

UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA



DEPARTMENT OF INFORMATION ENGINEERING

MASTER THESIS IN ICT FOR INTERNET AND MULTIMEDIA

# Visual-Based Anti-Spamming Techniques With Learning

**Supervisor**

Prof. Stefano Toamsin

**Master Candidate**

Ali Hossary

ACADEMIC YEAR 2023-2024

Graduation date 05/03/2024



# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Literature Review</b>	<b>5</b>
2.1	Introduction to Adversarial Machine Learning . . . . .	5
2.2	Overview of Spam Filtering Techniques . . . . .	6
2.2.1	Heuristic Filters . . . . .	7
2.2.2	Black listing . . . . .	7
2.2.3	Collaborative Spam Filtering . . . . .	7
2.2.4	Machine Learning Approach to E-mail Spam filtering: . . . . .	8
2.3	Adversarial Attacks on Spam Filtering . . . . .	14
2.3.1	Visual spoofing . . . . .	14
2.3.2	Ham Words Injection . . . . .	15
2.3.3	Spam Words Separation . . . . .	15
2.3.4	Dyslexia . . . . .	19
2.3.5	Synonym Replacement . . . . .	19
2.3.6	Tiny Nonsense . . . . .	20
2.3.7	Word Splitting . . . . .	21
2.3.8	dataset shift problem . . . . .	21
<b>3</b>	<b>Methodology</b>	<b>25</b>
3.1	Data Collection and preparation . . . . .	25
3.1.1	SpamAssassin public mail corpus . . . . .	25
3.1.2	Enron-Spam datasets . . . . .	27
3.1.3	Hand crafted datasets . . . . .	27
3.2	Supportive middle layer for Spam Classifier . . . . .	27
3.2.1	Summarization layer . . . . .	28
3.3	Vision-Based Spam Filter - VBSF model . . . . .	30
3.3.1	Incoming Emails . . . . .	31

3.3.2	Email Rendering . . . . .	31
3.3.3	Emails Capturing . . . . .	32
3.3.4	First Pipeline: Text Extraction and Content-Based Classification . . . . .	32
3.3.5	Second Pipeline : Image-Based Classification using CNN . . . . .	34
3.3.6	Final Prediction Decision . . . . .	36
3.3.7	Enhanced Vision-Based Spam Filter . . . . .	37
<b>4</b>	<b>Results and Analysis</b>	<b>39</b>
4.1	Dataset Analysis . . . . .	39
4.1.1	Analysis of Dataset distribution . . . . .	39
4.2	Data Summarization . . . . .	41
4.3	Analysis of the Vision-Based Spam Filter VBSF . . . . .	43
4.3.1	The First Pipeline . . . . .	45
4.3.2	The Second Pipeline . . . . .	53
4.3.3	Analysis on the final classification decision . . . . .	53
<b>5</b>	<b>Conclusion</b>	<b>65</b>
	<b>Bibliography</b>	<b>67</b>

# List of Figures

2.1	Adversary try to contaminating the training data of a classifier . . . . .	6
2.2	The roposed workflow of SVM spam Classifier. . . . .	10
2.3	Workflow of Stacking Classifier. . . . .	13
2.4	Types of ensemble learning methods. . . . .	14
2.5	Figure A illustrates Experiment A, which serves as the control experiment utilizing entirely unaltered datasets. In this experiment, the default character encoding of both the training and testing sets is maintained. [26] . . . . .	16
2.6	Figure B showcases Experiment B, where the encoding of the training set is retained, and specific characters in the testing set are encoded using corresponding confusables from the Cyrillic alphabet [26]. . . . .	17
2.7	Figure C depicts Experiment C, where confusables are introduced to both the training and testing sets. This experimental configuration involves training and evaluating each model with data from a single, mixed-script context containing confusables [26]. . . . .	18
2.8	Email similarity may be calculated using a cosine similarity metric between two email vectors $A$ and $B$ [27]. . . . .	20
2.9	Performance of the prediction for the spam filters, the combination used TF-IDF and NB [3]. . . . .	23
2.10	Performance of the prediction for the spam filters, the combination used BOW and NB [3]. . . . .	23
2.11	Performance of the prediction for the spam filters, the combination used TF-IDF and SVM [3]. . . . .	24
2.12	Performance of the prediction for the spam filters, the combination used BOW and SVM [3]. Table 5 . . . . .	24
3.1	Combined dataset distribution . . . . .	26
3.2	Emails classification after summarization process flow . . . . .	28
3.3	The proposed Vision-Based Spam Filter. . . . .	31
3.4	The accuracy of the NB model. . . . .	34

3.5	Architecture of CNN VGG19 Model . . . . .	36
3.6	Stacking Classifier Architecture. . . . .	37
3.7	Enhanced Vision-Based Spam Filter VBSF . . . . .	38
4.1	Frequency of occurrence of email lengths . . . . .	40
4.2	Frequency of occurrence of spam and ham email lengths . . . . .	40
4.3	The distribution of the training dataset before improving the balancing of the data.	41
4.4	The distribution of the training dataset after improving, increasing the number of samples and balancing. . . . .	42
4.5	Accuracy of NB after abstractive summarization . . . . .	44
4.6	Accuracy of NB after Extarctive summarization . . . . .	44
4.7	Text Extraction using OCR Logs. . . . .	45
4.8	NB accuracy for RAW Emails. . . . .	47
4.9	NB accuracy for Text content Extracted by OCR. . . . .	47
4.10	DT accuracy on Raw emails . . . . .	48
4.11	Decision Tree accuracy or Text content Extracted by OCR. . . . .	48
4.12	LR accuracy on raw emails. . . . .	49
4.13	LR accuracy for Text content Extracted by OCR. . . . .	49
4.14	SVM accuracy on raw emails. . . . .	50
4.15	SVM accuracy for Text content Extracted by OCR. . . . .	50
4.16	AdaBoost accuracy on raw emails. . . . .	51
4.17	AdaBoost accuracy for Text content Extracted by OCR. . . . .	51
4.18	KNN accuracy on raw emails. . . . .	52
4.19	KNN accuracy for Text content Extracted by OCR. . . . .	52
4.20	Learning rate of the vgg19 Model. . . . .	54
4.21	validation loss of the vgg19 Model. . . . .	55
4.22	Accuracy reaching 95 % after fine tuning the vgg19 CNN model. . . . .	56
4.23	Confusion matrix for vgg19 on the first version of the dataset. . . . .	57
4.24	The new validation loss of the vgg19 Model after increasing and enhancing the data. . . . .	58
4.25	Accuracy reaching 96 after fine tuning the vgg19 CNN model on the enhanced balanced dataset . . . . .	59
4.26	Confusion matrix for vgg19 after applied on the enhanced dataset. . . . .	60
4.27	Stacknig data shape . . . . .	62
4.28	NB test accuracy . . . . .	62
4.29	Test accuracy DT . . . . .	62
4.30	Test accuracy of CNN Model . . . . .	62

4.31 Meta classifiers accuracies . . . . .	63
4.32 Final stacknig test accuracy . . . . .	63







# Acronyms

- AdaBoost** Adaptive Boosting ix, 12, 46
- BERT** Bidirectional transformer ix, 27–30
- BOW** Bag Of Words ix, 23
- CNN** Convolutional Neural Network ix, 31, 35, 36
- DT** Decision Tree ix, 45, 46, 61
- GPT-2** Generative Pre-trained Transformer 2 ix, 27, 29
- HTML** Hyper Text Markup Language ix, 31, 32
- KNN** K-Nearest Neighbor ix, 12
- LR** Logistic Regression ix, 36, 45
- NB** Naive Bayes ix, 33, 34, 36, 45, 46, 61
- OCR** Optical Character Recognition ix, 32, 33
- RF** Random Forest ix
- SMTP** Simple Mail Transfer Protocol ix, 7
- SVM** Support Vector Machine ix, 37, 45, 46
- TF-IDF** term frequency-inverse document frequency ix, 15, 23
- VBSF** Vision Based Spam Filter ix, 33, 34, 36–38, 53, 61
- XGBoost** Extreme Gradient Boosting ix, 12
- XLNet** eXtreme Language understanding NETwork ix, 29, 30



## **Abstract**

In today's world, email remains a primary medium for information exchange. However, the flood of spam emails poses significant challenges to users, demanding innovative solutions to ensure a secure and efficient communication environment. This thesis recognizes the evolving scope of spam, where spammers employ tricks to dodge traditional spam filters, highlighting the need for advanced defense mechanisms. The proposed system introduces a visionary approach to anti-spamming techniques, leveraging computer vision, machine learning, and deep learning methodologies to emulate the human visual perception of emails. The multi-step process mimics the human eye's natural way of processing visual information, automatically rendering emails and capturing their visual content. Following this, two different processing pipelines are applied in parallel. The first pipeline involves extracting text content from emails using Optical Character Recognition (OCR), followed by a content-based classification using Naive Bayes (NB) and Decision Tree (DT) classifiers. These classifiers are utilized to differentiate between spam and legitimate emails (ham) based on their textual content. The benefit from this is that we got the text content cleaned from any HTML-related tricks since the HTML email content is already parsed, executed, and rendered to the browser. To augment the system's accuracy, a second pipeline involving Convolutional Neural Network (CNN) is introduced to analyze and classify screenshots of the email content. The CNN, serving as a visual perception model, learns patterns and features crucial for distinguishing between spam and ham emails. In response to the dynamic nature of spamming techniques, the proposed solution, namely Vision-Based Spam Filter (VBSF), uses a meta classifier that integrates both the text-based classifiers and the image-based CNN concurrently exploiting the stacking ensemble learning method. This approach enhances the system's adaptability and accuracy, providing a robust and comprehensive decision-making mechanism with a final classification accuracy of the meta classifier surpassing all the base models accuracies, exceeding 98%.



# Chapter 1

## Introduction

Email has become an intrinsic part of daily digital life, with over 200 billion messages sent globally per day. However, spam constitutes an estimated 80-90% of this traffic - over 170 billion unsolicited emails daily. Beyond mere annoyance, this relentless flood enables phishing, fraud, malware, and cybercrime. Early anti-spam efforts using blacklists and heuristics struggled against this volume. This necessitated scalable, adaptive techniques, leading to machine learning becoming the dominant approach for modern spam filtering. State-of-the-art Bayesian classifiers achieve over 90% accuracy by learning statistical patterns on labeled data. Some key factors in the rise of machine learning for spam defense include the sheer scale of global spam volumes, the ever-evolving sophistication of spam tactics for evasion and monetization, and the continuous refinement of algorithms tailored for textual data analysis. Furthermore, the exponential growth in computational resources has greatly enhanced the effectiveness of spam filtering models. Additionally, the ubiquity of machine learning across various IT systems' security functions has catalyzed its adoption in combating spam.

Nevertheless, the resilience of these machine learning models is now under scrutiny due to inherent vulnerabilities. Adversaries can exploit systematic weaknesses to manipulate models into misclassifying inputs. Techniques such as synonym replacement, appending legitimate strings, and strategically inserting whitespace within spam text have demonstrated their efficacy in evading filters. Moreover, the never-ending evolution of spam tactics poses an ongoing challenge, continually testing the robustness of spam filtering models against novel adversarial approaches. Dealing with these challenges requires a careful approach, understanding that typical protections against spam also have drawbacks. For example, updating models with new data takes extra effort, and making models more adaptable makes them more complicated. Tighter security constraints often come at the expense of accuracy, and more rigorous privacy protections may limit access to important data. As machine learning becomes more common in digital systems, it's increasingly important to keep these systems resilient and reliable against adver-

sarial threats. Key research goals include figuring out how attacks happen, adjusting defenses to stop them, and checking models regularly for weaknesses. The aim is also to promote systems' stability, improve their security features, and find a delicate balance between accuracy, privacy, and vulnerability.

# Chapter 2

## Literature Review

### 2.1 Introduction to Adversarial Machine Learning

Machine learning is a general tool for enabling reliable decision making in computer systems. The adaptive and pattern-finding nature of learning algorithms, while beneficial for prediction and decision-making, also introduces a security risk. Adversaries can manipulate the information fed to these algorithms, exposing systems that rely on machine learning to a novel type of vulnerability. Learners become susceptible to attacks that can disrupt the system's intended operation. To understand how well these techniques perform under adversarial conditions, it is necessary to investigate their robustness when exposed to attacks. Without thoroughly evaluating the performance of learning algorithms in adversarial settings, the systems will fail to gain wider adoption due to lack of trust. Worse, a vulnerable system could be exploited in a way that discourages practitioners from using machine learning in the future. Hence, practitioners must analyze the risks posed by learning algorithms and select techniques that adequately minimize these risks. When a learning algorithm performs well under realistic adversarial conditions, it demonstrates secure learning. Of course, whether an algorithm's performance is acceptable depends on the constraints imposed on the adversary and the task the algorithm must perform. This raises two key questions: What security criteria are needed to evaluate a learner's robustness to a given adversarial threat model? And do techniques exist capable of satisfying the security requirements for a particular problem while delivering good performance? By systematically assessing learning systems and selecting appropriate learning techniques, it is possible to diminish the potential impact of an attack.

Email spam filtering is one of the most well-known applications of machine learning. In this problem, known good (ham) and unwanted (spam), email messages are used to train a spam filter. The learning algorithm identifies characteristics that distinguish spam from ham (e.g. keywords or envelope features) and constructs a classifier that combines evidence of spam to

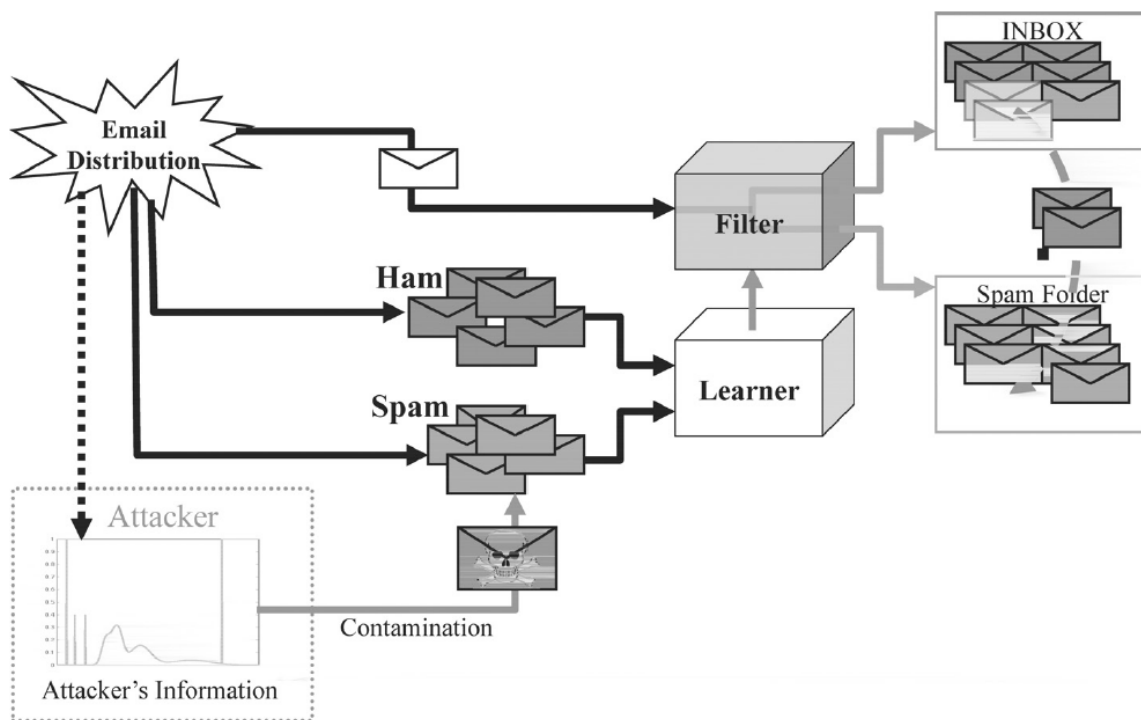


Figure 2.1: Adversary try to contaminating the training data of a classifier

label new messages. Spam filters have proven successful at correctly removing spam from users' inboxes. This has motivated spammers to evade detection by obfuscating messages or corrupting the learning process itself. As shown in figure 2.1, provided by [1], a spammer can construct attack spam that causes the filter to misclassify desired messages as spam, more specifically, spammer who has some access to the training data, try to contaminate the emails in a way to make the learning model vulnerable for future incoming spam emails. Ultimately, the spammer aims to make the filter so unreliable that the user can no longer trust its classifications and must manually sort through spam, ensuring important messages are not blocked.

## 2.2 Overview of Spam Filtering Techniques

Before the introduction of machine learning techniques, various technical measures were employed for spam filtering, including rule-based spam filtering, white lists, black lists, challenge-response (C/R) systems, honeypots, OCR filters, and numerous others. Each method has its own set of strengths and limitations. Black lists, white lists, challenge-response (C/R) systems, etc. are origin-based techniques employed by reputation-based filters.

### **2.2.1 Heuristic Filters**

Initial spam filters followed the ‘knowledge engineering’ approach and were based on coded rules or heuristics [2]. A content-based heuristic filter analyzes the contents of a message  $M$  and classifies it to spam or ham based on the occurrence of ‘spammy’ words like ‘viagra’ or ‘lottery’ in it. They were designed based on the knowledge of regularities or patterns observed in messages [3]. The drawback of heuristic filters is that maintaining an effective set of rules is time-consuming, moreover, the rules have to be constantly updated to keep up with the newest trends in spam. [4]

### **2.2.2 Black listing**

A blacklist of E-mail addresses or IP addresses of the server from which spam is found to originate is created and maintained either at the user or server level. If an email comes from certain addresses, it’s blocked automatically at the Simple Mail Transfer Protocol (SMTP) connection stage. This approach involves a simple check in a blacklist, like Real-time Blackhole Lists (RBL) and Domain Name System Blacklists, keeping computational costs low. These lists cover proxies, open relays, networks, or specific addresses used for spamming, such as Google’s and SpamHaus blacklists [5].

However, despite their effectiveness, blacklist methods have drawbacks, Legitimate addresses might be wrongly or arbitrarily blacklisted, Innocent users can be impacted, and entire domains, like Hotmail, might face blocks due to spammers using their email IDs or IP addresses without permission. To add, spammers use large Botnets to send spam, in its turn, creating extremely a huge number of IP addresses to be blacklisted.

### **2.2.3 Collaborative Spam Filtering**

Spammers commonly target a large number of recipients with their spam. It’s probable that the same spam has reached others as well. Collaborative spam filtering involves a distributed approach where a community collectively tackles spam, as outlined in [6], this method doesn’t analyze email content, instead, it gathers identifying information about spam, such as subject lines, senders, or results of mathematical functions applied to the email body. Early recipients share digital signatures of these spam messages within the community. Users then use these signatures to identify spam emails. Examples of such collaborative spam filters on the web include Vipuls Razor, Pyzor , and DCC (Distributed Checksum Clearinghouse)[7]. While collaborative techniques show promise, they face challenges related to scalability and underlying assumptions[6].

