Cybersecurity and machine learning - SMS Spam Detection

Diogo Marto
DMAT/DETI
Universidade de Aveiro
diogo.marto@ua.pt

Pedro Azevedo
DMAT/DETI
Universidade de Aveiro
pgca@ua.pt

***Abstract*—This report explores the application of machine learning (ML) techniques to build efficient and accurate SMS spam detection systems. It examines both traditional classifiers, such as Logistic Regression and Support Vector Machines (SVM), and advanced models like Long Short-Term Memory (LSTM) networks. The study includes a comprehensive review of state-of-the-art spam detection methods, focusing on data preprocessing, feature extraction using TF-IDF, and hyperparameter optimization through grid search. Comparative analyses of the models are provided to assess their performance. This research aims to contribute to the development of more robust and scalable cybersecurity solutions for SMS-based communications.**

***Index Terms*—SMS Spam Detection, Cybersecurity, Machine Learning, Natural Language Processing (NLP), Long Short-Term Memory (LSTM), TF-IDF**

This report was made for the FAA course with Petia Georgieva (petia@ua.pt) as its instructor.

I. Introduction

The exponential growth of mobile communications has brought about both opportunities and challenges in cybersecurity. With millions of users relying on SMS services daily, the threat of spam and malicious messages has become a pressing concern. SMS spam can range from unsolicited advertisements to phishing attacks engineered to compromise personal information and sensitive data. As traditional spam filters often fail to adapt to evolving spam tactics, machine learning (ML) has emerged as a powerful tool for developing more effective detection systems.

In this paper, Diogo Marto and Pedro Azevedo from the Universidade de Aveiro explore the intersection of cybersecurity and machine learning by focusing on SMS spam detection. This work examines how various machine learning models, including traditional classifiers and more advanced architectures like Long Short-Term Memory (LSTM) networks, can be employed to detect and prevent SMS spam.

The research addresses several key stages of the spam detection process, starting with data preprocessing and continuing through feature extraction using TF-IDF (Term Frequency-Inverse Document Frequency) representations. The authors then compare the performance of multiple machine learning models, including Logistic Regression, Random Forest, Decision Tree, Support Vector Machines (SVM), and LSTM networks, using grid search techniques to optimize model parameters and improve detection accuracy.

The core objective of this study is to evaluate the effectiveness of these approaches and provide a comparative analysis of their performance in detecting SMS spam. The findings presented in this work have the potential to enhance existing spam detection frameworks and pave the way for more robust security solutions in the telecommunications sector.

II. Problem Description

A many-to-one model is a neural network architecture designed to handle sequences of inputs and produce a single output. This type of model is particularly suited for tasks such as text classification, where the objective is to analyze a sequence of words or characters and predict a single label or category. In the context of this study, an LSTM model is applied to classify SMS messages as either spam or ham (non-spam).

The input to the model is a text message, which is processed as a sequence of words or tokens. The output is a binary label indicating whether the message is spam (1) or not (0). For example:

- Input: “Congratulations! You have won a free gift.”
- Output: Spam (1)
- Input: “Hey, are we meeting at 5 PM?”
- Output: Ham (0)

Besides that, there are techniques for text representation in Natural Language Processing (NLP) tasks such as TF-IDF (Term Frequency-Inverse Document Frequency). In the context of this study, TF-IDF plays a critical role in extracting meaningful features from SMS messages and converting them to structured data.

III. State of the Art

[1], by Gregorius Airlangga, provides a comprehensive evaluation of various machine-learning algorithms applied to SMS spam detection. The study contrasts traditional classifiers, such as Naïve Bayes and Support Vector Machines, with ensemble methods like Random Forests and AdaBoost, assessing their performance based on accuracy, precision, recall, and F1-score. The findings indicate that ensemble classifiers mostly outperform traditional methods, obtaining higher accuracy and better generalization capabilities in identifying spam messages. With a dataset with an approximate imbalance to our own, it achieved very respectable results. All of the models reached an accuracy of over 97%, with f1-scores being lower

for spam in general. These models included Random Forest, Gradient Boosting, AdaBoost, Support Vector Machine, Logistic Regression, and Ensemble Voting Classifier.

[2], by Himani Jain et al., provides a comprehensive overview of various machine learning algorithms applied to spam detection across different communication platforms, including email, SMS, and social networks. The study categorizes and evaluates techniques such as Naïve Bayes, Support Vector Machines, Decision Trees, and ensemble methods, analyzing their effectiveness in identifying spam content. They also apply TF-IDF, which they report to be an important factor in the extraction of information. They implement multiple classifiers, the more relevant models include Decision Tree and Random Forest, reaching an accuracy of 95.69% and 97.00% respectively, and a Naïve Bayes and SVM Classification that reached 98.02% and 97.48% accuracy. Their best model is a Multinominal Naïve Bayes model that reached 99.7% accuracy.

