HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

MACHINE LEARNING PROJECT

Email Spam Classifier

DƯƠNG MẠNH KIÊN
Kien.DM225576@sis.hust.edu.vn

HẠ NHẬT DUY
Duy.HN225544@sis.hust.edu.vn

LÊ TUẤN NAM
Nam.LT225581@sis.hust.edu.vn

NGÔ ANH QUÂN
Quan.NA225584@sis.hust.edu.vn

BÙI VIỆT HUY
Huy.BV225574@sis.hust.edu.vn

Supervisor: Dr. Thân Quang Khoát

Hanoi, 29/05/2024

ABSTRACT

Email spam detection is a critical task aimed at filtering out unwanted or malicious emails from users' inboxes. In this Email Spam Classifier project, we leverage machine learning methodologies to enhance the accuracy of our classifier model. By applying various algorithms and techniques, we aim to develop a robust classifier capable of accurately identifying spam emails and minimizing false positives.

Student
(Signature and full name)

Le Tuan Nam

TABLE OF CONTENTS

The Email Spam Classifier project utilizes machine learning methodologies to develop an effective model for classifying emails as either spam or non-spam (ham). Similar to the IMDb Reviews dataset used for binary sentiment classification, the Email Spam Classifier project aims to filter unwanted or malicious emails from users' inboxes.

The dataset for this project consists of a collection of labeled email examples : nearly 6000 emails, with an equal number of instances for training and testing purposes.

Upon preprocessing the text data and implementing word vectorization techniques, the project will explore various machine learning algorithms for classification. These include traditional supervised learning algorithms such as Support Vector Machine (SVM), Logistic Regression, and XGBoost.

An example a review classified as "Spam": "FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, å£1.50 to rcv".

An example of "Ham" email : "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat..."

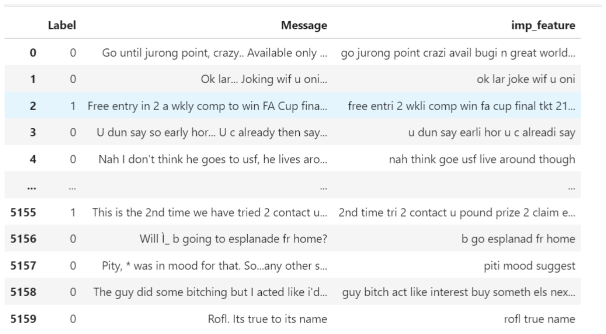
CHAPTER 2. DATA PREPROCESSING

2.1 Clean and tokenize

We do the basic learning techniques to clean the data: where:

- remove “(<.*?>)” markup,
- remove punctuation marks,
- remove all strings that contain a non-letter,
- convert to lower,
- remove stop words,
- Potter Stem: reduce words to their root form by removing common suffixes such as "ing," "ed," "s," etc., without regard to the meaning of the word,
- remove empty emails.

`Function "fit_transform" from library sklearn change "spam" and "ham" in "Label" row to 1 and 0, respectively. Also, function "drop_duplicates" in library pandas helps to remove duplicate emails.`



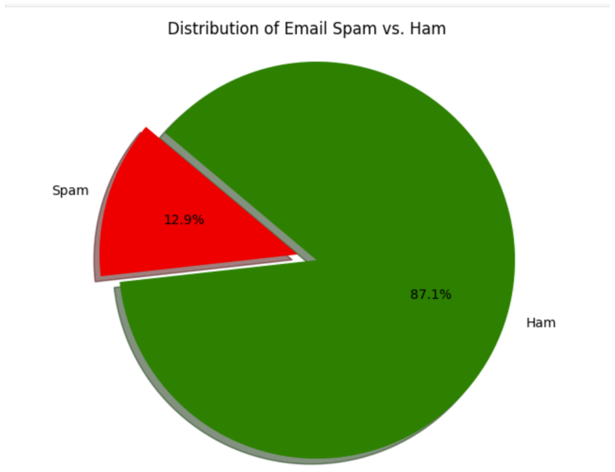
	Label	Message	imp_feature
0	0	Go until jurong point, crazy.. Available only ...	go jurong point crazi avail bugi n great world...
1	0	Ok lar... Joking wif u oni...	ok lar joke wif u oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	free entri 2 wkli comp win fa cup final tkt 21...
3	0	U dun say so early hor... U c already then say...	u dun say earli hor u c alreadi say
4	0	Nah I don't think he goes to usf, he lives aro...	nah think goe usf live around though
...	...	...	...
5155	1	This is the 2nd time we have tried 2 contact u...	2nd time tri 2 contact u pound prize 2 claim e...
5156	0	Will Ì_ b going to esplanade fr home?	b go esplanad fr home
5157	0	Pity, * was in mood for that. So...any other s...	piti mood suggest
5158	0	The guy did some bitching but I acted like i'd...	guy bitch act like interest buy someth els nex...
5159	0	Rofl. Its true to its name	rofl true name

Figure 2.1: Emails before and after cleaning

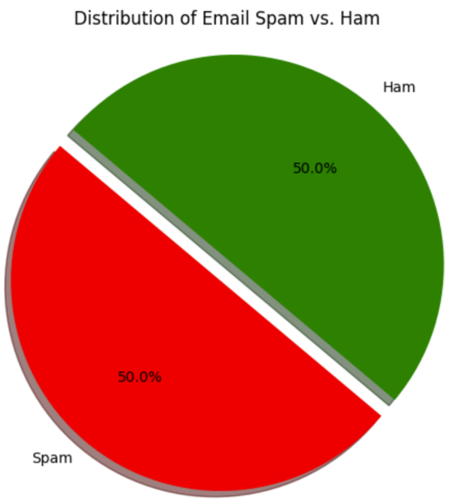
2.2 Balance data of spam and ham emails

Oversampling is a technique used to balance the number of samples between classes (in this case, between spam and ham emails) by generating new samples for the minority class (the class with fewer samples) to match the number of samples in the majority class. The process typically works:

