HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

PROJECT REPORT

VIETNAMESE SMS SPAM FILTERING

Course: Introduction to Artificial Intelligence

Supervisor: Assoc. Prof. Than Quang Khoat

Authors:	Student ID:
Nguyen Duc Binh	20225475
Tran An Khanh	20225447
Nguyen Nhu Giap	20225441
Nguyen Minh Duc	20225437
Truong Tuan Vinh	20225464

Hanoi, December 2023

ABSTRACT

For decades, spam emails and SMS have accounted for a large part of online traffic, consequently posing a threat to cybersecurity. Together with the evolution of spam contents, numerous spam filtering methods with varying degrees of sophistication have come into existence. This project aims to implement and evaluate the efficiency of different spam filtering algorithms, thereby giving a comprehensive comparison between these methods.

Contents

1 Introduction

For the past few decades, as telecommunications has become more accessible to the public, spamming has also rapidly grown. Despite global efforts against spamming, it remains viable thanks to low operating costs and exploitation of data leaks. On the other hand, it is difficult to hold spammers accountable for their actions.

According to a report in 2023 by Kaspersky Lab [7], an information security firm, in 2022, nearly half of all e-mails worldwide were identified as spam. Although there has been a general decline in the share of spam e-mails, they remain a major part of Internet traffic. On a similar note, in 2022, Americans received 225 billion spam texts, which is over 2.5 times that of the previous year [5].

Spam emails and SMS, in addition to being an inconvenience, can be a threat to users. Recipients may be scammed or lured into downloading malware, which further exposes their data to the scammer and facilitates further spamming. Spamming also interferes with internet infrastructure as it makes up a large portion of online traffic. For these reasons, more sophisticated methods of spam detection are required. They also need occasional updates to keep up with the most recent spamming trends and techniques.

This paper aims to present a system for Vietnamese SMS filtering using different machine learning algorithms and vector representation methods. We first introduce a procedure of preprocessing Vietnamese texts to extract the most important features, followed by vectorization for future computation. We then experiment with different methods and parameter settings to obtain results, then compare these results together and give some speculations explaining these results. We also attempt to combine these methods together to create an overall improved classifier.

2 Methodology

Our pipeline for the spam filtering system is demonstrated in the figure below.

