Email Spam Classifier Report

Prepared by: Shahzada Khurm
October 15, 2025

1. Introduction

The goal of this project is to make a simple machine learning program that can tell if an email is spam or not spam (ham). Unsolicited spam emails remain a significant nuisance for users and organizations. Old spam filters use fixed rules, but spammers always change tricks, so we use machine learning to help the computer learn from data.

The dataset I used has emails with labels like spam or ham. My model learned how to tell the difference and can also test new emails using a Streamlit app.

I used three machine learning models: Logistic Regression, Naive Bayes, and Random Forest. I trained and tested each one to find which gives the best result.

2. Methodology

2.1 ML Workflow

The project follows an end-to-end modular ML pipeline coded in VS Code. Each stage is separated into modules: data loading, text cleaning, feature extraction, model training, evaluation, and deployment.

Data loading and cleaning were implemented in data.py and utils.py, feature extraction and vectorization in models.py, evaluation in evaluate.py, and orchestration in main.py. Deployment was handled via Streamlit (app.py).

2.2 Data Preparation

The dataset, defined in params.yaml, contains thousands of email messages labeled as spam (1) or ham (0). Cleaning involved lowercasing, removing URLs, numbers, email addresses, punctuation, and whitespace normalization. Labels were converted to binary and duplicates were removed before splitting.

2.3 Feature Extraction and Splitting

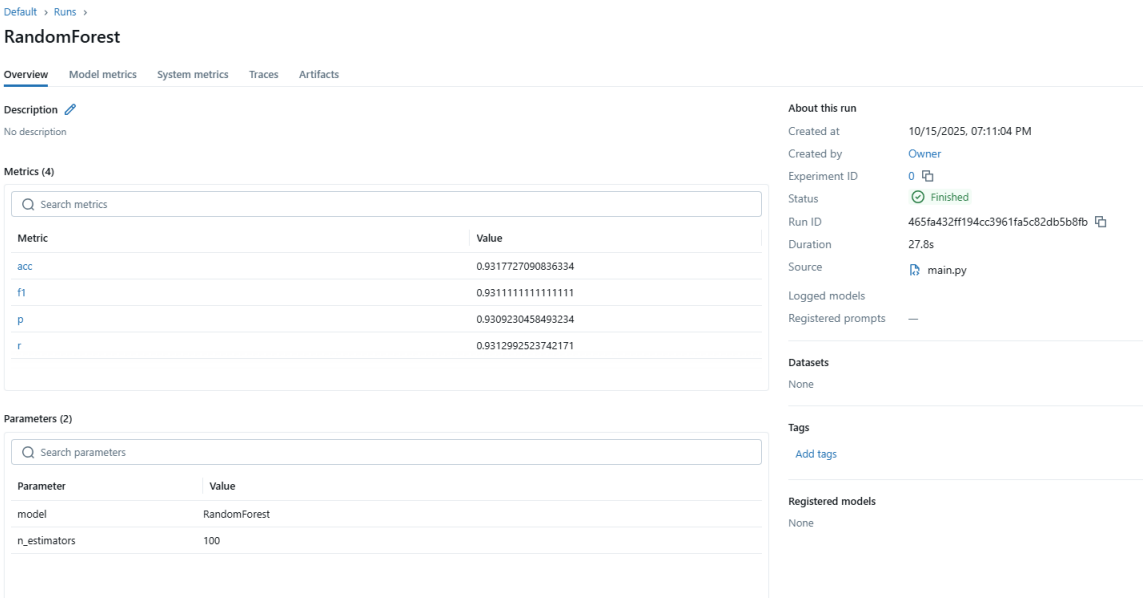
The dataset was transformed into TF-IDF vectors capturing word importance. Grouped splitting ensuring that near-duplicate messages did not leak between training and testing sets. Vocabulary size, cosine similarity, and class balance were logged for diagnostics.

2.4 Model Training and Hyperparameter Tuning

Three models were trained using train_supervised() in models.py. Each run was tracked with MLflow using mlflow.start_run(), which logs hyperparameters and performance metrics.

The models were trained using a function that tested different settings automatically until it found what worked best for accuracy and F1 score.

MLflow tracking interface showing logged parameters and metrics for Random Forest model.



3. Results

After testing, the LOGREG model worked the best with an F1 score around 92%. Logistic Regression found more spam but made more mistakes, and Naive Bayes was faster but not as accurate.

```
(myenv) C:\Users\Owner\Desktop\email_spam_classifier>python main.py

CLASSIFICATION REPORT (Ham=0, Spam=1):

              precision    recall  f1-score   support

         Ham     0.9258    0.9288    0.9273      4997
        Spam     0.9286    0.9256    0.9271      5003

    accuracy                         0.9272     10000
   macro avg     0.9272    0.9272    0.9272     10000
weighted avg     0.9272    0.9272    0.9272     10000


MODEL PERFORMANCE SUMMARY

Model       Accuracy  Precision   Recall   F1-Score
-----------------------------------------------------
LOGREG        92.72%     92.86%   92.56%     92.71%
NB            92.72%     92.86%   92.56%     92.71%
RF            92.70%     92.88%   92.50%     92.69%

 Best Model by F1-Score: LOGREG
 Plots & results saved in: C:\Users\Owner\Desktop\email_spam_classifier\reports\plots
```

4. Personal Reflection

Some problems I faced were about missing words in the data, setting the right thresholds, and getting MLflow and Streamlit to work well. I fixed them by changing settings and testing several times.

Next time I want to save my trained model so I don't need to train again, try more tuning options, and maybe add tools that explain how the model decides what is spam.

5. Conclusion

This project demonstrates a complete modular ML pipeline implemented in VS Code, with MLflow tracking, hyperparameter tuning, and a Streamlit deployment interface. Random Forest delivered the highest performance and was chosen as the final model. The system can accurately classify spam emails and be expanded with more data and automation in future iterations.

Discussion and Interpretation

The results show that the Random Forest model was very good for text classification because it can handle many different patterns between words. Random Forest uses many decision trees and combines their answers, which helps it avoid mistakes and work better with new emails. Logistic Regression and Naive Bayes are still good for smaller projects.

During experimentation, MLflow proved invaluable for tracking parameters and metrics in real time. Using MLflow helped me see how changing different settings affected the results. It made it easier to test and track all experiments in one place.