- Identify Minority and Majority Classes,
- Choose an Oversampling Method : Random Oversampling,
- Apply Oversampling: Implement the chosen Oversampling method to create new samples for the minority class. This may involve randomly replicating existing samples or generating synthetic samples based on existing ones,



(a) Before over-sampling



(b) After over-sampling

Figure 2.2: Comparison of two data

2.3 Machine Learning approach: Tf-idf

TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

$$tf(t,d)) = \frac{n_{t,d}}{\sum_k n_{k,d}}$$

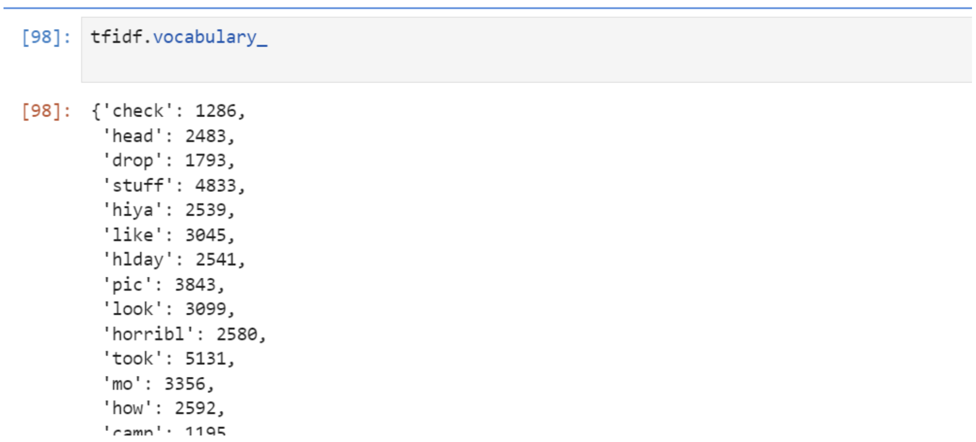
$$idf(t,D) = \log \frac{|D|}{|\{d \in D; t \in d\}| + 1} = \log \frac{|D|}{df(d,t) + 1}$$

$$tfidf(t,d,D) = tf(t,d) \times idf(t,D)$$

Where:

- $|D|$ is the number of documents in the collection.
- $df(d,t) = |\{d \in D; t \in d\}|$ is the frequency of the document $d \in D$ that the word t appears.
- $tf(t,d)$ is the frequency of the word t in the document d.
- $n_{t,d}$ is the number of times term t appears in document d.
- $\sum_k n_{k,d}$ is the total number of times all terms appear in document d.

The vocabulary of TF-IDF Vectorizer for our dataset would be roughly as below:



```
[98]: tfidf.vocabulary_

[98]: {'check': 1286,
       'head': 2483,
       'drop': 1793,
       'stuff': 4833,
       'hiya': 2539,
       'like': 3045,
       'hlday': 2541,
       'pic': 3843,
       'look': 3099,
       'horribl': 2580,
       'took': 5131,
       'mo': 3356,
       'how': 2592,
```

(a) A glance at TF-IDF vocabulary

3.1 Soft Margin Support Vector Machine

In similarity to the Perceptron Learning Algorithm (PLA), the Support Vector Machine (SVM) functions effectively only when dealing with linearly separable data between two classes. Naturally, there is an expectation for SVM to handle data that is almost linearly separable, akin to the capability demonstrated by Logistic Regression.

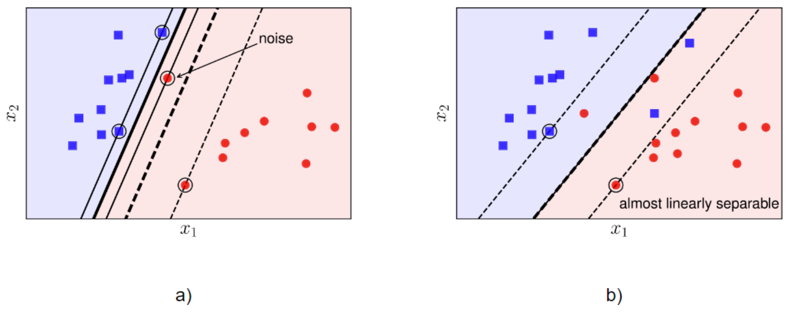


Figure 3.1: Soft margin SVM. When a) there is noise or b) the data is close to being linearly separable, pure SVM will not work effectively.

There are two cases where it is easy to see that SVM does not work effectively or even fails:

- The data is still linearly separable as in Figure 3.1.a, but there is a noisy point from the red circle class that is too close to the green square class. In this case, if using Hard Margin SVM, it will create a very small margin. Additionally, the separating hyperplane is too close to the green square class and far from the red circle class. However, if skipping this noisy point, it will get a much better margin described by the dashed lines. Hard Margin SVM is therefore considered sensitive to noise.

- The data is not linearly separable but is close to being linearly separable as in Figure 3.1.b. In this case, if using Hard Margin SVM, the optimization problem is clearly infeasible, meaning the feasible set is empty, so the SVM optimization problem becomes infeasible. However, if skipping some points near the boundary between the two classes, it can still create a fairly good separating hyperplane like the solid dashed line. The support vectors still help create a large margin for this classifier. For each point lying on the other side of the support lines (or margin lines, or boundary lines), it falls into the unsafe region. Note that the safe regions of the two classes are different, intersecting in the area between the two support lines.

the standard form of the optimization problem for Soft-margin SVM:

$$\underset{\mathbf{w},b,\boldsymbol{\xi}}{\operatorname{argmin}} \left(\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N} \xi_n^2 \right) \tag{3.1}$$

Subject to:

$$\begin{aligned} y_n(\mathbf{w}\cdot\mathbf{x}_n + b) &\geq 1 - \xi_n, & \forall n = 1,\dots,N \\ \xi_n &\geq 0, & \forall n = 1,\dots,N \end{aligned}$$

Evaluate:

If C is small, the amount of sacrifice does not significantly affect the objective function's value, and the algorithm will adjust to minimize $\|\mathbf{w}\|^2$, which means maximizing the margin. This will lead to a large $\sum_{n=1}^{N}\xi_n$. Conversely, if C is too large, to minimize the objective function's value, the algorithm will focus on reducing $\sum_{n=1}^{N}\xi_n$. In the case where C is very large and the two classes are linearly separable, we will obtain $\sum_{n=1}^{N}\xi_n = 0$. Note that this value cannot be less than 0. This means that no points need to be sacrificed, i.e., we obtain a solution for Hard Margin SVM. In other words, Hard Margin SVM is a special case of Soft Margin SVM.

