Knowledge Discovery and Data Mining Finals Report

1st 522H0036 - Luong Canh Phong
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
522H0036@student.tdtu.edu.vn

2nd 520H0341 - Nguyen Thai Bao
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
520H0341@student.tdtu.edu.vn

3rd 522H0030 - Le Tan Huy
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
522H0030@student.tdtu.edu.vn

4th 522H0008 - Dao Minh Phuc
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
522H0008@student.tdtu.edu.vn

5th 522H0136 - Nguyen Nhat Phuong Anh
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
522H0136@student.tdtu.edu.vn

***Abstract*—Spam emails are a common problem seen on the internet as it is an annoyance in daily life and a cyber security risks to any sensitive and important data of a person or an organization/business. With the number of spam emails increasing more and more significantly over the past few years, many more algorithms are created and improved in spam detection efficiency. Overall, this paper goes through the basic understanding of spam emails, understanding the necessity of a spam classification algorithm, and learn more about the methodologies, its effectiveness and usefulness when detecting spam emails.**

I. Introduction

Since the birth of the Internet, spam email has been a common occurrence. Along with the rapid growth and widespread of the Internet, the frequency has been increasing significantly, especially over the past decade. In addition to being nuisances, a waste of time and email storage, spam emails can be sent with malicious intent of stealing information, hijacking devices by storing malware within the content of the email itself. And with the nature of email spam being sent by botnets, it isn't easy to avoid the situation due to a new bot can be easily created in case another one got blocked or banned on the site. A common way how most platforms (such as Gmail, Yahoo!, Outlook) handle these spams is to develop a Machine Learning (ML) model to detect and get rid of the spam emails, lowering the number of spams getting into the inbox.

II. Importance of Spam Classification

To understand why spam classification is important to our lives, we must first understand the spam emails and its impact on daily life and businesses.

A. Different Types of Spam Emails

There are various forms of spam, sent with different intentions and purposes. But they are commonly grouped into:

- Phishing Emails
- Email Spoofing
- Technical support scams
- Current event scams
- Marketing/advertising email
- Malware scam

B. Problems with Spam Emails

According to statistics report in 2023, 160 billion spam emails are sent every day, which is 46% of the 347 billion emails sent daily. Out of which, the most common type being marketing/advertising emails, which take up around 36%, followed up with promotional of adult content, around 31.7% of total spam emails. Despite scam and fraudulent emails being the least common type, over 70% of them are phishing emails, which is still over 6 billion phishing emails are being sent to users daily.

A single spam email carbon emission is almost 0.03g of Carbon Dioxide Equivalent (CO_2e). With the amount of spam being sent daily, it can easily get nearly 5 tonnes of CO_2e being released every day. Additionally, two-thirds of spam receivers have been reported to have their mental health affected due to the number of spam or phishing scams.

For businesses, this spam can be sent as a way to get businesses to invest in nonexistent organizations under the guise of an investment and promise a high payback. For individuals, it would be under the form of bitcoin investment or for a charitable cause. Once the money is received, the sender would delete all traces and block the recipient contact.

III. Reading and Understanding the Dataset

After defining the business goals and the goals of data mining in the Business Understanding phase, the next crucial step in the CRISP-DM process is Data Understanding. This phase involves collecting the initial dataset, examining its structure, and verifying data quality. The primary goal is to become familiar with the dataset, identify potential quality issues, and extract early insights.

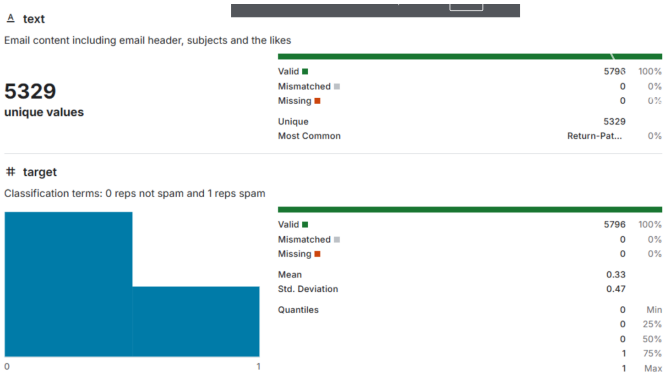

Fig. 1. Dataset Information

In this image, we can see that out of all 5679 text values which included email headers, subjects and the likes, 5329 of them are unique values. From this information, we can learn that the dataset has a large variety of values, and no duplicates are missing values that may impact the quality of the dataset significantly in any way.

Various NLP techniques were applied to explore the email content:

A. *Word Cloud*

Revealed high-frequency terms like number, url, date, mail, list, get, time, system, use, and HTML-related terms (font, nbsp, td, tr). The prevalence of number and url might indicate preprocessing/anonymization or simply be features of the emails. HTML terms suggest formatted emails are common.

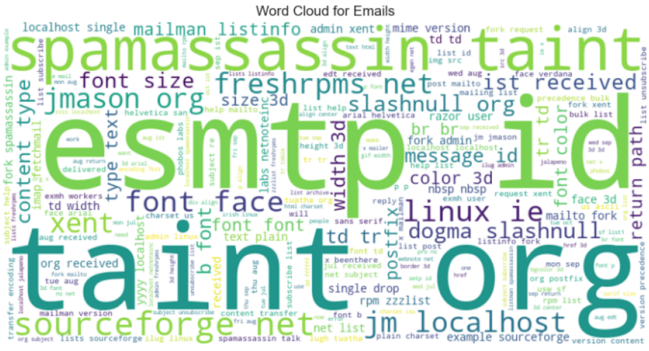

Fig. 2. All emails in the form of Word Cloud

Prominent Keywords:

- taint, spamassassin, esmtp, org, id, localhost, net, sourceforge: These are the largest and boldest words,indicating they are the most frequent and important terms within the analyzed email data.
- The rest of the word cloud: These words are smaller but still noticeable,suggesting they also play a significant role in the email context.

Possible Suggested Topics:

- Spam Filtering: The presence of “spamassassin” and “taint” (often associated with marking suspicious emails) suggests a significant topic might be the identification and handling of spam.
- Mailing List Management: Words like “list”,“listinfo”, “mailman”, and “mailing” hint at discussions related to managing and interacting with email lists.
- Technical Email Information: Terms such as “smtp” (Simple Mail Transfer Protocol), “mime” (Multipurpose Internet Mail Extensions), “version”,“id”, “localhost”, and “net” indicate potential discussions about the technical aspects of email.
- Formatting and Display: Words like “font”,“size”, “face”, “text”, and “color” might relate to how emails are displayed and formatted.
- Source and Path: “sourceforge” (an open-source software repository), “received”, and “path” could be related to tracking the origin and route of emails.
- Related Systems and Software: “linux”, “xent”, and “rpm” might be the names of operating systems or software mentioned within the email context.

B. *Word Frequency (Top 10)*

After removing basic English stopwords and non-alphabetic tokens, the bar chart confirmed the dominance of words like number, url, mail, date, list, get. Their persistence highlights their potential importance or noise factor.

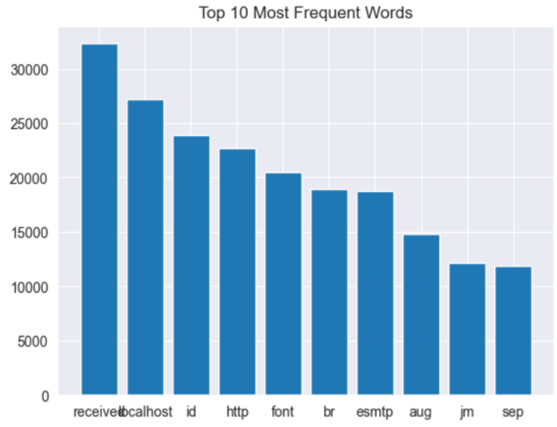

Fig. 3. Bar Chart for 10 Most Frequent Words

The most frequent word is “received”, with a significantly higher count (over 30,000) compared to the other top words. This suggests that the word “received” appears very often in the analyzed text data after the filtering.

