataset based on the emails1 pool of 1000 emails:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **model** | **feature\_matrix** | **accuracy** | **precision** | **recall** | **f1\_score** |
| Decision Tree | word\_1\_1000 | 0.976 | 0.96062992 | 0.99186992 | 0.976 |
| KNN | word\_1\_2000 | 0.96 | 0.93461538 | 0.98780488 | 0.96047431 |
| Logistic Regression | word\_1\_2000 | 0.994 | 0.98795181 | 1 | 0.9939394 |
| Multi-layer Perceptron | word\_1\_2000 | 1 | 1 | 1 | 1 |
| Multinomial Naive Bayes | word\_1\_6000 | 0.996 | 0.99593496 | 0.99593496 | 0.99593496 |
| Random Forest | word\_1\_6000 | 0.988 | 0.98 | 0.99593496 | 0.98790323 |
| Support Vector Machine | word\_1\_2000 | 0.998 | 0.99595142 | 1 | 0.9979716 |

* The results of the best models of each classifier, for a dataset based on the emails2 pool of 1000 emails:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **model** | **feature\_matrix** | **accuracy** | **precision** | **recall** | **f1\_score** |
| Decision Tree | word\_1\_1000 | 0.978 | 0.96047431 | 0.99590164 | 0.9778672 |
| KNN | word\_1\_100 | 0.948 | 0.92913386 | 0.96721311 | 0.94779116 |
| Logistic Regression | word\_1\_3000 | 0.99 | 0.99585062 | 0.98360656 | 0.98969072 |
| Multi-layer Perceptron | word\_1\_4000 | 0.998 | 0.99591837 | 1 | 0.99795501 |
| Multinomial Naive Bayes | word\_1\_4000 | 0.996 | 0.99590164 | 0.99590164 | 0.99590164 |
| Random Forest | word\_1\_100 | 0.998 | 1 | 0.99590164 | 0.99794661 |
| Support Vector Machine | word\_1\_2000 | 0.996 | 1 | 0.99180328 | 0.99588477 |

* The results of the best models of each classifier, for a dataset based on the emails1+2 pool of 1000 emails:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **model** | **feature\_matrix** | **accuracy** | **precision** | **recall** | **f1\_score** |
| Decision Tree | word\_1\_2000 | 0.94 | 0.94693878 | 0.93172691 | 0.93927126 |
| KNN | word\_1\_500 | 0.934 | 0.93902439 | 0.92771084 | 0.93333333 |
| Logistic Regression | word\_1\_6000 | 0.98 | 0.9877551 | 0.97188755 | 0.97975708 |
| Multi-layer Perceptron | word\_1\_6000 | 0.99 | 0.99593496 | 0.98393574 | 0.98989899 |
| Multinomial Naive Bayes | word\_1\_6000 | 0.982 | 0.99180328 | 0.97188755 | 0.98174442 |
| Random Forest | word\_1\_2000 | 0.982 | 0.98 | 0.98393574 | 0.98196393 |
| Support Vector Machine | word\_1\_2000 | 0.984 | 0.99183673 | 0.97590361 | 0.98380566 |

An example of a full results table can be seen in appendix B. all the full results tables are available in my GitHub account, in the project repository[[1]](#footnote-1) in the directory "trained models". Or can be received by running "train\_models.py".

# Discussion

Further to all the data presented in the last 2 chapters, some important conclusions can be distinguished:

* It seems that emails that came from emails1 and were not trained on - are correctly classified by a model trained on emails1 because the writing style is probably the same as the entire dataset and there is an overlap between key features in the emails. And the evidence is that I achieved very high classification percentages in each of the datasets separately. The difficulty starts when you take an email in a different writing style and then it seems that sometimes one database does not correctly classify emails that came from another database.
* The idea of combining the 2 datasets and creating a third dataset did create a general model that was able to correctly classify emails from each of the databases. And even handled well with new emails - for example, an email I wrote myself, or an email I received and Gmail classified as spam.
* As part of the test sample I conducted, it seems that the errors between the datasets are more common in non-spam mail, but the model is classified as spam. It could be that keywords that indicate spam in one database indicate ham in another database.
* In the 100-sized datasets, it seems that the models trained on emails2 were better, reaching an accuracy of 0.98-0.9 in each classifier. Emails1 models receive between 0.94-0.98 of this size, and correspondingly also the combined dataset emails1+2. It seems that emails2 is built in a way that makes it easier for the models.
* At a 500-sized dataset, emails1 gets slightly better results than the smaller dataset emails1 sized 100. But dataset 2 increases in the level of accuracy except for KNN which probably multi-neighbors harms its accuracy (too general). The integrated dataset improves and I guess more records allow it to learn the complex pattern better, and a few emails are not enough.
* In the 1000-sized dataset, all the datasets seem to achieve the best results among the many models (except KNN). Multiple examples allow better quality learning of this task.
* Among the classifier types, overwhelmingly, MLP achieves the highest accuracy. In each table you can see another outstanding model - sometimes RF, sometimes MNB, and sometimes SVM. The DT and KNN models are the weakest and this is not surprising. It is known that deep learning models have been a breakthrough when it comes to text classification, and even a basic and outdated model like MLP shows a good job.
* Concerning the number of features - the larger the dataset, the more features are needed because more types of emails need to be known. For example, in datasets of size 100, it can be seen that the outstanding models range from 500 to 2000 features. In datasets of 500 mails, there is an increase to 3000, 4000, and even 10000 features in the integrated dataset. And in the large datasets of 1000 words in the dataset, you can see large quantities of features but not very much, a maximum of 6000, and there are also quite a few models that excelled in low quantities of features of 1000-2000. It is not possible to point to one trend in the number of features in a relation to the size of the dataset.

# Conclusions and Future Work

In this paper, I describe my final project in the AI course – a system for Email spam detection.

I applied feature creation of word Ngrams based on tf-idf, 6 supervised machine learning methods and 1 deep learning method. Through various classification methods, I built a relatively strong system for assessing the spam classification of emails in English.

There are various ideas for future research regarding email spam detection:

* Additional repositories can be found and the accuracy of the model can be extended by making it more general.
* It is possible to improve existing ML models by performing parameter tuning, that is, choosing different parameters and comparing them to each model, rather than using the default Python setting.
* Features can be added using dictionaries containing words identified with spam.
* Advanced deep learning models can be trained - such as RNN, and LSTM networks that are particularly suitable for texts.
* It is possible to combine word embedding methods that were a breakthrough in the NLP world such as word2vec, BERT, and more.
* Oversampling methods can be used to increase the email pool. datasets 1-2 are originally unbalanced and contain much more non-spam emails. By oversampling, I can create a balanced pool by creating new spam samples and possibly improve the accuracy of the models.
* A combination of reinforcement learning tools allows the user to teach the system about an email that has been classified - whether it was successful or not. And it can improve the existing models.
* Using better computational resources to train much larger models.

In conclusion, I enjoyed doing the project. I learned a lot about the world of text classification and ended up with good and successful results. In recent years I have experienced the great benefits of automatically classifying emails as spam. I have a lot of emails classified as spam and do not litter the inbox. It seems that this type of model was significant in the past and will continue to be significant for improving the quality of life and use of emails in the modern world.