The optimization problem (2) includes the appearance of slack variables ξ_n. The $\xi_n = 0$ correspond to data points lying in the safe region. The $0 < \xi_n \leq 1$ correspond to points lying in the unsafe region but still correctly classified, i.e., they still lie on the correct side of the decision boundary. The $\xi_n > 1$ correspond to misclassified points.

The objective function in the optimization problem (2) is a convex function because it is the sum of two convex functions: the norm function and the linear function. The constraints are also linear functions of $(\mathbf{w}, b, \boldsymbol{\xi})$. Therefore, the optimization problem (2) is a convex problem and can be expressed as a Quadratic Programming (QP) problem.

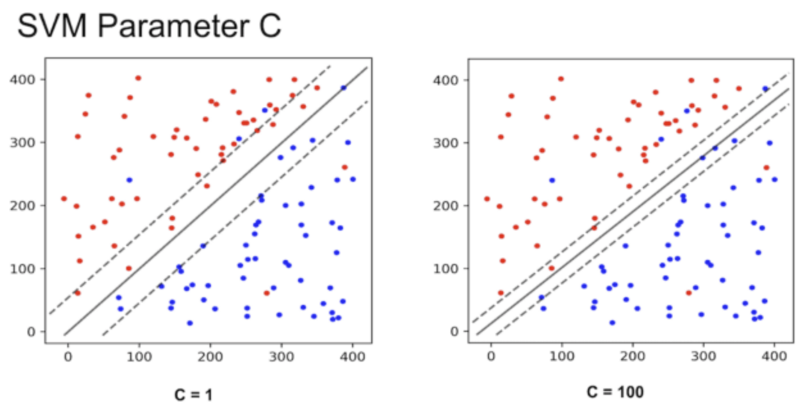


Figure 3.2: Example on the effect of C

The default model uses $C = 1.0$ and yields the accuracy on the training and test set as 99.996% and 97.75% respectively.

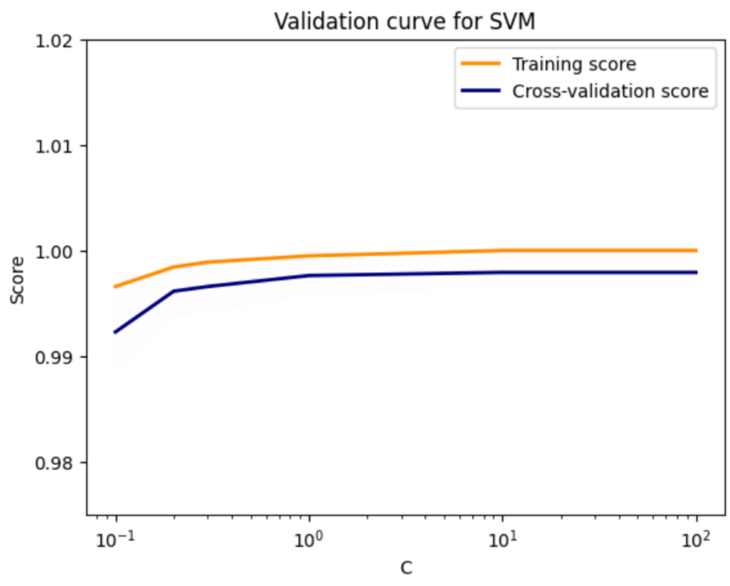


Figure 3.3: C from 0.1 to 100

We used the 10-fold cross validation schema to find the best value of C. The model seems to start overfit since C = 1.1, and a value of 1.0 should suffice to balance the running time, and slightly reduce overfitting.

The linear SVM model with C = 1.0 achieves 99.96% accuracy on the train set, and 97.75% on the test set.

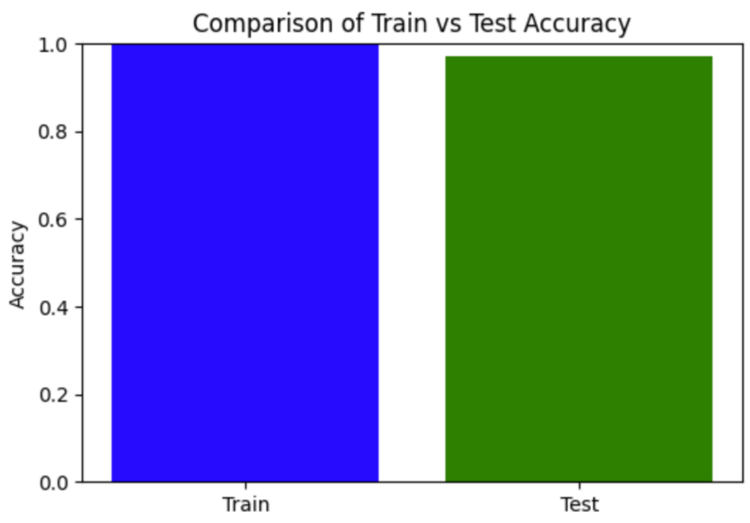


Figure 3.4: SVM model with C = 0.1

3.2 Logistic Regression

Logistic regression is a statistical method for analyzing datasets in which there are one or more independent variables that determine an outcome. The outcome is typically a binary variable (i.e., it has two possible outcomes). Logistic regression is widely used for binary classification problems in various fields such as medicine, finance,... or in this case, email classification.