The following most frequent words, in descending order, are “localhost”, “id”, “http”, “font”, “br”, “esmtp”, “aug”, “jm”, and “sep”. These words have frequencies ranging from approximately 11,000 to 27,000.

The nature of the top words suggests the analyzed text data might be related to email headers or technical logs. Words

like “received”, “localhost”, “id”, “http”, and “esmtp” are commonly found in email metadata. “aug” and “sep” likely refer to month abbreviations. “font” and “br” could appear in the content or formatting information.

The code successfully identifies and counts the occurrences of words after applying basic text cleaning. The use of nltk.word_tokenize splits the text into individual words, .lower() converts them to lowercase for consistent counting, and the list comprehension filters out non-alphabetic tokens and common English stop words.

The Counter object efficiently calculates word frequencies, and most_common(10) retrieves the top 10 most frequent words and their counts. This data is then used to generate the bar chart using matplotlib.pyplot.

The visualization clearly shows the relative frequency of the top 10 words. The height of each bar directly corresponds to the number of times that word appears in the processed text.

C. Email Length Distribution

The histogram showed a right-skewed distribution. Most emails are short (under 500 words), but a long tail indicates the presence of very long outlier emails, adding diversity to the dataset structure. Length could be a potential feature.

This code calculates how many words are in each email and then creates a bar chart (histogram) to show how many emails fall into different length categories (e.g., 0–500 words, 500–1000 words, etc.). It helps visualize the typical length of emails in our dataset.

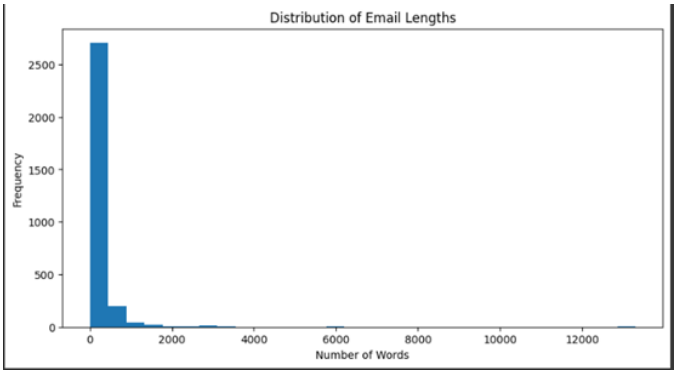

Fig. 4. Bar Chart for Email Length Distribution

Distribution Shape:

- Right-skewed: The majority of the bars are concentrated on the left side of the histogram, indicating that most emails are short in length.
- High Peak on the Left: The first bar (near zero words) has a significantly higher height than the other bars,showing a large number of very short emails.
- Long Tail to the Right: The bars gradually decrease towards the right, but there are still some emails with considerable length, extending to several thousand words.

D. Common N-grams (Trigrams)

Analysis of the top 10 trigrams showed patterns like “wed number aug”, “number aug number”, “date number number”, suggesting frequent date/number sequences or mailing list headers (list listinfo mail). These patterns might help identify specific email types.

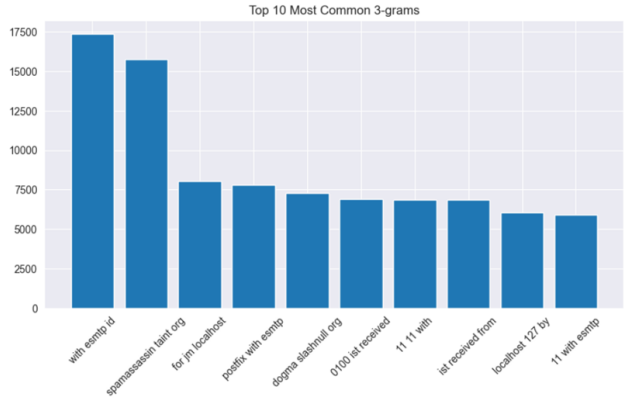

Fig. 5. Bar Chart for 10 Most Common Trigrams

Observations:

- The trigram “with esmtp id” is the most frequent, appearing significantly more often than the others.
- The trigram “spamassassin taint org” is the second most frequent.
- The remaining trigrams have lower and relatively similar frequencies.
- The trigrams suggest that the email data contains technical information, likely related to email headers and processing (“esmtp”, “localhost”, “received”) and spam filtering (“spamassassin”, “taint”).

Potential Meaning:

- The most frequent trigrams indicate the common occurrence of information related to email headers and the email transmission process. Phrases like “with esmtp id”, “postfix with esmtp”, “ist received”, “received from”, and “localhost 127 by” all pertain to how emails are sent and received.
- The presence of “spamassassin taint org” suggests that spam filtering activity is a significant aspect of this dataset.
- Other trigrams like “for jm local/host”, “dogma slashnull org”, “0100 ist received”, “11 11 with”, and “11 with esmtp” might relate to specific systems, software, or characteristic data formats within the context of this dataset.

Conclusion:

This chart reveals that information related to email processing and headers, as well as spam filtering activities, are prominent features in the analyzed text dataset. The other trigrams may provide further insights into specific systems or formats relevant to this data.

E. POS Tag Frequency

The distribution of Part-of-Speech tags (NN, NNP, NNS, VB variants, JJ, IN being most frequent) confirmed that the email text follows typical English grammatical structures.

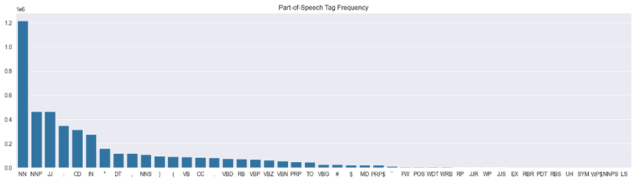

Fig. 6. Bar Chart for POS Tag Frequency

Observations:

- Nouns are highly frequent: The tags "NN" (singular nouns) and "NNP" (proper nouns) have the tallest bars,indicating they are the most frequent part-of-speech tags in this text.
- Adjectives and Prepositions/Conjunctions are also common: "JJ" (adjectives) and "IN" (prepositions/subordinating conjunctions) show relatively high frequencies as well.
- Verbs have varied frequencies: Different verb forms ("VB", "VBP", "VBZ","VBN", "VBG") show varying levels of frequency, with the base form ("VB") and non-3rd person singular present ("VBP") being more common than others.
- Determiners are significant: "DT" (determiners like "the", "a", "an") also appear frequently.
- Other parts of speech are less frequent: Tags like adverbs ("RB"), pronouns ("PRP", "PRP$"), modals ("MD"), coordinating conjunctions ("CC"), and various wh-words ("WDT", "WRB", "WP", "WP$") have noticeably lower frequencies compared to nouns and adjectives.
- Symbols and foreign words are rare: Tags like "$" (dollar sign), "FW" (foreign word), and other symbols ("SYM") have very short bars, indicating they are not common in this text.

F. TF-IDF Visualization (Top 20)

This analysis highlighted the importance of HTML tags and attributes (image, font, table, width, align, nbsp, size, color, etc.). These terms have high TF-IDF scores, indicating they are highly characteristic and useful for distinguishing between different emails, likely separating HTML-formatted emails from plain text ones.

TF-IDF combines two parts: Term Frequency (TF) and Inverse Document Frequency (IDF).

Term Frequency (TF): Measures how often a word appears in a document. A higher frequency suggests greater importance. If a term appears frequently in a document, it is likely relevant to the document's content. Formula:

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

where $f_{t,d}$ is the raw count of a term t in a document, i.e., the number of times that term t occurs in document d. Note the denominator is simply the total number of terms in document d (counting each occurrence of the same term separately). There are various other ways to define term frequency:

- The raw count itself: $\mathrm{tf}(t,d) = f_{t,d}$
- Boolean "frequencies": $\mathrm{tf}(t,d) = 1$ if t occurs in d and 0 otherwise;
- Logarithmically scaled frequency: $\mathrm{tf}(t,d) = \log(1+f_{t,d})$;
- Augmented frequency, to prevent a bias towards longer documents, e.g., raw frequency divided by the raw frequency of the most frequently occurring term in the document:

$$\mathrm{tf}(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

The inverse document frequency (IDF) is a measure of how much information the word provides, i.e., how common or rare it is across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient):

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d : d \in D \text{ and } t \in d\}|}$$

with

- N: Total number of documents in the corpus $N = |D|$
- $|\{d \in D : t \in d\}|$: Number of documents where the term t appears (e.g., $\mathrm{tf}(t,d) \neq 0$). If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust the numerator to $1+N$ and denominator to $1 + |\{d \in D : t \in d\}|$.