## 2.2.4 Machine Learning Approach to E-mail Spam filtering:

### The SpamBayes Spam Filter

SpamBayes is a content-based classifier that categorizes emails based on the occurrence of specific words or phrases (tokens) in a training dataset. It assigns a spam score to each token based on its frequency in spam and non-spam emails, reflecting the likelihood of an email containing that token being spam. The filter calculates the overall spam score of a message assuming token scores are independent and then uses Fisher's method to determine if the email's tokens point to one class or the other. The message score is compared to two thresholds to label it as spam, ham, or unsure. The spam classification model used by SpamBayes was invented by Robinson(2003)[8], Meyer, and Whateley(2004)[9] based on ideas from Graham (2002) [10] together with Fisher's method (1948) to combine independent tests. SpamBayes learns how likely a word is spam or ham by counting how often it appears in spam and ham emails. When SpamBayes sees a new email, it looks at all the words to see if they make it more likely to be spam or ham. It uses a statistical test to decide if there is enough evidence to call it either, or if it is unsure. SpamBayes tokenizes each email  $X$  based on words, URL components, header elements, and other character sequences that appear in  $X$ . Each is treated as a unique token of the email independent of their order within the message.

### SVM

Support vector machines (SVMs) are ranked as one of the best 'off-the-shelf' supervised learning algorithms. The effectiveness of SVM primarily stems from its strong generalization capability. Unlike numerous learning algorithms, SVM yields meaningful predictions without requiring the inclusion of extensive prior data [11]. It has shown power in binary classification, thanks to its wise theoretical foundation and well-defined learning algorithm rules. This leads to stable information classification.

In[11] the authors outlined methodology for detecting email spam using SVM. The workflow is depicted in figure 2.2 and explained as follows:

a. Pre-processing: The initial step involves pre-processing to eliminate irrelevant noise from emails. This phase encompasses:

- Removal of Numbers
- Elimination of Special Symbols
- Deletion of URLs
- Separation of HTML tags

- Word Stemming

b. Feature Extraction: The feature extraction technique is applied to derive significant features from the email body. These features replace numeric representations in a 2D vector space, with mappings obtained from a dictionary list.

c. SVM Training: For training purposes, email spam serves as the training dataset. The classifier undergoes training using these datasets, incorporating spam content. Post-training, the classifier is equipped to categorize spam emails.

d. Test Classifier: The classifier undergoes testing with various training data to ensure accurate functionality. The goal is to achieve a proposed solution with a 98% accuracy rate in email classification.

e. Test Email: Upon completion of the training phase, the classifier is provided with a sample email for classification. The classifier outputs 0 or 1, where 1 indicates spam, and 0 denotes non-spam, constituting the final classification.

## **Logistic Regression**

Logistic regression stands out as one of the most suitable and likely algorithms for dataset classification. Specifically, when it comes to classifying a dataset designated as "spam," logistic regression emerges as a versatile decision-based approach for identifying spam emails within the dataset. The logistic regression algorithm conducts fundamental tests on the data distribution, involving the calculation of statistical measures such as mean and standard deviation [12]. Additionally, it generates results from operations like word and character count, as well as max and min operations. Following the execution of these statistical and count tests, the logistic regression algorithm retrieves the outcomes and endeavors to establish connections between them. The paper [13] resulted that Logistic regression stands out as one of the most probable and suitable algorithms for classifying datasets. When dealing with the classification of a dataset labeled as spam, logistic regression proves to be a versatile decision-based approach, particularly adept at identifying spam emails within the dataset. The logistic regression algorithm conducts essential tests on the given data distribution, involving the determination and calculation of statistical parameters such as mean and standard deviation [16]. Additionally, it yields results from operations such as word and character count, as well as maximum and minimum operations. Following the execution and provision of these statistical and count tests, the logistic regression algorithm retrieves the outcomes and seeks to establish connections among them.

## **Clustering Techniques**

**Clustering** is the task of grouping a set of patterns into similar groups. Clustering techniques have been widely studied and used in a variety of application domains. Spam filtering datasets

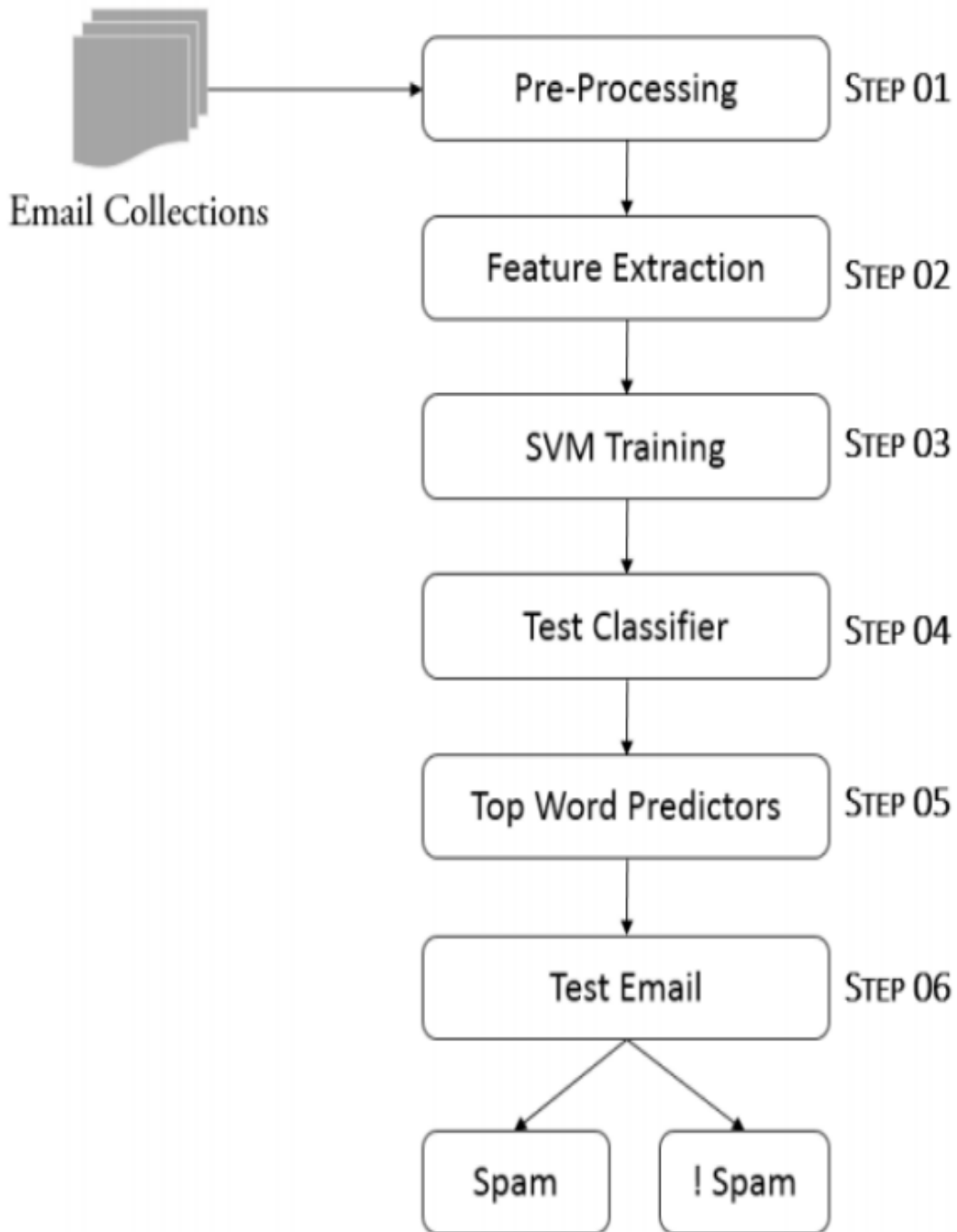


Figure 2.2: The roposed workflow of SVM spam Classifier.

often have true labels available and clustering algorithms, being unsupervised learning tools are not always closely related with true labelings. However given suitable representations, most clustering algorithms can partition e-mail spam datasets into ham and spam clusters. This was demonstrated by [14] in a novel investigation of e-mail spam clustering. The results were surprisingly significant as their clustering based approach bettered those of previously published state-of-the-art semi-supervised approaches, hence proving that clustering can be a powerful tool for e-mail spam filtering. Prior to this, Sasaki and Shinnou [15] had proposed spam detection technique making use of the text clustering through a vector space model. Basavaraju and Prabhakar [16], presented an effective clustering algorithm integrating K-means and BIRCH algorithm features. K-means algorithm worked well for small scale data sets. BIRCH with K-Nearest Neighbour Classifier (K-NNC) was found to be the ideal combination as it performed better with large data sets.

### **Other Machine Learning Classifiers**

- **Decision Tree Classifier (DT)** Decision Tree is a supervised learning method applicable to both classification and regression tasks, though it's predominantly favored for classification purposes. It operates as a tree-like classifier, with dataset features depicted in internal nodes, decision rules represented by branches, and outcomes shown in leaf nodes. It is a graphical representation that offers a systematic way to explore potential solutions or decisions based on specified conditions.[17] [18].
- **Random Forest Classifier (RF)** It is a machine learning technique for handling regression and classification problems [19] that averages the results of many decision trees.

### **Ensemble Classifiers**

Ensemble learning, a recent approach, involves training and combining several individual classifiers to improve the overall accuracy of classification, particularly in tasks like spam detection. This method proves highly effective and offers strong generalization. As spam filters encounter various spam types, they require continuous improvement to detect new forms of spam (future spam) while effectively capturing traditional spam. To address this challenge, [20] proposed the idea of merging old and new filters, such as using ensemble classifiers, as a promising strategy to handle the diversity of spam. Bagging, boosting and Stacking stand out as the most popular ensemble classifiers in this domain[21]2.4. **Primary Ensemble Method Categories:**

- **Bagging** , short for bootstrap aggregating, is predominantly utilized in classification and regression tasks. It enhances model accuracy through the use of decision trees, effectively minimizing variance. This reduction in variance contributes to increased accuracy

and mitigates overfitting, a common challenge in predictive modeling. Bagging involves two key components: bootstrapping and aggregation. Bootstrapping employs a sampling technique with replacement, creating randomized subsets from the entire dataset. The base learning algorithm is then applied to these subsets. Aggregation ensures that all potential outcomes are considered, contributing to more accurate predictions. Although advantageous in combining weak base learners into a robust model, bagging can be computationally expensive, potentially introducing bias if not implemented properly.

- Boosting is an ensemble technique that learns from prior predictor errors to enhance future predictions. It combines weak base learners to create a strong learner, significantly improving predictive performance. Various forms of boosting exist, including gradient boosting, Adaptive Boosting (AdaBoost), and Extreme Gradient Boosting (XGBoost).

**AdaBoost** utilizes decision trees, particularly decision stumps, where each stump is based on observations with similar weights. Gradient boosting sequentially adds predictors to correct errors in previous predictions, enhancing overall model accuracy. **XGBoost** employs decision trees with boosted gradient, focusing on computational speed and target model performance. While boosting enhances predictive capabilities, it can be computationally intensive, impacting implementation speed.

- Stacking, also known as stacked generalization, involves training an algorithm to ensemble predictions from multiple similar learning algorithms. This technique has found success in regression, density estimations, distance learning, and classifications. Stacking can also assess error rates associated with bagging [21].

As depicted in figure 2.3, The individual classification models undergo training using the entire training set. Subsequently, the meta-classifier is fitted based on the outputs, referred to as meta-features, produced by the individual classification models within the ensemble [22].

In [23], authors proposed the utilization of a stacking ensemble approach for spam classification. In this approach, the primary classifiers (Logistic Regression, K-Nearest Neighbor (KNN), Decision Tree, Gaussian Naive Bayes, and AdaBoost) generate predictions on the training data. The forecasted class probabilities from these base classifiers are aggregated to form the meta-training dataset (MetaTrain). Subsequently, the meta-classifier (logistic regression) undergoes training on the meta-training dataset.

For making predictions, the base classifiers provide predicted class probabilities for a novel sample, which are then collated to construct the meta-testing data (MetaTest). The meta-classifier utilizes this meta-testing data to make the final prediction.

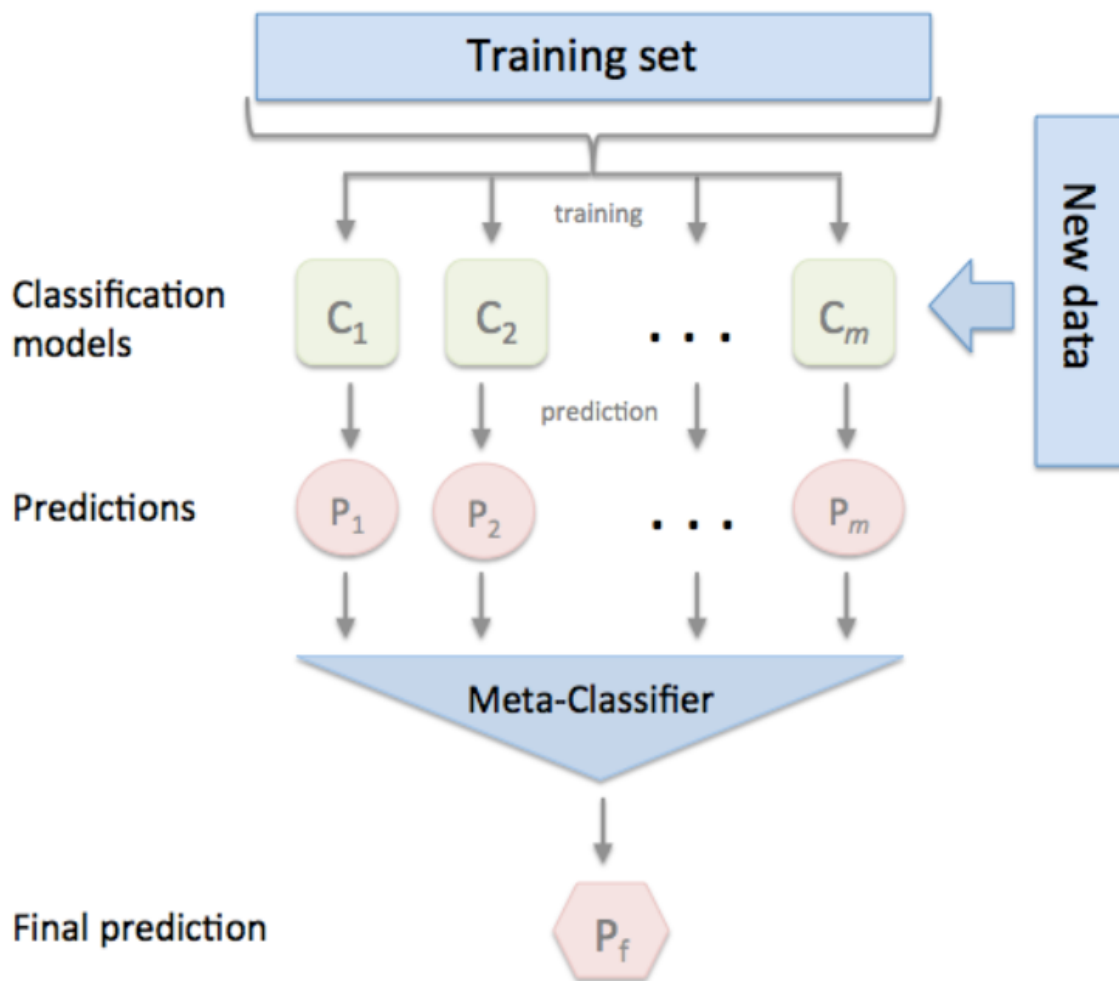


Figure 2.3: Workflow of Stacking Classifier.

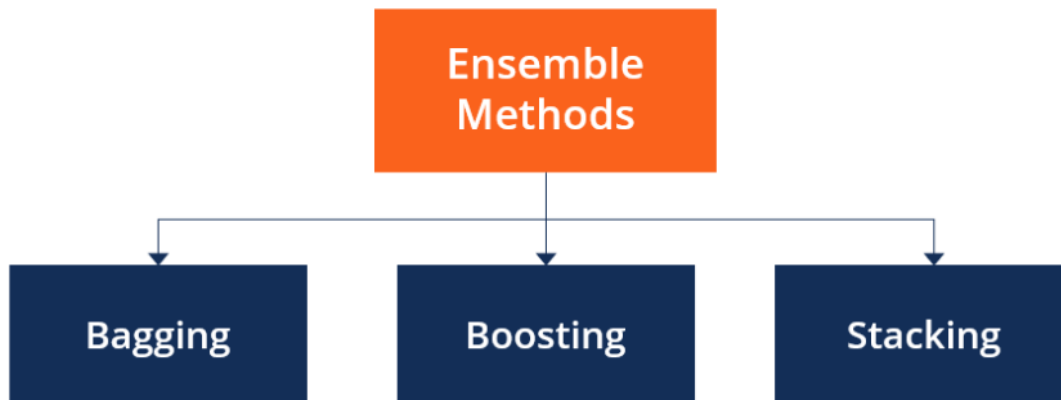


Figure 2.4: Types of ensemble learning methods.

- Variance Reduction Ensemble methods, including bagging, boosting, and stacking, are effective in reducing model variance, thereby improving prediction accuracy. The combination of multiple models ensures that the final prediction considers all possibilities from the diverse models, resulting in a more robust and accurate outcome. Ensemble methods excel in delivering optimal predictions by leveraging the strengths of different models and mitigating the weaknesses associated with individual models[21].

[24].

## 2.3 Adversarial Attacks on Spam Filtering

### 2.3.1 Visual spoofing

Despite notable advancements in addressing the challenge of spam classification, it is noteworthy that contemporary spam filters exhibit vulnerabilities that are susceptible to exploitation. In this context, a study introduced a specific vulnerability where certain characters can be substituted with visually similar counterparts from a different alphabet [25], each possessing distinct unicode encoding. This technique empowers spammers To craft messages with the capability to elude established spam filters effectively. Furthermore, the study demonstrates that this approach extends beyond spam evasion, as it can be employed to circumvent plagiarism detection mechanisms. Additionally, highlighted the broader implications of this method In various applications that rely on natural language processing for the automated analysis of textual documents. The primary objective of the experimental design is to comprehensively assess The influence

of confusables on the performance of traditional content-based Spam detection machine learning models. In pursuit of this goal, [25] executed Three distinct experiments to systematically investigate varying conditions. Experiment A in figure 2.5 , functioning as the control, employed entirely unaltered datasets, wherein the Default character encoding in both the training and testing sets was Meticulously preserved. Conversely, in experiment B 2.6, the encoding of the training set remained unaltered, while specific characters in the testing set were transformed from the default Latin alphabet to their corresponding confusables found in the cyrillic alphabet. Experiment C 2.7 broadened the scope by Introducing confusables to both the training and testing sets. Consequently, each model was trained and evaluated with data derived from a single, mixed-Script context that inherently incorporated confusables. This nuanced experimental approach facilitates a comprehensive exploration of the nuanced Impact of confusables on the efficacy of spam detection models. It evaluated these three scenarios with four machine learning algorithms frequently used for spam detection: decision tree, random forest, naïve Bayes, and support vector machine. The goal was not to compare these algorithms but rather to show that regardless of the algorithm used, this method leads to similar results. Each of these classifiers was evaluated using accuracy, precision, recall, F1 score, and confusion matrices.[25][26]

### 2.3.2 Ham Words Injection

As the spam filter primarily relies on the **term frequency-inverse document frequency (TF-IDF)** feature, altering the frequency of word occurrences shows potential. If there exists a publicly accessible database containing spam words with high probability of triggering the filter, any words not listed in the database are deemed ham words. Introducing ham words at various locations in messages, without significantly altering the original meaning, is a strategy. When a message contains a sufficient number of ham words and crosses a certain threshold, the model might misclassify a spam message as a ham message [27].

