PROFESSIONAL TRAINING PROJECT REPORT

TOPIC : CLOUD COMPUTING

AI-DRIVEN EMAIL SPAM CLASSIFIER: A MACHINE LEARNING APPROACH WITH AWS DEPLOYMENT

SELF CERTIFICATE

This is to certify that the project report entitled **"EMAIL Spam Classifier: A Machine Learning Approach with AWS Deployment"** is an original work carried out by **Harsh Kumar** in partial fulfilment of the requirements for the award of the **Bachelor of Technology (B.Tech)** degree.

I affirm that this project is a result of my independent research and implementation, and to the best of my knowledge and belief, it has not been submitted previously for the award of any degree or diploma at any institution. Furthermore, I acknowledge my understanding of the guidelines related to the **B.Tech project and project report** and confirm adherence to the prescribed academic standards.

_____

Name – Harsh Barnwal
Roll No. – 10330522027
Reg. No. - 221030110605 OF 2022-23

ACKNOWLEDGEMENT

We take immense pleasure in presenting our project, **"EMAIL Spam Classifier: A Machine Learning Approach with AWS Deployment."**

We express our heartfelt gratitude to **Debasish Sahoo Sir** for his invaluable guidance, continuous support, and encouragement throughout this project. His insights and expertise have been instrumental in shaping our work.

We also extend our sincere appreciation to our beloved parents for their unwavering support, thoughtful feedback, and motivation during the entire project journey. Their encouragement has been a driving force in our success.

Furthermore, we would like to thank all our teachers and friends for their constant support and motivation, which kept us moving forward. Their contributions, both directly and indirectly, have played a crucial role in the completion of this project.

We are deeply grateful to each and every one who has helped us along this journey.

NAME	ROLL NO.	REGISTRATION NO.	SIGNATURE
Harsh Barnwal	10330522027	221030110605	
Harsh Kumar	10330522028	221030110606	
Sovana Majhi	10330522059	221030110637	
Dipak Kumar	10330522023	221030110601	
Artisha Banerjee	10330522014	221030110592	

ABSTRACT

Spam emails are a significant problem in digital communication, leading to security threats, financial fraud, and productivity loss. The increasing volume of spam emails necessitates the development of **robust automated filtering solutions**. Traditional rule-based spam filters struggle to keep up with the evolving tactics used by spammers, making **machine learning-based classifiers** a more effective approach. This project focuses on developing an **email spam classifier using machine learning and Natural Language Processing (NLP) techniques**. The classifier processes email text, extracts relevant linguistic features, and applies a trained machine learning model to determine whether an email is spam or non-spam.

The system utilizes **TF-IDF vectorization and Naïve Bayes classification**, known for their efficiency in text-based classification tasks. Additionally, **Support Vector Machines (SVM) and Logistic Regression** models are explored to compare performance.

For real-time accessibility, the classifier is deployed on **AWS EC2** using a **Flask API**, allowing users to interact with the model through a web-based interface. The deployment on AWS ensures scalability, reliability, and seamless user experience.

By automating spam detection, this project contributes to improving **email security, filtering efficiency, and user productivity**. It highlights the effectiveness of **machine learning algorithms** in addressing real-world cybersecurity challenges and paves the way for future advancements in spam classification.

INTRODUCTION

In today's digital world, email spam poses a major challenge by flooding in boxes with unwanted messages, phishing attacks, and fraudulent content. Traditional rule-based spam detection methods fail to keep up with evolving spam patterns. **Machine learning** provides a powerful alternative by automatically learning patterns from email text and improving classification accuracy.

This project aims to develop an **AI-driven spam classifier** capable of accurately distinguishing between spam and non-spam emails. The classifier is built using **NLP and machine learning techniques**, leveraging algorithms such as **Naïve Bayes, Support Vector Machines (SVM), and Logistic Regression**. The model is deployed on **AWS EC2** for accessibility and scalability.

Project Objectives

- Develop an **AI-driven email spam classifier** with high accuracy.
- Use **NLP techniques** to preprocess and analyse email text.
- Deploy the model on **AWS EC2** with a Flask API for real-time predictions.

Technology Stack

- **Backend:** Flask (Python) for API development and model serving.
- **Machine Learning:** Scikit-learn, NLTK, TensorFlow (for model training).
- **Deployment:** Hosted on AWS EC2 with proper security configurations.