Term frequency–inverse document frequency (TF-IDF):

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D)$$

The TF-IDF scores highlight words that are important for distinguishing between different emails in your dataset. While widespread words like "the" have high term frequency, their low inverse document frequency reduces their TF-IDF score because they appear in almost all emails. Words with higher TF-IDF scores, like "number' and "url" are likely more specific to certain subsets of your email data and could be valuable features for a classification task like spam detection. The presence of "you" also suggests a potential distinction based on the direct address to the recipient. This visualization helps identify terms that are characteristic of individual emails within the collection.

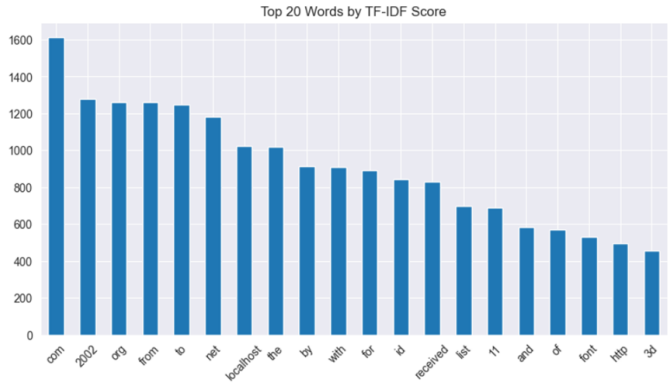

Fig. 7. Bar Chart for Top 20 Words by TF-IDF Score

Observations:

- "com" has the highest TF-IDF score: The word "com" has the tallest bar, indicating it has the highest TF-IDF score among the top 20.
- "2002" and "org" have the next highest scores: These words also have relatively high TF-IDF scores.
- Words like "from", "to", "net", and "localhost" have considerable scores: They suggest some importance in differentiating documents within the corpus.
- Common English words have lower scores: Words like "the", "by", "with", "for", "and", and "of" appear in the list but have lower TF-IDF scores compared to the more specific terms. This is expected because TF-IDF downweights words that appear frequently across many documents.
- Technical terms and identifiers are present: Words like "id", "font", "http", and "3d" suggest that these terms might be important in distinguishing certain types of documents within the corpus.
- "received" and "list" also have moderate TF-IDF scores. The number "11" also appears in the list.

Meaning of TF-IDF:

- Term Frequency (TF): Measures how frequently a term appears in a document.
- Inverse Document Frequency (IDF): Measures how rare a term is across the entire corpus of documents. Words that appear in many documents have a lower IDF.

A high TF-IDF score for a word means that the word is frequent in a particular document but rare in the overall corpus. Therefore, these words are considered more important for distinguishing documents.

In summary, this bar chart highlights the words that are most discriminative across the analyzed text documents based on their TF-IDF scores. The words with higher scores, such as "com", "2002", "org", and "localhost", are likely good indicators of the specific content or categories of the documents in which they appear. The presence of common English words with lower scores confirms the TF-IDF mechanism of prioritizing distinguishing terms. This analysis is often used in information retrieval and text mining to identify the most relevant terms in a collection of documents.

IV. Preprocessing the Data

A. Preprocessing Methodology

Preprocessing plays a pivotal role in natural language processing (NLP) pipelines, particularly in text classification tasks such as spam email detection. The objective is to transform raw and heterogeneous textual inputs into a structured, normalized, and semantically meaningful format that facilitates effective feature extraction and machine learning.

The implemented preprocessing pipeline in this study encompasses the following systematic procedures:

Duplicate and Null Removal: All duplicate entries and rows with missing email content are removed to ensure dataset integrity and eliminate redundant patterns that could skew model training.

Placeholder-Based Pattern Normalization: Regular expressions are employed to detect and replace specific patterns in the email content with standardized placeholders:

- HTML Tags: Replaced with the placeholder HTML to remove presentation-level markup irrelevant to semantic content.
- URLs: Substituted with URL to generalize link structures that are common in spam but non-specific in nature.
- Email Addresses: Masked using EMAIL_ADDRESS to prevent the model from learning user- or domain-specific identifiers.
- Numeric Values: Abstracted to NUMBER to normalize quantitative expressions across emails.
- Special Characters: Replaced with SPECIAL_CHARACTER to reduce lexical variance caused by symbols, punctuation, or encoded text.

Text Normalization and Linguistic Filtering: Using the SpaCy NLP library, the text is tokenized and linguistically normalized:

- Stopword Removal: Common functional words (e.g., "is", "the", "and") are eliminated as they carry minimal discriminative power.
- Lemmatization: Words are reduced to their base or dictionary form (e.g., "running" $\rightarrow$ "run") to consolidate variations of the same term.
- Token Categorization: Non-alphabetic tokens are conditionally replaced based on characteristics (e.g., numeric, symbolic), improving generalizability.
- Retention of Semantic Placeholders: Defined placeholders are retained throughout the pipeline to preserve informative markers (e.g., URL, NUMBER) critical in spam identification.

Deduplication of Tokens within Each Email: To further reduce token redundancy, duplicate words within individual emails are removed while preserving their first occurrence order. This reduces overemphasis on repeated terms often used in spam to attract attention (e.g., "FREE FREE FREE").

B. Justification and Rationale

The preprocessing strategy is grounded in established NLP practices that aim to:

- Minimize Noise: By filtering out irrelevant structures and symbols, the input data is denoised, enhancing signal clarity for classification.
- Standardize Lexical Patterns: Using placeholders ensures consistent representation of frequently variable components (e.g., URLs, email addresses), improving model robustness across domains.
- Reduce Dimensionality: Stopword removal, lemmatization, and token deduplication contribute to a more compact feature space, decreasing computational overhead and mitigating overfitting.
- Preserve Discriminative Features: Semantic placeholders and lemmatized content ensure that key spam indicators remain detectable by the model.

C. Expected Impact

The outcome of the preprocessing pipeline is a refined dataset composed of semantically informative, structurally consistent email content. This transformation is essential for:

- Enhancing feature extraction techniques such as TF-IDF vectorization.
- Improving the performance, generalization, and interpretability of downstream machine learning models.
- Enabling more accurate and efficient spam classification, especially when handling diverse and noisy real-world email data.

D. Feature Representation using TF-IDF Vectorization

Feature extraction is a crucial step in transforming preprocessed text into a numerical format suitable for machine learning algorithms. In this study, we apply Term Frequency-Inverse Document Frequency (TF-IDF) vectorization to encode textual information into structured numerical vectors.

E. Vectorization Configuration

The TF-IDF vectorizer was configured with the following parameters to optimize the feature space for the classification task:

- lowercase=False: Preserves the casing of tokens, enabling differentiation between certain placeholder tokens (e.g., EMAIL_ADDRESS, HTML) and regular terms.
- min_df=80: Filters out tokens that appear in fewer than 80 documents. This threshold eliminates infrequent noise terms while retaining commonly occurring features.
- max_features=1000: Restricts the final feature set to the top 1,000 most informative terms (by TF-IDF ranking), thereby reducing dimensionality and computational complexity.

An optional vocabulary parameter (commented in the code) can be used to explicitly preserve semantically relevant placeholders introduced during preprocessing.

Transformation Process

The vectorizer is trained (fit) on the preprocessed email corpus and applied (transform) to convert each email into a sparse numerical vector. These vectors are then converted to a dense array and formatted into a structured DataFrame (vector_df) where:

- Rows represent individual emails.
- Columns correspond to TF-IDF-weighted tokens.