### 2.3.3 Spam Words Separation

Introducing spaces between words presents an intriguing strategy to potentially bypass a spam filter. In this technique, spaces are inserted among the characters of words that could be highly likely to be labeled as spam. The intuition behind this is that by incorporating spaces between characters, a text parser might interpret each character as a distinct word, thereby disrupting the model's word frequency distribution [27].

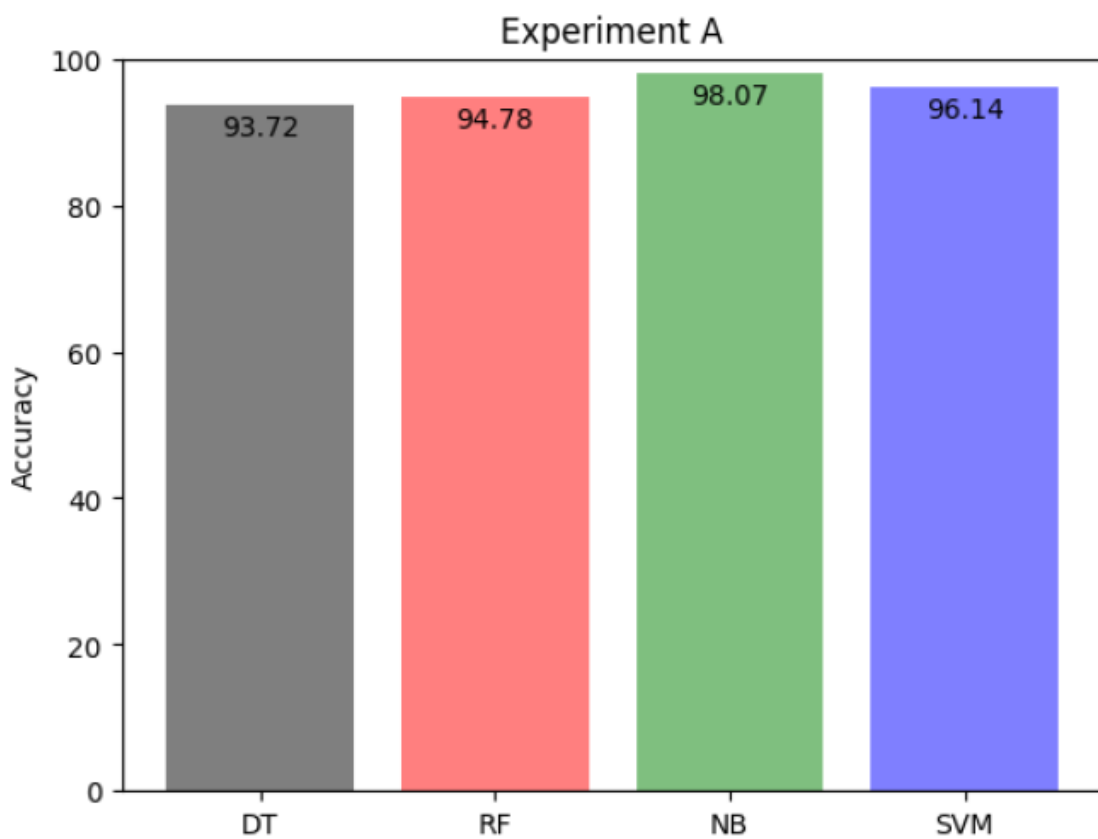


Figure 2.5: Figure A illustrates Experiment A, which serves as the control experiment utilizing entirely unaltered datasets. In this experiment, the default character encoding of both the training and testing sets is maintained. [26]

??

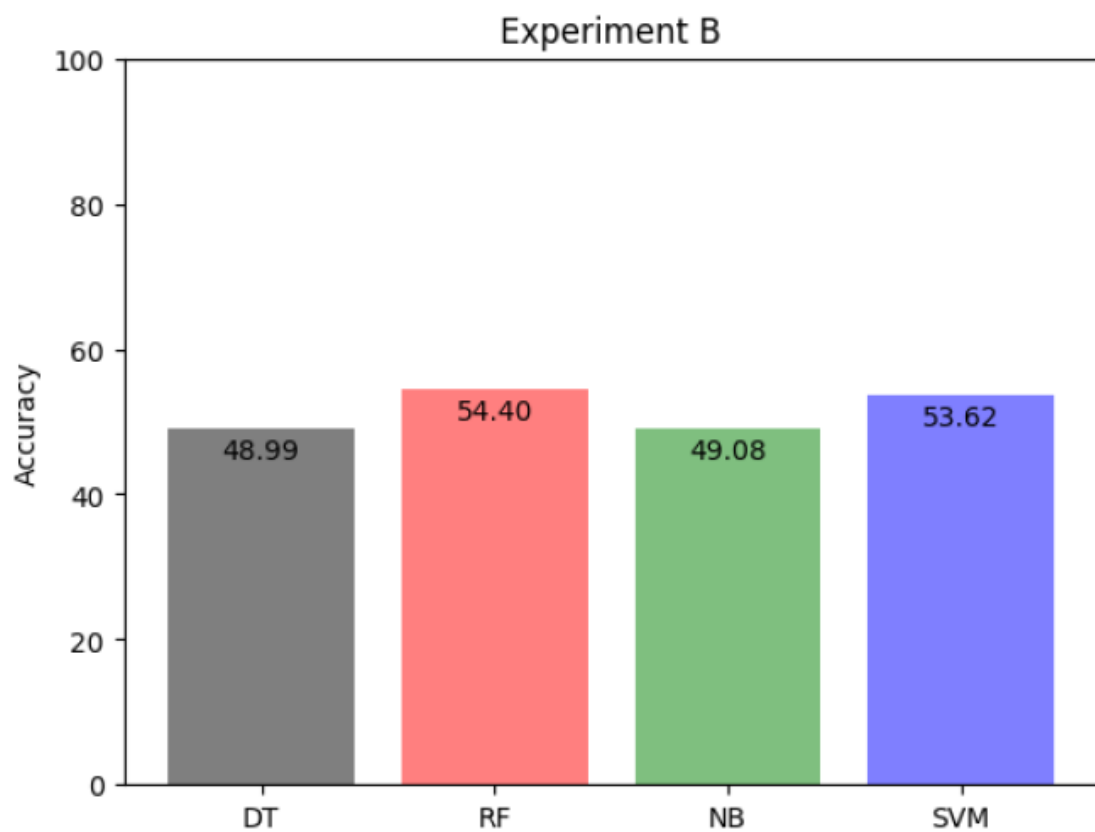


Figure 2.6: Figure B showcases Experiment B, where the encoding of the training set is retained, and specific characters in the testing set are encoded using corresponding confusables from the Cyrillic alphabet [26].

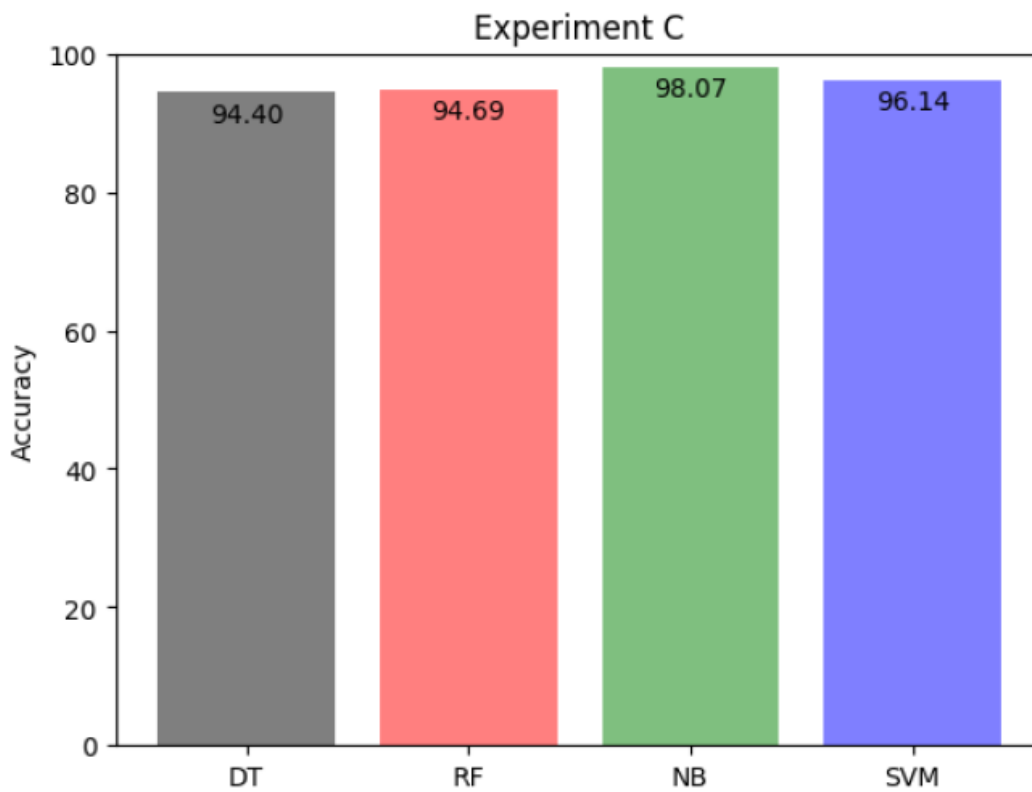


Figure 2.7: Figure C depicts Experiment C, where confusables are introduced to both the training and testing sets. This experimental configuration involves training and evaluating each model with data from a single, mixed-script context containing confusables [26].

### 2.3.4 Dyslexia

The human eye not only excels at pattern recognition but also tends to interpret visual data by fitting it into recognizable patterns. A well-known optical illusion illustrates that if the first and last letters of a word are in their correct positions, the remaining letters can be rearranged without hindering recognition of the word. Additionally, adding extra letters or using phonetic spellings can significantly alter the spelling of a word without rendering it illegible. Spammers exploit this phenomenon to generate an almost infinite array of spellings for words and phrases commonly found in spam messages [28]. Some examples include:

**FWREE HRBAL VGRAIA SAMPULS. Nekad scoohlgurl photos!!**. These examples, "FWREE HRBAL VGRAIA SAMPULS" and "Nekad scoohlgurl photos!!", illustrate how spammers employ this tactic. Despite the altered spellings and additions, the intended words ("**FREE HERBAL VIAGRA SAMPLES**" and "**Naked schoolgirl photos!!**") are still recognizable to most readers due to the brain's ability to interpret patterns. Overall, the example underscores the importance of understanding human perception and cognition in various contexts, including language processing, and how this understanding can be leveraged or exploited in different situations. Additionally, it highlights the challenges posed by such tactics in combating spam and other forms of deceptive communication.

### 2.3.5 Synonym Replacement

Given the reliance of the Naive Bayesian classifier on term frequencies, adjusting words within emails significantly influences spam classification. To enhance the likelihood of categorizing spam as ham without altering its original meaning extensively, [27] implemented a synonym replacement technique rooted in natural language processing (NLP).

the author's primary goal is to modify an email message (e.g., spam)  $M_0$  such that the modified sample  $M^*$  can both satisfy the needs (e.g., does not change the nature of spam) and bypass the spam classifier. In other words, the new messages must be similar to the original messages. Computed a similarity score between the new message and the original message using cosine similarity (2.8). Mathematically, cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space. The smaller the angle, the higher the cosine similarity.

For a given word  $W$ , find a set of synonyms  $S = \{W_1, W_2, W_3, \dots, W_n\}$ . A synonym word  $W^*$  is chosen from the set  $S$  to replace the word  $W$ . Replacing all the prominent words in an email with their closest synonyms to form a new message will deliver a similar meaning. The choice of synonyms depends on the similarity metrics. Since stopwords (common words), such as "a", "the", and "it", do not play much

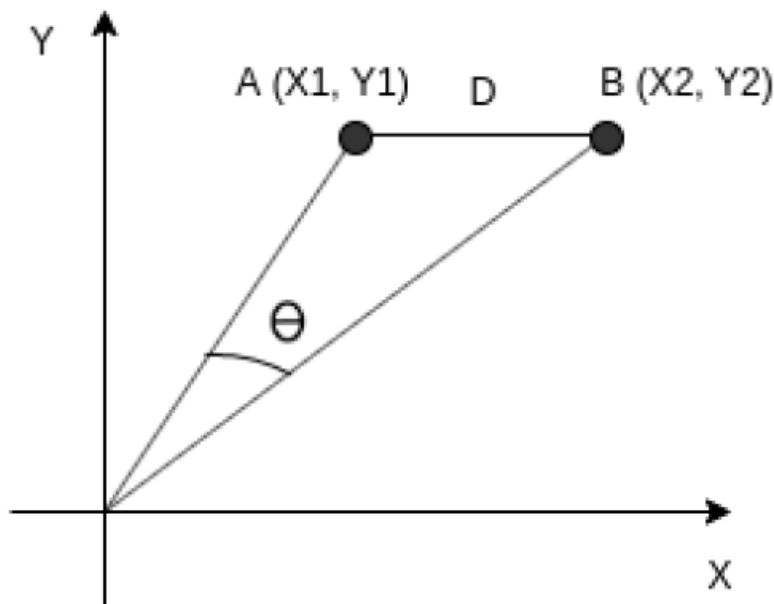


Figure 2.8: Email similarity may be calculated using a cosine similarity metric between two email vectors  $A$  and  $B$  [27].

role in changing the meaning of a message and have little effect on classification, stop words have been excluded from synonym replacement.

[27]

### 2.3.6 Tiny Nonsense

If a spammer manages to evade detection by a spam filter for a word deemed "spammy," a significant portion of their task is accomplished. Certain spam filters conduct basic HTML parsing by removing HTML tags and analyzing the remaining text. Spammers exploit these systems by inserting random letters into "spammy" words while utilizing HTML `<FONT>` tags to shrink them to a size that goes unnoticed by users reading the message.

For instance, the HTML code:

```
Mo<FONT SIZE="1">sdf</FONT>rtga<FONT SIZE="1">lax</FONT>ge
```

would render as:

"Mortgage"

However, it would appear as:

"mosdfrtgalaxge" to a spam filter that doesn't recognize the `<FONT>` tags.

A variant of this technique involves employing extremely small (one pixel wide) images to disrupt words. These images may not even exist - they're so tiny that the email client's "broken

image” placeholder won’t be visible.[28]

### 2.3.7 Word Splitting

A widely-used tactic involves breaking up ”interesting” words using HTML comments. For instance, the HTML:

```
Fr>!-- 63 --<ee mor>!-- adf --<tgage fin>!-- sdf --<anci>!-- e --<ng
```

would appear as: Free mortgage financing in an email client. However, merely disregarding HTML comments isn’t sufficient. Most HTML rendering engines overlook any tag they fail to recognize, enabling spammers to invent their own nonsensical HTML tags and employ them to divide words.[28]

### 2.3.8 dataset shift problem

A foundational assumption in supervised learning is that the training and test data share the same, albeit unknown, distribution. However, real-world scenarios may introduce mismatches, leading to a violation of this assumption. This challenge is commonly referred to by various terms such as dataset shift, concept shift, concept drift, or simply drift [29]). Over the past decade, this issue has garnered increasing attention [30]; [31]; [32]).

It is crucial to emphasize that in the presence of dataset shift, classification models often struggle to generalize in the deployment environment, leading to a significant deterioration in their performance [33]. [29] and [34] have conducted a thorough analysis of the effects of dataset shift on probability distributions, including a categorization of different types of shifts: covariate shift (distribution shift in features), prior probability shift (shift in classes), concept shift (shift in the relationship between features and classes), and other types of shift.

Some of the initial strategies to address the inherent dataset shift, also known as concept drift, present in email spam data have leaned towards the utilization of lazy learners ([35]; [36]). The fundamental idea proposed by [35] involves (i) daily incorporating the misclassified cases into the case-base for system updates at the end of each day and (ii) periodically retraining the system and reselecting features based on the most recent cases.

In [36], two additional techniques were introduced for monitoring concept drift in this domain, employing a lazy learner. The first technique, RTI (Relevant Term Identification), involves the selection of representative terms based on the information contained in each email. The second technique, RMS (Representative Message Selection), selects emails deemed more applicable given the current context implementation.

The investigation into the dynamic attributes of the spam email domain is also explored in [37]. The authors delve into the notion of a cyclical concept drift in this domain, suggesting

that the list of characteristics employed for identifying spam emails may periodically vanish and reappear. This study not only addresses the concept drift but also tackles other challenges such as catastrophic forgetting, involving past strategies employed by spammers. The authors aim to establish a lifelong classification model utilizing an ensemble learning strategy. Their approach relies on the implementation of the Early Drift Detection Method (EDDM) to verify the occurrence of concept drift. In cases of confirmed drift, the proposed Ensemble-based Lifelong Classification using Adjustable Dataset Partitioning (ELCADP) endeavors to adapt the spam filter to accommodate any alterations in the class distribution.

The majority of publicly accessible datasets employed for training and evaluating new spam email filtering models in the literature pertain to the timeframe between 2000 and 2010. In the experimental approach, authors have opted for five widely recognized datasets that encompass both spam and ham emails, enabling to span epochs with substantial time gaps. These datasets include Ling-Spam by [38], SpamAssassin5 [38], Enron-Spam [38], TREC07 [39], and CSDMS 2010.

Ling-Spam consists of 2893 emails gathered from the Linguist List, focusing on linguistic interests such as job postings, research opportunities, software availability announcements, and flame-like responses, while excluding attachments, HTML tags, and duplicate spam emails.

SpamAssassin, developed by Justin Mason of Network Associates, comprises 6047 emails sourced from public forums or user donations, exhibiting less topic specificity compared to single-user datasets.