[3] by Mustanger showcases advanced techniques for spam message classification. By leveraging deep learning architectures like LSTM (Long Short-Term Memory) and embedding layers, this model processes textual data effectively, capturing complex patterns in message content. Preprocessing steps, such as tokenization and padding, ensure the model's ability to handle varying input sizes. Evaluation metrics like accuracy and precision further highlight the method's effectiveness in detecting spam, making it a notable contribution to the field of text classification and natural language processing. The model LSTM presented achieved an F1 score of 0.87 and had 168 thousand trainable parameters.

[4] The paper explores by AndresHG the application of various machine-learning techniques for text classification and sentiment analysis tasks. It leverages a dataset of tweets to distinguish between real and fake disaster events. The document delves into Naïve Bayes and LSTM models, demonstrating their effectiveness in text classification. Finally, it explores the GloVe word embedding model, highlighting its potential to enhance the performance of text classification models. Incorporating LSTM with GloVe embeddings is an intriguing approach to text classification. It achieved an accuracy of 0.9662 with XGboost and 0.9174 with an LSTM.

IV. Dataset Description

The dataset used for this study is the SMS Spam Collection Dataset, sourced from Kaggle [5]. It is a publicly available dataset that contains 5,572 SMS messages, which are labeled as either "ham" (non-spam) or "spam". This dataset is widely used in spam detection research due to its diverse range of text messages, including personal, promotional, and fraudulent messages.

The dataset consists of two main columns:

- class: The target label indicating whether the message is spam or ham.
- text: The SMS message content in raw text form.

Upon initial inspection, several redundant columns were found, namely Unnamed: 2, Unnamed: 3, and Unnamed: 4. These columns contained missing or irrelevant data and were removed to streamline the dataset for analysis. The dataset is imbalanced, with 4,827 ham messages (86.6%) and 747 spam messages (13.4%) as shown Fig. 1. Due to this imbalance we consider our main metric F1-score since accuracy could be misleading in certain cases.

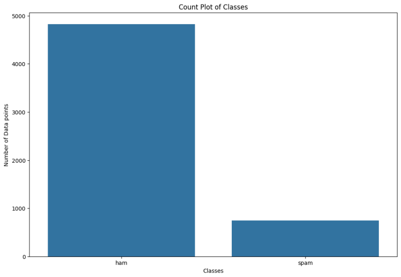


Fig. 1: Class Distribution

To understand the underlying data structure we made a pair plot comparing various text-related features such as the number of characters, number of words, and number of sentences in messages labeled as either ham (non-spam) or spam. Each scatter plot on Fig. 2 displays the relationship between two features, while the diagonal plots show the distribution of individual features. The color legend distinguishes between the two classes: blue represents ham, and orange represents spam. The data shows that spam messages generally tend to have higher counts of characters and words compared to ham messages, suggesting that longer messages are more likely to be classified as spam. Additionally, we can conclude that by only using these metrics it's impossible to separate the 2 classes.

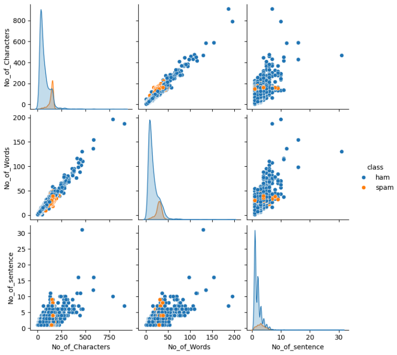


Fig. 2: Pair plot comparing number of characters, number of words, and number of sentences in messages

Fig. 3 and Fig. 4 present a bar chart and comparing the two we can see differences in top word frequency between classes. Similarly, Fig. 5 and Fig. 6 show the top 10 bigrams for each class. The charts show that spam messages tend to contain more promotional and urgent language, with frequent occurrences of words like "free," "win," and "claim." In contrast, ham messages often contain more casual, everyday language. This difference in word frequency shows that there is a clear difference in the vocabulary between the two classes and that applying methods such as TF-IDF or Bag of Words could prove successful.