This matrix serves as the input feature set for the downstream machine learning classifiers.

V. Models in use

The project employs six different models for email classification, including five traditional machine learning models and one deep learning model. Below is a detailed explanation of each model, its algorithm, implementation, and results.

A. Random Forest Model

Random Forest is a popular machine learning model in the Ensemble Learning family, used for both classification and regression tasks. It consists of a collection of decision trees trained on different subsets of data, with the final prediction aggregated from these trees. The core idea is to combine multiple weak learners (individual decision trees) into a strong learner with higher accuracy and reduced overfitting compared to a single decision tree.

Random Forest relies on two key principles:

- **Bagging (Bootstrap Aggregating)**: Creates multiple data subsets by randomly sampling with replacement from the original dataset.
- **Feature Randomness**: At each split in a decision tree, only a random subset of features is considered.

1) Basic Components:

Decision Tree: The fundamental unit of Random Forest. Each tree is built independently on a data subset.

Forest: A collection of many decision trees (typically hundreds or thousands).

Bootstrap Sample: Each tree is trained on a randomly sampled subset of the original dataset.

Random Feature Subset: At each node, only a random subset of features is evaluated for the best split.

2) Detailed Algorithm:

Step 1: Data Preparation

- Assume an original dataset D with N samples and M features.

Step 2: Create Bootstrap Samples

- Generate T subsets $D1, D2, \ldots, DT$ (where T is the number of trees), each of size N, by randomly sampling with replacement from D.
- Due to sampling with replacement, some samples may appear multiple times, while others may not appear (approximately 36.8% of the data is left out, called Out-of-Bag (OOB) data).

Step 3: Build Each Decision Tree

- For each subset D_t:
  a) Initialize a decision tree: Trees are grown fully without pruning.
  b) At each node:
     – Randomly select a subset of m features from the total M features ($m < M$). Typically:
       * For classification: $m = \sqrt{M}$
       * For regression: $m = M/3$
     – Find the best feature among these m features to split the node, based on criteria like Gini Index, Entropy (classification), or Mean Squared Error (regression).
  c) Repeat the splitting process until the tree is complete (reaches maximum depth or cannot split further).
- Result: T independent decision trees.

Step 4: Aggregate Results

For classification:

- Each tree predicts a class.
- The final class is the one with the most votes (Majority Voting).

For regression:

- Each tree predicts a numerical value.
- The final value is the average of all predictions.

Step 5: Model Evaluation (Using OOB Error)

- OOB data (not used to train a given tree) is used to test the performance of each tree.
- Aggregate OOB results to estimate the overall accuracy of the Random Forest without a separate test set.

3) Mathematical Formulas:

There are splitting criteria at each node in the tree, and it is different for each task.

For Classification:

- Gini Index:

$$Gini = 1 - \sum_{i=1}^{C} p_i^2$$

where p_i is the proportion of class i in the node.

- Entropy:

$$Entropy = -\sum_{i=1}^{C} p_i \log_2 p_i$$

- Final Result:

$$\hat{y} = mode(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_T)$$

where $\hat{y}_t$ is the prediction from a tree t.

For Regression:

- Splitting criterion: Minimize Mean Squared Error:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- Final Result:

$$\hat{y} = \frac{1}{T} \sum_{t=1}^{T} \hat{y}_t$$

4) Advantages:

- High accuracy due to combining multiple trees.
- Resistance to overfitting thanks to randomness and aggregation.
- Handles large datasets with many features and samples effectively.
- No need for data normalization since it is based on decision trees.
- Provides feature importance measurement based on impurity reduction.

5) Disadvantages:

- Resource-intensive: Requires significant memory and computation for large numbers of trees.
- Less interpretable than a single decision tree.
- Reduced performance with linear data, where linear regression might be more suitable.

B. Support Vector Machine (SVM)

SVM is a supervised machine learning algorithm used for classification and regression, though it is primarily applied to classification tasks. The core idea of SVM is to find the optimal hyperplane that best separates data points of different classes in a high-dimensional space, maximizing the margin between the classes. For cases where data is not linearly separable, SVM uses the kernel trick to transform the data into a higher-dimensional space where a linear boundary can be established.

SVM is known for its robustness and effectiveness in handling high-dimensional datasets and is widely used in tasks like text classification, image recognition, and bioinformatics.

1) Basic Components:

Hyperplane: A decision boundary that separates data points of different classes. In d-dimensional space, a hyperplane is defined by $w^t + b = 0$, where w is the weight vector, b is the bias, and x is the input vector.

Margin: The distance between the hyperplane and the nearest data point from either class. SVM aims to maximize this margin.

Support Vectors: The data points closest to the hyperplane, which define the margin and are critical to determining the hyperplane's position.

Kernel Function: A function that transforms non-linearly separable data into a higher-dimensional space where a linear boundary can be found (e.g., linear, polynomial, or radial basis function (RBF) kernels).

Regularization Parameter (C): Controls the trade-off between maximizing the margin and minimizing classification errors.

Slack Variables: Allow for some misclassification in soft-margin SVM to handle non-linearly separable data.

2) Detailed Algorithm:

Step 1: Data Preparation

- Assume an original dataset

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

where $x_i \in \mathbb{R}^d$ (feature vector with d dimensions) and $y_i \in \{-1, +1\}$ (binary class labels).

Step 2: Define the Objective

Hard-Margin SVM (Linearly Separable Data):

- Find the hyperplane $w^T x + b = 0$ that separates the classes with the maximum margin.
- The margin is defined as the distance from the hyperplane to the nearest data point, given by $\frac{2}{\|w\|}$, where $\|w\| = \sqrt{w^T w}$.
- Objective: Maximize the margin, i.e., minimize $\frac{1}{2}\|w\|^2$, subject to:

$$y_i(w^T x_i + b) \geq 1, \quad \forall i = 1, \ldots, N$$

Soft-Margin SVM (Non-Linearly Separable Data):

- Introduce slack variables $\xi_i \geq 0$ to allow some misclassification.
- Objective: Minimize:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i$$

Subject to:

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i$$

where C is the regularization parameter controlling the trade-off between margin maximization and classification error.

Step 3: Solve the Optimization Problem

The optimization problem is typically solved in its **dual form** using Lagrange multipliers to handle constraints efficiently.

- Dual Problem:
  - Introduce Lagrange multipliers $\alpha_i \geq 0$.
  - The dual optimization problem is:

Maximize

$$L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i^T x_j)$$

Subject to:

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad \forall i$$

- The solution α_i determines the weight vector:

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i$$

- The bias b is computed using support vectors (where $\alpha_i > 0$).
- Support vectors are the points where $y_i(w^T x_i + b) = 1$ (on the margin) or $\xi_i > 0$ (misclassified or within the margin).

Step 4: Handle Non-Linear Data with Kernels For non-linearly separable data, map the input data to a higher-dimensional space using a kernel function $K(x_i, x_j)$.

- Common kernels:
  - Linear Kernel:

$$K(x_i, x_j) = x_i^T x_j$$

  - Polynomial Kernel:

$$K(x_i, x_j) = (x_i^T x_j + c)^d$$

  - Radial Basis Function (RBF) Kernel:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$

- The dual problem becomes: Maximize

$$L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

Subject to the same constraints as above.

Step 5: Prediction

For a new input x:

- Compute the decision function:

$$f(x) = \sum_{i \in \text{SV}} \alpha_i y_i K(x_i, x) + b$$

where SV is the set of support vectors.

- Predict the class:

$$\hat{y} = sign(f(x))$$

- For probability estimates (if enabled), use techniques like Platt scaling to convert f(x) into probabilities

3) Advantages:

- Effective in High-Dimensional Spaces: Works well with datasets having many features (e.g., text classification).
- Robust to Outliers: Maximizing the margin focuses on support vectors, ignoring points far from the boundary.
- Flexible with Kernels: The kernel trick allows SVM to handle non-linearly separable data effectively.
- Global Optimization: The convex optimization problem ensures a unique solution.
- Sparse Solution: Only support vectors (a subset of the data) determine the model, making it memory-efficient.