Enron-Spam combines ham emails from six Enron employees with spam emails obtained from diverse sources, including the SpamAssassin corpus, the Honeypot Project, Bruce Guenter’s collection, and one dataset creator’s personal mailbox. The emphasis here is on constructing a personalized spam dataset, resulting in six sub-datasets that emulate various real-world user scenarios.

TREC07 encompasses all emails from a specific server with multiple accounts and honeypot accounts, published on the web between April 8 and July 6, 2007. This corpus was distributed for a competition aimed at developing a spam filter.

CSDMC, a dataset associated with the ICONIP 2010 data mining competition, consists of messages published on public forums and received from non-spam-trap sources, resembling non-personalized email datasets like SpamAssassin and Ling-Spam.

For model evaluation, two additional datasets from the Spam Archive of Bruce Guenter were employed. This repository publicly shares spam emails from their mailbox monthly since 1998. Specifically, we utilized the folders from 2010 and 2018 for our experimentation to gauge the current performance of models on recent spam emails. This approach enables us to assess the generalization of models trained on datasets from the last decade against contemporary spam

Test set	Train set									
	Ling-Spam		SpamAssassin		EnronSpam		TREC07		CSDMC	
	2000		2002–2006		2006		2007		2010	
	Acc	FPR	Acc	FPR	Acc	FPR	Acc	FPR	Acc	FPR
Ling-Spam	99.14	0.00	86.52	13.76	91.98	8.95	94.05	5.97	51.19	56.38
SpamAssassin	74.54	30.75	97.41	1.04	64.28	46.79	84.09	17.11	93.86	4.30
EnronSpam	81.83	11.54	75.64	18.44	97.55	1.49	83.50	23.72	72.22	28.29
TREC07	75.45	20.76	75.89	6.74	81.68	34.13	96.65	1.41	77.37	20.23
CSDMC	74.81	27.05	88.84	0.48	70.81	39.71	85.66	9.29	95.51	2.96

Figure 2.9: Performance of the prediction for the spam filters, the combination used TF-IDF and NB [3].

Test set	Train set									
	Ling-Spam		SpamAssassin		EnronSpam		TREC07		CSDMC	
	2000		2002–2006		2006		2007		2010	
	Acc	FPR	Acc	FPR	Acc	FPR	Acc	FPR	Acc	FPR
Ling-Spam	98.93	1.00	87.80	13.06	92.84	7.75	94.50	4.52	33.08	79.27
SpamAssassin	37.07	83.25	97.78	0.91	68.31	41.41	86.61	11.22	92.88	6.37
EnronSpam	62.65	59.68	73.74	27.46	97.24	1.28	85.21	17.39	66.87	44.99
TREC07	68.72	72.55	78.52	7.90	82.18	29.85	96.05	0.94	81.63	21.30
CSDMC	36.67	86.83	92.24	0.54	74.28	33.68	84.89	5.95	95.58	3.98

Figure 2.10: Performance of the prediction for the spam filters, the combination used BOW and NB [3].

environments.

designed four spam email filters for each dataset, based on a text classification pipeline which comprises two machine learning algorithms widely used for detecting spam, NB and SVM, and two text encoders, TF-IDF and Bag Of Words (BOW).

Test set	Train set									
	Ling-Spam		SpamAssassin		EnronSpam		TREC07		CSDMC	
	2000		2002–2006		2006		2007		2010	
	Acc	FPR	Acc	FPR	Acc	FPR	Acc	FPR	Acc	FPR
Ling-Spam	99.52	0.04	56.10	50.70	63.64	41.46	37.37	74.38	54.72	52.86
SpamAssassin	69.11	39.17	99.33	0.31	61.11	50.36	63.87	45.24	98.64	0.85
EnronSpam	80.92	12.95	63.48	59.23	97.97	2.64	56.90	73.27	72.76	36.69
TREC07	69.52	26.49	85.17	11.01	78.10	47.56	99.38	0.83	83.24	14.55
CSDMC	70.00	29.36	92.24	0.88	66.74	45.15	76.45	27.42	99.04	0.58

Figure 2.11: Performance of the prediction for the spam filters, the combination used TF-IDF and SVM [3].

Test set	Train set									
	Ling-Spam		SpamAssassin		EnronSpam		TREC07		CSDMC	
	2000		2002–2006		2006		2007		2010	
	Acc	FPR	Acc	FPR	Acc	FPR	Acc	FPR	Acc	FPR
Ling-Spam	99.38	0.17	52.89	54.06	71.07	31.34	39.13	71.56	51.54	56.34
SpamAssassin	48.55	66.96	99.16	0.45	64.83	44.13	65.95	42.56	97.94	1.68
EnronSpam	76.12	17.64	63.06	59.20	96.86	3.38	59.74	67.12	61.69	60.19
TREC07	55.25	55.60	86.11	7.37	74.89	45.24	98.78	1.45	86.29	16.34
CSDMC	50.58	60.77	93.36	1.80	62.44	50.36	77.62	26.13	98.36	1.16

Models are trained with different datasets (columns) and accuracy and false positive rate (FPR) metrics are reported when they are applied to different scenarios (rows). Values highlighted in grey refer to performance estimated with 10-fold cross-validation using a specific dataset

Figure 2.12: Performance of the prediction for the spam filters, the combination used BOW and SVM [3]. Table 5

# Chapter 3

## Methodology

### 3.1 Data Collection and preparation

A mix of Publicly available datasets has been used, some of them with minimal modifications in addition to a dataset of hand-crafted emails for specific testing purposes. the Enron 1 and pre-processed Spam Assassin email corpus, which will be explained later. This combination of parts of these two datasets was the best fit for our proposed model and led to better generalization, since Enron 1 has enough textual info, while it lacks colors and visuals, unlike Spam Assassin, which is rich in colors and visual emails but alone was not big enough. As a result, we have 3151 benign emails (ham) and 1995 spam emails. The combined dataset had to some extent imbalanced class distributions (40% for spam and 60% for ham emails), so we increased the number of spam emails to balance the dataset and prevent overfitting toward the majority class. This ensured that our model had equal representation of both classes, which is essential for accurate classification performance. Figure 3.1 shows the achieved balanced dataset.

All the previous emails were in.txt format, so each email needed to be converted to HTML email, rendered, then captured as a screenshot, after that OCR technique was applied to extract the text again and a data cleansing process was applied, at the end we have two identical datasets, one is the images(screenshots) and the other is the text extracted from these images. Each sample in a dataset has a corresponding, with the same ID in the other dataset. Now let us Describe the structure of the email itself.

#### 3.1.1 SpamAssassin public mail corpus

The email data collection has 6,047 total labeled messages separated into 5 groups:

- spam – 500 real-world spam emails that got past traps, showing true threat tactics
- easy-ham – 2,500 non-spam emails that are easy to recognize as safe, often with simple text
- hard-ham – 250

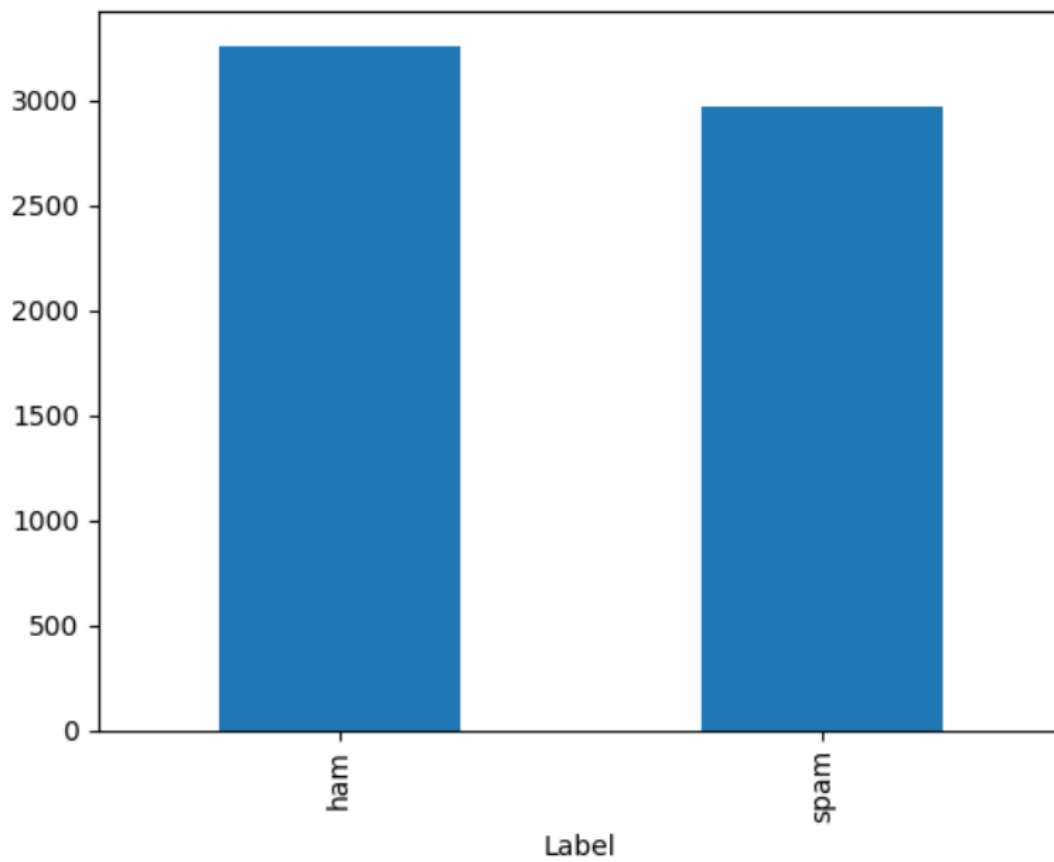


Figure 3.1: Combined dataset distribution

non-spam emails made to mimic spam, with tricky features like HTML, graphics easy-ham-2 – 1,400 additional straightforward clean emails added more recently spam-2 – 1,397 new spam emails also just added to expand threats covered So about 31% of emails represent actual spam, the rest are non-spam examples divided into obvious “easy” cases and tricky “hard” cases made to appear potentially malicious.

### 3.1.2 Enron-Spam datasets

The Enron-spam directory contains the Enron-Spam datasets, as described in paper [38] :

The ”preprocessed” subdirectory contains the messages in the preprocessed format that was used in the experiments of the paper. Each message is in a separate text file. The number at the beginning of each filename is the ”order of arrival”.

The ”raw” subdirectory contains the messages in their original form. Spam messages in non-Latin encodings, ham messages sent by the owners of the mailboxes to themselves (sender in ”To:”, ”Cc:”, or ”Bcc” field), and a handful of virus-infected messages have been removed, but no other modification has been made. The messages in the ”raw” subdirectory are more than the corresponding messages in the ”preprocessed” subdirectory, because (a) duplicates are preserved in the ”raw” form, and (b) during the preprocessing, ham and/or spam messages were randomly subsampled to obtain the desired ham:spam ratios.

### 3.1.3 Hand crafted datasets

After converting the text emails to HTML emails, a few samples have been minimally modified by applying some spam tricks, trying to Emulate an Adversary Environment to test it later, like spam word spacing using HTML comments, ham and spam words injection, size and bold effects, at the end most of these samples succeeded in misleading the spam classifier.

## 3.2 Supportive middle layer for Spam Classifier

**Summarization** is the task of producing a shorter version of a document while preserving its important information. Some models can extract text from the original input, while other models can generate entirely new text [40].

Even though language models like **Bidirectional transformer (BERT)** and **Generative Pre-trained Transformer 2 (GPT-2)** are great at text classification on their own, we wanted to try a new approach. Instead of relying solely on these powerful models, we decided to experiment with a different strategy. As illustrated in figure 3.2, we aimed to first summarize the text using various techniques and then run classification on these condensed versions.

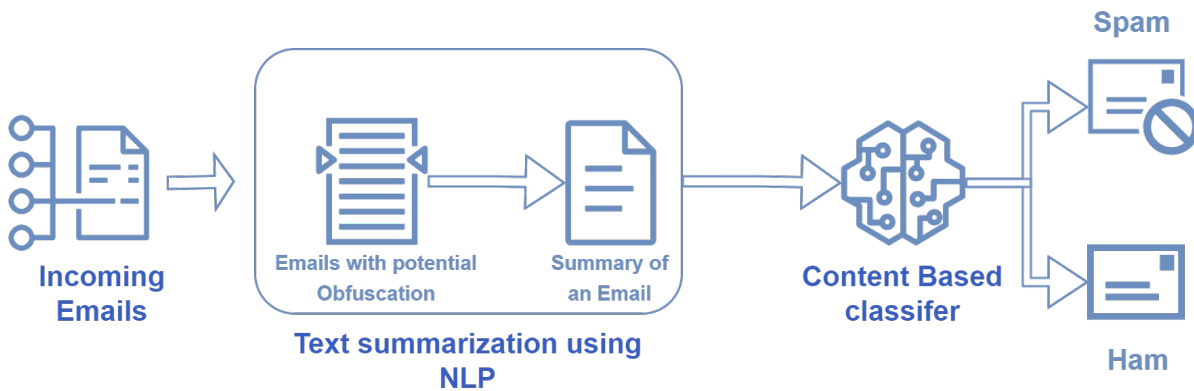


Figure 3.2: Emails classification after summarization process flow

### 3.2.1 Summarization layer

Leveraging the capabilities of the Hugging Face platform [41], we seamlessly incorporated and fine-tuned various language models designed specifically for the text summarization process within our experimental framework. This allowed us to harness the power of advanced natural language processing tools to examine the efficiency and effectiveness of such summarization techniques. Now let's explore the two primary types of text summarization:

- **Extractive summarization:** This method follows a traditional approach, aiming to identify key sentences within the text and incorporate them into the summary. It's important to note that the resulting summary comprises sentences directly lifted from the original text [42]. [43]
- **Abstractive summarization:** An advanced technique that focuses on grasping essential sections, interpreting the contextual meaning, and generating a condensed version that effectively communicates the core information. Unlike extractive methods, here the summary sentences are created by the model, not simply extracted from the source text. This approach ensures the delivery of critical information in the most succinct form possible [42]. [43]

#### Text Summarization with BERT

The paper "Attention is all you need" introduced a groundbreaking architecture known as Transformers. This network employs the encoder-decoder architecture, resembling the RNN architecture but with the notable capability of processing input sequences in parallel [44]. BERT is a transformer used to overcome the limitations of RNN and other neural networks as long-term dependencies. It is a pre-trained model that is naturally bidirectional. This pre-trained model can be tuned to easily perform the NLP tasks as specified, summarization in our case.

In [43], the authors showcase how BERT can be usefully applied in text summarization and propose a general framework for both extractive and abstractive models. introducing a novel document-level encoder based on BERT which can express the semantics of a document and obtain representations for its sentences.

```
port transformers
om summarizer import Summarizer, TransformerSummarizer
rt_model = Summarizer()
r index, row in tqdm(df_bert.iterrows(), total=len(df_bert), desc="Processing"):
    processed_text = ''.join(bert_model(row['content'], min_length=60))
    df_bert.at[index, 'content'] = processed_text
```

## Text summarization with GPT-2

The GPT-2 language model was introduced for the first time in 2018 in the paper “Language Models are Unsupervised Multitask Learners”[45], to develop a system that can learn from previously existing text. It would be able to produce multiple options for completing a phrase in this way, saving time and adding diversity and linguistic depth to the text. GPT-2 is capable of several tasks, including summarization, generation, and translation. To add, that was all done with a complete absence of grammatical errors[46]. At the heart of the GPT-2 architecture lies the Attention layer. Without delving too deeply into its technical intricacies, let’s outline its fundamental features:

GPT2 uses Byte Pair Encoding to create the tokens in its vocabulary. This means the tokens are usually parts of words.

GPT-2 was trained with the goal of causal language modeling (CLM) and is thus capable of predicting the next token in a sequence. GPT-2 may create syntactically coherent text by utilizing this capability.

GPT-2 generates synthetic text samples in response to the model being primed with an arbitrary input. The model is chameleon-like — it adapts to the style and content of the conditioning text.

## Text Summarization with XLNet

eXtreme Language understanding NETwork (XLNet), an advanced transformer-based language model, is an acronym for “eXtreme Language understanding NETwork.” It was specifically crafted to address certain constraints found in earlier

models such as BERT (Bidirectional Encoder Representations from Transformers). While BERT models typically handle input sequences in a bidirectional fashion, XLNet distinguishes itself by enabling the model to consider all conceivable permutations of the input sequence through permutation-based training. This approach empowers XLNet to capture a broader range of contextual information, enhancing its performance across diverse natural language processing tasks.[44] XLNet brings forth the concept of "autoregressive" training, autoregressive models predict the subsequent token in a sequence by considering the preceding tokens. This approach enhances the ability to generate more coherent and contextually relevant texts. By seamlessly integrating autoregressive and permutation-based training, XLNet achieves improved performance across a range of tasks.[44]

### **Does summarization affect text Classifying?**

The goal was to investigate whether summarizing the text before classification could potentially enhance the results. By taking this two-step process - summarizing and then classifying, we hoped to uncover if this approach could offer improved accuracy or efficiency in detecting spam emails compared to just using the standalone capabilities of these advanced models. In the "Results and Analysis chapter", all the pros and cons of this proposed idea have been discussed.

## **3.3 Vision-Based Spam Filter - VBSF model**

The proposed idea involves automatically rendering each incoming email through a web browser and subsequently capturing screenshots of the entire email content. The rationale behind this approach is to provide the classifier with a representation of the email as it would be visually perceived by a human eye. Figure 3.3 shows the workflow of the proposed Vision-Based Filter, which is composed of the following elements :

- Incoming emails
- Rendering emails
- Screenshot capturing
- First pipeline, which is mainly text extraction from images using OCR followed by content-based filter

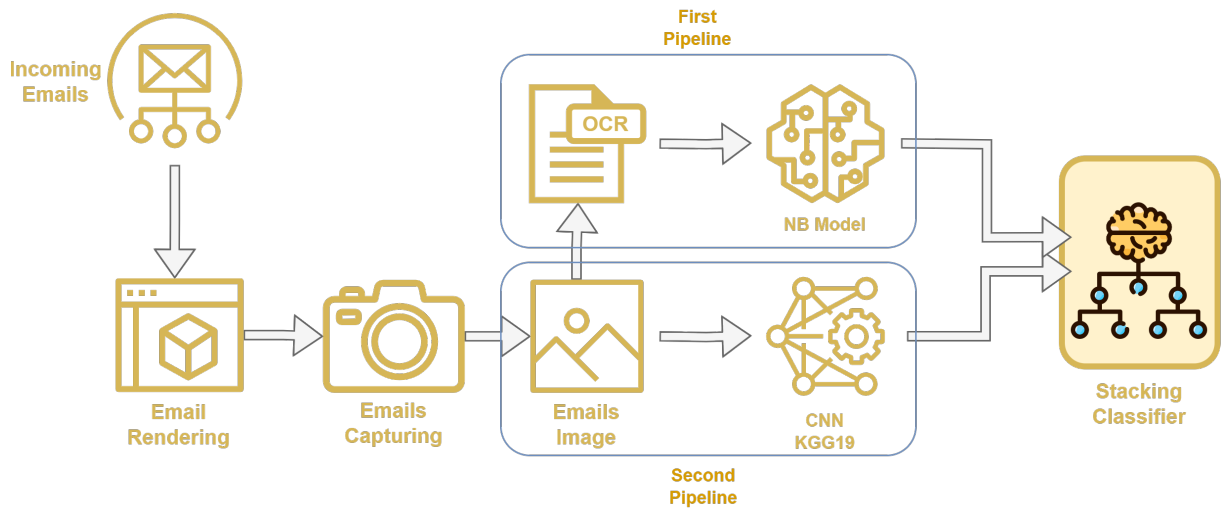


Figure 3.3: The proposed Vision-Based Spam Filter.