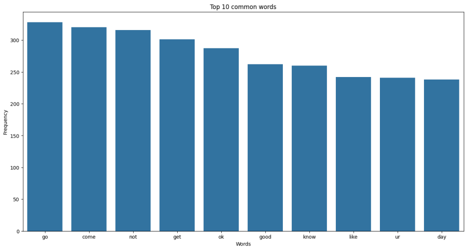


Fig. 3: Top 10 common words in ham class

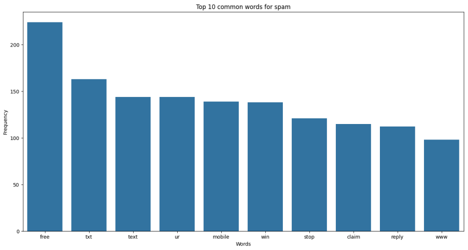


Fig. 4: Top 10 common words in spam class

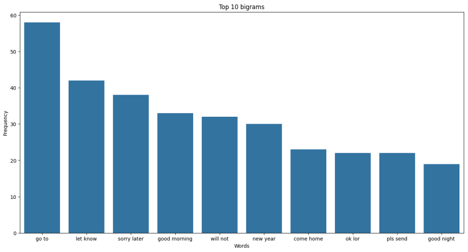


Fig. 5: Top 10 bigrams in ham class

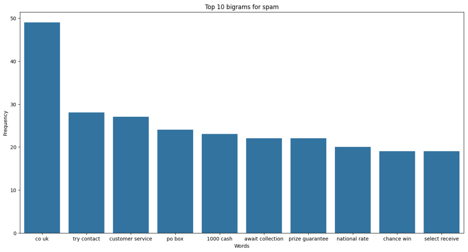


Fig. 6: Top 10 bigrams in spam class

V. Data Preprocessing

In this project, a comprehensive data preprocessing pipeline was developed to prepare the raw SMS text messages for classification into spam or ham (non-spam). The preprocessing steps involved text cleaning, lemmatization, removal of stop words, and the application of the TF-IDF (Term Frequency-Inverse Document Frequency) transformation to convert text data into numerical features for models that will use them.

1) Text Cleaning

The first step of preprocessing was text cleaning, where the raw text data was stripped of unwanted characters and symbols. Special characters such as "&lt;", "&gt;", and "åÕ" were removed from the text. Additionally, redundant columns in the original dataset were dropped to streamline the data for further analysis.

2) Lemmatization

Lemmatization is a critical NLP technique that involves reducing words to their base or root form (lemma). Unlike stemming, which often removes word endings without considering the context, lemmatization ensures that words are reduced to a linguistically correct form. For example, words such as "running" and "ran" are lemmatized to their base form "run," improving the uniformity of the text data. This reduces the noise in textual data and the number of features and can help with model training.

The project utilized SpaCy's pre-trained language model (en_core_web_sm) for lemmatization. A function was then defined that converts text to lowercase, tokenizes it, and applies lemmatization to each token while simultaneously removing stop words.

3) Removal of Stop Words

Stop words are commonly used words with little semantic meaning in a text and are often removed to reduce noise in text data. Words such as "is," "and," "the," and "of" are examples of stop words that appear frequently but do not contribute significantly to understanding the content or intent of a message. This can help models focus on more important features.

In this project, stop words were filtered out during the lemmatization process using SpaCy's built-in stop word list. This helped in reducing the dimensionality of the text data and ensuring that only meaningful words were retained for further analysis.

4) TF-IDF Transformation

The cleaned text data was then transformed using TF-IDF (Term Frequency-Inverse Document Frequency), a widely used feature extraction technique in text classification tasks. TF-IDF is used to convert textual data into numerical features that can be processed by machine learning algorithms. It measures the importance of a word in a document relative to its occurrence in the entire corpus.

Term Frequency (TF) [6]: This measures how frequently a word appears in a given document. Inverse Document Frequency (IDF): This measures how unique or rare a word is across all documents in the corpus. The TF-IDF score is computed as the product of TF and IDF:

$$TFIDF(t) = TF(t)IDF(t) \tag{1}$$

where t is a term, $TF(t)$ is the frequency of term t in a document, and $IDF(t)$ is the logarithmically scaled inverse fraction of the documents that contain t.

The TF-IDF vectorizer we used was the one included in scikit-learn [7]. The resulting 7473 numerical features were then concatenated to the original dataset.

5) Tokenizer

To preprocess the text data for the LSTM model, we employed tokenization to convert raw text into numerical sequences suitable for sequential modeling. Using TensorFlow's Tokenizer [8], we limited the vocabulary size to the top 2000 most frequent words to reduce noise and computational complexity. The tokenizer mapped each word to a unique integer ID based on its frequency, and the text samples were then transformed into sequences of these IDs. To ensure uniform input length, the sequences were padded or truncated to a fixed of 100, accommodating the LSTM's input shape requirements. This process preserved the semantic order of the text while standardizing its structure, enabling the LSTM model to make use of it.

6) Data Split

To ensure the model's performance is evaluated accurately and generalizes well to unseen data, the dataset was split into a training set (80%) and a test set (20%). The training set contained 4457 samples and the test set had 1115. The training set was used to train the classification models, while the test set was reserved to evaluate the model's predictive performance on unseen messages. This split ratio balances the need for sufficient training data with the requirement of a reliable evaluation set.