3.2.1 Logistic sigmoid function

Sigmoid function as the output of this model:

$$\sigma(w^T \cdot x) = \frac{1}{1 + e^{-w^T \cdot x}} \tag{3.2}$$

where:

- w^T represents the transpose of the weight vector w.
- x is the feature vector.
- $\sigma(w^T \cdot x)$ is the logistic sigmoid function applied to $w^T \cdot x$.

The logistic function outputs values between 0 and 1, which can be interpreted as probabilities.

3.2.2 The objective function with Regularization L2

The loss function in logistic regression with L2 regularization (Ridge regularization) can be written as follows:

$$\begin{aligned} J(w,b) &= \frac{1}{N}\sum_{i=1}^{N}[-y_i \log(h_w(x_i)) - (1-y_i)\log(1-h_w(x_i))] + \frac{\lambda}{2}\|w\|^2 \\ &= \frac{1}{N}\sum_{i=1}^{N}\left[-y_i \log\left(\sigma(w^T x_i)\right) - (1-y_i)\log\left(1-\sigma(w^T x_i)\right)\right] + \frac{\lambda}{2}\|w\|^2 \end{aligned}$$

Where:

- w is the weight vector.
- N is the number of samples.
- y_i is the true label of sample i.
- x_i is the feature vector of sample i.
- $\|w\|^2$ is the L2 norm of the weight vector.
- λ is the regularization parameter.

The parameter λ in the above formula is related to C as follows:

$$\lambda = \frac{1}{C}$$

When C is small (meaning λ is large), regularization is stronger, and the model will try to make the weights smaller to prevent overfitting, even if this leads to some errors in predictions.

When C is large (meaning λ is small), regularization is weaker, and the model will try to fit the training data better, but there is a higher risk of overfitting.

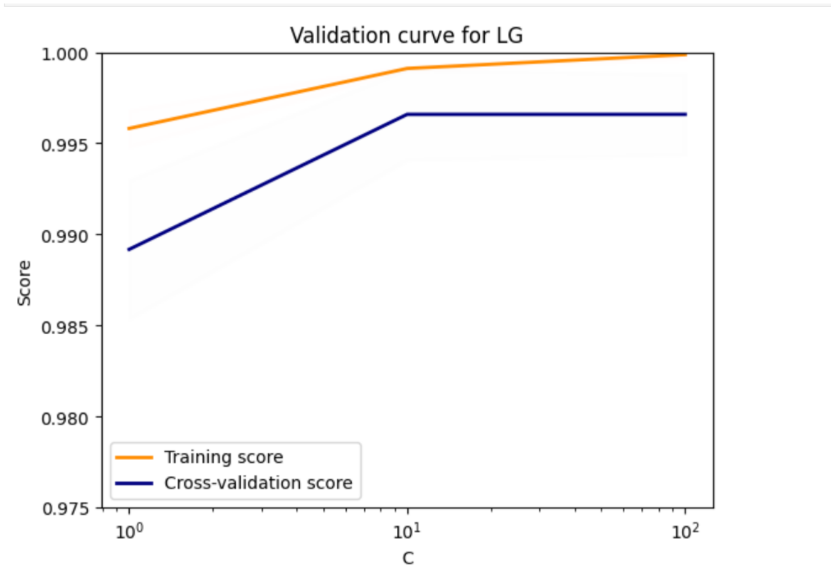


Figure 3.5: C from 1 to 100

Using the 10-fold cross validation schema to find the best value of C. The model seems to start overfit since C = 20, and a value of 10 should suffice to balance the running time, and slightly reduce overfitting.

The LG model with C = 10 achieves 99.91% accuracy on the train set, and 97.29% on the test set.

3.3 XGBoost

XGBoost (Extreme Gradient Boosting) is an algorithm based on gradient boosting, however, it is accompanied by great improvements in terms of algorithm optimization, in the perfect combination of software and hardware power, helping achieve outstanding results in both training time and memory usage.

In the first step is identifying a training set with any number of factors and y as the target variable, as well as a differentiable loss function L(y, F(x)). A loss function simply compares the actual value of the target variable with the predicted value of the target variable.

Next, initialize the XGBoost model with a constant value:

$$\hat{f}_0(x) = \arg\min_{\theta} \sum_{i=1}^{N} L(y_i, \theta) \tag{3.3}$$

Here, θ represents any arbitrary value (its value does not matter) that serves as the first value of estimation for the regression algorithm. The error of estimation with θ will be very large but will get smaller and smaller with each additive iteration of the algorithm.

Then, for all $m \in \{1, 2, 3, \ldots, M\}$, compute the gradients and Hessian matrices for the gradient boosting of the trees themselves:

$$g_i = \frac{\partial L(y_i, \hat{y}_i^{(m-1)})}{\partial \hat{y}_i^{(m-1)}} = \hat{f}_{(m-1)}(x) \tag{3.4}$$

$$h_i = \frac{\partial^2 L(y_i, \hat{y}_i^{(m-1)})}{\partial (\hat{y}_i^{(m-1)})^2} = \hat{f}_{(m-1)}(x) \tag{3.5}$$

The gradients, often referred to as the "pseudo-residuals," show the change in the loss function for one unit change in the feature value. The Hessian is the derivative of the gradient, which is the rate of change of the loss function in the feature value. The Hessian will help determine how much the gradient is changing, and therefore how much the model will change. Each of these is imperative for the gradient descent process.

Using these new matrices, another tree is added to the algorithm by completing the following optimization problem for each iteration of the algorithm:

$$\hat{\phi}_m = \arg\min_{\phi \in \Phi} \sum_{i=1}^{N} \frac{1}{2} \hat{h}_m(x_i) \left[-\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)} - \phi_m(x_i) \right]^2 \tag{3.6}$$

$$\hat{p}_m(x) = \alpha \hat{\phi}_m(x) \tag{3.7}$$

This optimization problem uses a Taylor approximation, which is necessary to be able to use traditional optimization techniques. What this is doing is estimating the point that the algorithm is at in the gradient boosting process. If the rate of change of the gradient is steep, meaning that the residuals are large, then the algorithm still needs significant change. On the other hand, if the rate of change of the gradient is flat, the algorithm is close to completion.