4) Disadvantages:

- Computationally Expensive: Training time scales poorly with large datasets ($O(N^2) to O(N^3)$ for solving the quadratic optimization problem).
- Sensitive to Parameter Tuning: Requires careful tuning of C and kernel parameters (e.g., γ for RBF kernel).
- Not Interpretable: The resulting hyperplane and support vectors are not as intuitive as decision trees.

- Poor with Noisy Data: Overlapping classes or noisy data can degrade performance.
- Not Ideal for Large Datasets: Due to high computational cost, SVM is less suitable for massive datasets compared to models like Random Fores

C. K-Nearest Neighbors (KNN)

KNN is a simple, non-parametric, and instance-based supervised machine learning algorithm used for both classification and regression. It works by finding the K closest data points (neighbors) to a new input in the feature space and making a prediction based on their labels (for classification) or values (for regression). KNN is often described as a "lazy learning" algorithm because it does not build an explicit model during training; instead, it stores the training data and performs calculations at prediction time.

The core idea is: "A point is likely to belong to the same class (or have a similar value) as its nearest neighbors."

1) Basic Components:

Training Data: The entire dataset

$$D = (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$

where $x_i \in \mathbb{R}^d$ is a feature vector with d dimensions, and y_i is the label (for classification, e.g., $y_i \in \{0, 1\}$) or value (for regression, e.g., $y_i \in \mathbb{R}$)

Distance Metric: A function to measure the "closeness" between points, typically Euclidean distance.

2) Detailed Algorithm:

Here are the steps to implement and use the KNN algorithm:

Step 1: Data Preparation

- Prepare the dataset

  $$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

  where x_i are feature vectors and y_i are labels (classification) or continuous values (regression).
- Normalize or standardize features, as KNN relies on distance calculations, and differing feature scales can skew results.

Step 2: Choose Parameters

- Select K, the number of neighbors to consider.
- Choose a **distance metric** (e.g., Euclidean, Manhattan).
- **For classification**, decide on a voting method (e.g., majority voting or weighted voting).
- **For regression**, decide on an aggregation method (e.g., mean or weighted mean).

Step 3: Prediction

- For a new input x:
  a) Compute the distance between x and all points x_i in the training set using the chosen distance metric.
  b) Identify the K nearest neighbors (the K points with the smallest distances).
  c) **Classification**: Predict the class by majority voting among the K neighbors' labels.
  d) **Regression**: Predict the value by averaging the K neighbors' values (or using a weighted average).
- Weighted KNN: Assign weights to neighbors based on their distance (e.g., inverse distance), giving closer neighbors more influence.

3) Mathematical Formulas:

KNN's mathematics is centered around **distance calculations** and **aggregation** of neighbors' outputs. Below are the key formulas:

- Euclidean Distance (L2 Norm):

  $$d(x, x_i) = \sqrt{\sum_{j=1}^{d} (x_j - x_{i,j})^2}$$

  where x_j and $x_{i,j}$ are the j-th features of points x and x_i, and d is the number of features.
- Manhattan Distance (L1 Norm):

  $$d(x, x_i) = \sum_{j=1}^{d} |x_j - x_{i,j}|$$

- Minkowski Distance (Generalization):

  $$d(x, x_i) = \left(\sum_{j=1}^{d} |x_j - x_{i,j}|^p \right)^{1/p}$$

  – $p = 2$: Euclidean distance.
  – $p = 1$: Manhattan distance.
- Cosine Similarity (for high-dimensional data):

  $$d(x, x_i) = 1 - \text{cosine similarity} = 1 - \frac{x \cdot x_i}{\|x\| \|x_i\|}$$

- Weighted Distance (if features have different importance):

  $$d(x, x_i) = \sqrt{\sum_{j=1}^{d} w_j (x_j - x_{i,j})^2}$$

  where w_j is the weight for feature j.

Classification Prediction

- Majority Voting:

  $$\hat{y} = \text{mode}(y_{i1}, y_{i2}, \ldots, y_{iK})$$

  where $y_{i1}, \ldots, y_{iK}$ are the labels of the K nearest neighbors.
- Weighted Voting:

  $$\hat{y} = \arg\max_c \sum_{i \in \text{KNN}} w_i I(y_i = c)$$

  where:
  – $w_i = \frac{1}{d(x, x_i)}$ (inverse distance) or another weighting function.
  – $I(y_i = c) = 1$ if $y_i = c$, else 0.
  – c is a class label.

Regression Prediction

- Mean:

$$\hat{y} = \frac{1}{K} \sum_{i \in \text{KNN}} y_i$$

- Weighted Mean:

$$\hat{y} = \frac{\sum_{i \in \text{KNN}} w_i y_i}{\sum_{i \in \text{KNN}} w_i}$$

where $w_i = \frac{1}{d(x, x_i)}$ or similar.

Error Metrics

- Classification Error:

$$\text{Error} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} I(\hat{y}_i \neq y_i)$$

- Regression Error (Mean Squared Error):

$$\text{MSE} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} (\hat{y}_i - y_i)^2$$

4) Advantages:

- Simple and Intuitive: Easy to understand and implement.
- No Training Phase: No model is built, making it flexible for dynamic datasets.
- Non-Parametric: Makes no assumptions about data distribution, effective for non-linear patterns.
- Versatile: Works for both classification and regression.
- Robust to Multimodal Data: Can handle complex decision boundaries.

5) Disadvantages:

- Computationally Expensive at Prediction Time: Requires calculating distances to all training points ($O(Nd)$ per prediction, where N is the number of samples, d is the number of features).
- Memory-Intensive: Stores the entire training dataset.
- Sensitive to Noise and Outliers: Noisy points can skew predictions.
- Curse of Dimensionality: Performance degrades in high-dimensional spaces due to sparse data.
- Requires Feature Scaling: Distances are sensitive to feature scales.

D. Naive Bayes

Naive Bayes is a family of simple, probabilistic, supervised machine learning algorithms used primarily for classification tasks, though it can be adapted for other purposes. It is based on Bayes' Theorem and assumes that features are conditionally independent given the class label (the "naive" assumption). Despite this simplifying assumption, Naive Bayes performs surprisingly well in many real-world applications, especially in text classification and spam filtering.

The core idea is: Given a set of features, predict the most likely class by calculating probabilities based on prior observations.

1) Basic Components:

- **Training Data**:

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

where $x_i = (x_{i1}, x_{i2}, \ldots, x_{id}) \in \mathbb{R}^d$ is a feature vector with d dimensions, and $y_i \in \{C_1, C_2, \ldots, C_k\}$ is a class label from k classes.
- **Bayes' Theorem**: The foundation for calculating probabilities of classes given features.
- **Conditional Independence Assumption**: Assumes that features $x_{i1}, x_{i2}, \ldots, x_{id}$ are independent given the class y.
- **Prior Probability**: The probability of each class before observing the features.
- **Likelihood**: The probability of observing the features given a class.
- **Posterior Probability**: The probability of a class given the observed features, which is used for prediction.
- **Smoothing**: A technique (e.g., Laplace smoothing) to handle zero probabilities for unseen feature-class combinations.

2) Detailed Algorithm: Here are the steps to implement and use the Naive Bayes algorithm:

Step 1: Data Preparation

- Prepare the dataset D, where each sample has features x_i and a class label y_i.
- For categorical features, Naive Bayes is straightforward. For continuous features, assume a distribution (e.g., Gaussian) or discretize the data.

Step 2: Training (Model Building)

- Estimate Prior Probabilities: Calculate the probability of each class C_j:

$$P(C_j) = \frac{\text{Number of samples with class } C_j}{\text{Total number of samples}}$$

- Estimate Likelihoods: For each feature x_m and class C_j, compute the conditional probability $P(x_m|C_j)$:
  - Categorical Features: Use frequency counts.
  - Continuous Features: Assume a distribution (e.g., Gaussian) and estimate parameters (mean, variance).
  - Apply smoothing (e.g., Laplace smoothing) to avoid zero probabilities.
- Store these probabilities for use during prediction.