# References

1. Pang Bo, Lillian Lee, and Shivakumar Vaithyanathan. 2002. [Thumbs up? Sentiment Classification using Machine Learning Techniques](https://aclanthology.org/W02-1011). In Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002), pages 79–86. Association for Computational Linguistics.
2. Pang Bo, and Lillian Lee, 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In: Proceedings of the 43rd annual meeting on association for computational linguistics. pp. 115–124. Association for Computational Linguistics.
3. Leo Breiman. 1996. Bagging predictors. Machine learning 24(2), 123-140.
4. Leo Breiman. 2001. Random forests. Machine learning 45(1), 5-32.
5. Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, & Gaël Varoquaux, 2013. API design for machine learning software: experiences from the scikit-learn project. In ECML PKDD Workshop: Languages for Data Mining and Machine Learning (pp. 108–122).
6. Chih-Chung Chang and Chih-Jen Lin, 2011. LIBSVM: A library for support vector machines. ACM transactions on intelligent systems and technology (TIST) 2(3), 1-27.
7. Corinna Cortes and Vladimir Vapnik, 1995. Support-vector networks. Machine learning 20.3 : 273-297.
8. David R. Cox. 1958. The regression analysis of binary sequences, Journal of the Royal Statistical Society: Series B (Methodological), 20, 215–232.
9. Guo, G., Wang, H., Bell, D., Bi, Y., & Greer, K. (2003). KNN model-based approach in classification. In On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings (pp. 986-996). Springer Berlin Heidelberg.‏
10. Yaakov HaCohen-Kerner, Ariel Kass, and Ariel Peretz, 2008. Combined one sense disambiguation of abbreviations. In Proceedings of ACL-08: HLT, Short Papers, Association for Computational Linguistics, pages 61-64, Columbus, Ohio, Association for Computational Linguistics. URL: https://aclanthology.org/P08-2.
11. Yaakov HaCohen-Kerner, Yair Yigal, and Daniel Miller. 2019. The impact of Preprocessing on Classification of Mental Disorders, in Proc. of the 19th Industrial Conference on Data Mining, (ICDM 2019), New York.
12. Ramchoun Hassan, Mohammed Amine Janati Idrissi, Youssef Ghanou, and Mohamed Ettaouil, 2016. "Multilayer Perceptron: Architecture Optimization and Training." International Journal of Interactive Multimedia and Artificial Intelligence 4, no. 1 (2016): 26+.
13. David W. Hosmer, Stanley Lemeshow, and Rodney X. Sturdivant. 2013. Applied logistic regression, Vol. 398, John Wiley & Sons.
14. Agarwal, Kriti, and Tarun Kumar. "Email spam detection using integrated approach of Naïve Bayes and particle swarm optimization." 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS). IEEE, 2018.‏
15. Kumar, Nikhil, and Sanket Sonowal. "Email spam detection using machine learning algorithms." 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA). IEEE, 2020.‏
16. Yan-yan Song and Ying Lu. 2015. Decision tree methods: applications for classification and prediction. Shanghai archives of psychiatry, 27(2), 130.
17. Sang-Bum Kim, Kyoung-Soo Han, Hae-Chang Rim and Sung Hyon Myaeng, 2006. "Some Effective Techniques for Naive Bayes Text Classification," in IEEE Transactions on Knowledge and Data Engineering, vol. 18, no. 11, pp. 1457-1466, Nov. 2006, doi: 10.1109/TKDE.2006.180.
18. Wes McKinney, 2010. Data Structures for Statistical Computing in Python. In Proceedings of the 9th Python in Science Conference (pp. 56 - 61 ).
19. Ramos, Juan. "Using tf-idf to determine word relevance in document queries." Proceedings of the first instructional conference on machine learning. Vol. 242. No. 1. 2003.‏
20. Olatunji, Sunday Olusanya. "Improved email spam detection model based on support vector machines." Neural Computing and Applications 31 (2019): 691-699.‏
21. Guido Van Rossum & Fred Drake, 2009. Python 3 Reference Manual. CreateSpace.

# Appendices

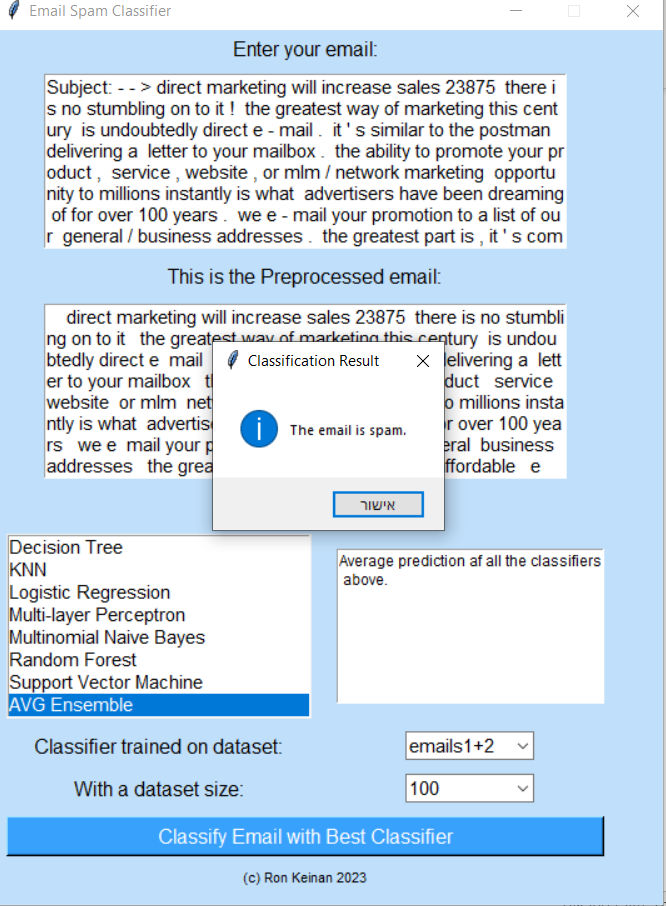
**A Examples of dev classifications with the GUI**

* First email - email that came from database #1 and the models were not trained on it. The email is classified as 1, which means spam:

The classifications of chosen models were as follows -

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **data size** | **classifier** | **classification** |
| Emails1 | 100 | AVG ensemble | spam |
| Emails2 | 100 | AVG ensemble | spam |
| Emails1+2 | 100 | AVG ensemble | spam |