The model is then updated by adding the new trees to the previous model:

$$\hat{f}_{(m)}(x) = \hat{f}_{(m-1)}(x) + \hat{p}_{(m)}(x) \tag{3.8}$$

This process is repeated for every single weak learner, where each $m \in \{1, 2, \ldots, M\}$. Weak learners are utilized iteratively to improve the model's accuracy by minimizing the loss function. The final output of the XGBoost algorithm can then be expressed as the sum of each individual weak learner's predictions:

$$\hat{f}(x) = \sum_{m=0}^{M} \hat{f}_m(x) \tag{3.9}$$

The initial accuracy on the test set of XGBoost (with default values for parameters) is 97.09%. After experimenting with tuning parameters such as *max_ depth*, *learning_ rate*, *n_ estimators*, *gamma*, and *subsample*, we observed that *learning_ rate*, *max_ depth*, and *subsample* have the most significant impacts on the accuracy of the XGBoost classifier. Narrowing down the scope and range of these parameters, we employed GridSearch with 10-fold cross-validation to find the best combination.

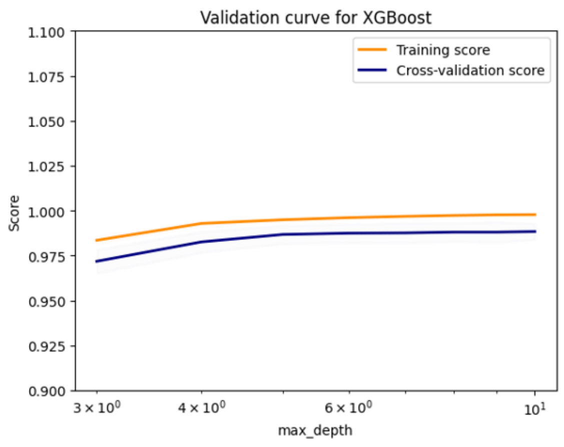


Figure 3.6: max_depth from 3 to 10

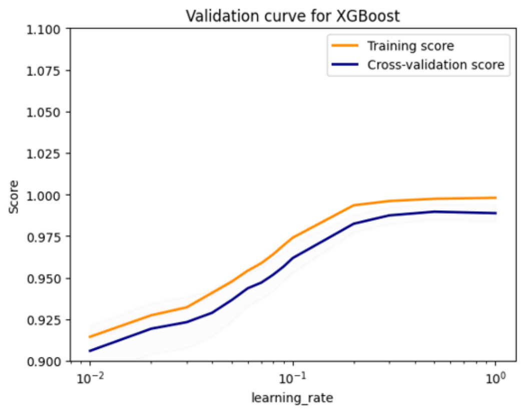


Figure 3.7: learning_rate from 0.01 to 1

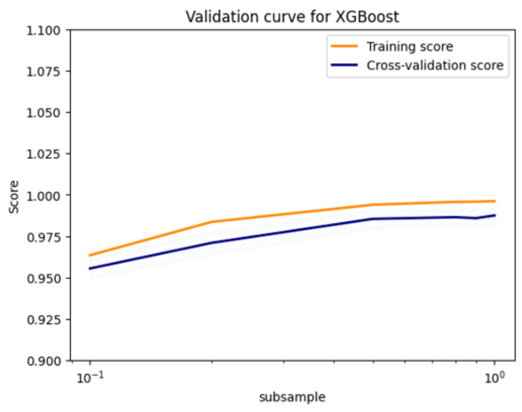


Figure 3.8: subsample from 0.1 to 1

The best combination (learning_rate = 0.5, max_depth = 5, and subsample = 1.0) yields an accuracy of 99.78% for the training set and 96.43% for the test set.

3.4 Random Forest

Random forest is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputting the class that is the mode of the classes (classification) of the individual trees. Each tree in the random forest is trained on a random subset of the training data and a random subset of features. The randomness introduced during both the bootstrap sampling of the training data and the feature selection helps to decorrelate the trees and reduce overfitting.

3.4.1 Random Forest Algorithm Steps

1. **Bootstrap Sampling**:
   - Randomly select a subset of the training data with replacement (bootstrap sampling). Let N be the size of the training set, then for each tree:

     $$\text{Sampled\_data}_k = \{(x_i, y_i)\} \quad \text{where } x_i \text{ is a feature vector and } y_i \text{ is its label.}$$

2. **Feature Subset Selection**:
   - Randomly select a subset of features (columns) from the training data. Let M be the total number of features, then for each tree:

     $$\text{Selected\_features}_k = \{f_j\} \quad \text{where } j \in \{1, 2, ..., M\}.$$

3. **Grow Decision Trees**:
   - Construct a decision tree using the bootstrap sample and the selected features. Let Tree_k denote the k^{th} decision tree.

     $$\text{Tree}_k = \text{BuildTree}(\text{Sampled\_data}_k, \text{Selected\_features}_k)$$

4. **Ensemble Decision Trees**:

   - Repeat steps 1-3 to create multiple decision trees. Let K be the number of trees in the forest. Then, the output for a given input x is:

     $$\text{Output}(x) = \text{mode}\left(\text{Predictions}(x; \text{Tree}_1), \text{Predictions}(x; \text{Tree}_2), ..., \text{Predictions}(x; \text{Tree}_K)\right).$$

   - Where $\text{Predictions}(x; \text{Tree}_k)$ represents the predicted class of input x by the k^{th} decision tree.