- Second pipeline, which is image-based classifier, mainly Convolutional Neural Network (CNN) model, applied on emails images
- Meta classifier, utilizing stacking ensemble method, as ensemble learning technique

These elements are described in detail below.

### 3.3.1 Incoming Emails

This element represents all kinds of emails that might arrive to an email client, in our case, all emails in our diverse data set are in their raw format, it was imperative to transform all the emails into HTML format as they required rendering within a browser tab, given that these emails lacked a specific file type and were presented solely as text.

By adopting this method, the aim is to mitigate common spam tricks that exploit HTML syntax and parsers(which are mostly visual) to conceal spam content effectively.

### 3.3.2 Email Rendering

The aim from this step is to mitigate common spam tricks that exploit Hyper Text Markup Language (HTML) syntax and parsers(which are mostly visual) to conceal

spam content effectively. Puppeteer[47], was adopted for the seamless automation of email rendering and capture. Puppeteer, a Node.js library, furnishes a sophisticated API that enables control over Chrome/Chromium through the DevTools Protocol. Operating by default in a headless mode, Puppeteer can also be configured to function in the full (“headful”) mode of Chrome/Chromium, offering versatile options for automated email processing [47].

### 3.3.3 Emails Capturing

The reasoning behind this step is explained with the following example, spammers often use HTML techniques such as hidden text, invisible elements, HTML comments, or misleading formatting to trick traditional text-based classifiers. By rendering the email in a browser and capturing screenshots, these visual tricks are exposed, allowing the final classifier to analyze the content like how a human user would view it. This approach can enhance the classifier’s ability to detect and classify spam accurately, as it takes into account the visual presentation of the email, uncovering potential malicious elements that might be hidden in the HTML code.

### 3.3.4 First Pipeline: Text Extraction and Content-Based Classification

#### Google Tesseract OCR engine

Tesseract was originally developed at Hewlett-Packard Laboratories. In 2005 Tesseract was open-sourced by HP. From 2006 until November 2018 it was developed by Google [48].

**Python-tesseract** is an optical character recognition (Optical Character Recognition (OCR)) tool for Python. That is, it will recognize and “read” the text embedded in images [49].

**Python-tesseract** is a wrapper for Google’s Tesseract-OCR Engine. It is also useful as a stand-alone invocation script to Tesseract, as it can read all image types supported by the Pillow and Leptonica imaging libraries, including jpeg, png, gif, bmp, tiff, and others. Additionally, if used as a script, Python-tesseract will print the recognized text instead of writing it to a file [49].<https://www.overleaf.com/project/6559d6efae94708a97d8376f>

As part of our methodology, Tesseract OCR has a crucial role in the proposed Vision Based Spam Filter (VBSF), it is used to extract the text part of the emails screenshots, store them in separate text files, and make them ready for further processing in the following steps.

### **Grammar and spelling Correction**

As part of the cleaning and enhancement process for the output of the OCR, a grammar and spelling analysis and correction has been applied, resulting in clearer and more correct text emails, which in turn helped in improving the accuracy of the following spam filter, since some spam tricks would be exposed from such spelling errors, such as Typosquatting. `pyspellchecker` is used in our case to correct the typos and grammar errors that exist in the extracted texts. **pyspellchecker** is Pure Python Spell Checking based on Peter Norvig's blog post [50] on setting up a simple spell-checking algorithm.

It uses a Levenshtein Distance algorithm to find permutations within an edit distance of 2 from the original word. It then compares all permutations (insertions, deletions, replacements, and transpositions) to known words in a word frequency list. Those words that are found more often in the frequency list are more likely the correct results.[51]

`pyspellchecker` supports multiple languages including English, Spanish, German, French, Portuguese, Arabic and Basque.

### **Naive Bayes Spam Filter**

The selection of the Naive Bayes (NB) classifier for our spam filter holds significance, as it stands out as one of the most renowned and efficacious content-based spam filters. We integrated the Naive Bayes (NB) classifier into the first pipeline to scrutinize the extracted text from the emails. Interestingly, when applied to the raw emails in our new dataset, the performance of a conventional NB classifier did not match the effectiveness observed in well-known email datasets.

However, things changed when we used the NB classifier on emails that went through Optical Character Recognition OCR after rendering and capturing. In this case, the NB classifier as shown in figure 3.4 demonstrated a remarkable improvement in performance. The adaptation of OCR technology appeared to enhance the classifier's ability to discern spam characteristics within the text, showcasing the

```

Model: Naive Bayes
Classification Report:
              precision    recall  f1-score   support

     0           0.94       0.97       0.96         816
     1           0.97       0.94       0.95         744

 accuracy                   0.96         1560
 macro avg           0.96       0.95       0.95         1560
 weighted avg        0.96       0.96       0.96         1560

Confusion Matrix:
[[794  22]
 [ 48 696]]

```

Figure 3.4: The accuracy of the NB model.

versatility of the NB classifier in the context of our VBSF’s first pipeline. This nuanced observation underscores the importance of tailoring spam filters to the unique characteristics of the dataset at hand, optimizing their performance for diverse sources and formats of email content.

### 3.3.5 Second Pipeline : Image-Based Classification using CNN

#### CNN for Image Classification

Image classification refers to the process of assigning labels or classes to input images, constituting a supervised learning task. In this scenario, a model is trained on labeled image data to forecast the class of images it hasn’t encountered before. Convolutional Neural Networks (CNNs) are frequently employed for image classification due to their capability to grasp hierarchical features such as edges, textures, and shapes. This ability facilitates precise recognition of objects within images. The strength of CNNs in this task lies in their capacity to autonomously extract meaningful spatial features from images, contributing to their effectiveness in image classification.[52]

## **Embedded text recognition using CNN**

Text embedded within an image has crucial information and conveys high-level semantics, rendering it a significant source of data and a subject of extensive research. Numerous investigations have demonstrated the efficacy and accuracy of CNN-based neural networks in image classification, forming the foundation for text recognition, which is specifically what we need in the case of Email classification. In [53], The research results showed that the best accuracy is the model that trained using VGG-16 architecture applied with some data augmentation techniques. These findings lead to the conclusion that a pre-trained CNN can achieve high accuracy in text recognition. The model architecture employed in this study can serve as a valuable reference for the future development of text detection systems.

## **VGG-19 CNN**

Simonyan and Zisserman from the university of Oxford developed a CNN with 19 layers (16 convolutional, 3 fully-connected) known as the VGG-19 model. This architecture strictly employs  $3 \times 3$  filters with a stride and padding of 1, accompanied by  $2 \times 2$  max-pooling layers with a stride of 2. In comparison to AlexNet, VGG-19 as its architecture figure depicts 3.5 is a deeper CNN with an increased number of layers. To manage the parameters in these deep networks effectively, VGG-19 utilizes small  $3 \times 3$  filters across all convolutional layers, resulting in optimal performance reflected in its 7.3 error rate [54].

## **Implementation**

While improving our neural network, specifically using the VGG-19 architecture, we went through a process of fine-tuning various settings to attain optimal predictive performance. Several key hyperparameters were precisely adjusted, including the learning rate (the most important), the number of training epochs, and the number of fine-tuning epochs. Additionally, we incorporated data augmentation techniques to further enhance the model's ability to generalize patterns from our dataset.

To add, choosing an appropriate learning rate was a critical step in achieving the best results. Through a systematic exploration of various learning rates, we identified the spot that led to a remarkable validation loss, without overfitting or underfitting the data.

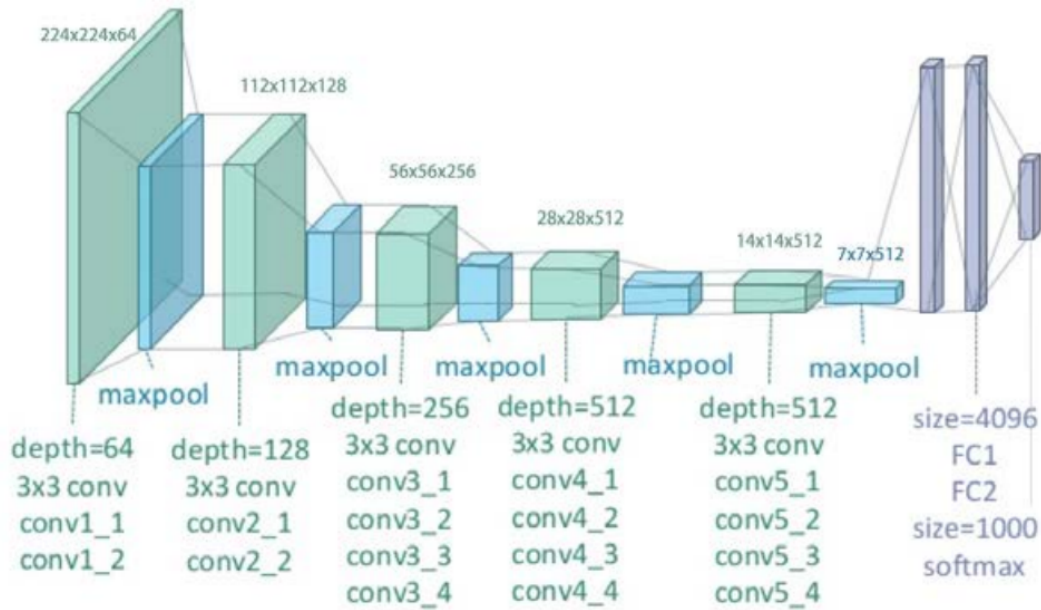


Figure 3.5: Architecture of CNN VGG19 Model

### 3.3.6 Final Prediction Decision

In order to consolidate the classification predictions from both pipelines, which are predictions of baseline models, namely CNN VGG19 and the Naive Bayes (NB) model, within the VBSF, a sophisticated approach is adopted. This involves the incorporation of a meta-classifier utilizing the stacking ensemble method, as depicted in Figure 3.6. The stacking classifier architecture elements depicted in the figure are as follows:

- Training data, the training portion of the dataset we have.
- Baseline classification models, at least two, no specific number, in our case NB and CNN models.
- Predictions of the base models, the training predictions of the models in binary form, also called stacking features, which are fed as input training data for the meta classifier
- Meta classifier, in our case is Logistic Regression (LR) classifier, which is trained on the training predictions of the base classifiers, in its output the final binary prediction of the whole architecture.

The chosen stacking classifier in this context is Logistic Regression. This pivotal component is strategically fed with the predictions generated by the CNN classifier

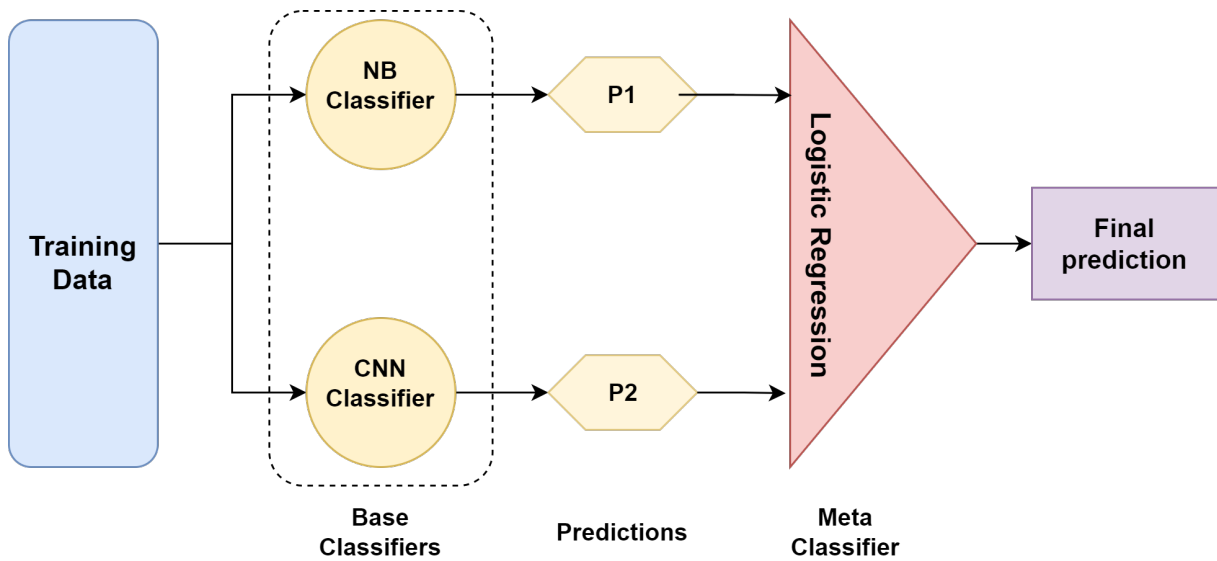


Figure 3.6: Stacking Classifier Architecture.

and the NB classifier. The amalgamation of these predictions serves as the input to the stacking classifier, and its ultimate output becomes the conclusive prediction for the final classification decision.

The stacking ensemble methodology empowers the system to leverage the unique strengths and capabilities of each individual classifier, enhancing the overall predictive accuracy and robustness of the final decision-making process. This strategic combination ensures a more comprehensive and nuanced analysis of the input data, leading to a more reliable and informed final classification outcome. Interestingly, we experimented with various models, such as Support Vector Machine (SVM), decision trees, and random forests, as potential stacking classifiers. However, after rigorous evaluation, the results revealed that Logistic Regression emerged as the best fit. The utilization of Logistic Regression as the stacking classifier further refines the integration process, providing a well-balanced synthesis of the predictions from both baseline models. This holistic approach underscores the commitment to achieving an accurate final classification decision in the VBSF architecture.

### 3.3.7 Enhanced Vision-Based Spam Filter

After employing a meta-classifier, utilizing the stacking ensemble method, to consolidate predictions from our proposed model, we noted an enhancement in our final test accuracy, achieving an improvement of the highest classifier accuracy by approximately 1.2 percent. Encouraged by this progress, we were driven to explore

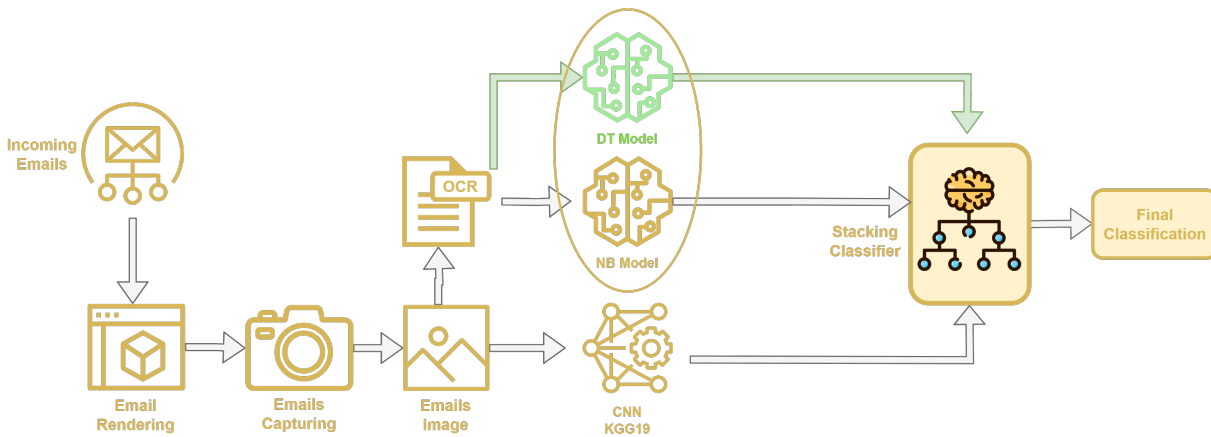


Figure 3.7: Enhanced Vision-Based Spam Filter VBSF

avenues for further enhancement. Recognizing the potential for increased performance with a diverse set of base classifiers, we embarked on refining our approach.

We proposed an enhanced variant of the VBSF as depicted in figure ??, augmenting our existing model, more specifically, the first pipeline, with a decision tree classifier alongside the established Naive Bayes (NB), working in parallel with the second pipeline, Convolutional Neural Network (CNN vgg19) model. This strategic addition aimed to capitalize on the inherent strengths of each classifier, leveraging their diverse nature to improve predictive accuracy.

The outcomes were compelling. Through experimentation and evaluation, we observed a remarkable increase in testing accuracy. The integration of the decision tree classifier proved to be particularly impactful, contributing to a significant enhancement in predictive performance.

These findings underscore the importance of strategic model composition and the value of incorporating diverse classifiers to achieve superior results. Our enhanced VBSF represents a promising advancement in predictive modeling, offering a pathway for further refinement and optimization of our approach.

# Chapter 4

## Results and Analysis

### 4.1 Dataset Analysis

Below, two figures depict the lengths of emails in the dataset, measured in characters. The first figure 4.1 illustrates the distribution and lengths for the entire emails dataset, while the subsequent figure 4.2 differentiate between spam and ham emails. It is recognizable that spam emails tend to exhibit greater length. These plots provide insights into potential features that could be exploited in email classification algorithms. Leveraging email length as a distinguishing attribute may enhance the efficacy of spam detection systems.