VI. LSTM

A. *LSTM description*

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) designed to address the vanishing gradient problem in standard RNNs, making them particularly effective for sequence prediction tasks like SMS spam classification. LSTM networks function through a series of gates (input, forget, and output), each controlling the flow of information within the network. Mathematically, the update mechanism is governed by the following key equations:

- The forget gate decides what information should be discarded,
- The input gate determines what new information should be stored,
- The output gate controls the output based on the current memory and the input sequence.

This enables LSTMs to capture both short- and long-term dependencies in sequential data, making them suitable for tasks like SMS spam classification where the context of previous words influences the classification decision. [9]

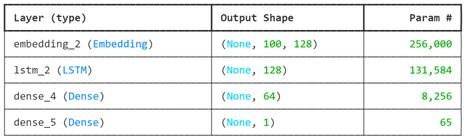
B. *Methodology*

a) *Architecture:*

To train the LSTM model, we employed a structured approach that used early stopping and grid search techniques for optimization. These functions were available on the TensorFlow Python library [8].

The architecture of the LSTM model included:

1) Embedding Layer: Converts input tokens into dense vector representations of specified dimensions (embedding_dim), capturing the semantic meaning of words. Hyperparameter: embedding_dim was varied across values [50, 100] during grid search.
2) LSTM Layer: Core component of the architecture, designed to capture temporal dependencies in sequences through its gated mechanisms (input, forget, and output gates). Configured with units (lstm_units) controlling the size of the hidden state, and regularization using dropout and recurrent_dropout to mitigate overfitting. Hyperparameter: lstm_units was tested with values [24, 32, 48] and dropout_rate was varied across [0.1, 0.2, 0.4].
3) Dense Layers: A fully connected layer with ReLU activation, followed by a single-unit dense layer with a sigmoid activation for binary classification. Hyperparameter: The number of units in the dense layer (dense_units) was tuned with values [32, 64, 96].



Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 100, 128)	256,000
lstm_2 (LSTM)	(None, 128)	131,584
dense_4 (Dense)	(None, 64)	8,256
dense_5 (Dense)	(None, 1)	65

Fig. 7: Example of LSTM Architecture

b) *Hyperparameter Tuning Using Grid Search:*

To identify the best-performing model, we employed a grid search strategy. The grid search explored the hyperparameter space, including:

- embedding_dim: [50, 100]
- lstm_units: [24, 32, 48]
- dense_units: [32, 64, 96]
- dropout_rate: [0.1, 0.2, 0.4]

For all 54 parameter combinations, the model was instantiated and trained on train data and evaluated on test data. To pick the best model we compared the F1-score of the models.

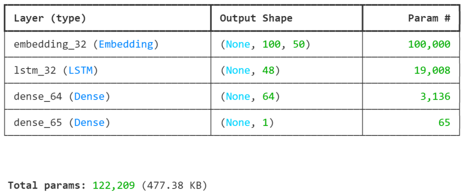
The training process utilized a batch size of 128 and was run for up to 20 epochs, although early stopping typically terminated training earlier. The Adam optimizer and categorical cross-entropy loss were used for optimization and evaluation, respectively.

c) *Visualizing results:*

The best model obtained had the parameters:

- 'dense_units': 64,
- 'dropout_rate': 0.2,
- 'embedding_dim': 50,
- 'lstm_units': 48

As shown in Fig. 8



Layer (type)	Output Shape	Param #
embedding_32 (Embedding)	(None, 100, 50)	100,000
lstm_32 (LSTM)	(None, 48)	19,008
dense_64 (Dense)	(None, 64)	3,136
dense_65 (Dense)	(None, 1)	65

Total params: 122,209 (477.38 KB)

Fig. 8: Architecture of LSTM

It obtained an F1-score of 0.9872, an accuracy of 0.9874, a recall of 0.9874, a precision of 0.9876, and AUC score (area under the curve) of 0.9533. In Fig. 9 we can see accuracy and loss over epochs and we can see that the model is slightly overfitted but the performance on the test set is satisfactory.

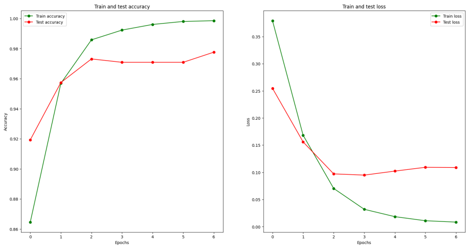


Fig. 9: Accuracy and loss functions over training epochs

The confusion matrix on Fig. 10 shows 1 false negative and 17 false positives out of 1115 samples.