3.4.2 Predict and evaluate model

Precision: 0.9923664122137404
Recall: 0.87248322147651
Accuracy Score: 0.9844961240310077
F1 Score: 0.9285714285714286

3.4.3 Optimization and Performance

The Random Forest model was optimized using GridSearchCV with 10-fold cross-validation. The best parameters found were:

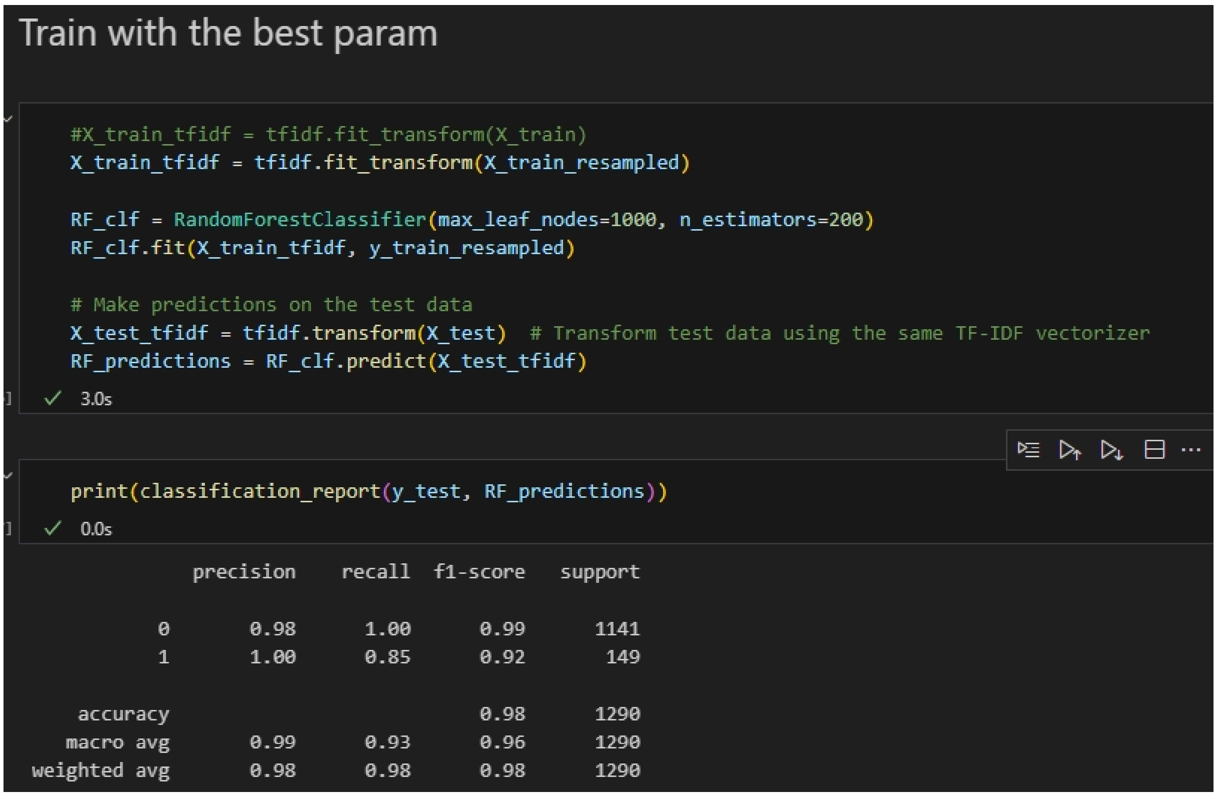
- **max_leaf_nodes**: 10000
- **n_estimators**: 200

The $n\_estimators$ parameter in Random Forest determines the number of trees in the ensemble. It plays a crucial role in balancing model complexity and overfitting.

- **Small** $n\_estimators$: When $n\_estimators$ is small, regularization is stronger. The model tends to prioritize smaller weights to prevent overfitting, even if this results in some errors in predictions. This encourages a simpler model structure and helps in generalization to unseen data.
- **Large** $n\_estimators$: Conversely, when $n\_estimators$ is large, regularization is weaker. The model tries to fit the training data more closely, potentially leading to overfitting. While this may improve performance on the training data, it also increases the risk of poor generalization to new data.

Choice of $n\_estimators$ with Cross-Validation

To determine the optimal value of $n\_estimators$, a 10-fold cross-validation schema was employed. This approach assesses model performance across different subsets of the data, helping to identify the best value of $n\_estimators$ that balances model complexity and performance.


Train with the best param

```
#X_train_tfidf = tfidf.fit_transform(X_train)
X_train_tfidf = tfidf.fit_transform(X_train_resampled)

RF_clf = RandomForestClassifier(max_leaf_nodes=1000, n_estimators=200)
RF_clf.fit(X_train_tfidf, y_train_resampled)

# Make predictions on the test data
X_test_tfidf = tfidf.transform(X_test)  # Transform test data using the same TF-IDF vectorizer
RF_predictions = RF_clf.predict(X_test_tfidf)
```
✓ 3.0s

```
print(classification_report(y_test, RF_predictions))
```
✓ 0.0s

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1141
           1       1.00      0.85      0.92       149