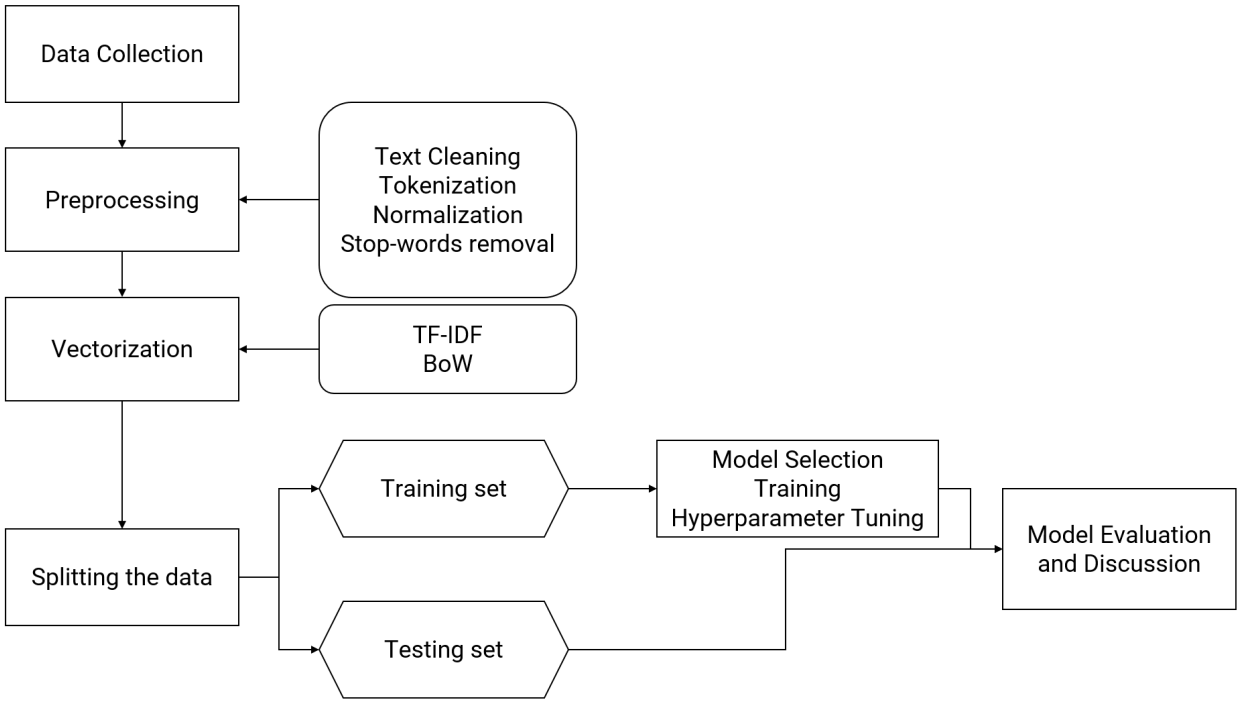

Figure 1: The pipeline of our SMS spam filtering system

We will describe in detail three main parts: Preprocessing, Vector Representation, and Classifiers.

2.1 Preprocessing

2.1.1 Tokenization

When dealing with Vietnamese SMS spam classification, tokenization, which is the process of breaking down a text or a sentence into smaller units called tokens, allows for a better understanding of the content and structure of the message, making it easier to analyze and process. Those tokens can be words, phrases, or even characters. One more problem is that in Vietnamese, the space symbol is not only used to separate words but also to separate syllables within the same word. There are public tools to solve this problem such as ViTokenizer, Underthesea Tool Kit.

2.1.2 Stop words removal

On the other hand, this technique involves eliminating frequently occurring but less informative words that do not contribute much to distinguishing between spam and legitimate messages, and it can help the algorithms focus on more meaningful content words and thus enhance the accuracy of the classification process. In Vietnamese, some words such as “và”, “anh”, “là” might be considered stop words.

2.2 Vector representation

The raw data type is text, so we need to transform it into vector form for computation in machine learning models. There are two common methods for converting raw text to numeric vector – Bag of Words (BoW) and Term frequency-Inverse document frequency (TF-IDF).

2.2.1 Bag of Words (BoW)

The Bag of Words model represents text as an unordered collection or a "bag" of words, disregarding grammar and word order. It focuses solely on the presence and frequency of words in a document. Each document is represented as a numerical vector, where each dimension corresponds to a word from the vocabulary, and the value in each dimension represents the frequency of that word in the document.

2.2.2 Term frequency-Inverse document frequency (TF-IDF)

TF-IDF aims to evaluate the importance of a word in a document relative to its frequency in the document and the entire corpus. TF measures how frequently a term appears in a document. It is calculated as the number of times a word appears in a document divided by the total number of words in that document. IDF measures the rarity or importance of a word across the entire corpus. It's calculated as the logarithm of the ratio of the total number of documents to the number of documents containing the word. The overall TF-IDF score is obtained by multiplying the TF value and the IDF value for each term in a document. TF-IDF will assign higher values to words that are frequent in a document but rare across other documents.

$$\text{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

$$\text{IDF}(t, D) = \log\left(\frac{\text{Total number of documents in the corpus } |D|}{\text{Number of documents containing term } t}\right)$$

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

2.2.3 Additional features

Based on our observations and experiments, we found that some additional features may be useful for this particular problem. Those include: the number of non-Vietnamese characters, the number of hyperlinks, phone numbers, money amount, strange capitalization recognized, the capitalization proportion of the document, and the maximum token length.

2.3 Classifiers

2.3.1 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, yet effective, supervised machine learning algorithm used for both classification and regression tasks. It's a non-parametric and instance-based learning algorithm that classifies new data points based on the majority vote or average of the K nearest neighbors' classes or values.