Step 3: Prediction

- For a new input $x = (x_1, x_2, \ldots, x_d)$:
  a) Compute the posterior probability for each class C_j using Bayes' Theorem:

$$P(C_j|x) \propto P(C_j) \prod_{m=1}^{d} P(x_m|C_j)$$

(The proportionality $\propto$ is used because the denominator P(x) is constant across classes.)

b) Predict the class with the highest posterior probability:

$$\hat{y} = \arg\max_{C_j} P(C_j) \prod_{m=1}^{d} P(x_m|C_j)$$

c) Optionally, compute normalized probabilities by dividing by the sum of all class posteriors

3) Mathematical Formulas:

Naive Bayes is grounded in Bayes' Theorem and the conditional independence assumption. Below are the key formulas, explained intuitively.

Bayes' Theorem

For a class C_j and feature vector $x = (x_1, x_2, \ldots, x_d)$:

$$P(C_j|x) = \frac{P(C_j)P(x|C_j)}{P(x)}$$

- $P(C_j|x)$: Posterior probability (probability of class C_j given features x).
- $P(C_j)$: Prior probability (probability of class C_j before seeing x).
- $P(x|C_j)$: Likelihood (probability of observing x given class C_j).
- $P(x)$: Evidence (probability of observing x, a normalizing constant).

Since $P(x)$ is the same for all classes, we can ignore it for classification:

$$P(C_j|x) \propto P(C_j)P(x|C_j)$$

Conditional Independence Assumption

Naive Bayes assumes that features are independent given the class:

$$P(x|C_j) = P(x_1, x_2, \ldots, x_d|C_j) = \prod_{m=1}^{d} P(x_m|C_j)$$

Intuition: If you know the class (e.g., "spam email"), the presence of one feature (e.g., the word "free") doesn't affect the probability of another feature (e.g., the word "win"). This is often unrealistic but simplifies calculations.

Prior Probability

$$P(C_j) = \frac{\text{Count}(y = C_j)}{N}$$

where $\text{Count}(y = C_j)$ is the number of samples with class C_j, and N is the total number of samples.

Likelihood

- Categorical Features:

$$P(x_m = v|C_j) = \frac{\text{Count}(x_m = v, y = C_j)}{\text{Count}(y = C_j)}$$

where v is a specific value of feature x_m.

- Laplace Smoothing: To avoid zero probabilities:

$$P(x_m = v|C_j) = \frac{\text{Count}(x_m = v, y = C_j) + \alpha}{\text{Count}(y = C_j) + \alpha \cdot |\text{Values}(x_m)|}$$

where α (e.g., 1) is the smoothing parameter, and $|\text{Values}(x_m)|$ is the number of possible values for x_m.

- Continuous Features (Gaussian Naive Bayes): Assume feature x_m follows a Gaussian distribution for class C_j:

$$P(x_m|C_j) = \frac{1}{\sqrt{2\pi\sigma_{mj}^2}} \exp\left(-\frac{(x_m - \mu_{mj})^2}{2\sigma_{mj}^2}\right)$$

where:

– μ_{mj}: Mean of feature x_m for class C_j.
– σ_{mj}^2: Variance of feature x_m for class C_j.

Error Metrics

- Classification Error:

$$\text{Error} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} I(\hat{y}_i \neq y_i)$$

- Log Loss (if probabilities are used):

$$\text{Log Loss} = -\frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \sum_{j=1}^{k} I(y_i = C_j) \log P(C_j|x_i)$$

4) Advantages:

- Simple and Fast: Easy to implement and computationally efficient, especially for training.
- Effective with Small Datasets: Performs well even with limited data, unlike complex models.
- Handles High-Dimensional Data: Common in text classification (e.g., bag-of-words models).
- Probabilistic Output: Provides probability estimates for each class, useful for decision-making.
- Robust to Irrelevant Features: The independence assumption mitigates the impact of irrelevant features.

5) Disadvantages:

- Naive Assumption: The conditional independence assumption is often unrealistic, leading to suboptimal performance when features are correlated.
- Sensitive to Zero Probabilities: Requires smoothing to handle unseen feature-class combinations.
- Poor with Continuous Features: Gaussian assumptions may not fit all data distributions.
- Outperformed by Complex Models: Often less accurate than SVM or Random Forest on complex datasets.
- Imbalanced Data Issues: May favor majority classes without proper adjustments.

E. XGBoost

XGBoost (Extreme Gradient Boosting) is a powerful, scalable, and highly optimized machine learning algorithm used for both classification and regression tasks, though it excels in structured/tabular data. It belongs to the family of gradient boosting methods, which build an ensemble of decision trees sequentially, where each tree corrects the errors of the previous ones. XGBoost enhances gradient boosting with advanced regularization, parallel processing, and handling of missing

data, making it a go-to choice in data science competitions and real-world applications.

The core idea is: Combine many weak decision trees (weak learners) into a strong predictive model by iteratively minimizing a loss function using gradient-based optimization.

1) Basic Components:

- **Decision Trees**: The base learners in XGBoost, typically shallow trees (e.g., depth 3–10) to prevent overfitting.
- **Ensemble**: A collection of trees whose predictions are combined (summed for regression, aggregated for classification).
- **Loss Function**: Measures the difference between predicted and actual values (e.g., mean squared error for regression, log loss for classification).
- **Regularization**: Penalties on tree complexity to prevent overfitting (e.g., L1/L2 regularization on leaf weights).
- **Gradient and Hessian**: First-order (gradient) and second-order (Hessian) derivatives of the loss function guide tree construction.
- **Boosting**: Sequential addition of trees, where each tree focuses on correcting the residuals (errors) of the previous trees.
- **Hyperparameters**: Parameters like learning rate, max depth, and regularization terms control model behavior.

2) Detailed Algorithm:

Step 1: Data Preparation

- Prepare the dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, where $x_i \in \mathbb{R}^d$ is a feature vector with d dimensions, and y_i is a label (e.g., $y_i \in \{0, 1\}$ for binary classification, $y_i \in \mathbb{R}$ for regression).
- Handle missing values (XGBoost can automatically learn how to treat them).
- No feature scaling is required, as XGBoost is tree-based.

Step 2: Initialize the Model

- Start with an initial prediction (e.g., mean of target values for regression, log-odds for classification).
- Define the loss function (e.g., MSE for regression, log loss for classification) and regularization terms.

Step 3: Build Trees Sequentially

- For T iterations (number of trees):
  a) Compute **gradients** (first derivative of the loss with respect to predictions) and **Hessians** (second derivative) for each sample.
  b) Build a decision tree to fit the gradients, using a specialized objective function that balances loss reduction and tree complexity.
  c) Update the model's predictions by adding the new tree's output, scaled by a **learning rate** (η).
- Each tree focuses on the residuals (errors) of the current model.

Step 4: Prediction

- For a new input x:
  - Sum the predictions from all trees (for regression) or compute a weighted sum and apply a sigmoid function (for classification).
  - Output the final class or value.

3) Mathematical Formulas: XGBoost's mathematics combines **gradient boosting**, **decision tree construction**, and **regularized optimization**. Below are the key formulas, explained intuitively.

Objective Function

The goal is to minimize a loss function plus regularization:

$$\text{Obj} = \sum_{i=1}^{N} \ell(y_i, \hat{y}_i) + \sum_{t=1}^{T} \Omega(f_t)$$

- $\ell(y_i, \hat{y}_i)$: Loss function measuring the error between true label y_i and prediction $\hat{y}_i$.
  - Regression (MSE):
  $$\ell(y_i, \hat{y}_i) = \frac{1}{2}(y_i - \hat{y}_i)^2$$
  - Classification (Log Loss):
  $$\ell(y_i, \hat{y}_i) = -[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$
- $\Omega(f_t)$: Regularization term for tree t:
$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + \alpha \sum_{j=1}^{T} |w_j|$$
  - T: Number of leaves in the tree.
  - w_j: Leaf weight (output value of leaf j).
  - γ: Penalty for adding leaves (controls tree size).
  - λ: L2 regularization on leaf weights.
  - α: L1 regularization on leaf weights.