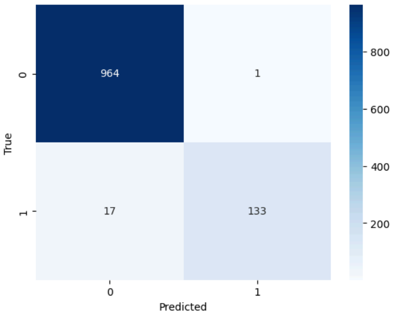


Fig. 10: Confusion Matrix of the best LSTM model

Some of misclassified samples are (1 for spam 0 for ham):

```
Funny fact Nobody teaches volcanoes 2 erupt, tsunamis 2 arise, hurricanes 2 sway aroundn no 1 teaches hw 2 choose a wife Natural disasters just happens (Predicted: [1.], Actual: 0)

Loans for any purpose even if you have Bad Credit! Tenants Welcome. Call NoWorriesLoans.com on 08717111821 (Predicted: [0.], Actual: 1)

ringtoneking 84484 (Predicted: [0.], Actual: 1)

LookAtMe!: Thanks for your purchase of a video clip from LookAtMe!, you've been charged 35p. Think you can do better? Why not send a video in a MMSto 32323. (Predicted: [0.], Actual: 1)

INTERFLORA - åÒIt's not too late to order Interflora flowers for christmas call 0800 505060 to place your order before Midnight tomorrow. (Predicted: [0.], Actual: 1)
```

VII. Models on TF-IDF

A. General Approach

The models of this section used the TF-IDF features we extracted from the dataset and for all these models we used a grid search to find the best parameters, the best model was evaluated by picking the model with the best F1 score. We used the scikit-learn Python library [7] to train all these models.

B. Logistic Regression

Logistic Regression is a statistical model used for binary classification tasks, where the goal is to categorize input data into one of two classes. Logistic regression predicts probabilities of an observation belonging to a particular class. These probabilities are then mapped to discrete labels (e.g., spam or not spam) using a threshold, commonly 0.5. The parameter grid for the Logistic Regression model was designed to explore the impact of different regularization techniques, solvers, and regularization strength values on model performance. The specific parameters evaluated were as follows: Regularization (regularization): Two types of regularization techniques were tested:

- L2 regularization (Ridge regression)
- L1 regularization (Lasso regression)

Solver (solver): The optimization algorithm used for weight updates was fixed to:

- 'lbfgs' (Limited-memory Broyden-Fletcher-Goldfarb-Shanno)

Inverse Regularization Strength (C): A range of values was tested to determine the optimal regularization strength, spanning several orders of magnitude: 10^{-3},10^{-2}, 10^{-1},0.5,1,10,100,1000,10^{4},10^{5}

The optimal configuration used L2 regularization, the 'lbfgs' solver, and a high value of $C = 10^5$. Our Logistic Regression model reached a 97.70% f1-score and confusion matrix presented on Fig. 11, being one of the best-performing models.

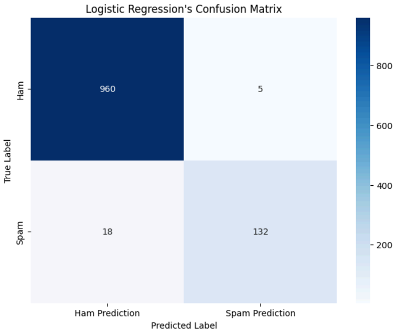


Fig. 11: Logistic Regression's Confusion Matrix

C. Random Forest

Random Forest is an ensemble machine learning algorithm primarily used for classification and regression tasks. It operates by constructing multiple decision trees during training and combining their outputs to make a prediction. After many attempts to find the best combination for the model, including manipulating the class weights to adjust for the imbalanced dataset, and using similar settings to the ones mentioned in [1] to tune its performance. We believe not setting a limit to the maximum depth helped the model's performance, as manipulating this value shows worse results overall. The parameter grid for the Random Forest model was:

- Maximum Depth (max_depth): Set to None, allowing trees to expand fully until all leaves are pure or contain fewer samples than required for splitting. This configuration enables the model to capture intricate patterns in the data without imposing a predefined depth limit.
- Number of Estimators (n_estimators): Evaluated across 100, 200, and 300 trees to determine the impact of ensemble size on performance. The optimal configuration used 100 estimators, balancing computational efficiency with accuracy.
- Minimum Samples for Splitting (min_samples_split): Tested with values of 5, 10, and 20. The optimal value of 20 was selected, effectively reducing overfitting by enforcing a stricter threshold for further splits in the trees.
- Class Weights (class_weight): To address the class imbalance in the dataset, the "balanced" setting was employed. This configuration assigns weights inversely proportional to class frequencies, ensuring that both classes are adequately represented during training.