תמונה שמכילה טקסט

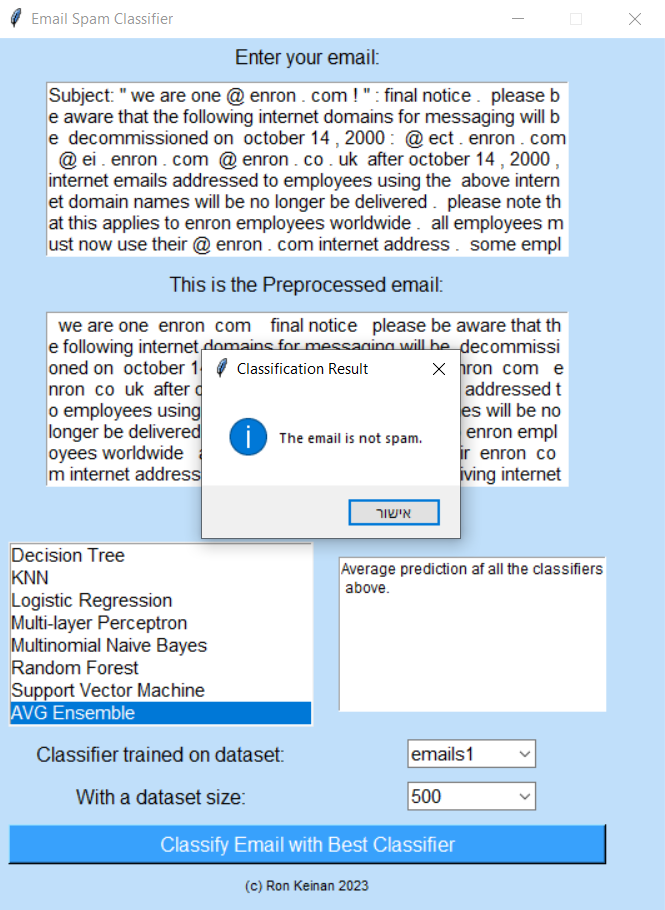
התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט

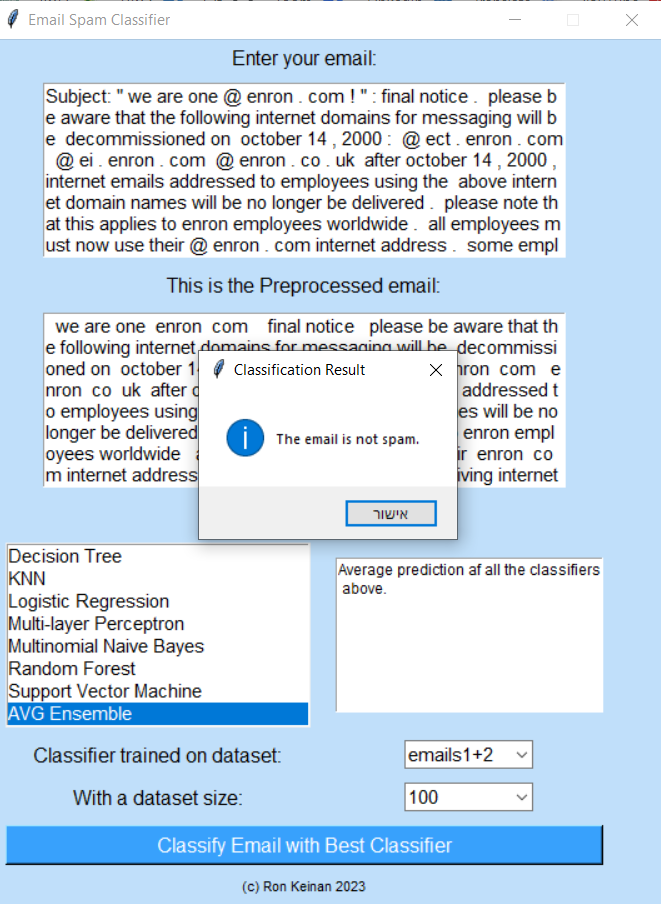
התיאור נוצר באופן אוטומטי

* Second email - email that came from database #1 and the models were not trained on it. The email is classified as 0, meaning not spam:

The classifications of chosen models were as follows -

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **data size** | **classifier** | **classification** |
| Emails1 | 500 | AVG ensemble | Not spam |
| Emails2 | 500 | AVG ensemble | Spam |
| Emails1+2 | 100 | AVG ensemble | Not spam |

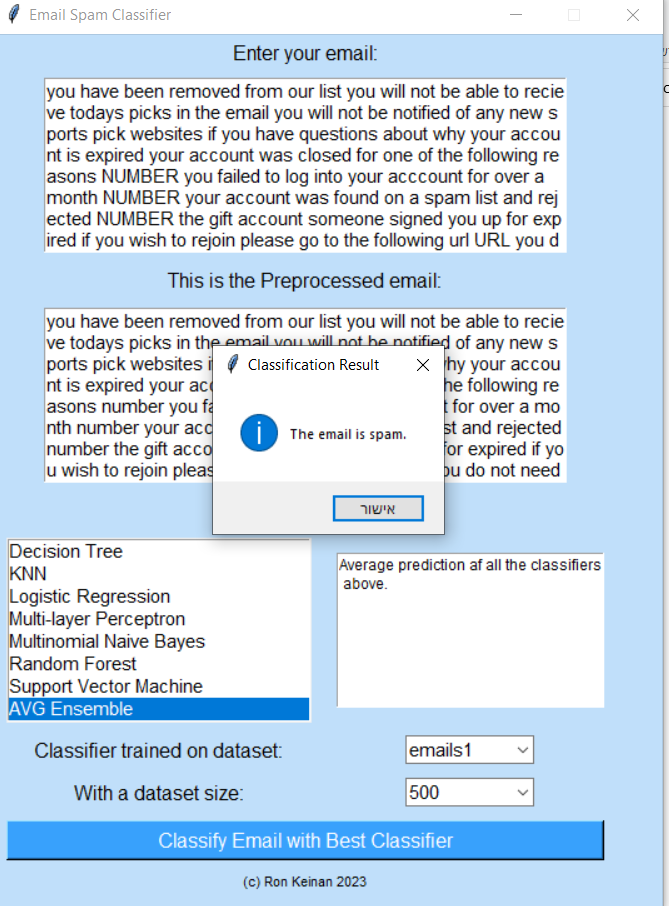
תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

* Third email - email that came from database #2 and the models were not trained on it. The email is classified as 1, which means spam:

The classifications of chosen models were as follows -

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **data size** | **classifier** | **classification** |
| Emails1 | 500 | AVG ensemble | spam |
| Emails2 | 500 | AVG ensemble | spam |
| Emails1+2 | 500 | AVG ensemble | spam |

תמונה שמכילה טקסט

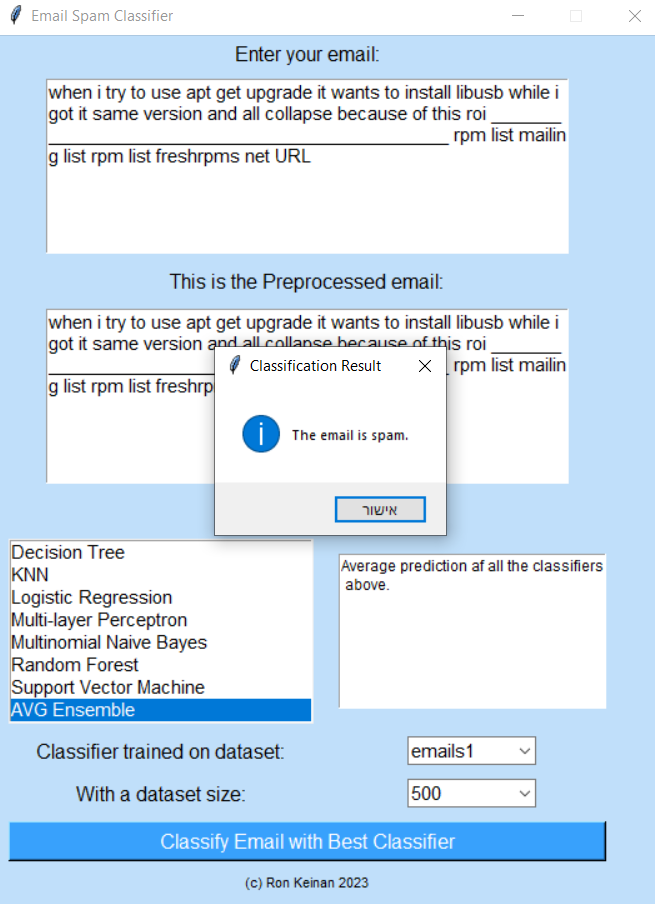
התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