#### 4.1.1 Analysis of Dataset distribution

In the dataset creation phase, after merging parts from two datasets (Enron1 and SpamAssassin) to achieve the desired diversity, we noticed an imbalance, the frequency of ham emails outweighed that of spam, with a ratio of nearly 35% for spam and 65% for ham emails, as illustrated in Fig 4.3. When dealing with classification tasks, accuracy is a commonly used metric to evaluate the performance of a model. Accuracy measures the portion of correctly classified instances out of the total instances. However, accuracy alone can be misleading, especially in the presence of imbalanced datasets. An imbalanced dataset occurs when one class (e.g., spam) is significantly more prevalent than the other class (e.g., ham). In the context of email classification, it's not uncommon to have a higher proportion of ham emails compared to spam emails. Let's consider a scenario where we have a dataset of 10,000 emails, out of which only 500 emails are spam (5% of the total) and the rest are

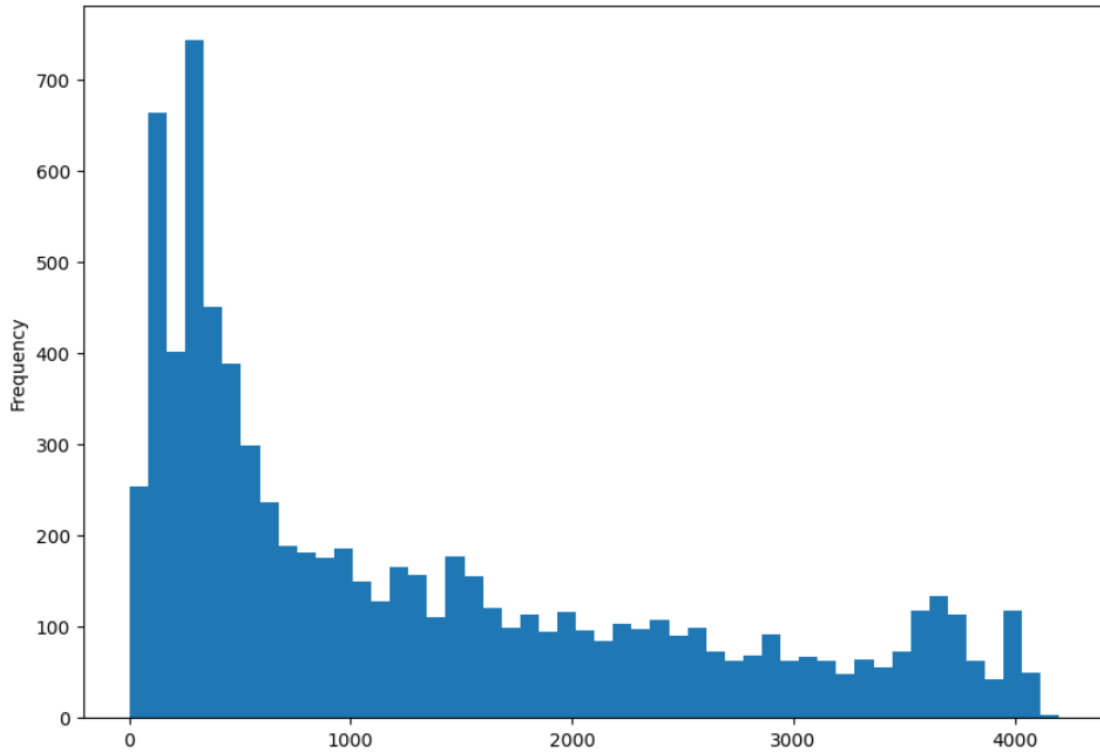


Figure 4.1: Frequency of occurrence of email lengths

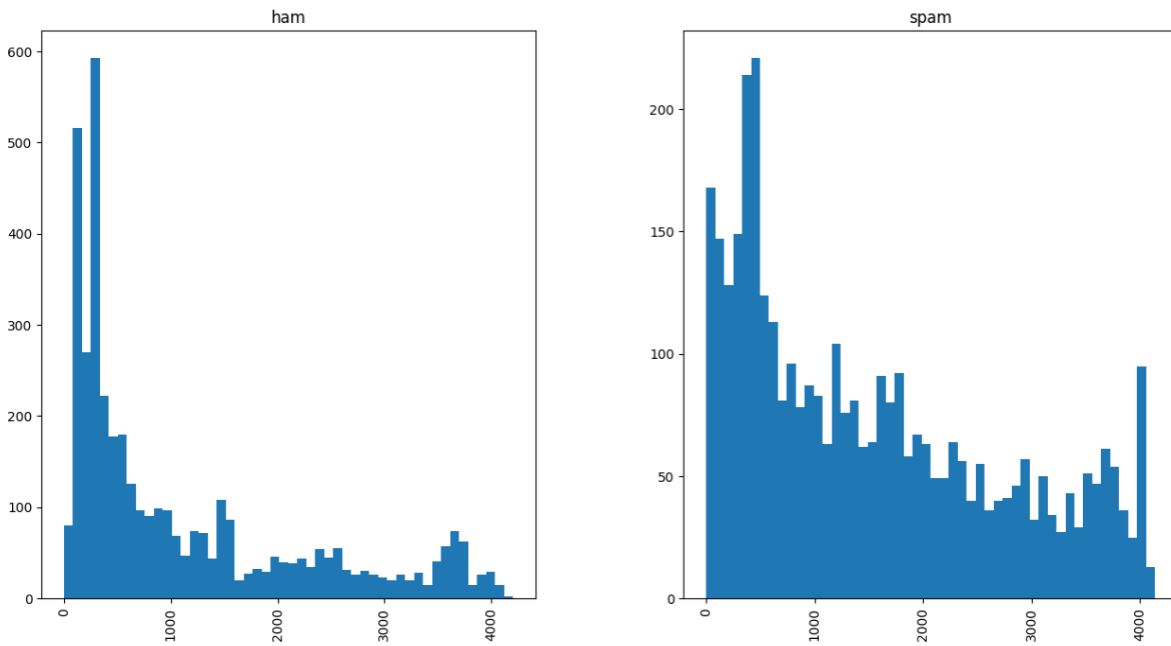


Figure 4.2: Frequency of occurrence of spam and ham email lengths

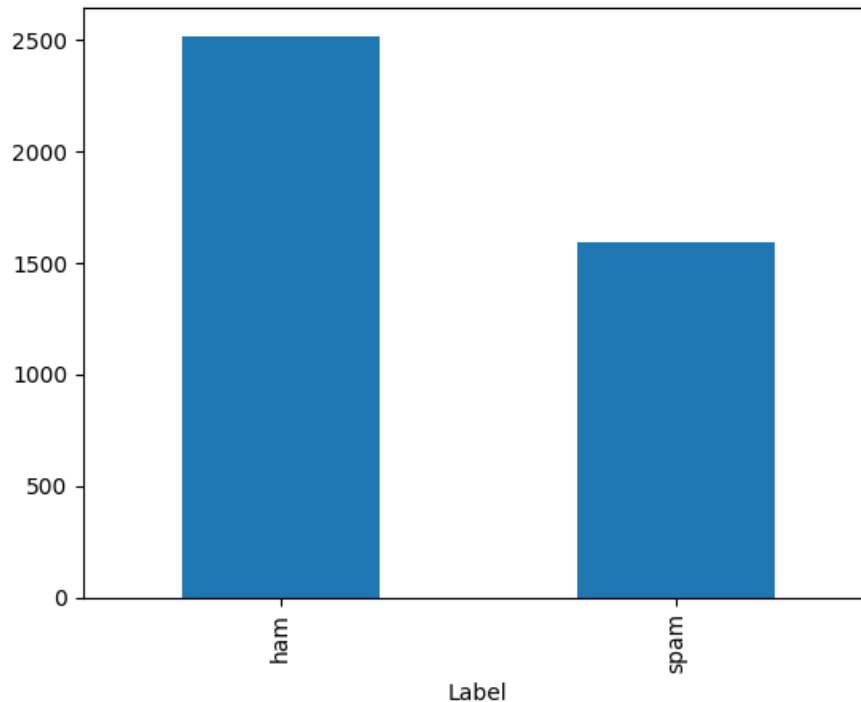


Figure 4.3: The distribution of the training dataset before improving the balancing of the data.

ham. If we train a classifier on this imbalanced dataset and it predicts all emails as ham, we would still achieve an accuracy of 95% because the majority class (ham) dominates the dataset, that’s why a more balanced email dataset was needed, and we oversampled the minority class (spam) and added samples from (Enron4) to get the desired balance as in figure 4.4.

## 4.2 Data Summarization

Our experiment involved summarizing emails before subjecting them to spam-ham classification. We observed that this preprocessing step yielded promising results, particularly for long emails. The summarization process helped to condense lengthy emails into concise representations, which facilitated understanding and classification. One notable finding from our analysis was that the type of summarization technique played a significant role in the effectiveness of the classification process. We experimented with both extractive and abstractive summarization methods. Extractive summarization involves selecting and extracting important sentences or phrases directly from the text, while abstractive summarization involves generating new sentences that capture the essence of the original text. We found that extractive sum-

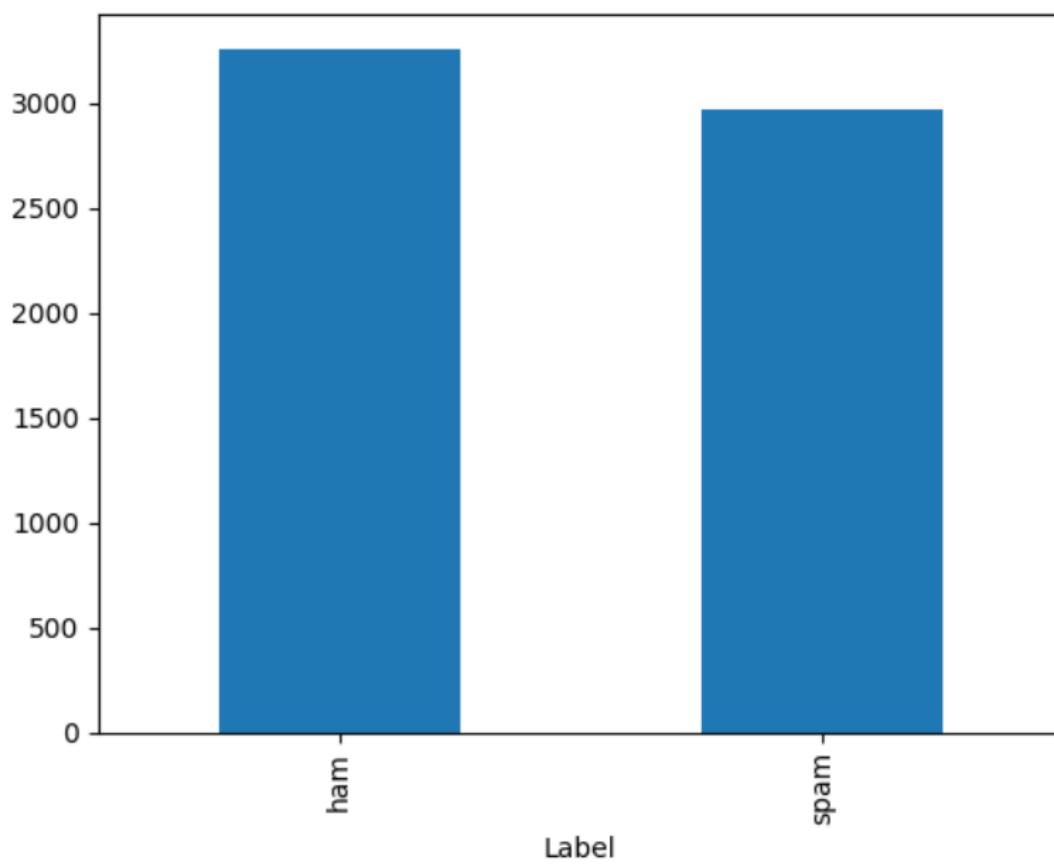


Figure 4.4: The distribution of the training dataset after improving, increasing the number of samples and balancing.

marization performs slightly better for spam-ham classification tasks. This could be attributed to the fact that extractive summarization retains the original wording and context more accurately, making it easier for the classification algorithm to identify spam-triggering words or patterns. However, despite the overall success of the summarization approach, we encountered some limitations. One notable drawback was the potential loss of crucial information during the summarization process. Fig4.5 shows the decrease in accuracy of Naive Bayes classifier after it was 94% on emails before processing, and figure4.6 shows a slight improvement on classification accuracy, however, both are still lower than the before-summary one. To justify, more specifically, some spam-triggering words or phrases might be deleted or overlooked in the summarization, leading to misclassification of certain emails. Nevertheless, the summarization process proved to be beneficial in mitigating certain spam tactics, such as ham words injection. In cases where spam emails were designed to mimic legitimate correspondence by injecting ham words or phrases, the summarization process helped to detect and remove anomaly-injected words. This is because the summarization model, to some extent, understands the semantic context of the email content and can identify and filter out anomalous or irrelevant terms.

The table 4.1 represents summary of results:

Model	Acc. after Summ.		2*Acc. before
	Extractive	Abstractive	
NB	91% (4.6)	89% (4.5)	94%

Table 4.1: Accuracy comparison of different models before and after summarization

In conclusion, while the summarization of emails before spam-ham classification provided significant improvements, especially for long emails, the choice between extractive and abstractive summarization depends on the specific requirements and characteristics of the classification task. Despite its limitations, summarization proves to be a valuable preprocessing step in enhancing the effectiveness of spam-ham classification systems.

### 4.3 Analysis of the Vision-Based Spam Filter VBSF

Rendering and capturing the incoming emails is the first step in the proposed filtering model, this process was done with the help of **Puppeteer** javascript framework, it is worth to mention that we needed to set a specific timeout in milliseconds while

```

Model: Naive Bayes
Classification Report:

```

	precision	recall	f1-score	support
0	0.84	0.98	0.90	1525
1	0.98	0.80	0.88	1464
accuracy			0.89	2989
macro avg	0.91	0.89	0.89	2989
weighted avg	0.91	0.89	0.89	2989

Figure 4.5: Accuracy of NB after abstractive summarization

```

100%|██████████| 1/1 [00:00<00:00, 1.35it/s]
Model: Naive Bayes
Classification Report:

```

	precision	recall	f1-score	support
0	0.89	1.00	0.94	464
1	0.98	0.69	0.81	188
accuracy			0.91	652
macro avg	0.94	0.84	0.88	652
weighted avg	0.92	0.91	0.90	652

Figure 4.6: Accuracy of NB after Extarctive summarization

Time	#	Log Message
25327.5s	1	OCR complete for images in /kaggle/input/emails-screenshots-enron-1-2/Emails screenshots Enron 1+2/screenshots_spam_E1+E2 , extracted text saved in /kaggle/working/extracted_text_spam_E1+E2/
25331.0s	2	/opt/conda/lib/python3.10/site-packages/traitlets/traitlets.py:2930: FutureWarning: --Exporter.preprocessors=["remove_papermill_header.RemovePapermillHeader"] for containers is deprecated in traitlets 5.0. You can pass `--Exporter.preprocessors item` ... multiple times to add items to a list.

Figure 4.7: Text Extraction using OCR Logs.

automatically rendering a capturing [55], this is justified by the fact that some pages in our case HTML emails contain many elements to appear [56], in our case, many emails have tables, colored features and other elements that take more time to parse and render, then capturing the content after it is fairly rendered to the browser, in this way, we had a full uninterrupted view of the emails, as a human might see.

### 4.3.1 The First Pipeline

#### Analysis of Google Tesseract OCR

Extracting the embedded text in all email images was one of the most time-consuming processes in our model, this could explained by the huge amounts of emails we got, many emails have tricky text positioning content combined with colors and some visual features. These samples consumed much more time than text-only emails screenshots for the OCR engine to process and extract in an acceptable precision. Figure 4.7 shows the logs of the text extraction process using OCR, which also indicate more than 7 hours as the duration of processing part of our dataset.

#### Analysis of the Naive Bayes Filter

The text content obtained from the images by applying OCR is fed to the Bayesian classifier. Fig.?? shows the accuracy of NB classifier on these emails, which is around 96%, however, we might ask, what if we just applied NB classifier on these emails in their raw format, before rendering and capturing and extracting content using OCR, Actually, due to the diversity of our dataset, it seems after experimenting some well-known classifiers (NB, SVM, Decision Tree (DT), LR) on raw emails

without any processing, all of these classifiers performed less in terms of accuracy. As we see in figure 4.8, the accuracy of NB classifier was fluctuating between 93% and 94%, mostly 93%, while in our first pipeline, NB accuracy reached 96%. This could be explained by the fact that, when rendering emails, or in real world, When an email client renders an email for display, it processes the HTML content included in the email body. This includes parsing all HTML tags and formatting the content according to their specifications. Whether it's normal content, known spam content hiding tricks, or crafty spam tactics, all elements are visually visible and ready for further investigation, in the first pipeline case, text extraction, and content-based classification.

### Analysis of the Decision Tree Filter

In a later phase, DT classifier was added as an improvement to the first pipeline, the choice of DT was because it performed very well on our data compared with others like SVM and AdaBoost and because it has a different nature from the used base model, which in turn helped a lot in enhancing the performance of the stacking classifier. Figure ?? displays the accuracy of DT on our dataset in its raw format 94%, figure 4.11 displays the accuracy of the dataset after applying the visual trick. Noticeable 3% improvement in the accuracy proves again, the success of the proposed technique.

The table 4.2 represents summary of the results of the used models, including some other models, that support our statement.

2*ML Model	Acc. on raw emails		Acc. after applying OCR	
	Acc.	Ref. Fig.	Acc.	Ref. Fig.
<b>NB</b>	94	4.8	96	4.9
<b>DT</b>	95	4.10	97	4.11
<b>LR</b>	96	4.12	97	4.13
<b>SVM</b>	80	4.14	96	4.15
<b>AdaBoost</b>	96	4.16	96	4.17
<b>KNN</b>	89	4.18	91	4.19

Table 4.2: Accuracy comparison of different machine learning models

```

100%|██████████| 3/3 [04:14<00:00, 84.88s/it]
Model: Naive Bayes
Classification Report:
              precision    recall  f1-score   support

     0       0.90      0.99      0.94       750
     1       0.98      0.85      0.91       557

 accuracy      0.93      1307
 macro avg     0.94      0.92      0.92      1307
weighted avg     0.93      0.93      0.93      1307

Confusion Matrix:
[[740  10]
 [ 86 471]]

```

Figure 4.8: NB accuracy for RAW Emails.

```

Model: Naive Bayes
Classification Report:
              precision    recall  f1-score   support

     0       0.94      0.97      0.96       816
     1       0.97      0.94      0.95       744

 accuracy      0.96      1560
 macro avg     0.96      0.95      0.95      1560
weighted avg     0.96      0.96      0.96      1560

Confusion Matrix:
[[794  22]
 [ 48 696]]

```

Figure 4.9: NB accuracy for Text content Extracted by OCR.

```

Model: Decision Tree
Classification Report:
              precision    recall  f1-score   support

     0         0.95         0.94         0.94         750
     1         0.92         0.94         0.93         557

 accuracy         0.94         1307
 macro avg         0.93         0.94         0.94         1307
 weighted avg         0.94         0.94         0.94         1307

Confusion Matrix:
[[702  48]
 [ 35 522]]

```

Figure 4.10: DT accuracy on Raw emails

```

Model: Decision Tree
Classification Report:
              precision    recall  f1-score   support

     0         0.98         0.96         0.97         816
     1         0.95         0.98         0.96         744

 accuracy         0.97         1560
 macro avg         0.97         0.97         0.97         1560
 weighted avg         0.97         0.97         0.97         1560

Confusion Matrix:
[[780  36]
 [ 17 727]]

```

Figure 4.11: Decision Tree accuracy on Text content Extracted by OCR.

```

Model: Logistic Regression
Classification Report:
              precision    recall  f1-score   support

     0         0.97         0.95         0.96         769
     1         0.95         0.97         0.96         538

 accuracy         0.96         0.96         0.96         1307
 macro avg         0.96         0.96         0.96         1307
 weighted avg         0.96         0.96         0.96         1307

Confusion Matrix:
[[744  25]
 [  12 526]]

```

Figure 4.12: LR accuracy on raw emails.

```

Model: Logistic Regression
Classification Report:
              precision    recall  f1-score   support

     0         0.98         0.98         0.97         816
     1         0.98         0.98         0.97         744

 accuracy         0.97         0.97         0.97         1560
 macro avg         0.97         0.97         0.97         1560
 weighted avg         0.97         0.97         0.97         1560

Confusion Matrix:
[[798  18]
 [   7 737]]

```

Figure 4.13: LR accuracy for Text content Extracted by OCR.