The optimal configuration used the following settings: 'class_weight': 'balanced', 'max_depth': None, 'min_samples_split': 20, 'n_estimators': 100. This configuration resulted in an F1-score of 94.87% and the following confusion matrix presented in Fig. 12.

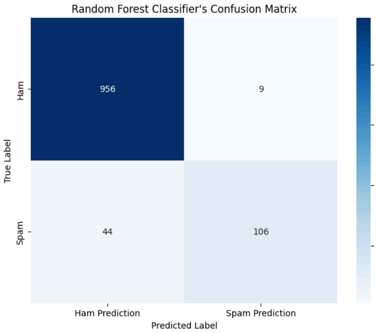


Fig. 12: Random Forest Classifier's Confusion Matrix

D. Decision Tree

A Decision Tree model is a supervised learning algorithm that splits the data into subsets based on feature values, creating a tree-like structure with decision nodes and leaf nodes. It makes predictions by traversing the tree from the root to a leaf, where each leaf represents a predicted outcome based on the majority class or average value in that subset. The parameter grid for the Decision Tree model was:

- Maximum Depth (max_depth): Evaluated across the values 10, 20, 30, and 50 to assess the impact of tree depth on model performance. The optimal configuration selected a maximum depth of 50, allowing the tree to capture complex patterns while maintaining generalization.

The optimal configuration used the following setting: 'max_depth': 50. This configuration resulted in an F1-score of 97.62% and the following confusion matrix presented on Fig. 13.

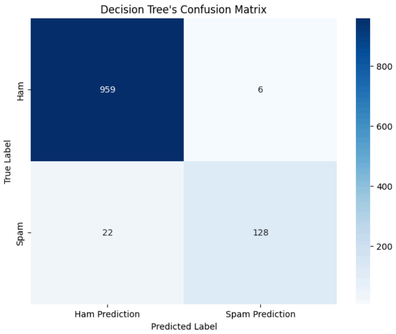


Fig. 13: Decision Tree's Confusion Matrix

E. Support Vector Machine

To implement a Support Vector Machine, we used scikit-learn's SVC (Support Vector Classifier). The core idea behind SVC is to find the best hyperplane that separates data points belonging to different classes in a high-dimensional space. The parameter grid for the SVM model was:

- C: Tested with values of 0.1, 10, and 100 to assess the regularization strength and its impact on model performance.
- Gamma: Set to 'scale' to automatically adjust the gamma value based on the number of features.
- Kernel: Evaluated with both 'linear' and 'rbf' kernels to determine the best transformation of the data space for separation.

The optimal configuration used the following settings: 'C': 10, 'gamma': 'scale', and 'kernel': 'linear'. This configuration resulted in an F1 score of 97.32% and the following confusion matrix presented on Fig. 14. To note that the 'linear' kernel consistently outperformed the 'rbf' kernel suggesting that this data is very linear.

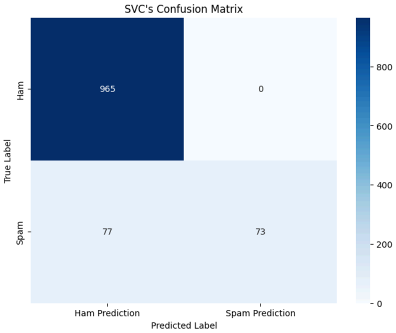


Fig. 14: Support Vector Machine's Confusion Matrix

F. Visualizing the results

Fig. 15 shows a comparison between all the models in this section to note that all models achieved an F1 score above 94% and the Logistic Regression was the best with a score of 0.9770.

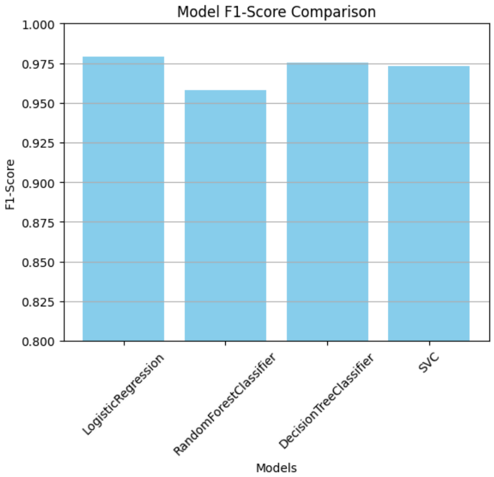


Fig. 15: F1-Score Comparison

VIII. Comparison

In this section we will analyze the performance of the models used for SMS spam detection, focusing on accuracy and F1-score as primary evaluation metrics. Table I highlights the key performance metrics of Logistic Regression, Random Forest Classifier, Decision Tree, Support Vector Machine (SVM), and Long Short-Term Memory (LSTM) of our models.