* Fourth email - email that came from database #2 and the models were not trained on it. The email is classified as 0, meaning not spam:

The classifications of chosen models were as follows -

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **data size** | **classifier** | **classification** |
| Emails1 | 500 | AVG ensemble | Spam |
| Emails2 | 500 | AVG ensemble | Not spam |
| Emails1+2 | 500 | AVG ensemble | Not spam |

תמונה שמכילה טקסט

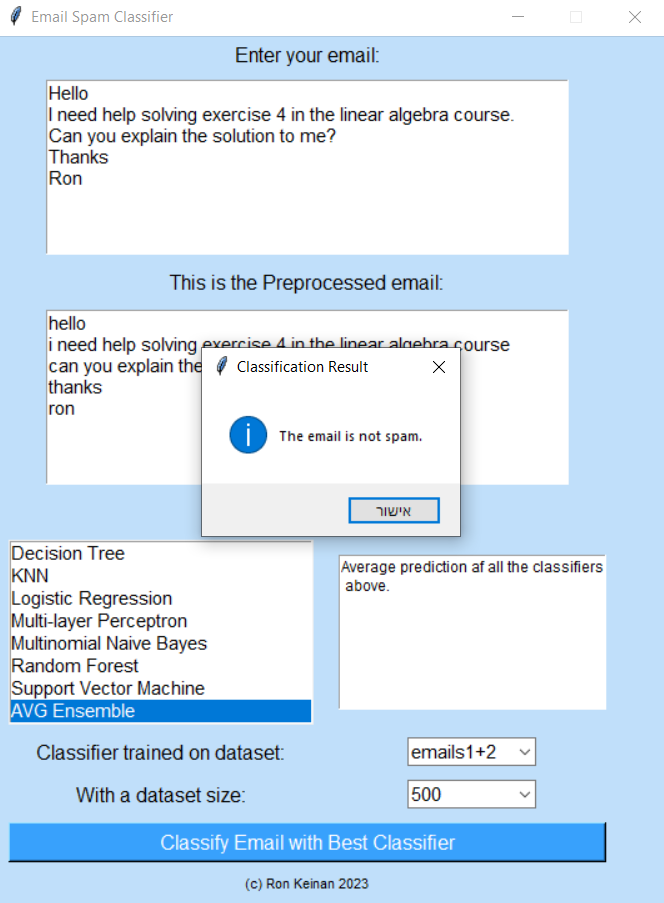
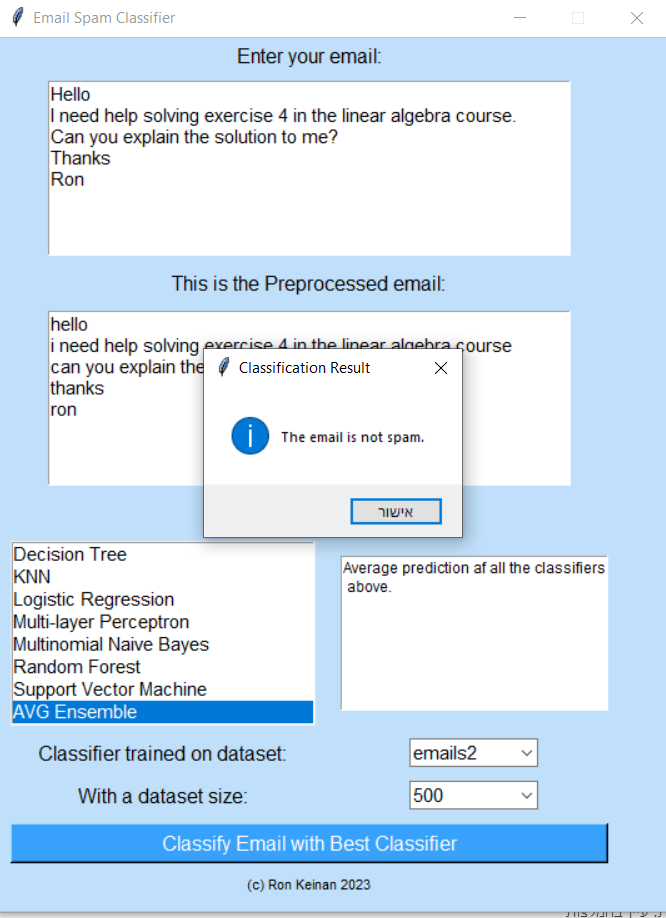
התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

* Fifth email - just an email I made up and is not spam:

The classifications of chosen models were as follows -

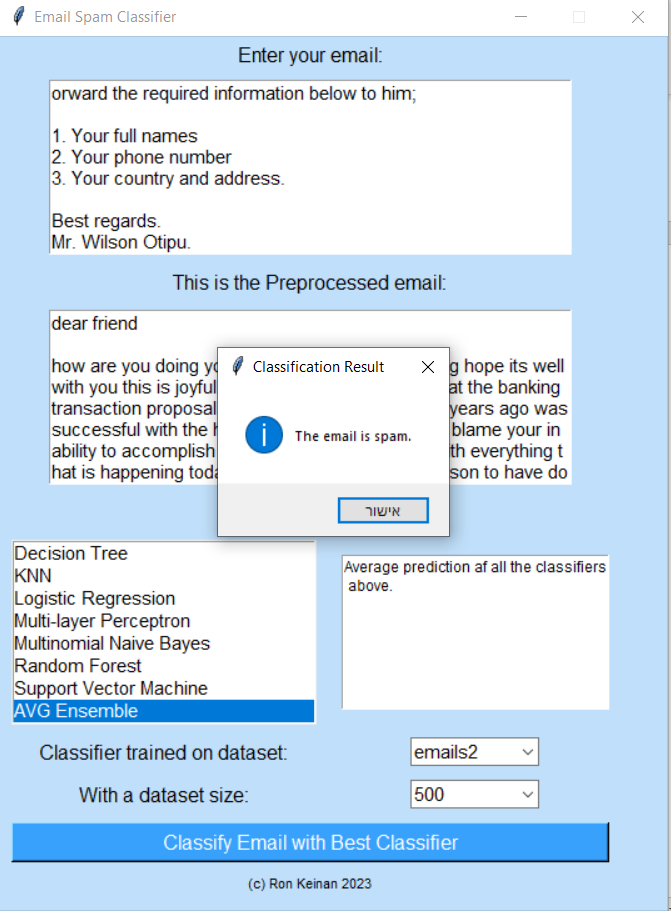
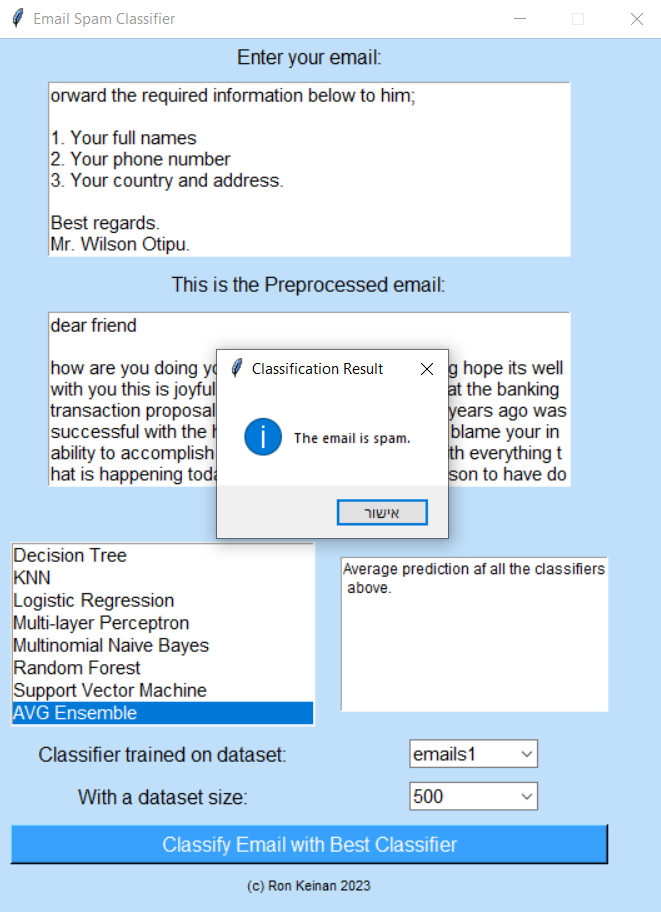
|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **data size** | **classifier** | **classification** |
| Emails1 | 500 | AVG ensemble | Not spam |
| Emails2 | 500 | AVG ensemble | Not spam |
| Emails1+2 | 500 | AVG ensemble | Not spam |



* Sixth email - an email I took from my inbox and which Gmail classified as spam:

The classifications of chosen models were as follows -

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **data size** | **classifier** | **classification** |
| Emails1 | 500 | AVG ensemble | Spam |
| Emails2 | 500 | AVG ensemble | Spam |
| Emails1+2 | 500 | AVG ensemble | Spam |