Gradient Boosting

Predictions are the sum of outputs from T trees:

$$\hat{y}_i = \sum_{t=1}^{T} f_t(x_i)$$

where $f_t(x_i)$ is the output of tree t for sample x_i.

At iteration t, the prediction is:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta f_t(x_i)$$

- η: Learning rate (shrinks the contribution of each tree to prevent overfitting).

Gradient and Hessian

For each sample i, compute:

- Gradient (first derivative): $g_i = \frac{\partial \ell(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$
  - For MSE: $g_i = \hat{y}_i^{(t-1)} - y_i$.
  - For log loss: $g_i = \hat{y}_i^{(t-1)} - y_i$, where $\hat{y}_i^{(t-1)}$ is the predicted probability.
- Hessian (second derivative): $h_i = \frac{\partial^2 \ell(y_i, \hat{y}_i^{(t-1)})}{\partial (\hat{y}_i^{(t-1)})^2}$
  - For MSE: $h_i = 1$.
  - For log loss: $h_i = \hat{y}_i^{(t-1)}(1 - \hat{y}_i^{(t-1)})$.

These guide the tree to focus on samples with larger errors.

Tree Construction

Each tree is built to minimize:

$$\text{Obj}^{(t)} = \sum_{i=1}^{N} [g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2] + \Omega(f_t)$$

Intuition: The tree predicts values that reduce the loss (via gradients) while keeping the tree simple (via regularization).

- For a leaf j with samples I_j, the optimal leaf weight is:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

- The gain from splitting a node into left (I_L) and right (I_R) branches is:

$$\text{Gain} = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma$$

  - Split if Gain > 0, choosing the feature and threshold that maximizes Gain.

Prediction

- Regression:

$$\hat{y}_i = \sum_{t=1}^{T} f_t(x_i)$$

- Classification (Binary):

$$\hat{y}_i = \sigma \left(\sum_{t=1}^{T} f_t(x_i) \right)$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function for probability output.

4) Advantages:

- High Accuracy: Often outperforms other models on structured data due to sequential error correction.
- High Accuracy: Often outperforms other models on structured data due to sequential error correction.
- Handles Missing Data: Automatically learns how to treat missing values.
- Regularization: Prevents overfitting with L1/L2 penalties and tree pruning.
- Scalable: Optimized for speed with parallel processing and efficient tree construction.
- Feature Importance: Provides insights into which features drive predictions.

5) Disadvantages:

- Computationally Intensive: Training can be slow for large datasets with many trees.
- Complex Tuning: Requires careful tuning of hyperparameters (e.g., learning rate, max depth).
- Less Interpretable: Ensemble of trees is harder to interpret than a single tree.
- Poor with Sparse Data: Less effective for high-dimensional, sparse data (e.g., text) compared to Naive Bayes.
- Overfitting Risk: Without proper regularization, can overfit noisy data.

VI. Evaluations

A. On the Training Dataset

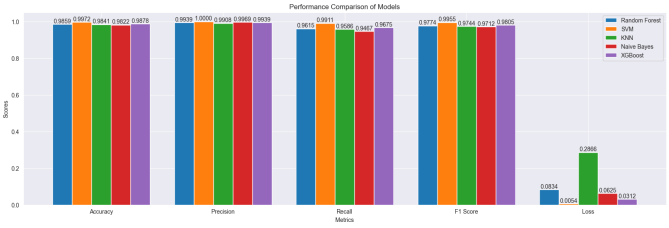

Fig. 8. Comparison of Models' Performance on Training Dataset

- **Accuracy:**
  - SVM exhibits the highest accuracy (0.9972).
  - KNN (0.9841), Random Forest (0.9859), Naive Bayes (0.9822), and XGBoost (0.9878) show very competitive and high accuracy scores, all above 0.98.
- **Precision:**
  - SVM achieves perfect precision (1.0000).
  - Random Forest (0.9939), KNN (0.9908), Naive Bayes (0.9969), and XGBoost (0.9939) also demonstrate exceptionally high precision, all above 0.99.
- **Recall:**
  - SVM shows the highest recall (0.9911).
  - XGBoost (0.9675) and Random Forest (0.9615) follow with high recall scores.
  - KNN (0.9586) and Naive Bayes (0.9467) have slightly lower recall compared to the others, but still maintain strong performance.
- **F1 Score:**
  - SVM leads with the highest F1 Score (0.9955).
  - XGBoost (0.9805) and Random Forest (0.9774) also have excellent F1 Scores, indicating a good balance between precision and recall.
  - KNN (0.9744) and Naive Bayes (0.9712) show robust F1 Scores as well.
- **Loss:**
  - SVM has the lowest loss value (0.0054), indicating the best model fit according to this metric.
  - XGBoost (0.0312) and Random Forest (0.0834) also have relatively low loss values.
  - Naive Bayes (0.0625) has a moderate loss.
  - KNN exhibits the highest loss among the models (0.2866).
- **Overall Observations from the Chart:**
  - Across the metrics of Accuracy, Precision, Recall, and F1 Score, SVM consistently performs as the top model or among the top performers.
  - All models demonstrate very high performance (scores generally > 0.94) for Accuracy, Precision, Recall, and F1 Score, suggesting the effectiveness of the underlying task.

– In terms of Loss (where lower is better), SVM is distinctly superior, followed by XGBoost. KNN shows a significantly higher loss compared to the other models.

TABLE I
MODEL PERFORMANCE COMPARISON (BASED ON BAR CHART)

Model	Key Analysis
Random Forest	High overall performance. Excellent precision, strong recall & F1. Moderate loss.
SVM	Top performer across all metrics. Perfect precision, highest Acc, Recall, F1. Lowest loss.
KNN	Strong performance, slightly below RF / XGBoost. Highest loss among models.
Naive Bayes	Very high precision. Recall is the lowest among models, impacting F1 slightly. Low loss.
XGBoost	Excellent performance, second to SVM overall. Very low loss. Strong recall & F1.

B. On the Validating Dataset

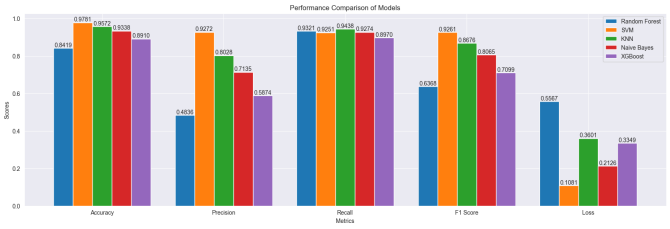

Fig. 9.