Model: Support Vector Machine

Classification Report:

	precision	recall	f1-score	support
0	0.76	0.98	0.86	769
1	0.96	0.55	0.70	538
accuracy			0.80	1307
macro avg	0.86	0.77	0.78	1307
weighted avg	0.84	0.80	0.79	1307

Confusion Matrix:

```
[[756  13]
 [242 296]]
```

Figure 4.14: SVM accuracy on raw emails.

Model: Support Vector Machine

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.92	0.96	816
1	0.92	1.00	0.96	744
accuracy			0.96	1560
macro avg	0.96	0.96	0.96	1560
weighted avg	0.96	0.96	0.96	1560

Confusion Matrix:

```
[[754  62]
 [  3 741]]
```

Figure 4.15: SVM accuracy for Text content Extracted by OCR.

```

Model: AdaBoost
Classification Report:
              precision    recall  f1-score   support

     0       0.96      0.96      0.96     769
     1       0.95      0.97      0.96     538

 accuracy          0.96     1307
 macro avg         0.96     1307
 weighted avg      0.96     1307

Confusion Matrix:
[[740  29]
 [ 17 521]]

```

Figure 4.16: AdaBoost accuracy on raw emails.

```

100%|██████████| 3/3 [1:10:51<00:00, 1417.28s/it]
Model: AdaBoost
Classification Report:
              precision    recall  f1-score   support

     0       0.98      0.93      0.96     816
     1       0.93      0.98      0.95     744

 accuracy          0.96    1560
 macro avg         0.96    1560
 weighted avg      0.96    1560

Confusion Matrix:
[[762  54]
 [ 16 728]]

```

Figure 4.17: AdaBoost accuracy for Text content Extracted by OCR.

Model: KNN

Classification Report:

	precision	recall	f1-score	support
0	0.95	0.86	0.90	769
1	0.83	0.93	0.88	538
accuracy			0.89	1307
macro avg	0.89	0.90	0.89	1307
weighted avg	0.90	0.89	0.89	1307

Confusion Matrix:

```
[[665 104]
 [ 36 502]]
```

Figure 4.18: KNN accuracy on raw emails.

---

Model: KNN

Classification Report:

	precision	recall	f1-score	support
0	0.96	0.86	0.91	816
1	0.86	0.96	0.91	744
accuracy			0.91	1560
macro avg	0.91	0.91	0.91	1560
weighted avg	0.91	0.91	0.91	1560

Confusion Matrix:

```
[[704 112]
 [ 32 712]]
```

Figure 4.19: KNN accuracy for Text content Extracted by OCR.

### 4.3.2 The Second Pipeline

#### Analysis of the CNN vgg19 Classifier

In the process of optimizing our neural network, specifically using the VGG-19 architecture, we underwent a hyperparameter tuning journey to attain optimal predictive performance. Several key hyperparameters were meticulously adjusted, including the learning rate (the most important), number of training epochs, and the number of fine-tuning epochs. Fine-tuning, a crucial phase in neural network training, was undertaken to refine the model's performance. It was noticeable while fine-tuning the pre-trained vgg19 model, starting with the first dataset before resampling and balancing it, we got the following outcomes, fig. 4.20 shows the learning process and helps to pick the appropriate learning rate of the model that gave the best accuracy and validation loss as appears in fig. 4.21, which after intense experiments of tuning the parameters, like the number of training and fine-tuning epochs and learning rate, reached about 95% accuracy, as depicted in figure 4.22, signifying that the model reached a level of proficiency after approximately 15 epochs. Figure 4.23 shows the confusion matrix that provides a comprehensive evaluation of the model's performance, this visual representation offers insights into how well our CNN classifier performed in classifying instances. However, since deep learning models are voracious for data, the more data, the better it will perform, this was noticeable while fine-tuning the model on the bigger and more balanced dataset of emails, ofcourse, we needed to retune the parameters, through immense number of combinations of these params to adapt better outcome results for the new dataset figure 4.24 shows the new validation loss plot, and finally we got 96% validation accuracy, 1 percent higher than previous one, as depicted in figure 4.25 and figure 4.26 shows the confusion matrix for detailed performance information of the model.

### 4.3.3 Analysis on the final classification decision

Since the chosen technique to combine the result of the VBSF model is a meta classifier, as a stacking ensemble technique, one of the most important steps was creating the training stacking features which are the training predictions of the base models, and fed to the meta classifier, and then testing stacking features for testing the model (meta classifier) which they are the testing predictions of baseline models, as the following code shows :

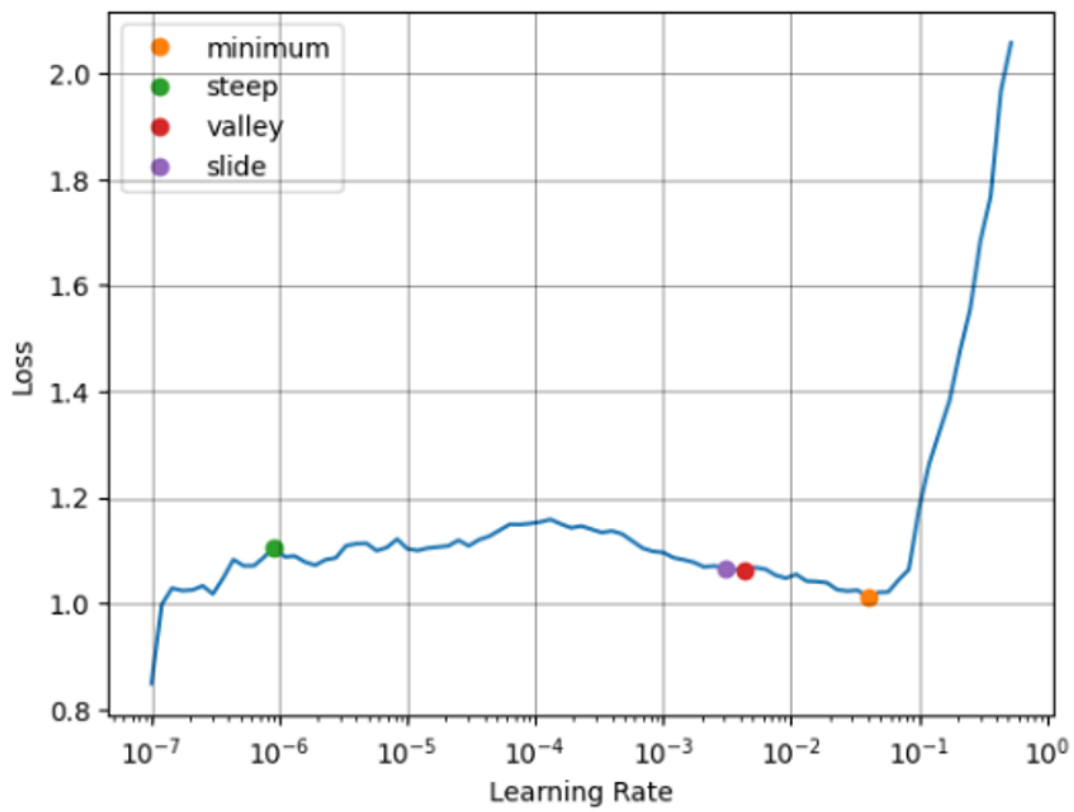


Figure 4.20: Learning rate of the vgg19 Model.

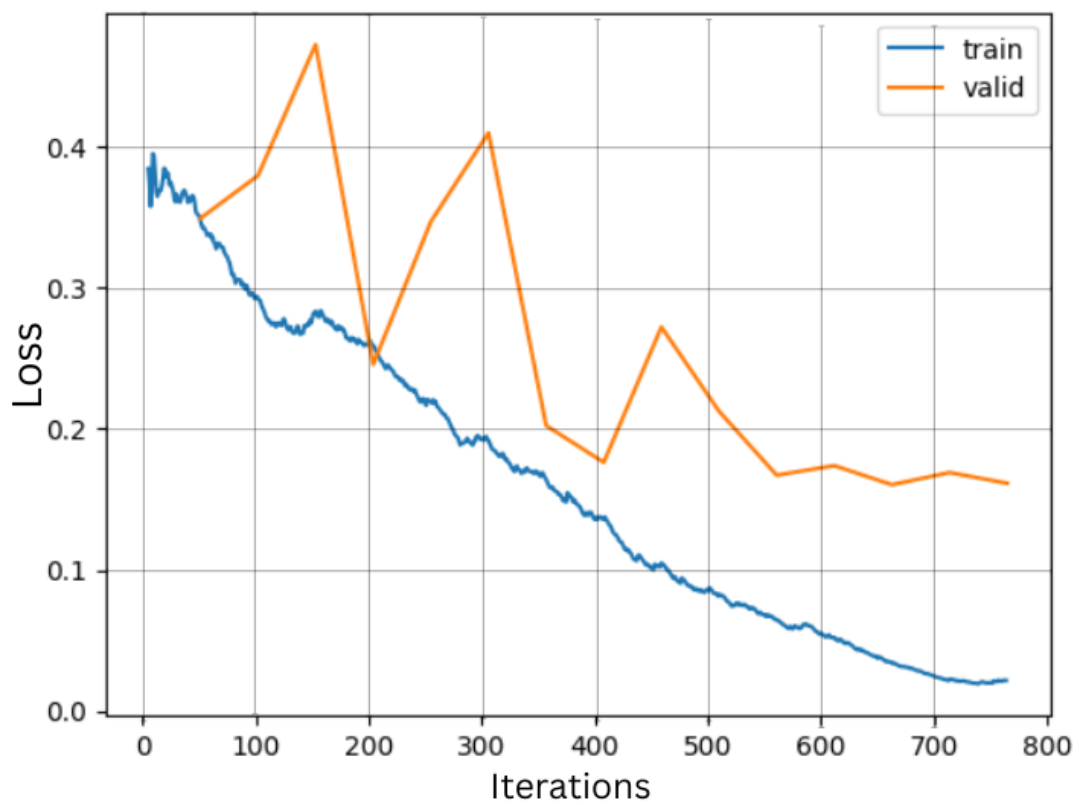


Figure 4.21: validation loss of the vgg19 Model.

epoch	train_loss	valid_loss	error_rate	accuracy	time
0	0.350998	0.349431	0.153098	0.846902	01:38
1	0.294199	0.379784	0.164034	0.835966	01:38
2	0.282551	0.472761	0.251519	0.748481	01:40
3	0.259852	0.245801	0.105711	0.894289	01:40
4	0.220030	0.347160	0.167679	0.832321	01:40
5	0.192567	0.410115	0.143378	0.856622	01:39
6	0.165496	0.202436	0.081409	0.918591	01:39
7	0.137330	0.176349	0.071689	0.928311	01:38
8	0.102355	0.272356	0.102066	0.897934	01:39
9	0.081476	0.212590	0.065614	0.934386	01:39
10	0.065034	0.167237	0.051033	0.948967	01:38
11	0.051733	0.173986	0.052248	0.947752	01:39
12	0.034350	0.160477	0.047388	0.952612	01:44
13	0.021542	0.169098	0.046173	0.953827	01:42
14	0.021616	0.161510	0.046173	0.953827	01:40

Figure 4.22: Accuracy reaching 95 % after fine tuning the vgg19 CNN model.

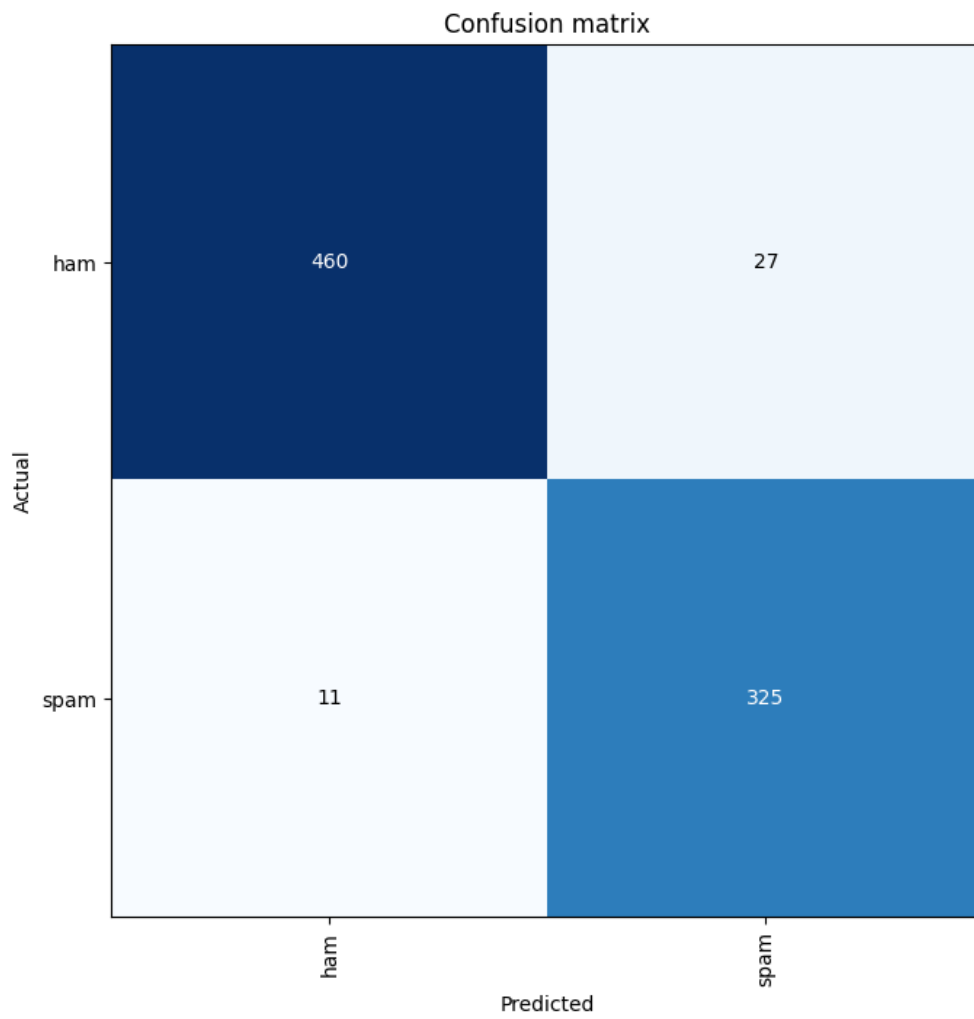


Figure 4.23: Confusion matrix for vgg19 on the first version of the dataset.



Figure 4.24: The new validation loss of the vgg19 Model after increasing and enhancing the data.

epoch	train_loss	valid_loss	error_rate	accuracy	time
0	0.346235	0.285582	0.128308	0.871692	02:33
1	0.274890	0.394381	0.165998	0.834002	02:32
2	0.272834	0.453315	0.177225	0.822775	02:32
3	0.253280	0.853118	0.207698	0.792302	02:33
4	0.242370	1.148375	0.391339	0.608661	02:32
5	0.214832	0.722867	0.268645	0.731355	02:31
6	0.171879	0.266362	0.093023	0.906977	02:32
7	0.152272	0.588061	0.149960	0.850040	02:30
8	0.130725	0.282588	0.105052	0.894948	02:31
9	0.102808	0.227434	0.070569	0.929431	02:30
10	0.083890	0.275016	0.068164	0.931836	02:31
11	0.064075	0.206691	0.057739	0.942261	02:31
12	0.051089	0.182422	0.048115	0.951885	02:34
13	0.029928	0.165781	0.043304	0.956696	02:33
14	0.029107	0.132280	0.039294	0.960706	02:32
15	0.019319	0.156171	0.039294	0.960706	02:32
16	0.013609	0.148674	0.039294	0.960706	02:32

Figure 4.25: Accuracy reaching 96 after fine tuning the vgg19 CNN model on the enhanced balanced dataset

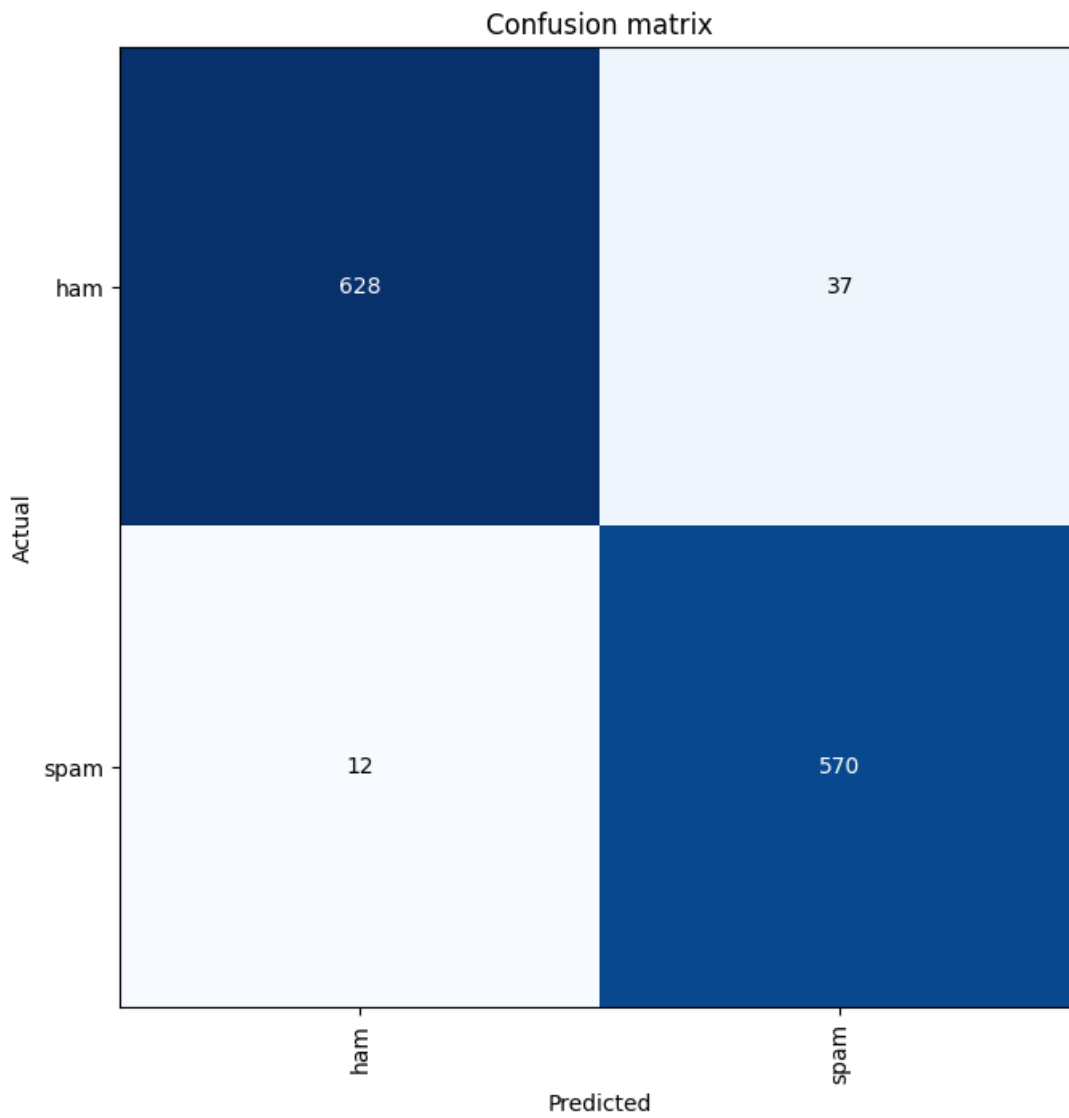


Figure 4.26: Confusion matrix for vgg19 after applied on the enhanced dataset.

```

stacked_features_train =
np.column_stack(
(CNN_Train_Pred["Label"],
NB_Train_Pred["Label"],
DT_Train_Pred["Label"]))

stacked_features_test =
np.column_stack(
(CNN_Test_Pred["Label"],
NB_Test_Pred["Label"],
DT_Test_Pred["Label"]))

```

We were expecting a final result equal to or exceeding the best base classifier's accuracy, so we ensured to investigate the testing accuracy of each classifier, figure 4.31 shows the test accuracy of the meta classifier, many meta classifiers has been tested, and Logistic regression seemed to give better than others, it was the chosen one.