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

**B Results of all models scores**

Models results in example – training on dataset emails1, size 100:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **model** | **feature\_matrix** | **accuracy** | **precision** | **recall** | **f1\_score** |
| KNN | word\_1\_100 | 0.84 | 0.9 | 0.84375 | 0.87096774 |
| KNN | word\_2\_100 | 0.78 | 0.76923077 | 0.9375 | 0.84507042 |
| KNN | word\_3\_100 | 0.48 | 0.65 | 0.40625 | 0.5 |
| KNN | word\_1\_500 | 0.92 | 0.9375 | 0.9375 | 0.9375 |
| KNN | word\_2\_500 | 0.66 | 0.65306122 | 1 | 0.79012345 |
| KNN | word\_3\_500 | 0.66 | 0.65957447 | 0.96875 | 0.78481013 |
| KNN | word\_1\_1000 | 0.96 | 1 | 0.9375 | 0.96774194 |
| KNN | word\_2\_1000 | 0.86 | 0.93103448 | 0.84375 | 0.8852459 |
| KNN | word\_3\_1000 | 0.76 | 0.77777778 | 0.875 | 0.82352941 |
| KNN | word\_1\_2000 | 0.9 | 0.93548387 | 0.90625 | 0.92063492 |
| KNN | word\_2\_2000 | 0.88 | 0.96428571 | 0.84375 | 0.9 |
| KNN | word\_3\_2000 | 0.76 | 0.77777778 | 0.875 | 0.82352941 |
| KNN | word\_1\_3000 | 0.9 | 0.93548387 | 0.90625 | 0.92063492 |
| KNN | word\_2\_3000 | 0.88 | 0.96428571 | 0.84375 | 0.9 |
| KNN | word\_3\_3000 | 0.76 | 0.77777778 | 0.875 | 0.82352941 |
| KNN | word\_1\_4000 | 0.9 | 0.93548387 | 0.90625 | 0.92063492 |
| KNN | word\_2\_4000 | 0.88 | 0.96428571 | 0.84375 | 0.9 |
| KNN | word\_3\_4000 | 0.76 | 0.77777778 | 0.875 | 0.82352941 |
| KNN | word\_1\_5000 | 0.9 | 0.93548387 | 0.90625 | 0.92063492 |
| KNN | word\_2\_5000 | 0.88 | 0.96428571 | 0.84375 | 0.9 |
| KNN | word\_3\_5000 | 0.76 | 0.77777778 | 0.875 | 0.82352941 |
| KNN | word\_1\_6000 | 0.9 | 0.93548387 | 0.90625 | 0.92063492 |
| KNN | word\_2\_6000 | 0.88 | 0.96428571 | 0.84375 | 0.9 |
| KNN | word\_3\_6000 | 0.76 | 0.77777778 | 0.875 | 0.82352941 |
| KNN | word\_1\_7000 | 0.9 | 0.93548387 | 0.90625 | 0.92063492 |
| KNN | word\_2\_7000 | 0.88 | 0.96428571 | 0.84375 | 0.9 |
| KNN | word\_3\_7000 | 0.76 | 0.77777778 | 0.875 | 0.82352941 |
| KNN | word\_1\_8000 | 0.9 | 0.93548387 | 0.90625 | 0.92063492 |
| KNN | word\_2\_8000 | 0.88 | 0.96428571 | 0.84375 | 0.9 |
| KNN | word\_3\_8000 | 0.76 | 0.77777778 | 0.875 | 0.82352941 |
| KNN | word\_1\_9000 | 0.9 | 0.93548387 | 0.90625 | 0.92063492 |
| KNN | word\_2\_9000 | 0.88 | 0.96428571 | 0.84375 | 0.9 |
| KNN | word\_3\_9000 | 0.76 | 0.77777778 | 0.875 | 0.82352941 |
| KNN | word\_1\_10000 | 0.9 | 0.93548387 | 0.90625 | 0.92063492 |
| KNN | word\_2\_10000 | 0.88 | 0.96428571 | 0.84375 | 0.9 |
| KNN | word\_3\_10000 | 0.76 | 0.77777778 | 0.875 | 0.82352941 |
| Logistic Regression | word\_1\_100 | 0.9 | 0.96551724 | 0.875 | 0.91803279 |
| Logistic Regression | word\_2\_100 | 0.86 | 0.90322581 | 0.875 | 0.88888889 |
| Logistic Regression | word\_3\_100 | 0.92 | 0.88888889 | 1 | 0.94117647 |
| Logistic Regression | word\_1\_500 | 0.94 | 1 | 0.90625 | 0.95081967 |
| Logistic Regression | word\_2\_500 | 0.9 | 0.96551724 | 0.875 | 0.91803279 |
| Logistic Regression | word\_3\_500 | 0.92 | 0.9375 | 0.9375 | 0.9375 |
| Logistic Regression | word\_1\_1000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Logistic Regression | word\_2\_1000 | 0.86 | 1 | 0.78125 | 0.87719298 |
| Logistic Regression | word\_3\_1000 | 0.78 | 0.95652174 | 0.6875 | 0.8 |
| Logistic Regression | word\_1\_2000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Logistic Regression | word\_2\_2000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Logistic Regression | word\_3\_2000 | 0.78 | 0.95652174 | 0.6875 | 0.8 |
| Logistic Regression | word\_1\_3000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Logistic Regression | word\_2\_3000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Logistic Regression | word\_3\_3000 | 0.78 | 0.95652174 | 0.6875 | 0.8 |
| Logistic Regression | word\_1\_4000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Logistic Regression | word\_2\_4000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Logistic Regression | word\_3\_4000 | 0.78 | 0.95652174 | 0.6875 | 0.8 |
| Logistic Regression | word\_1\_5000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Logistic Regression | word\_2\_5000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Logistic Regression | word\_3\_5000 | 0.78 | 0.95652174 | 0.6875 | 0.8 |
| Logistic Regression | word\_1\_6000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Logistic Regression | word\_2\_6000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Logistic Regression | word\_3\_6000 | 0.78 | 0.95652174 | 0.6875 | 0.8 |
| Logistic Regression | word\_1\_7000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Logistic Regression | word\_2\_7000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Logistic Regression | word\_3\_7000 | 0.78 | 0.95652174 | 0.6875 | 0.8 |
| Logistic Regression | word\_1\_8000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Logistic Regression | word\_2\_8000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Logistic Regression | word\_3\_8000 | 0.78 | 0.95652174 | 0.6875 | 0.8 |
| Logistic Regression | word\_1\_9000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Logistic Regression | word\_2\_9000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Logistic Regression | word\_3\_9000 | 0.78 | 0.95652174 | 0.6875 | 0.8 |
| Logistic Regression | word\_1\_10000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Logistic Regression | word\_2\_10000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Logistic Regression | word\_3\_10000 | 0.78 | 0.95652174 | 0.6875 | 0.8 |
| Multinomial Naive Bayes | word\_1\_100 | 0.94 | 0.96774194 | 0.9375 | 0.95238095 |
| Multinomial Naive Bayes | word\_2\_100 | 0.78 | 0.86206897 | 0.78125 | 0.81967213 |
| Multinomial Naive Bayes | word\_3\_100 | 0.66 | 0.94117647 | 0.5 | 0.65306122 |
| Multinomial Naive Bayes | word\_1\_500 | 0.96 | 1 | 0.9375 | 0.96774194 |
| Multinomial Naive Bayes | word\_2\_500 | 0.82 | 1 | 0.71875 | 0.83636364 |
| Multinomial Naive Bayes | word\_3\_500 | 0.7 | 1 | 0.53125 | 0.69387755 |
| Multinomial Naive Bayes | word\_1\_1000 | 0.86 | 1 | 0.78125 | 0.87719298 |
| Multinomial Naive Bayes | word\_2\_1000 | 0.82 | 1 | 0.71875 | 0.83636364 |
| Multinomial Naive Bayes | word\_3\_1000 | 0.68 | 0.94444444 | 0.53125 | 0.68 |
| Multinomial Naive Bayes | word\_1\_2000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Multinomial Naive Bayes | word\_2\_2000 | 0.8 | 1 | 0.6875 | 0.81481481 |
| Multinomial Naive Bayes | word\_3\_2000 | 0.68 | 0.94444444 | 0.53125 | 0.68 |
| Multinomial Naive Bayes | word\_1\_3000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Multinomial Naive Bayes | word\_2\_3000 | 0.8 | 1 | 0.6875 | 0.81481481 |