    accuracy                           0.98      1290
   macro avg       0.99      0.93      0.96      1290
weighted avg       0.98      0.98      0.98      1290
```

Figure 3.9: classification report

Observation and Recommendation

Through the cross-validation process, it was observed that the model starts to exhibit signs of overfitting when $n\_estimators$ reaches 200. Consequently, a value of 100 was deemed sufficient to balance computational efficiency and reduce overfitting tendencies. This choice aims to maintain a reasonable model complexity while achieving satisfactory performance on both training and validation data.

The Random Forest model with **n_estimator** = 200 achieves 99.98% accuracy on the train set, and 97.67% on the test set.

3.4.4 Conclusion

The Random Forest model demonstrates high performance on the given dataset, achieving excellent accuracy and precision-recall balance. The optimized model with tuned hyperparameters ensures robustness and generalization to unseen data.

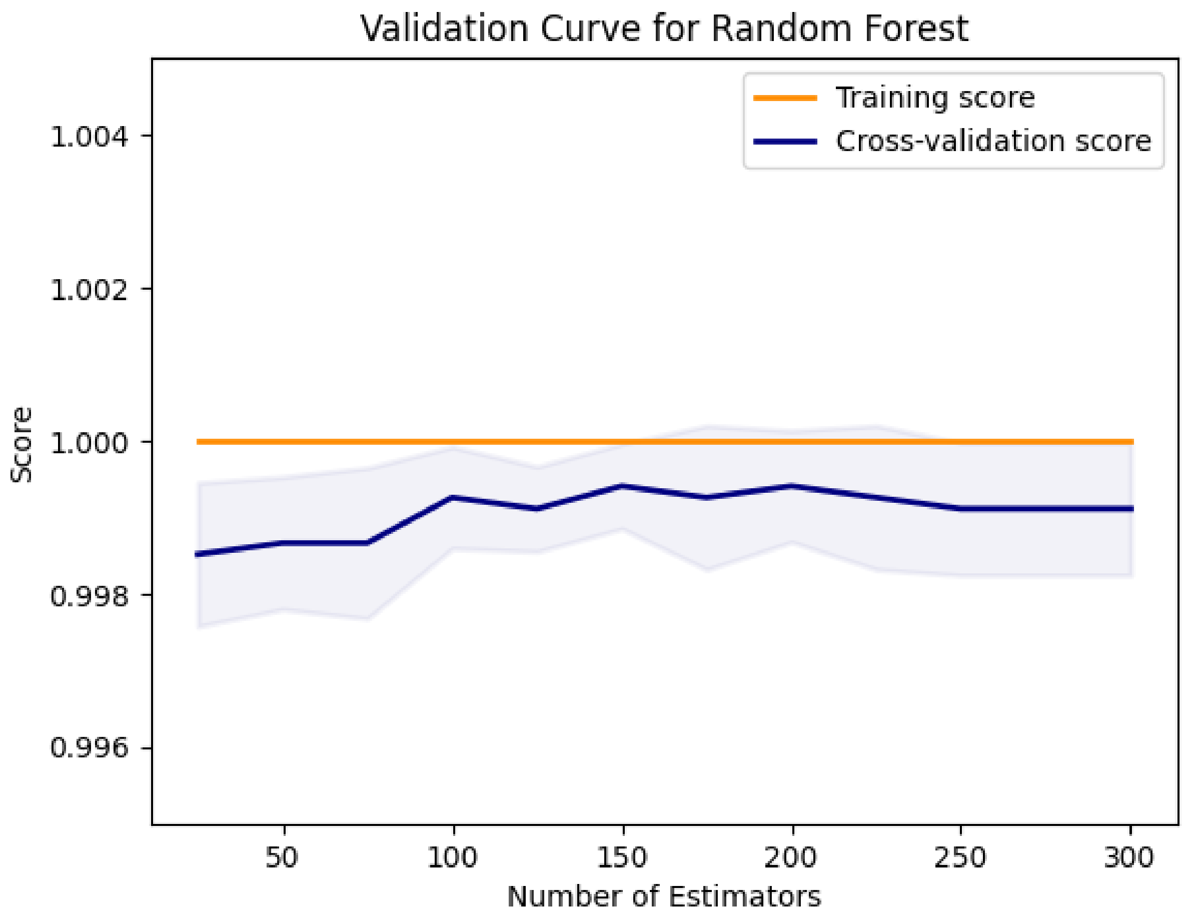


Figure 3.10: validation curve

4.1 Test Accuracy

We chose accuracy as the metric to measure the performance of our Supervised Machine Learning algorithms. Below is the plot of training and test accuracy of Machine Learning models. It can be seen that all three models achieved quite good accuracy of about 97 percent in average.

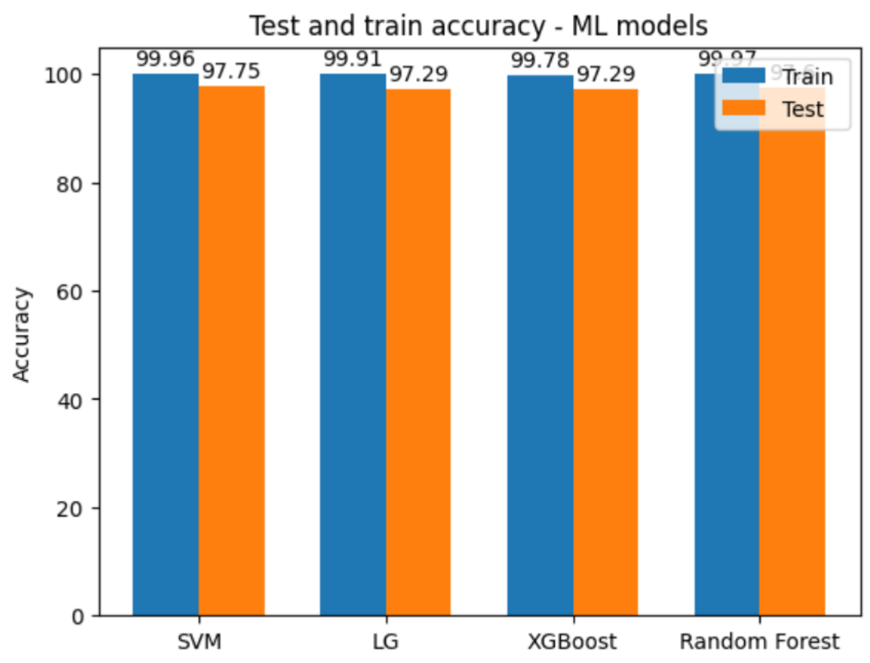


Figure 4.1: compare 3 models

4.2 Confusion Matrix

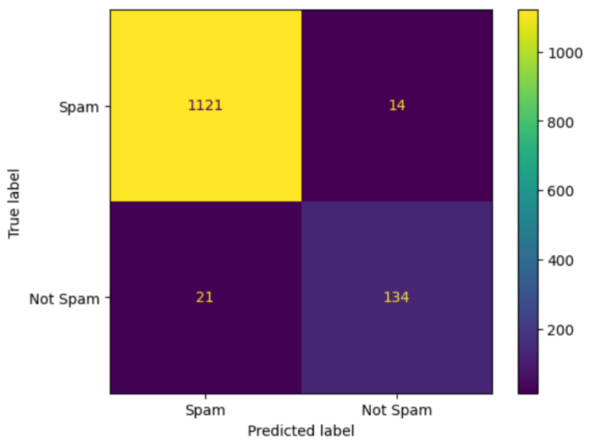


Figure 4.2: Confusion Matrix of LG

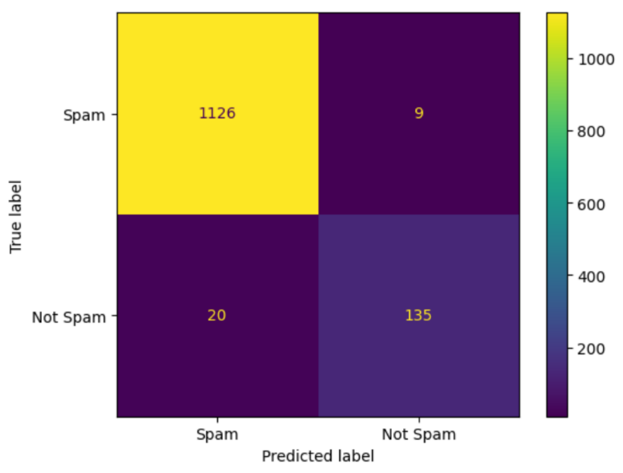


Figure 4.3: Confusion Matrix of SVM

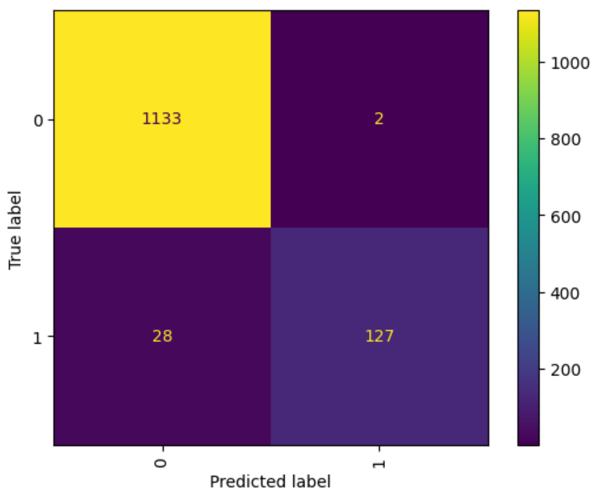


Figure 4.4: Confusion Matrix of Random Forest

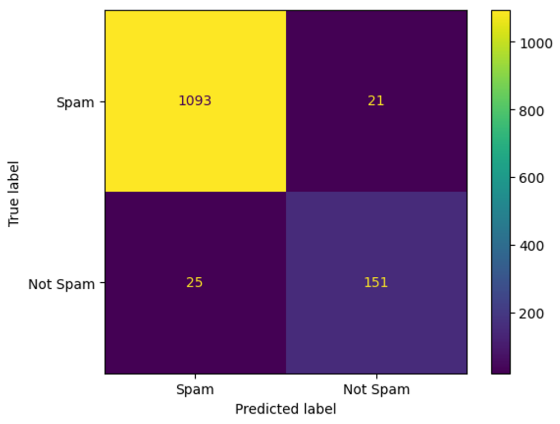


Figure 4.5: Confusion Matrix of XGBoost

When evaluating the confusion matrices of the three models: SVM, Logistic Regression, Random Forest and XGBoost for email spam classifier we can see that: All four models have a certain number of misclassified instances, represented by the false positive and false negative values in the confusion matrices. However, the degree of these misclassifications varies across the models. The Random Forest model has a higher false positive rate, meaning it incorrectly predicted many negative instances as positive. On the other hand, the XGBoost model has a higher false negative rate, missing many actual positive instances.