TABLE I: Comparison between models

Model	**Accuracy**	**F1-score**
Logistic Regression	0.9793	0.9789
Random Forest Classifier	0.9596	0.9579
Decision Tree Classifier	0.9757	0.9751
SVC (State Vector Classifier)	0.9739	0.9732
LSTM	0.9874	0.9872

Among the models tested, the LSTM model achieved the highest performance, with an accuracy of 98.74% and an F1-score of 98.72%. Logistic Regression followed with an accuracy of 97.93% and an F1-score of 97.89%. Random Forest showed a slightly lower performance with an F1-score of 95.79%. Decision Tree and SVM also demonstrated robust results, achieving F1 scores of 97.51% and 97.32%, respectively.

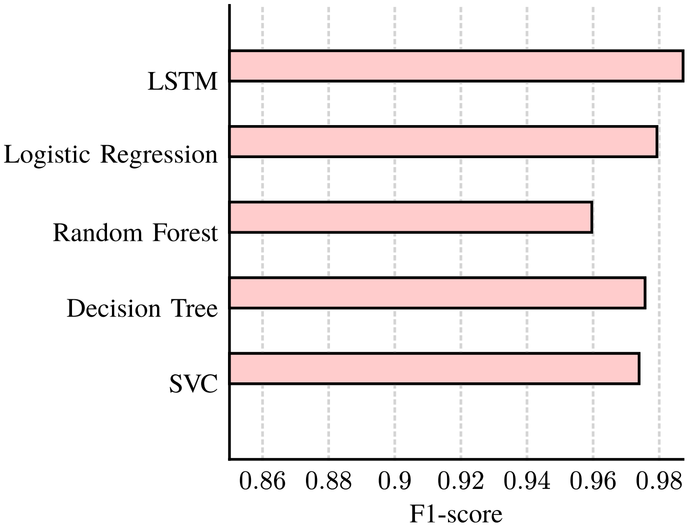


Fig. 16: Barchart of F1-score per model (Our models)

When comparing with previous studies, our models performed on par or slightly better, particularly with the LSTM model achieving higher F1 scores compared to benchmarks cited in related works. For instance, the study by Himani Jain et al. reported their highest-performing Multinomial Naïve Bayes model reaching 99.7% accuracy. Our results, while not surpassing this benchmark, align closely with state-of-the-art results in the field and reflect robust model training and optimization strategies. To note that our LSTM model significantly outperformed other LSTM models.

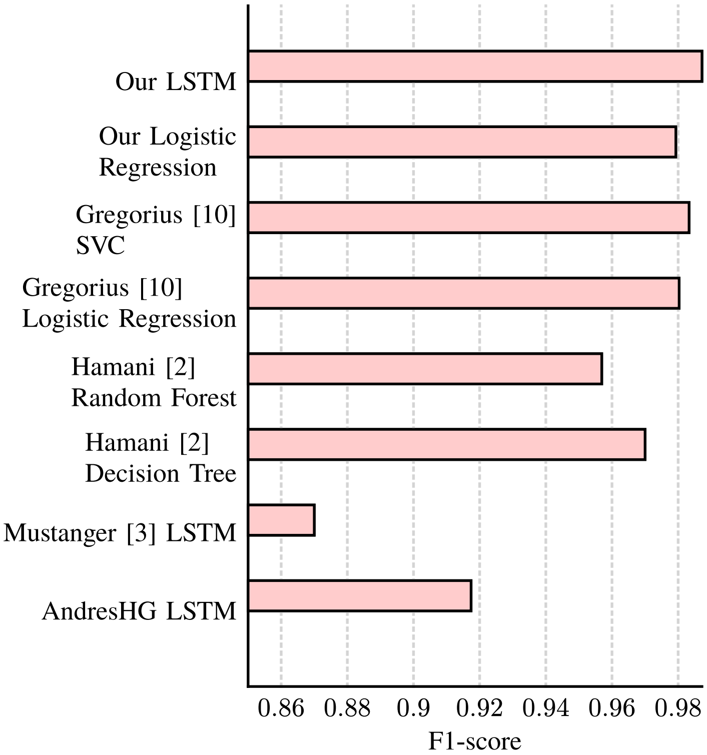


Fig. 17: Barchart of F1-score per model (Our best models vs references)

IX. Discussion

1) Time to Compute

The computational time for training and evaluating each model varied significantly. Logistic Regression and Support Vector Machine (SVM) with a linear kernel exhibited the fastest training times due to their relatively straightforward mathematical formulations. These models required only a few seconds to train, making them well-suited for scenarios where computational efficiency is critical.