Algorithm Overview

Given a dataset with labeled points, KNN classifies new instances by finding the majority class among the K nearest neighbors (data points) to the new instance. For regression, it predicts the value for the new instance by averaging the values of the K nearest neighbors.

Parameters

- **K (Number of Neighbors)**: It is the most crucial parameter in KNN. K defines how many nearest neighbors will be considered to make a classification or regression prediction.
- **Distance Metric**: KNN uses distance metrics (e.g., Euclidean distance, Manhattan distance, etc.) to measure the distance between data points. The choice of distance metric can significantly impact the algorithm's performance.
  - **Euclidean Distance**: Calculates the straight-line distance between two points in the space, commonly used for continuous numerical data where the scale of features is similar.

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

  - **Manhattan Distance**: Computes the distance between two points by summing the absolute differences of their coordinates, suitable for cases where the distance along the axes (or dimensions) is more important than the direct Euclidean distance, such as in grid-based representations.

$$\sum_{i=1}^{n}|x_i - y_i|$$

  - **Minkowski Distance**: A generalized form of Euclidean and Manhattan distance, where the distance metric parameter 'p' determines the specific distance (p=1 for Manhattan, p=2 for Euclidean).

$$(\sum_{i=1}^{n}|x_i - y_i|^p)^{\frac{1}{p}}$$

  - **Cosine Similarity and Distance**: measures the cosine of the angle between two non-zero vectors in a multidimensional space, commonly used for text data, high-

dimensional data, or in cases where the magnitude of the vectors is less important than the orientation or direction

$$\frac{\sum_{i=1}^{n} x_i \times y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \times \sqrt{\sum_{i=1}^{n} y_i^2}}$$

Choosing the suitable parameters:

Larger values of K lead to smoother decision boundaries and less complex models. However, too large a value of K may lead to underfitting. On the other hand, smaller values of K can capture intricate patterns in the data, potentially leading to overfitting, especially with noisy data.

For this specific application, to pick the right parameters, We perform a grid search over a range of hyperparameters to identify the optimal values, using the GridSearchCV class in Python's scikit-learn library to automate this process. The parameter grid is:
{'n_neighbors': [3, 5, 7, 9, 11, 13], 'metric': ['euclidean', 'manhattan', 'cosine']}

After performing the search on the data scaled by maximum absolute scaling, We found {'metric': 'euclidean', 'n_neighbors': 3} to be the optimal parameters with the following results:

Testing set accuracy	98.38%
Precision	66.67%
Recall	29.63%
F1 score	41.03%

Table 1: Test Results for Hyperparameters tuning for the KNN algorithm

The advantage of KNN is that it is simple and easy to implement, with no training involved. It works well with small to moderately-sized datasets and where decision boundaries are not highly complex. However, it is sensitive to irrelevant or redundant features, computationally expensive for large datasets as it needs to compute distances for each new instance, and its performance might also degrade with high-dimensional data.

2.3.2 Naive Bayes

The Naïve Bayes Algorithm is a popular classification technique based on Bayes' Theorem. It considers the features of the input data, with the assumption that these features appear independently of each other, to predict which class an item belongs to. Despite that this assumption generally does not hold in real-life situations, the Naïve Bayes classifier is still widely used because of its high accuracy and low computational complexity.

Algorithm Overview

The likelihood of features has an impact on Naïve Bayes' probability calculation. Hence, there are three popular types of Naïve Bayes: Gaussian, Multinomial, and Bernoulli. The nature of our problem will decide which types would be best.

For our problem of filtering spam email, Multinomial Naïve Bayes is the most suitable option where the data are represented as word vector counts. This allows us to account for multiple occurrences of words in the same datum, as opposed to Bernoulli Naïve Bayes where we only have a 'bag of words' model that returns Boolean values. Let c denote the hypothesis that an email is spam, and $\overline{c}$ denotes the hypothesis that an email is not spam. Each datum X represents an email or SMS that contains words $x_1, x_2, x_3, ..., x_n$.