| Multinomial Naive Bayes | word\_3\_3000 | 0.68 | 0.94444444 | 0.53125 | 0.68 |
| Multinomial Naive Bayes | word\_1\_4000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Multinomial Naive Bayes | word\_2\_4000 | 0.8 | 1 | 0.6875 | 0.81481481 |
| Multinomial Naive Bayes | word\_3\_4000 | 0.68 | 0.94444444 | 0.53125 | 0.68 |
| Multinomial Naive Bayes | word\_1\_5000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Multinomial Naive Bayes | word\_2\_5000 | 0.8 | 1 | 0.6875 | 0.81481481 |
| Multinomial Naive Bayes | word\_3\_5000 | 0.68 | 0.94444444 | 0.53125 | 0.68 |
| Multinomial Naive Bayes | word\_1\_6000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Multinomial Naive Bayes | word\_2\_6000 | 0.8 | 1 | 0.6875 | 0.81481481 |
| Multinomial Naive Bayes | word\_3\_6000 | 0.68 | 0.94444444 | 0.53125 | 0.68 |
| Multinomial Naive Bayes | word\_1\_7000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Multinomial Naive Bayes | word\_2\_7000 | 0.8 | 1 | 0.6875 | 0.81481481 |
| Multinomial Naive Bayes | word\_3\_7000 | 0.68 | 0.94444444 | 0.53125 | 0.68 |
| Multinomial Naive Bayes | word\_1\_8000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Multinomial Naive Bayes | word\_2\_8000 | 0.8 | 1 | 0.6875 | 0.81481481 |
| Multinomial Naive Bayes | word\_3\_8000 | 0.68 | 0.94444444 | 0.53125 | 0.68 |
| Multinomial Naive Bayes | word\_1\_9000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Multinomial Naive Bayes | word\_2\_9000 | 0.8 | 1 | 0.6875 | 0.81481481 |
| Multinomial Naive Bayes | word\_3\_9000 | 0.68 | 0.94444444 | 0.53125 | 0.68 |
| Multinomial Naive Bayes | word\_1\_10000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Multinomial Naive Bayes | word\_2\_10000 | 0.8 | 1 | 0.6875 | 0.81481481 |
| Multinomial Naive Bayes | word\_3\_10000 | 0.68 | 0.94444444 | 0.53125 | 0.68 |
| Support Vector Machine | word\_1\_100 | 0.9 | 0.96551724 | 0.875 | 0.91803279 |
| Support Vector Machine | word\_2\_100 | 0.86 | 0.90322581 | 0.875 | 0.88888889 |
| Support Vector Machine | word\_3\_100 | 0.9 | 0.90909091 | 0.9375 | 0.92307692 |
| Support Vector Machine | word\_1\_500 | 0.96 | 1 | 0.9375 | 0.96774194 |
| Support Vector Machine | word\_2\_500 | 0.84 | 1 | 0.75 | 0.85714286 |
| Support Vector Machine | word\_3\_500 | 0.82 | 0.89655172 | 0.8125 | 0.85245901 |
| Support Vector Machine | word\_1\_1000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Support Vector Machine | word\_2\_1000 | 0.82 | 1 | 0.71875 | 0.83636364 |
| Support Vector Machine | word\_3\_1000 | 0.8 | 0.89285714 | 0.78125 | 0.83333333 |
| Support Vector Machine | word\_1\_2000 | 0.94 | 1 | 0.90625 | 0.95081967 |
| Support Vector Machine | word\_2\_2000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Support Vector Machine | word\_3\_2000 | 0.8 | 0.89285714 | 0.78125 | 0.83333333 |
| Support Vector Machine | word\_1\_3000 | 0.94 | 1 | 0.90625 | 0.95081967 |
| Support Vector Machine | word\_2\_3000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Support Vector Machine | word\_3\_3000 | 0.8 | 0.89285714 | 0.78125 | 0.83333333 |
| Support Vector Machine | word\_1\_4000 | 0.94 | 1 | 0.90625 | 0.95081967 |
| Support Vector Machine | word\_2\_4000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Support Vector Machine | word\_3\_4000 | 0.8 | 0.89285714 | 0.78125 | 0.83333333 |
| Support Vector Machine | word\_1\_5000 | 0.94 | 1 | 0.90625 | 0.95081967 |
| Support Vector Machine | word\_2\_5000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Support Vector Machine | word\_3\_5000 | 0.8 | 0.89285714 | 0.78125 | 0.83333333 |
| Support Vector Machine | word\_1\_6000 | 0.94 | 1 | 0.90625 | 0.95081967 |
| Support Vector Machine | word\_2\_6000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Support Vector Machine | word\_3\_6000 | 0.8 | 0.89285714 | 0.78125 | 0.83333333 |
| Support Vector Machine | word\_1\_7000 | 0.94 | 1 | 0.90625 | 0.95081967 |
| Support Vector Machine | word\_2\_7000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Support Vector Machine | word\_3\_7000 | 0.8 | 0.89285714 | 0.78125 | 0.83333333 |
| Support Vector Machine | word\_1\_8000 | 0.94 | 1 | 0.90625 | 0.95081967 |
| Support Vector Machine | word\_2\_8000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Support Vector Machine | word\_3\_8000 | 0.8 | 0.89285714 | 0.78125 | 0.83333333 |
| Support Vector Machine | word\_1\_9000 | 0.94 | 1 | 0.90625 | 0.95081967 |
| Support Vector Machine | word\_2\_9000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Support Vector Machine | word\_3\_9000 | 0.8 | 0.89285714 | 0.78125 | 0.83333333 |
| Support Vector Machine | word\_1\_10000 | 0.94 | 1 | 0.90625 | 0.95081967 |
| Support Vector Machine | word\_2\_10000 | 0.84 | 1 | 0.75 | 0.85714286 |
| Support Vector Machine | word\_3\_10000 | 0.8 | 0.89285714 | 0.78125 | 0.83333333 |
| Decision Tree | word\_1\_100 | 0.84 | 0.9 | 0.84375 | 0.87096774 |
| Decision Tree | word\_2\_100 | 0.8 | 0.86666667 | 0.8125 | 0.83870968 |
| Decision Tree | word\_3\_100 | 0.88 | 0.84210526 | 1 | 0.91428571 |
| Decision Tree | word\_1\_500 | 0.84 | 0.9 | 0.84375 | 0.87096774 |
| Decision Tree | word\_2\_500 | 0.74 | 0.88 | 0.6875 | 0.77192982 |
| Decision Tree | word\_3\_500 | 0.88 | 0.86111111 | 0.96875 | 0.91176471 |
| Decision Tree | word\_1\_1000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Decision Tree | word\_2\_1000 | 0.74 | 0.82758621 | 0.75 | 0.78688525 |
| Decision Tree | word\_3\_1000 | 0.9 | 0.88571429 | 0.96875 | 0.92537314 |
| Decision Tree | word\_1\_2000 | 0.94 | 0.91428571 | 1 | 0.95522388 |
| Decision Tree | word\_2\_2000 | 0.78 | 0.86206897 | 0.78125 | 0.81967213 |
| Decision Tree | word\_3\_2000 | 0.88 | 0.90625 | 0.90625 | 0.90625 |
| Decision Tree | word\_1\_3000 | 0.9 | 0.90909091 | 0.9375 | 0.92307692 |
| Decision Tree | word\_2\_3000 | 0.76 | 0.85714286 | 0.75 | 0.8 |
| Decision Tree | word\_3\_3000 | 0.84 | 0.85294118 | 0.90625 | 0.87878788 |
| Decision Tree | word\_1\_4000 | 0.94 | 0.91428571 | 1 | 0.95522388 |
| Decision Tree | word\_2\_4000 | 0.8 | 0.86666667 | 0.8125 | 0.83870968 |
| Decision Tree | word\_3\_4000 | 0.88 | 0.86111111 | 0.96875 | 0.91176471 |
| Decision Tree | word\_1\_5000 | 0.86 | 0.90322581 | 0.875 | 0.88888889 |
| Decision Tree | word\_2\_5000 | 0.82 | 0.89655172 | 0.8125 | 0.85245901 |
| Decision Tree | word\_3\_5000 | 0.88 | 0.86111111 | 0.96875 | 0.91176471 |
| Decision Tree | word\_1\_6000 | 0.94 | 0.91428571 | 1 | 0.95522388 |
| Decision Tree | word\_2\_6000 | 0.76 | 0.88461538 | 0.71875 | 0.79310345 |
| Decision Tree | word\_3\_6000 | 0.88 | 0.86111111 | 0.96875 | 0.91176471 |