Conversely, the LSTM model demanded significantly more computational resources, with training times extending into several minutes per epoch. The increased training time can be attributed to the sequential processing of input data and the complexity of the gated mechanisms inherent in recurrent neural networks. While LSTM provides slightly superior performance, its higher computational requirements may limit its applicability in real-time or resource-constrained environments. Similarly, ensemble methods like Random Forest required moderate computational time, primarily due to the construction and aggregation of multiple decision trees.

To balance these trade-offs, it is important to consider the specific application requirements. We observed that the LSTM model is 10.8x times slower at classifying the test data.

2) Concept Drift

An additional challenge in SMS spam detection lies in concept drift, where the statistical properties of incoming data change over time. This phenomenon is particularly relevant in cybersecurity applications, as spam tactics continually evolve to bypass detection systems. Models trained on static datasets, such as the SMS Spam Collection Dataset, may degrade in performance when applied to new or unseen spam messages. Addressing concept drift requires periodic retraining or adaptation of models to incorporate new data. Techniques like online learning or incremental updates can help maintain model relevance over time. Since some of our models rely on TF-IDF which relies on a specific corpus of documents this problem can become more evident over time, we suspect that since LSTM learns more about the relations between words this problem is less evident compared to TF-IDF models.

3) Implications and Future Directions

The interplay between computational time and the need to handle changes to spam email structure over time is a complex optimization problem. Real-time applications must balance the need for fast, efficient computation with the ability to adapt to new data patterns. Future work should explore:

- Active learning approaches: Incorporating user feedback or adaptive mechanisms to address evolving spam behaviors dynamically.
- Transfer learning: Utilizing pre-trained models like BERT or GPT for continual learning, reducing training time while addressing concept drift effectively.
- Data Augmentation: Implementing data augmentation techniques to generate diverse training examples can help in addressing the challenge of limited labeled data and reduce overfitting and concept drift. We could create such structured e-mails, for example, through generative AI(such as Chat-GPT).

Besides that, for usability purposes, we consider the following topics important areas to improve on

- Developing methods to improve the interpretability of models, such as visualization tools and techniques for extracting feature importance can make these models more transparent and easier to trust in practical applications.
- Allow for the classification of spam e-mails in multiple languages. This would be possible by training multiple models, with different languages as sources or training one generalist model.

X. Conclusion

The study successfully explored the use of machine learning techniques to detect SMS spam, addressing critical cybersecurity concerns in the telecommunications sector. By analyzing traditional classifiers like Logistic Regression and advanced architectures such as LSTM, the research demonstrated the comparative strengths and weaknesses of various approaches in spam detection tasks. Among these, the LSTM model emerged as the most effective, achieving the highest accuracy and F1-score, though it required significantly more computational resources compared to simpler models like Logistic Regression.

References

[1] G. Airlangga, "SMS Spam Detection: A Comparative Analysis of Machine Learning Algorithms," *Indonesian Journal of Network and Cyber Security*, vol. 12, no. 2, pp. 47–58, 2023, doi: 10.47709/cnahpc.v6i4.4822.

[2] H. Jain and M. Mahadev, "An Analysis of SMS Spam Detection using Machine Learning Model," in *2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT)*, 2022, pp. 151–156. doi: 10.1109/CCiCT56684.2022.00038.

[3] U. Eisgandar, "Spam SMS Detector: Deep Learning Methods." [Online]. Available: https://www.kaggle.com/code/eisgandar/spam-sms-detector-deep-learning-methods

[4] AndresHG, "NLP 📝 GloVe, BERT, TF-IDF, LSTM... 📝 Explained." 2021.

[5] U. M. L. Repository, "SMS Spam Collection Dataset." [Online]. Available: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset/data

[6] C. O. Tech, "Understanding TF-IDF: A Simple Introduction." [Online]. Available: https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/

[7] F. Pedregosa *et al.*, "scikit-learn: Machine Learning in Python - API Reference." 2011. [Online]. Available: https://scikit-learn.org/stable/api/index.html

[8] T. Developers, "TensorFlow: API Documentation." 2015. [Online]. Available: https://www.tensorflow.org/api_docs

[9] W. contributors, "Long Short-Term Memory." [Online]. Available: https://en.wikipedia.org/wiki/Long_short-term_memory

[10] S. Narayanan, B. Murthy, S. Pal, and K. Shanker, "A new species of the genus *Cnemaspis* Strauch, 1887 (Squamata: Gekkonidae) from the Western Ghats, India," *Herpetology Notes*, vol. 6, no. 4, pp. 589–597, 2013, doi: 10.47709/cnahpc.v6i4.4822.