### **The Enhanced VBSF Analysis**

We improved the proposed model by supporting NB classifier with the Decision Tree classifier(applied on the text extracted by the OCR engine) in the first pipeline. Figure 4.27 shows an output representing the stacking data shape for training and testing. We demonstrated the test accuracies of all baseline models in figures 4.30, 4.28 and 4.29 , for comparison with the final meta classifier as the following, NB classifier 95.5%, DT classifier 96.6%, CNN VGG19 95.7%. Figure 4.32 displays the test accuracy of the meta classifier after augmenting the first pipeline of the VBSF. Several meta-classifiers underwent testing, again, among which Logistic Regression produced superior performance compared to others reaching more than 98% accuracy, hence it was selected as the preferred choice.

The outcomes were compelling. Through experimentation and evaluation, we observed a remarkable increase in testing accuracy. The integration of the decision tree classifier proved to be particularly impactful, contributing to a significant enhancement in predictive performance. These findings underscore the importance of strategic model composition and the value of incorporating diverse classifiers to achieve superior results. Our enhanced variant of VBSF represents a promising

```
CNN Training Prediction DataFrame shape: (6239, 2)
NB Training Prediction DataFrame shape: (6239, 2)
CNN Testing Prediction DataFrame shape: (1560, 2)
NB Testing Prediction DataFrame shape: (1560, 2)
DT Train Prediction DataFrame shape: (6239, 2)
DT Test Prediction DataFrame shape: (1560, 2)
```

Figure 4.27: Stacknig data shape

```
In [12]: accuracy_score(Test_GT["Label"], NB_Test_Pred["Label"])
Out[12]: 0.9551282051282052
```

Figure 4.28: NB test accuracy

```
In [11]: accuracy_score(Test_GT["Label"], DT_Test_Pred["Label"])
Out[11]: 0.966025641025641
```

Figure 4.29: Test accuracy DT

```
In [10]: accuracy_score(Test_GT["Label"], CNN_Test_Pred["Label"])
Out[10]: 0.9576923076923077
```

Figure 4.30: Test accuracy of CNN Model

Results:

	<b>Classifier</b>	<b>Training Accuracy</b>	<b>Test Accuracy</b>
<b>0</b>	Logistic Regression	0.992225	0.956311
<b>1</b>	Random Forest	0.992225	0.956311
<b>2</b>	Gradient Boosting	0.992225	0.956311
<b>3</b>	SVM	0.992225	0.956311
<b>4</b>	K-Nearest Neighbors	0.969874	0.938835
<b>5</b>	Decision Tree	0.992225	0.956311
<b>6</b>	Naive Bayes	0.992225	0.956311
<b>7</b>	Neural Network	0.992225	0.956311
<b>8</b>	AdaBoost	0.992225	0.956311

Figure 4.31: Meta classifiers accuracies

Results:

	Classifier	Training Accurcay :	Test Accuracy
0	Logistic Regression	0.99984	0.983333
1	Random Forest	0.99984	0.973077
2	Gradient Boosting	0.99984	0.966026
3	Decision Tree	0.99984	0.966026

Figure 4.32: Final stacknig test accuracy

advancement in predictive modeling, offering a pathway for further refinement and optimization of our approach.

# Chapter 5

## Conclusion

This thesis has addressed the challenge of battling spam emails, which persistently overwhelm email clients worldwide, presenting a range of threats from just being an annoyance to cybercrime. The evolution of spam tactics has demanded the adoption of more advanced defense mechanisms beyond traditional approaches. Our study has come up with a new idea that merges computer vision and machine learning techniques to replicate human visual perception in email analysis.

By employing a multi-step process imitating the natural processing of visual information by the human eye, alongside text extraction of email snapshots using Optical Character Recognition (OCR) followed by textual content classification using a Naive Bayes (NB) classifier, augmented by Decision Tree (DT) classifier, our system efficiently cleans and analyzes email text content. Moreover, integrating a Convolutional Neural Network (CNN) as a visual perception classification model enhances the system's ability to discern between spam and legitimate emails based on visual features and cues.

A remarkable strength of our proposed solution lies in its adaptability to the dynamic nature of spamming techniques, especially the visual ones. By integrating text-based and image-based classifiers in a meta-classifier using stacking ensemble learning, our system achieves an impressive final classification accuracy exceeding 98%. This holistic approach not only enhances accuracy but also ensures resilience against evolving spam tactics.

Going forward, the researchers in anti-spam technology need to prioritize continuous innovation and adaptation to emerging threats. Areas such as threat modeling, evaluation of ML model architectures' security, and advancements in security properties for learning algorithms deserve further exploration. In addition, achieving the

right balance between reliability objectives, including the accuracy, and privacy of machine learning models, remains a critical challenge.

In conclusion, our research represents a significant advancement in the ongoing battle against spam emails, offering a resilient and adaptable solution that addresses the evolving threat landscape. Through continuous collaboration and innovation, we can leverage these findings to foster a safer and more secure digital environment for all users.

# Bibliography

- [1] B. Nelson, R. B. I. P., and J. D. Tygar, “Introduction/example 1.1 (spam filter and data sanitization),” in *Adversarial machine learning*. Cambridge University Press, 2019.
- [2] E. Sanz, J. Gomez Hidalgo, and J. Cortizo, “Email spam filtering,” *Advances in Computers*, vol. 74, pp. 45–114, Jan. 2008.
- [3] F. Jáñez-Martino, R. Alaiz-Rodríguez, V. González-Castro, E. Fidalgo, and E. Alegre, “A review of spam email detection: Analysis of spammer strategies and the dataset shift problem,” *Artificial Intelligence Review*, vol. 56, no. 2, pp. 1163–1164, 2022. doi: 10.1007/s10462-022-10195-4.
- [4] A. Bhowmick and S. M. Hazarika, “E-mail spam filtering: A review of techniques and trends,” *Lecture Notes in Electrical Engineering*, pp. 583–590, 2017. doi: 10.1007/978-981-10-4765-7\_61.
- [5] Spamhaus, *The spamhaus block list*. [Online]. Available: <https://www.spamhaus.org/sbl/>.
- [6] P. Sousa, A. Machado, M. Rocha, P. Cortez, and M. Rio, “A collaborative approach for spam detection,” in *2010 2nd International Conference on Evolving Internet*, 2010, pp. 92–97. doi: 10.1109/INTERNET.2010.25.
- [7] R. Puzanghera, *Razor2, pyzor, spamcop and dcc setup*, 2023. [Online]. Available: <https://notes.sagredo.eu/en/qmail-notes-185/razor2-pyzor-spamcop-and-dcc-setup-251.html>.
- [8] G. Robinson, *A Statistical Approach to the Spam Problem*, 2003.
- [9] T. A. Meyer and B. Whateley, “Spambayes: Effective open-source, bayesian based, email classification system,” in *International Conference on Email and Anti-Spam*, 2004. [Online]. Available: <https://api.semanticscholar.org/CorpusID:2368172>.
- [10] P. Graham, *A plan for spam*, 2002.

- [11] S. Suryavanshi, A. Taralekar, R. Yadav, S. Deshmukh, and S. Pandey, "E-mail spam detection and classification using svm," *(IJCSIT) International Journal of Computer Science and Information Technologies*, vol. 11, 2020.
- [12] Z. Burgansky-Eliash, G. Wollstein, T. Chu, *et al.*, "Optical coherence tomography machine learning classifiers for glaucoma detection: A preliminary study," *Investigative Ophthalmology amp; Visual Science*, vol. 46, no. 11, p. 4147, 2005. doi: 10.1167/iovs.05-0366.
- [13] S. Khanday and S. Parveen, "Logistic regression based classification of spam and non-spam emails," *Proceedings of the 2nd International Conference on ICT for Digital, Smart, and Sustainable Development, ICIDSSD 2020, 27-28 February 2020, Jamia Hamdard, New Delhi, India, 2021*. doi: 10.4108/eai.27-2-2020.2303291.
- [14] J. S. Whissell and C. L. Clarke, "Clustering for semi-supervised spam filtering," *Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference*, 2011. doi: 10.1145/2030376.2030391.
- [15] M. Sasaki and H. Shinnou, "Spam detection using text clustering," *2005 International Conference on Cyberworlds (CW'05)*, 2005. doi: 10.1109/cw.2005.83.
- [16] M Basavaraju and R Prabhakar, "Clustered distributed index for efficient text retrieval using threads," *International Journal of Grid Computing amp; Applications*, vol. 1, no. 2, pp. 1–13, 2010. doi: 10.5121/ijgca.2010.1201.
- [17] [Online]. Available: <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>.
- [18] Alagesan, Rubin, and Julis, "Spam detection in sms using machine learning through textmining," *nternational Journal Of Scientific Technology Research*, vol. 9, 2020.
- [19] N. N. Amir Sjarif, N. F. Mohd Azmi, S. Chuprat, H. M. Sarkan, Y. Yahya, and S. M. Sam, "Sms spam message detection using term frequency-inverse document frequency and random forest algorithm," *Procedia Computer Science*, vol. 161, pp. 509–515, 2019. doi: 10.1016/j.procs.2019.11.150.
- [20] L. P. Guerra Velasco, *Semi-supervised subspace Clustering and applications to neuroscience*, 2010. doi: 10.20868/upm.thesis.14703.

- [21] 2023. [Online]. Available: <https://corporatefinanceinstitute.com/resources/data-science/ensemble-methods/>.
- [22] S. Raschka, *Stackingclassifier: Simple stacking*. [Online]. Available: [https://rasbt.github.io/mlxtend/user\\_guide/classifier/StackingClassifier/](https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/).
- [23] M. Adnan, M. O. Imam, M. F. Javed, and I. Murtza, “Improving spam email classification accuracy using ensemble techniques: A stacking approach,” *International Journal of Information Security*, vol. 23, no. 1, pp. 505–517, 2023. doi: 10.1007/s10207-023-00756-1.
- [24] A. D. Joseph, B. Nelson, R. B. I. P., and J. D. Tygar, “Availability attack case study: Spambayes,” in *Adversarial machine learning*. Cambridge University Press, 2019.
- [25] M. Sokolov, K. Olufowobi, and N. Herndon, “Visual spoofing in content-based spam detection,” *13th International Conference on Security of Information and Networks*, 2020. doi: 10.1145/3433174.3433605.
- [26] M. Sokolov, K. Olufowobi, and N. Herndon, “Visual spoofing in content-based spam detection,” *13th International Conference on Security of Information and Networks*, pp. 3–3, 2020. doi: 10.1145/3433174.3433605.
- [27] B. Kuchipudi, R. T. Nannapaneni, and Q. Liao, “Adversarial machine learning for spam filters,” *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020. doi: 10.1145/3407023.3407079.
- [28] [Online]. Available: [https://www.process.com/products/pmas/whitepapers/common\\_spammer\\_tricks.html](https://www.process.com/products/pmas/whitepapers/common_spammer_tricks.html).
- [29] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla, and F. Herrera, “A unifying view on dataset shift in classification,” *Pattern Recognition*, vol. 45, no. 1, pp. 521–530, 2012. doi: 10.1016/j.patcog.2011.06.019.
- [30] B. Biggio and F. Roli, “Wild patterns: Ten years after the rise of adversarial machine learning,” *Pattern Recognition*, vol. 84, pp. 317–331, 2018. doi: 10.1016/j.patcog.2018.07.023.
- [31] B. Kim, S. Abuadbba, and H. Kim, “Deepcapture: Image spam detection using deep learning and data augmentation,” *Information Security and Privacy*, pp. 461–475, 2020. doi: 10.1007/978-3-030-55304-3\_24.

- [32] D. Simester, A. Timoshenko, and S. I. Zoumpoulis, “Targeting prospective customers: Robustness of machine-learning methods to typical data challenges,” *Management Science*, vol. 66, no. 6, pp. 2495–2522, 2020. doi: 10.1287/mnsc.2019.3308.
- [33] R. Alaiz-Rodríguez and N. Japkowicz, “Assessing the impact of changing environments on classifier performance,” *Advances in Artificial Intelligence*, pp. 13–24, 2008. doi: 10.1007/978-3-540-68825-9\_2.
- [34] Quionero-Candela, Lawrence, Schwaighofer, and Sugiyama, *Dataset shift in machine learning*, 2008. doi: 10.7551/mitpress/9780262170055.001.0001.
- [35] S. J. Delany, P. Cunningham, A. Tsybal, and L. Coyle, “A case-based technique for tracking concept drift in spam filtering,” *Knowledge-Based Systems*, vol. 18, no. 4–5, pp. 187–195, 2005. doi: 10.1016/j.knosys.2004.10.002.
- [36] F. Fdez-Riverola, E. Iglesias, F. Díaz, J. Méndez, and J. Corchado, “Applying lazy learning algorithms to tackle concept drift in spam filtering,” *Expert Systems with Applications*, vol. 33, no. 1, pp. 36–48, 2007. doi: 10.1016/j.eswa.2006.04.011.
- [37] R. M. Mohammad, “A lifelong spam emails classification model,” *Applied Computing and Informatics*, vol. 20, no. 1/2, pp. 35–54, 2020. doi: 10.1016/j.aci.2020.01.002.
- [38] V. Metsis, I. Androutsopoulos, and G. Paliouras, “Spam filtering with naive bayes - which naive bayes?,” Jan. 2006.
- [39] G. V. Cormack, “Trec 2007 spam track overview,” in *The sixteenth Text REtrieval Conference (TREC 2007) Proceedings*, 2007, pp. 1–9.
- [40] H. Face, *What is summarization? - hugging face*. [Online]. Available: <https://huggingface.co/tasks/summarization>.
- [41] [Online]. Available: <https://huggingface.co/>.
- [42] S. Bag, *Text summarization using bert, gpt2, xlnet*, 2023. [Online]. Available: <https://medium.com/analytics-vidhya/text-summarization-using-bert-gpt2-xlnet-5ee80608e961>.

- [43] Y. Liu and M. Lapata, "Text summarization with pretrained encoders," *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019. doi: 10.18653/v1/d19-1387.
- [44] K. Amrutha, *Revolutionizing text summarization: Exploring gpt-2 and xlnet transformers*, 2023. [Online]. Available: <https://www.analyticsvidhya.com/blog/2023/07/exploring-gpt-2-and-xlnet-transformers/>.
- [45] A. Radford, J. Wu, R. Child, D. Amodei, and I. Sutskever, *Language Models are Unsupervised Multitask Learners*, 2018.
- [46] A. Hachcham, *Text summarization with gpt2 and layer ai*, 2022. [Online]. Available: <https://towardsdatascience.com/text-summarization-with-gpt2-and-layer-ai-599625085d8e>.
- [47] [Online]. Available: <https://pptr.dev/>.
- [48] Tesseract-Ocr, *Tesseract-ocr/docs: Various documents related to tesseract ocr*. [Online]. Available: <https://github.com/tesseract-ocr/docs>.
- [49] [Online]. Available: <https://pypi.org/project/pytesseract/>.
- [50] P. Norvig, *How to write a spelling corrector*. [Online]. Available: <https://norvig.com/spell-correct.html>.
- [51] T. Barrus, *Pyspellchecker*. [Online]. Available: <https://pyspellchecker.readthedocs.io/en/latest/>.
- [52] D. Sharma, *Image classification using cnn: Step-wise tutorial*, 2024. [Online]. Available: <https://www.analyticsvidhya.com/blog/2021/01/image-classification-using-convolutional-neural-networks-a-step-by-step-guide/#:~:text=CNN%20classifier%20for%20image%20classification,classes%2C%20enabling%20accurate%20image%20classification..>
- [53] A. F. Rizky, N. Yudistira, and E. Santoso, *Text recognition on images using pre-trained CNN*, 2023.
- [54] Y. Zheng, C. Yang, and A. Merkulov, "Breast cancer screening using convolutional neural network and follow-up digital mammography," *Computational Imaging III*, 2018. doi: 10.1117/12.2304564.

- [55] P. Thaleikis, *Dealing with timeouts in puppeteer*, 2022. [Online]. Available: <https://releasecandidate.dev/posts/2022/puppeteer-timeouts/>.
- [56] [Online]. Available: [https://www.webshare.io/blog/puppeteer-timeout#:~:text=Puppeteer%20allows%20setting%20a%20default,seconds%20\(10000%20milliseconds\)%20page..](https://www.webshare.io/blog/puppeteer-timeout#:~:text=Puppeteer%20allows%20setting%20a%20default,seconds%20(10000%20milliseconds)%20page..)