| Decision Tree | word\_1\_7000 | 0.9 | 0.90909091 | 0.9375 | 0.92307692 |
| Decision Tree | word\_2\_7000 | 0.8 | 0.89285714 | 0.78125 | 0.83333333 |
| Decision Tree | word\_3\_7000 | 0.82 | 0.82857143 | 0.90625 | 0.86567164 |
| Decision Tree | word\_1\_8000 | 0.88 | 0.90625 | 0.90625 | 0.90625 |
| Decision Tree | word\_2\_8000 | 0.76 | 0.83333333 | 0.78125 | 0.80645161 |
| Decision Tree | word\_3\_8000 | 0.86 | 0.83783784 | 0.96875 | 0.89855073 |
| Decision Tree | word\_1\_9000 | 0.88 | 0.90625 | 0.90625 | 0.90625 |
| Decision Tree | word\_2\_9000 | 0.76 | 0.85714286 | 0.75 | 0.8 |
| Decision Tree | word\_3\_9000 | 0.86 | 0.83783784 | 0.96875 | 0.89855073 |
| Decision Tree | word\_1\_10000 | 0.86 | 0.90322581 | 0.875 | 0.88888889 |
| Decision Tree | word\_2\_10000 | 0.78 | 0.88888889 | 0.75 | 0.81355932 |
| Decision Tree | word\_3\_10000 | 0.86 | 0.87878788 | 0.90625 | 0.89230769 |
| Random Forest | word\_1\_100 | 0.94 | 0.93939394 | 0.96875 | 0.95384615 |
| Random Forest | word\_2\_100 | 0.82 | 0.89655172 | 0.8125 | 0.85245901 |
| Random Forest | word\_3\_100 | 0.92 | 0.88888889 | 1 | 0.94117647 |
| Random Forest | word\_1\_500 | 0.96 | 0.96875 | 0.96875 | 0.96875 |
| Random Forest | word\_2\_500 | 0.82 | 0.87096774 | 0.84375 | 0.85714286 |
| Random Forest | word\_3\_500 | 0.92 | 0.88888889 | 1 | 0.94117647 |
| Random Forest | word\_1\_1000 | 0.92 | 0.9375 | 0.9375 | 0.9375 |
| Random Forest | word\_2\_1000 | 0.86 | 0.87878788 | 0.90625 | 0.89230769 |
| Random Forest | word\_3\_1000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Random Forest | word\_1\_2000 | 0.96 | 0.96875 | 0.96875 | 0.96875 |
| Random Forest | word\_2\_2000 | 0.88 | 0.86111111 | 0.96875 | 0.91176471 |
| Random Forest | word\_3\_2000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Random Forest | word\_1\_3000 | 0.96 | 0.94117647 | 1 | 0.96969697 |
| Random Forest | word\_2\_3000 | 0.88 | 0.90625 | 0.90625 | 0.90625 |
| Random Forest | word\_3\_3000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Random Forest | word\_1\_4000 | 0.94 | 0.93939394 | 0.96875 | 0.95384615 |
| Random Forest | word\_2\_4000 | 0.88 | 0.88235294 | 0.9375 | 0.90909091 |
| Random Forest | word\_3\_4000 | 0.9 | 0.88571429 | 0.96875 | 0.92537314 |
| Random Forest | word\_1\_5000 | 0.98 | 0.96969697 | 1 | 0.98461538 |
| Random Forest | word\_2\_5000 | 0.86 | 0.85714286 | 0.9375 | 0.89552239 |
| Random Forest | word\_3\_5000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Random Forest | word\_1\_6000 | 0.96 | 0.96875 | 0.96875 | 0.96875 |
| Random Forest | word\_2\_6000 | 0.88 | 0.88235294 | 0.9375 | 0.90909091 |
| Random Forest | word\_3\_6000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Random Forest | word\_1\_7000 | 0.96 | 0.96875 | 0.96875 | 0.96875 |
| Random Forest | word\_2\_7000 | 0.9 | 0.90909091 | 0.9375 | 0.92307692 |
| Random Forest | word\_3\_7000 | 0.94 | 0.91428571 | 1 | 0.95522388 |
| Random Forest | word\_1\_8000 | 0.98 | 0.96969697 | 1 | 0.98461538 |
| Random Forest | word\_2\_8000 | 0.86 | 0.87878788 | 0.90625 | 0.89230769 |
| Random Forest | word\_3\_8000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Random Forest | word\_1\_9000 | 0.96 | 0.96875 | 0.96875 | 0.96875 |
| Random Forest | word\_2\_9000 | 0.9 | 0.90909091 | 0.9375 | 0.92307692 |
| Random Forest | word\_3\_9000 | 0.94 | 0.91428571 | 1 | 0.95522388 |
| Random Forest | word\_1\_10000 | 0.94 | 0.96774194 | 0.9375 | 0.95238095 |
| Random Forest | word\_2\_10000 | 0.88 | 0.90625 | 0.90625 | 0.90625 |
| Random Forest | word\_3\_10000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Multi-layer Perceptron | word\_1\_100 | 0.9 | 1 | 0.84375 | 0.91525424 |
| Multi-layer Perceptron | word\_2\_100 | 0.86 | 0.93103448 | 0.84375 | 0.8852459 |
| Multi-layer Perceptron | word\_3\_100 | 0.92 | 0.88888889 | 1 | 0.94117647 |
| Multi-layer Perceptron | word\_1\_500 | 0.94 | 1 | 0.90625 | 0.95081967 |
| Multi-layer Perceptron | word\_2\_500 | 0.88 | 1 | 0.8125 | 0.89655172 |
| Multi-layer Perceptron | word\_3\_500 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Multi-layer Perceptron | word\_1\_1000 | 0.96 | 1 | 0.9375 | 0.96774194 |
| Multi-layer Perceptron | word\_2\_1000 | 0.84 | 0.96153846 | 0.78125 | 0.86206896 |
| Multi-layer Perceptron | word\_3\_1000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Multi-layer Perceptron | word\_1\_2000 | 0.98 | 1 | 0.96875 | 0.98412698 |
| Multi-layer Perceptron | word\_2\_2000 | 0.9 | 0.96551724 | 0.875 | 0.91803279 |
| Multi-layer Perceptron | word\_3\_2000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Multi-layer Perceptron | word\_1\_3000 | 0.98 | 1 | 0.96875 | 0.98412698 |
| Multi-layer Perceptron | word\_2\_3000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Multi-layer Perceptron | word\_3\_3000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Multi-layer Perceptron | word\_1\_4000 | 0.98 | 1 | 0.96875 | 0.98412698 |
| Multi-layer Perceptron | word\_2\_4000 | 0.9 | 0.96551724 | 0.875 | 0.91803279 |
| Multi-layer Perceptron | word\_3\_4000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Multi-layer Perceptron | word\_1\_5000 | 0.98 | 1 | 0.96875 | 0.98412698 |
| Multi-layer Perceptron | word\_2\_5000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Multi-layer Perceptron | word\_3\_5000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Multi-layer Perceptron | word\_1\_6000 | 0.98 | 1 | 0.96875 | 0.98412698 |
| Multi-layer Perceptron | word\_2\_6000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Multi-layer Perceptron | word\_3\_6000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Multi-layer Perceptron | word\_1\_7000 | 0.98 | 1 | 0.96875 | 0.98412698 |
| Multi-layer Perceptron | word\_2\_7000 | 0.9 | 0.96551724 | 0.875 | 0.91803279 |
| Multi-layer Perceptron | word\_3\_7000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Multi-layer Perceptron | word\_1\_8000 | 0.98 | 1 | 0.96875 | 0.98412698 |
| Multi-layer Perceptron | word\_2\_8000 | 0.92 | 1 | 0.875 | 0.93333333 |
| Multi-layer Perceptron | word\_3\_8000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Multi-layer Perceptron | word\_1\_9000 | 0.98 | 1 | 0.96875 | 0.98412698 |
| Multi-layer Perceptron | word\_2\_9000 | 0.9 | 0.96551724 | 0.875 | 0.91803279 |
| Multi-layer Perceptron | word\_3\_9000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |
| Multi-layer Perceptron | word\_1\_10000 | 0.98 | 1 | 0.96875 | 0.98412698 |
| Multi-layer Perceptron | word\_2\_10000 | 0.9 | 0.96551724 | 0.875 | 0.91803279 |
| Multi-layer Perceptron | word\_3\_10000 | 0.92 | 0.91176471 | 0.96875 | 0.93939394 |

1. https://github.com/Ronke21?tab=repositories [↑](#footnote-ref-1)