KEYWORDS

- **Email Spam Detection**
- **Machine Learning**
- **Natural Language Processing (NLP)**
- **Naïve Bayes Classifier**
- **Support Vector Machines (SVM)**
- **Logistic Regression**
- **Text Classification**
- **Feature Extraction**
- **AWS EC2 Deployment**
- **Flask API**

HEADINGS

- **Implement Machine Learning Techniques for Accurate Email Spam Classification**
- **Develop an Interactive and User-Friendly Interface Using Flask**
- **Enhance Prediction Accuracy by Analysing Email Content Patterns**
- **Deploy a Scalable Spam Classification System Using Flask and AWS EC2**

LITERATURE SURVEY

Traditional spam filters relied on rule-based approaches, such as keyword filtering and blacklisting, which were ineffective against evolving spam techniques. The introduction of machine learning-based classifiers significantly improved spam detection accuracy.
Several algorithms have been used in spam classification, including:

- Naïve Bayes: Works well for text classification and is computationally efficient.
- Support Vector Machines (SVM): Effective in handling high-dimensional data.
- Deep Learning Models: More recent approaches use LSTMs and Transformers but require high computational resources.

This project implements a Naïve Bayes classifier due to its simplicity and effectiveness in spam detection.

DATA SOURCES AND APPROACH

Data Sources

- The dataset used in this project is the SMS Spam Collection dataset, which is publicly available on Kaggle. This dataset contains 5,572 labelled email messages categorized as spam or non-spam (ham).
- The dataset includes a mix of formal, informal, and marketing emails, making it a comprehensive resource for training the spam classification model.
- Additional datasets from public repositories and academic sources were reviewed to understand variations in spam characteristics across different domains.

Approach

1. **Data Preprocessing:**
    - Text cleaning: Removing stop words, special characters, and redundant whitespace.
    - Tokenization: Splitting text into individual words or phrases.
    - Feature Extraction: Using TF-IDF vectorization to convert text into numerical form.
2. **Model Training:**
    - Multiple classification models were tested, including Naïve Bayes, Support Vector Machines (SVM), and Logistic Regression.
    - The models were evaluated using accuracy, precision, recall, and F1-score to determine the most effective approach.
    - Naïve Bayes was selected as the final model due to its superior performance in spam detection.
3. **Deployment Strategy:**
    - A Flask-based API was developed to serve the model and process user input.
    - The model and API were deployed on AWS EC2, ensuring scalability and accessibility.
    - The web interface allows users to enter email content and receive instant classification results.
4. **Performance Optimization:**
    - Implemented hyperparameter tuning to enhance model accuracy.
    - Optimized API request handling for faster response times.
    - Ensured secure communication between the frontend and backend components.

MATERIALS

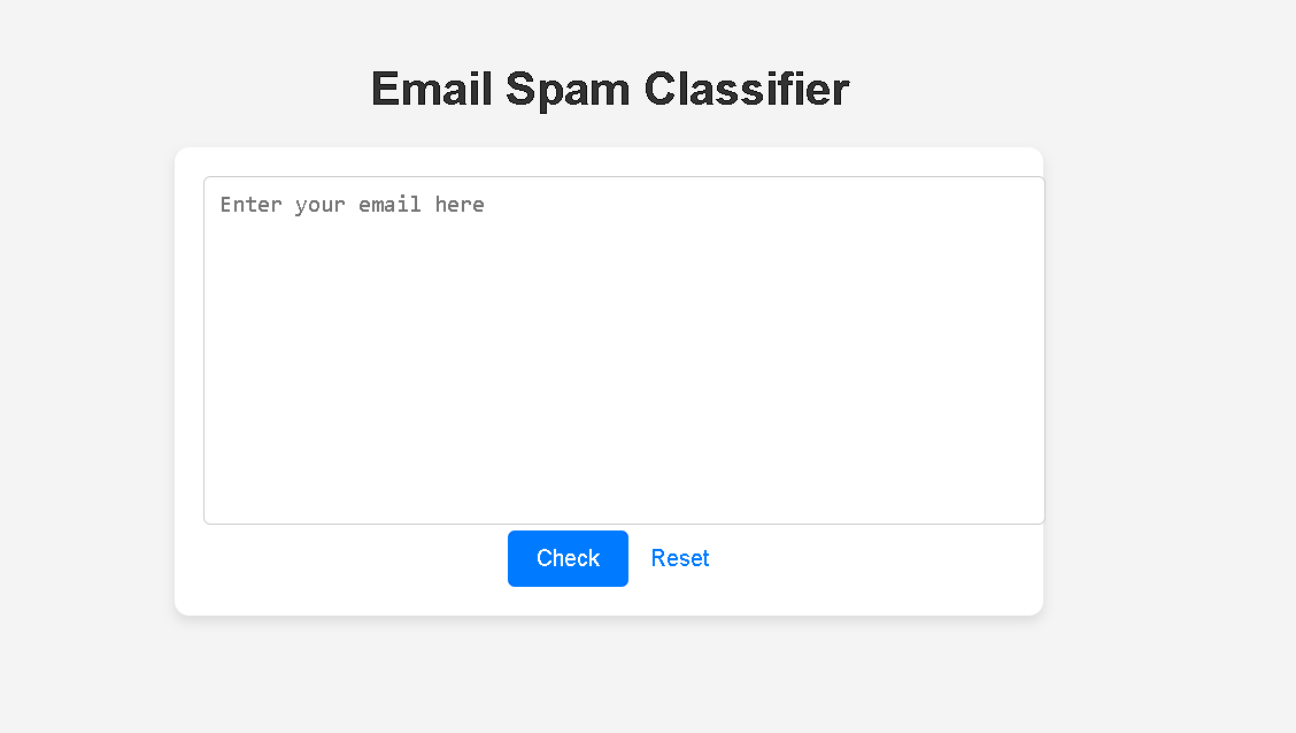
Datasets:

- SMS Spam Collection Dataset (Kaggle) - Used for model training and evaluation**.**
- Additional spam email datasets from public repositories for extended analysis.

Tools & Technologies:

- Programming Language: Python
- Machine Learning Libraries:
    - Scikit-learn (for model training and evaluation)

    - NLTK (for text processing and NLP tasks)
    - Pandas & NumPy (for data manipulation)
- **Model Training & Development:**
    - Jupyter Notebook / Google Colab for model experimentation
    - Flask for API development
- **Deployment Environment:**
    - AWS EC2 (for hosting the Flask API and ML model)
    - Docker (optional, for containerized deployment)
    - GitHub (for version control and collaboration)
- **Security & Performance Enhancements:**
    - HTTPS (for secure API communication)
    - Load balancing (future scalability consideration)



This image showcases the look and feel of our project, offering a glimpse into its design and functionality. (Image extracted from project)

PROPOSED SYSTEM

The proposed system aims to implement a machine learning-based spam classification model that accurately detects spam emails and provides a user-friendly API for seamless interaction. The system includes the following components:

1. **Preprocessing Module**: Cleans and transforms email text for model input.
2. **Machine Learning Model:** A Naïve Bayes classifier trained to distinguish spam from non-spam emails.
3. **Flask-based API**: Provides an interface for real-time email classification.
4. **Deployment on AWS EC2**: Ensures scalability and availability for multiple users.

PROPOSED WORK

The proposed system focuses on deploying a machine learning-based email spam classification model using cloud computing technologies. The key objectives and steps involved in this project are detailed below:

1. Model Hosting & Backend Development

- The backend serves as the core component of the system, where the trained Naïve Bayes-based email spam classification model is hosted. This backend is built using Flask, a lightweight Python-based web framework known for its efficiency in handling API requests.
- Key Responsibilities of the Backend:
- Processing API Requests: The backend exposes REST API endpoints that allow the frontend or other systems to send email text for spam prediction.
- Running the ML Model: When a request is received, the backend loads

the pre-trained Naïve Bayes model and processes the input data to generate a spam classification result.

- Returning Predictions: The backend formats the prediction results (e.g., Spam or Not Spam) and sends them back as a response to the API call.
- Deployment on AWS EC2
- Once the backend is developed and tested, it is deployed on AWS EC2, ensuring stability and scalability for handling multiple API requests

2. Frontend Development & Cloud Hosting

- The frontend serves as the user interface (UI) for interacting with the spam classification model. It is developed using HTML, CSS, and JavaScript, ensuring a responsive and user-friendly experience.
- Key Responsibilities of the Frontend:
- User Input Handling:
- The interface allows users to enter email content.
- Forms validate user input before sending requests to the backend.
- API Communication:
- The frontend sends email text to the Flask backend via REST API calls.
- Once the backend processes the data and returns the classification, the frontend displays the prediction results.
- Dynamic UI & Data Visualization:
- The application updates dynamically without requiring page reloads, enhancing the user experience.
- Additional charts and graphs can be added to visualize spam trends.
- Cloud Hosting on AWS EC2
- The frontend is hosted on AWS EC2, providing a scalable and cost-effective solution for static website hosting.

3. Cloud-Based Deployment & API Integration

- Cloud-based deployment ensures users can access predictions in real-time without requiring local installations. This is achieved through API integration between the frontend and the Flask backend, allowing seamless communication.
- Key Responsibilities:
- REST API Development:
- The backend provides API endpoints for email text classification.