According to Bayes' Theorem:

$$P(c|X) = \frac{P(X|c)P(c)}{P(X)}; P(\overline{c}|X) = \frac{P(X|\overline{c})P(\overline{c})}{P(X)}$$

where $P(c|X)$ and $P(\overline{c}|X)$ are posterior probabilities, $P(X|c)$ and $P(X|\overline{c})$ are likelihoods, $P(c)$ and $P(\overline{c})$ are class prior probabilities, and $P(X)$ is predictor prior probability.

To decide whether an email/SMS is spam or non-spam, we compare $P(c|X)$ and $P(\overline{c}|X)$. To simplify, we can compare $P(X|c)P(c)$ and $P(X|\overline{c})P(\overline{c})$.
With our assumption of feature independence, we have:

$$P(X|c)P(c) = P(x_1|c) \times P(x_2|c) \times ... \times P(x_n|c) \times P(c)$$

$$P(X|\overline{c})P(\overline{c}) = P(x_1|\overline{c}) \times P(x_2|\overline{c}) \times ... \times P(x_n|\overline{c}) \times P(\overline{c})$$

$P(c)$ is the proportion of spam emails in the training dataset, and $P(\overline{c}) = 1 - P(c)$. To determine $P(x_i|c)$, we consider all the spam emails in the training dataset, count the number of occurrences of x_i and divide it by the total number of words in these emails:

$$P(x_i|c) = \frac{N_c x_i}{N_c}$$

$P(x_i|\overline{c})$ is determined in the same way.

In this process, we may face the issue that the test email contains a word x_i not seen in the training data. The probability $P(x_i|c)$ and/or $P(x_i|\overline{c})$, when computed, would equal to 0. A workaround for this problem is to introduce a nonnegative smoothing prior α such that $P(x_i|c) = \frac{N_{cx_i}+\alpha}{N_c+\alpha n}$. The optimal value of this parameter will be discussed in further detail below.

Parameters

This project uses scikit-learn 1.3.2's MultinomialNB to implement the solution mentioned above. This classifier allows us to adjust the α smoothing parameter and classes' prior probabilities.

The smoothing factor α accounts for absent features in the training dataset and prevents zero probabilities. Setting α=1 is called Laplace smoothing, while $\alpha < 1$ is called Lidstone smoothing. For MultinomialNB, α is a hyper-parameter and requires fine-tuning to search for an optimal value. After intensive testing, α=0,001 was found to bring the most positive results of all.

Adjusting classes' prior probabilities can be a useful feature when the given dataset is biased. According to Statista, spam emails currently account for about 49% of total emails sent. Our dataset for this project has a spam proportion of approximately 10%, which inaccurately reflects real-life situations. For this reason, we may adjust the prior probability of non-spam and spam classes to 51% and 49%, respectively. Alternatively, we can pass in an argument fit_prior = False to assume uniform priors (50% spam and 50% non-spam). Although this setting may reduce the recorded performance of our algorithm with the given dataset, it is expected to help improve performance in real-life circumstances.

Test results for MultinomialNB with the respective settings are as follows:

Testing set accuracy	98.52%
Precision	65.00%
Recall	48.15%
F1 score	55.32%

Table 2: Test Results for MultinomialNB with α=1, fit_prior = True

Testing set accuracy	98.31%
Precision	53.85%
Recall	77.78%
F1 score	63.64%

Table 3: Test Results for MultinomialNB with α=0.001, fit_prior = False

The first table indicates that the settings α=1, fit_prior = True make MultinomialNB classification more likely to classify an email as non-spam, partially due to the imbalance in the dataset. With a tuned alpha value and ignoring prior probabilities, MultinomialNB proves to be more effective.

The advantage of Multinomial Naïve Bayes is that it is easy to implement, runs fast, and yields relatively good results. The main disadvantage of Multinomial Naïve Bayes in text classification is that the assumption of feature independence would weaken the results; however, in this particular problem, spam emails are often loosely structured, hence this independence becomes less problematic.