- **General Trend Compared to Previous (Assumed Training) Data:**
  - A significant drop in performance is observed across all models and most metrics (Accuracy, Precision, F1 Score) when comparing to the previously analyzed dataset (which had near-perfect scores). Recall scores are somewhat more varied, with some models maintaining high recall. Loss values have generally increased.
  - This substantial decrease strongly suggests that the models may have overfit to the previous dataset, or that this new dataset presents a more challenging distribution, is inherently more complex, or has a different class balance. The previous high scores might not have been indicative of true generalization capability.
- **Accuracy:**
  - SVM exhibits the highest accuracy (0.9781) on this dataset, maintaining strong performance, though lower than its previously observed near-perfect score.
  - KNN follows with a good accuracy of 0.9572.
  - Naive Bayes (0.9338), XGBoost (0.8910), and Random Forest (0.8419) show a more considerable drop in accuracy. Random Forest's accuracy is now the lowest.
- **Precision:**
  - SVM leads with the highest precision (0.9272), a significant decrease from its previous perfect score but still the best.
  - KNN (0.8028) and Naive Bayes (0.7135) maintain moderate precision.
  - Random Forest (0.4836) and XGBoost (0.5874) show a very sharp decline in precision, indicating a large increase in false positives on this new dataset. This is a major change from their previously high precision.
- **Recall:**
  - KNN achieves the highest recall (0.9438) on this dataset.
  - Random Forest (0.9321), SVM (0.9251), and Naive Bayes (0.9274) also maintain high recall scores, suggesting they are still effective at identifying positive instances.
  - XGBoost's recall (0.8970) is also high, though slightly lower than the others in this group.
  - The high recall for models like RF and KNN, despite lower precision, suggests they are casting a wide net, correctly identifying positives but also incorrectly flagging many negatives.
- **F1 Score:**
  - SVM has the highest F1 Score (0.9261), indicating the best balance between its high precision and high recall on this dataset.
  - KNN (0.8676) also shows a good F1 Score.
  - Naive Bayes (0.8065), XGBoost (0.7099), and Random Forest (0.6368) have lower F1 Scores, primarily impacted by their significant drop in precision (for RF and XGBoost) or a less optimal balance.
- **Loss:**
  - SVM maintains the lowest loss (0.1081), consistent with its strong performance across other metrics, though this loss is much higher than its previous near-zero loss.
  - Naive Bayes (0.2126) has the next lowest loss.
  - XGBoost (0.3349) and KNN (0.3601) have moderate loss values.
  - Random Forest exhibits the highest loss (0.5567) on this dataset, correlating with its lower precision and accuracy.

- **Overall Observations and Key Changes from Previous Dataset:**
  - SVM demonstrates the most robust generalization, leading in Accuracy, Precision, F1 Score, and having the lowest Loss on this new, more challenging dataset. While its scores have decreased from the (assumed) training set, the drop is less severe compared to some other models.
  - Random Forest and XGBoost show a dramatic decrease in Precision, leading to significantly lower F1 Scores. This suggests they might be particularly sensitive to changes in data distribution or may have learned overly specific patterns from the previous dataset. Their high recall now comes at the cost of many false positives.
  - KNN, while having the highest loss previously, now shows a more competitive F1 score due to maintaining high recall and moderate precision. Its loss is still relatively high but not the outlier it was.
  - Naive Bayes maintains relatively stable performance, with its precision dropping but recall remaining high.
  - The major changes in performance indicate that this new dataset likely differs significantly from the one previously evaluated (assumed to be training/easier validation). This could be due to different underlying data characteristics, a shift in the data distribution (concept drift), or the presence of more noise or outliers. The previous near-perfect scores for some models were likely not representative of real-world generalization.

TABLE II
MODEL PERFORMANCE COMPARISON (BASED ON VALIDATING DATASET)

Model	**Key Analysis**
Random Forest	Lowest Acc. Very low Prec., high Recall (many FPs). Highest Loss. Significant performance drop.
SVM	Best performer overall. Highest Acc, Prec, F1. Lowest Loss. Most robust generalization.
KNN	Good Acc. Highest Recall. Moderate Prec & F1. High Loss. Performance drop but recall strong.
Naive Bayes	Moderate Acc & Prec. High Recall. Second lowest Loss. Performance drop but relatively stable.
XGBoost	Moderate Acc. Low Prec., high Recall (many FPs). Moderate Loss. Significant drop in Prec/F1.

VII. CONTRIBUTION

The following table represents the contribution of each member, note that whichever member handles whichever task will also write the report for that task.

TABLE III
MEMBERS CONTRIBUTIONS

ID	**Member**	**Contribution**	**Progress**
522H0036	Luong Canh Phong	Overseer, Implementation and Handling Report	100%
520H0341	Nguyen Thai Bao	Research and Evaluating Models	100%
522H0030	Le Tan Huy	Preprocessing the Dataset	100%
522H0008	Dao Minh Phuc	Visualizing the Dataset	100%
522H0136	Nguyen Nhat Phuong Anh	Research Business and Visualizing the Dataset	100%

VIII. CONCLUSION

This paper addressed spam email detection through data preprocessing and the application of several machine learning models. Initial NLP-driven data exploration identified key email characteristics like headers, HTML elements, and spam-related terms, alongside a right-skewed email length distribution, underscoring the need for the implemented robust preprocessing pipeline. Five models—Random Forest, SVM, KNN, Naive Bayes, and XGBoost—were evaluated.

While all models performed exceptionally well on the training data, with SVM achieving near-perfect scores, the validation dataset provided a clearer picture of generalization.

SVM consistently emerged as the most robust model, leading in accuracy (0.9781), precision (0.9272), F1-score (0.9261), and achieving the lowest loss. In contrast, other models, particularly Random Forest and XGBoost, experienced significant drops in precision and overall performance, indicating potential overfitting or sensitivity to data variations. KNN maintained high recall but at the cost of precision, while Naive Bayes showed more stable, albeit moderate, performance.

The general performance decline from training to validation highlights the critical challenge of overfitting and the necessity of evaluating models on unseen data. SVM's superior generalization capabilities make it a strong candidate for spam classification. This comparative analysis provides valuable insights for model selection in similar tasks. Future directions could involve exploring advanced feature engineering or deep learning approaches.

REFERENCES

[1] *XGBoost Documentation — xgboost 3.0.1 documentation*. (n.d.). https://xgboost.readthedocs.io/

[2] *scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation*. (n.d.). https://scikit-learn.org/

[3] TensorFlow. (n.d.). *TensorFlow*. https://www.tensorflow.org/

[4] Wikipedia contributors. (2025, April 21). *Cross-entropy*. Wikipedia. https://en.wikipedia.org/wiki/Cross_entropy#Cross-entropy_loss_function_and_logistic_regression

[5] Shannon, C. E., & Weaver, W. (1949). A Mathematical Theory of Communication. *Bell System Technical Journal*, *27*(4), 623–656. https://doi.org/10.1002/j.1538-7305.1948.tb00917.x

[6] Chen, T., & Guestrin, C. (2016). XGBoost: a Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, *1*(1), 785–794. https://doi.org/10.1145/2939672.2939785

[7] Breiman, L. (2001). *Random Forests*. https://www.stat.berkeley.edu/ breiman/randomforest2001.pdf

[8] Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/bf00994018

[9] *Pattern Recognition and Machine Learning (Information Science and Statistics): Bishop, Christopher M.: 9780387310732: Amazon.com: Books*. (n.d.). https://www.amazon.com/Pattern-Recognition-Machine-Learning-Information/dp/0387310738

[10] *Elements of Statistical Learning: data mining, inference, and prediction. 2nd Edition.* (n.d.). https://hastie.su.domains/ElemStatLearn/

[11] *Machine Learning: A Probabilistic Perspective (Adaptive Computation and Machine Learning series): Murphy, Kevin P.: 9780262018029: Amazon.com: Books*. (n.d.). https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020

[12] *Introduction to Machine Learning, fourth edition (Adaptive Computation and Machine Learning series): Alpaydin, Ethem: 9780262043793: Amazon.com: Books*. (n.d.). https://www.amazon.com/Introduction-Machine-Learning-Ethem-Alpaydin/dp/0262043793

[13] *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems): Han, Jiawei, Kamber, Micheline, Pei, Jian: 9780123814791: Amazon.com: Books*. (n.d.). https://www.amazon.com/Data-Mining-Concepts-Techniques-Management/dp/0123814790

[14] *David MacKay: Information Theory, Inference, and Learning Algorithms: The book*. (n.d.). https://www.inference.org.uk/mackay/itila/book.html

[15] *Support vector Machines (Information Science and Statistics): Steinwart, Ingo, Christmann, Andreas: 9780387772417: Amazon.com: Books*. (n.d.). https://www.amazon.com/Support-Machines-Information-Science-Statistics/dp/0387772413

[16] Godoy, D. (2025, March 7). *Understanding binary cross-entropy / log loss: a visual explanation*. Towards Data Science. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

[17] Yıldırım, S. (2025, January 18). *K-Nearest Neighbors (KNN) – explained*. Towards Data Science. https://towardsdatascience.com/k-nearest-neighbors-knn-explained