CHAPTER 5. CONCLUSION

In conclusion, the developed spam classifier shows significant promise in reducing the volume of spam emails received by users, thereby improving their overall email experience and productivity.From data from file csv, we proposed many different algorithms to find the best sentiment classifier for our problem. In terms of Machine Learning models, Linear Support Vector Machine has outperformed other algorithms with an accuracy of 97.75

Although the Linear SVM model has achieved remarkable performance, it is essential to acknowledge that no single model can be considered a universal solution. The choice of the most appropriate algorithm depends on various factors, including the characteristics of the dataset, the specific requirements of the problem, and the trade-offs between model complexity, interpretability, and computational resources. Moving forward, it would be beneficial to explore ensemble techniques that combine the strengths of multiple models, potentially leading to even better performance and robustness. Additionally, continuous monitoring and updating of the spam classifier are crucial as spam patterns evolve over time, and new strategies emerge to circumvent existing filters.

Furthermore, it is essential to address the ethical considerations and potential biases that may arise from the use of automated spam filtering systems. Transparency, user control, and the ability to appeal misclassifications should be integrated into the system to ensure fairness and accountability. Overall, the successful development of an effective spam classifier represents a significant step towards enhancing user experience and productivity in the realm of email communication. However, ongoing research, adaptation, and a commitment to ethical practices are necessary to maintain the reliability and trustworthiness of such systems in the long run.

References

[1] YONATHA WIJAYA, Komang Dhiyo; KARYAWATI, Anak Agung Istri Ngurah Eka. The Effects of Different Kernels in SVM Sentiment Analysis on Mass Social Distancing. JELIKU (Jurnal Elektronik Ilmu Komputer Udayana), [S.l.], v. 9, n. 2, p. 161-168, nov. 2020. ISSN 2654-5101

[2] https://www.youtube.com/@thanquangkhoat4070

[3] https://dominhhai.github.io/vi/2017/12/ml-logistic-regression/

[4] https://machinelearningcoban.com/