2.3.3 Logistic Regression

Logistic regression is a classification model that categorizes observations into one of two classes or multiple classes. In the context of spam filtering, it attempts to assign higher weights to text features that enhance its ability to distinguish between spam and non-spam. (Jurafsky and Martin, 2014)

Algorithm Overview

- **The sigmoid function**
  The sigmoid function acts as a classifier that calculates $\hat{y}$, the predicted class based on the probability that the text belongs to a class. Logistic regression computes this probability by learning, from a training set, a weight vector, and a bias term.

  Denote:

  x: a vector of input features.

  w: the weight vector representing how important that input feature is to the classification decision.

  b: the bias term that gives the model the freedom to vertically shift the weights of the features in a way that fits the data.

  $P(y = 1|x)$: the probability that the text is spam, $P(y = 0|x)$: the probability that the text is non-spam.

  The output of linear transformation Z is:

$$z = \mathbf{w}.\mathbf{x} + b$$

  To convert z into a probability between 0 and 1, we plug it into a sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$P(y = 1|x) = \sigma(\mathbf{w}.\mathbf{x} + b)$$

The closer the result of this function is to 1, the more likely the text is to be spam.
Given a test set with multiple instances, the input feature vectors for each instance are consolidated into a single input matrix.

- **The cross-entropy loss function**
  The cross-entropy loss function is an objective function for learning. The model adjusts the parameters to make the predicted label $\hat{y}$ as close as possible to the true label y, or in other words, minimize the loss function.
  Denote $p = P(y = 1|x)$
  Since the event whether a text is spam or not is a Bernoulli distribution, we have:

$$P(y|x) = p^y.(1-p)^{1-y}$$

By using Maximum Likelihood Estimation, the cross-entropy loss function is:

$$L_{CE}(\hat{y}, y) = -[ylog\sigma(\mathbf{w}.\mathbf{x} + b) + (1-y)log(1-\sigma(\mathbf{w}.\mathbf{x} + b))]$$

Parameters

L1 and L2 regularization are techniques used to prevent overfitting in machine learning models. The level each technique improves the performance of our system depends on the characteristics of our dataset. We continue to use the GridSearchCV class in Python's scikit-learn library to find the suitable parameters among:
'penalty': ['l1', 'l2'], 'solver': ['liblinear'], 'C': $[10^4, 10^2, 10^6]$, 'max_iter': [100, 200, 500]

The optimal parameters we found are 'C': 10000.0, 'max_iter': 200, 'penalty': 'l2', 'solver': 'liblinear' with the following results:

Testing set accuracy	99.23%
Precision	76.67%
Recall	85.19%
F1 score	80.70%

Table 4: Test Results for Hyperparameters tuning for the Logistic Regression algorithm

2.3.4 Support Vector Machines (SVM)

Support Vector Machines is an advanced technique used for classification and regression tasks. They work by finding a hyperplane, or a decision boundary, that maximizes the margin

between two data classes, therefore providing predictions with high accuracy. SVM is efficient in terms of memory and computation since only a small group of data points have direct influence on the decision boundary.

Algorithm Overview

The SVM algorithm can be briefly described as follows: They express each data point in a space of n dimensions, with n being the number of features that are included in the data. SVMs then will try to find a boundary that can best be used to divide the data points into their respective classes. The boundary, which is called a "hyperplane", should be optimized so that it maximizes the margin - the distance between the hyperplane and the closest data points of each class while maintaining the correctness of the data classification method. Once the optimal hyperplane is found, the data points can be classified: the points belonging to one side of the hyperplane are of one category, and the rest of the data will be categorized into the other type.

There can be challenges that appear when trying to categorize given data. The margins concluded by SVMs can turn out to be too tightly strict. In this case, a "soft margin" can be introduced. In clearer explanation, the hyperplane corresponding to this soft margin will accept some data points that fall into the false category, paving the path for a more effective classification with a higher confidence level.

The data presented to SVMs can sometimes be unable to be classified linearly. However, SVMs can try to handle the situation by applying kernel functions to transform the data to a space of higher dimensions, thus giving possibilities of finding a feasible hyperplane. While common kernel functions such as linear, polynomial, or radial basis function (RBF) kernel can be utilized, users can also specify their own function to aid in classifying the given data.

Parameters

- **C (Regularization Parameter)**: Directly affects how false classifications are allowed during the classification process. A large value of C means a stricter separation, while a small value will create a wider margin, therefore giving better data generalization.
- **kernel ("linear", "poly", "rbf")**: Specifies the kernel type to be used in the algorithm.
  - "linear" determines that the Linear Function is selected:

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}$$

– "poly" refers to the Polynomial Function:

$$k(\mathbf{x}, \mathbf{z}) = (r + \gamma \mathbf{x}^T \mathbf{z})^d$$

– "rbf" is the abbreviation of Radial Basis Function:

$$k(x, z) = e^{-(\gamma ||\mathbf{x}-\mathbf{z}||^2)}$$

Adjusting the Parameters

As stated above, changes in the value of C and the use of kernel functions can directly affect the quality of data classification. Therefore, experiments are to be conducted in order to determine the optimal choices.

After finishing the experiments, the most feasible combinations of C and the corresponding kernel function are found as C = 1.0 and the Linear kernel function.

2.3.5 Artificial Neural Network (Multi-Layer Perceptrons)

Artificial neural networks (ANNs) are comprised of node layers, including an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Once an input layer is determined, weights are assigned. These weights help determine the importance of any given variable, with larger ones contributing more significantly to the output compared to other inputs. All inputs are then multiplied by their respective weights and then summed. Afterward, the output is passed through an activation function, which determines the output.

There are some common activation functions like Sigmoid, Tanh, ReLU(Rectified Linear Units).

ReLU function:

$$R(z) = max(0, z)$$

After choosing the activation function, we need to train the Neural Network. Suppose we have K classes and N samples, in order to train the Neural Network, we need to minimize the categorical cross-entropy function:

$$L = -\sum_{i=1}^{n}\sum_{k=1}^{n} t_{ik} log(y_{ik})$$

t_{ik}: the real target output with respect to the sample i
y_{ik}: the predicted probability of class k with respect to the sample i

To minimize this objective function, we have applied the idea of the Backpropagation algorithm to compute the gradient of each of the weights in the model, and use Gradient Descent to minimize it.

After finishing training the model, we can feedforward back the model to obtain the probabilities of each class given the input. And we will predict the class with the highest probability of all.

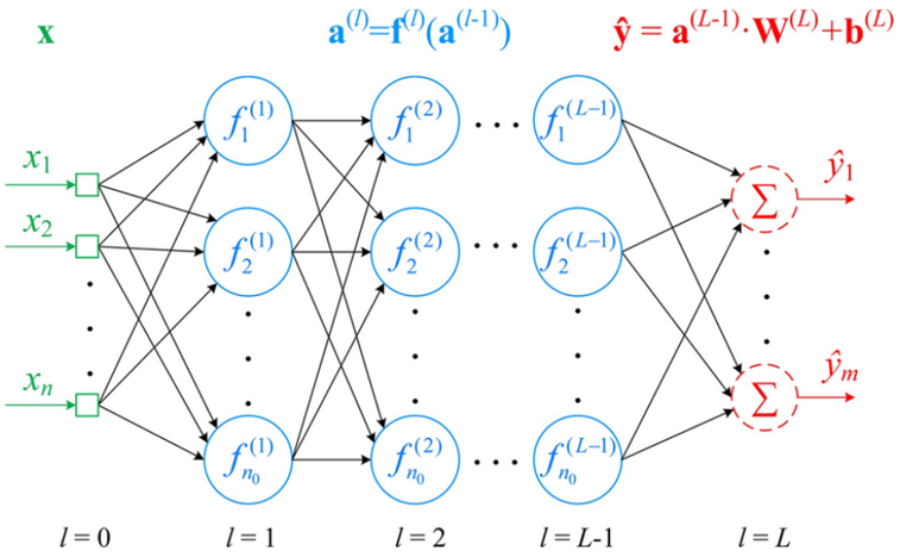

Figure 2: A visualization of how Neural Network predicts

In our Spam SMS Classification problem, we use the Keras API of TensorFlow v2.0 to implement the Neural Network model. The model allows us to choose the hyper-parameters: Activation Functions, Hidden Layer Sizes, and Number of Hidden Layers.

To choose the Activation Function, we choose the ReLU function since the dataset of the problem consists of large matrix data representation. And the ReLU function helps us to compute the gradient more efficiently than the other two functions: Sigmoid, Tanh.

And to choose the Hidden Layer Sizes and the Number of Hidden Layers, we can use K-Fold Cross Validation to test whether which sizes and how many layers we should choose for the model. In our current problem, we will choose 5-Fold Cross Validation and then compute the means and variances to test their efficiency for each value of the hyper-parameters.

After experimenting, we find that choosing three 8-node hidden layers with the activation function ReLU performs well on the data set and on the Cross Validation. We may choose more hidden layers, but it may cause overfitting and the computation time will increase significantly.

3 Experiments

3.1 Experimental setup

3.1.1 Dataset

The Vietnamese SMS dataset we use in this project contains 5692 messages, including 5569 non-spams and 123 spams, and is collected using the social media accounts and Short Message Service (SMS) of our team members and volunteers of the experiment. The dataset is shuffled after preprocessing, then split into a training set and a testing set with the test set size equal to a quarter of the total number of instances in the whole dataset. The English emails dataset we initially used to experiment with different algorithms is retrieved from Kaggle [9].

3.1.2 Evaluation metrics

We denote:

TP: the number of true positives

TN: the number of true negatives

FP: the number of false positives

FN: the number of false negatives

We evaluate the performance of our system based of the following criteria:

- Accuracy = $\dfrac{TP + TN}{TP + TN + FP + FN}$
- Precision = $\dfrac{TP}{TP + FP}$
- Recall = $\dfrac{TP}{TP + FN}$
- F1 score = $\dfrac{2 \times Precision \times Recall}{Precision + Recall}$

3.2 Experimental results

3.2.1 Performance of baseline system

Our baseline system classifies a text as spam if it satisfies one of the following conditions:

1. Contains at least 5 non-Vietnamese characters
2. Contains a hyperlink
3. Contains one word that has strange capitalization
4. Contains a money amount

Testing set accuracy	86.58%
Precision	12.04%
Recall	96.30%
F1 score	21.40%

Table 5: Performance of the baseline system for spam classification

5. Contains a phone number

From this result, we can infer that the specific rules of the baseline system seem to have a significant correlation with distinguishing between spam and non-spam messages in the dataset. However, this does not necessarily cover the full spectrum of characteristics that differentiate the two classes since the results of other experiments show that the accuracy, precision and recall metrics can be improved using machine learning algorithms. Our dataset likely contains samples where the chosen features (maximum token length, count of weird characters, ...) show consistent differences between spam and non-spam messages. These characteristics are prominent enough to achieve an 86% accuracy level, but any further enhancement requires the implementation of Natural Language Processing Techniques and careful machine learning model selection and tuning.

3.2.2 Performance with different vector representations

In this experiment, we compare the performance of 2 methods of text representation - BoW and TF-IDF. We combine the predictions of all 5 classifiers: an instance is labeled as the class that the majority of the classifiers predict it to be (majority vote model).

	BoW	**TF-IDF**
Testing set accuracy	99.23%	99.02%
Precision	90.00%	69.70%
Recall	66.67%	85.19%
F1 score	76.60%	76.67%

Table 6: Performance of different vector representations

Both BoW and TF-IDF models attain excellent accuracy, with BoW achieving a slightly higher accuracy (The difference of 0.21% is equivalent to 3 test cases in our experiment because the size of the test set is one-fourth of the total size of the dataset (which is approximately 1500). BoW demonstrates higher precision, whereas TF-IDF has a better recall and F1 score, which suggests that there is a trade-off between precision and recall. Still, we choose TF-IDF for this project because it is known for the capability of giving each unique token the weights corresponding to its importance in the whole dataset, which is something the BoW technique

can not do. Besides, since the proportion of spam messages in our dataset is very small, we would prefer a vectorizer with higher recall because a higher recall score indicates that the model is better at capturing actual spam messages and minimizing the number of spam messages that are wrongly classified as non-spam.

3.2.3 Performance of different classifiers

From the last experiment, we conclude that TF-IDF vectorizer is more suitable than BoW. Therefore, this experiment is carried out using TF-IDF as the vectorization method.

Logistic Regression, SVM, and ANN demonstrate strong overall accuracy, indicating their effectiveness in correctly categorizing instances. The precision of all models varies from low to moderately high levels, implying a high number of non-spam instances being misclassified as spam. Notably, KNN and Naive Bayes exhibit the lowest precision among these models. While SVM achieves a perfect recall of 100%, KNN lags significantly with a recall of only 29.63%, indicating its suboptimal ability to identify spam compared to other models. Finally, Logistic Regression stands out with a high F1 score of 80.70%, indicating a good balance between precision and recall.

Although, it should be noted that there may be some characteristics of the dataset used in the evaluation process that lead to the overall high accuracy. The dataset might be well-structured and have a clear decision boundary between spam and non-spam instances. Especially when we look at the result of Logistic Regression which returns the most mails classified correctly, there might be many linear relationships among the features that the algorithms are capturing effectively.

	KNN	**Naive Bayes**	**Logistic Regression**	**SVM**	**ANN**
Testing set accuracy	98.38%	98.31%	99.23%	98.81%	98.81%
Precision	66.67%	53.85%	76.67%	61.36%	62.50%
Recall	29.63%	77.78%	85.19%	100.00%	92.59%
F1 score	41.03%	63.64%	80.70%	76.06%	74.63%

Table 7: Test Results for different classifiers

4 Discussion and Conclusions

In conclusion, we have successfully implemented an SMS spam filtering system with the majority vote model reaching a high accuracy of 99.02%. While the performance metrics are promising, it is important to note that the system's ability to correctly classify non-spam still requires improvement. In the future, we hope to be able to collect more diverse and representative samples from various sources, including different platforms, communication channels, and time periods to capture a wide range of spamming techniques and variations. Besides the text content of messages, we can consider including additional metadata of features associated with messages, such as sender information, timestamps, message length, or any other relevant contextual information. These additional features might provide valuable signals for better classification.

References

[1] GeeksforGeeks. (2020). Artificial Neural Networks and its Applications. [online] Available at: https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/.

[2] IBM (2023). What Are Neural Networks? — IBM. [online] www.ibm.com. Available at: https://www.ibm.com/topics/neural-networks.

[3] Jiang, Q., Zhu, L., Shu, C. and Sekar, V. (2021). An efficient multilayer RBF neural network and its application to regression problems. Neural Computing and Applications. doi:https://doi.org/10.1007/s00521-021-06373-0.

[4] Jurafsky, D. and Martin, J.H. (2014). Speech and Language Processing: an Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. [online] India: Dorling Kindersley Pvt, Ltd. Available at: https://web.stanford.edu/ jurafsky/slp3/ed3book.pdf.

[5] Orred, K. (n.d.). 2022 Spam Text Statistics: Are Spam Texts on the Rise? [online] www.text-em-all.com. Available at: https://www.text-em-all.com/blog/spam-text-statistics.

[6] Pham, T.-H. and Le-Hong, P. (2017). Content-based Approach for Vietnamese Spam SMS Filtering. arXiv (Cornell University). doi:https://doi.org/10.48550/arxiv.1705.04003.

[7] Statista. (n.d.). Spam e-mail traffic share 2019. [online] Available at: https://www.statista.com/statistics/420400/spam-email-traffic-share-annual/.

[8] Wikipedia. (2019). Sigmoid. [online] Available at: https://en.wikipedia.org/wiki/Sigmoid [Accessed 29 Dec. 2023].

[9] www.kaggle.com. (n.d.). Spam email Dataset. [online] Available at: https://www.kaggle.com/datasets/jackksoncsie/spam-email-dataset.