- API responses are structured for efficient integration with various frontend applications.
- Efficient Cloud-Based Model Deployment:
- Hosting the backend on AWS EC2 ensures reliability and high availability.
- Future Scalability Considerations:
- Migration to AWS Lambda for a fully serverless backend.
- Implementing load balancing and auto-scaling on AWS for high-traffic handling.
- Enhancing security by integrating OAuth-based authentication**.**

4. System Scalability & Performance Optimization

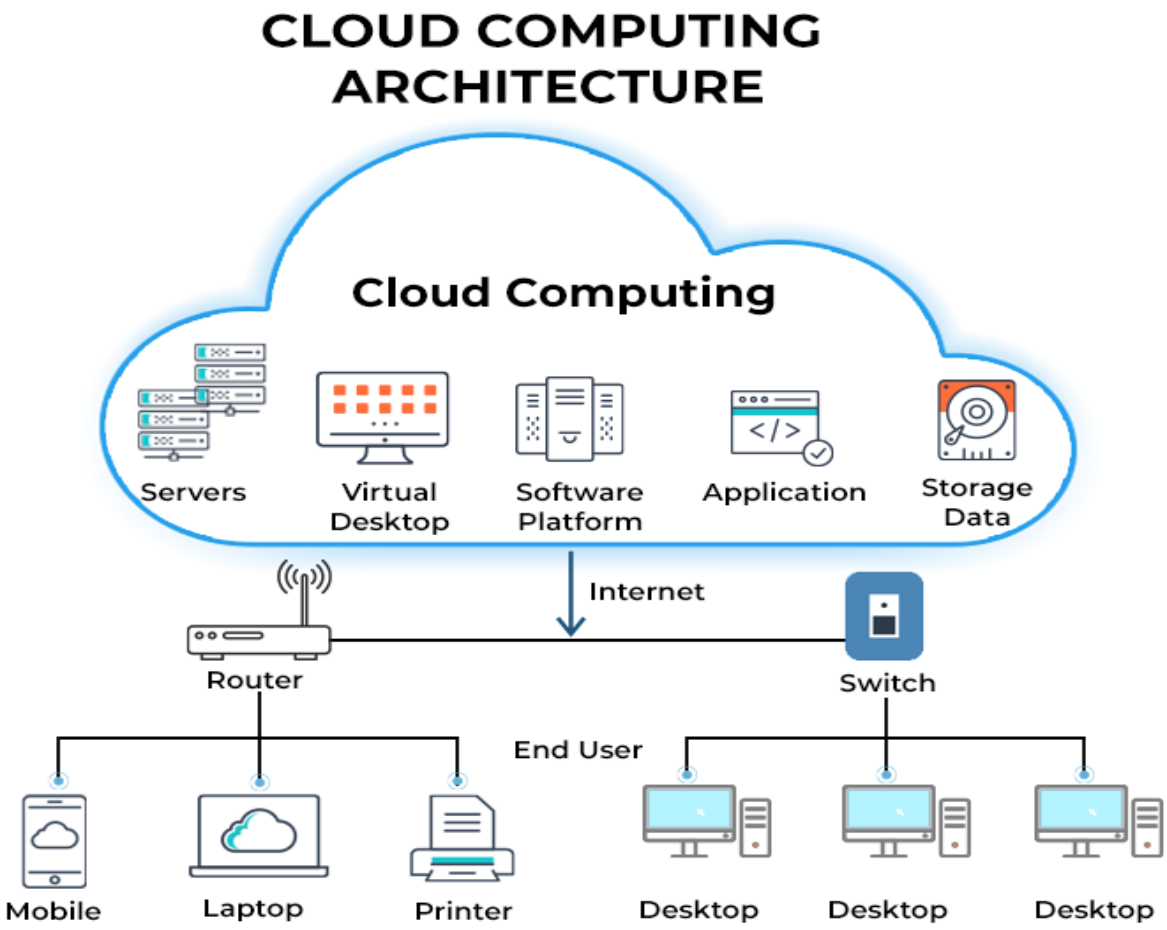
- Cloud-based deployment is designed to handle multiple API requests efficiently while ensuring fast response times and reliability.
- Key Areas of Optimization:
- Optimized API Routing & Backend Configuration:
- Structured API endpoints to minimize latency.
- Implemented caching mechanisms (e.g., Redis) to reduce redundant computations.
- Minimizing Model Inference Time:
- Preloading the Naïve Bayes model in memory to reduce request processing time.
- Optimizing request handling to reduce API response times.
- Load Balancing & High Availability Considerations:
- Deploying the backend with AWS Elastic Load Balancer (ELB) for distributing incoming traffic.
- Implementing auto-scaling policies to increase or decrease compute resources dynamically based on demand.
- **Future Enhancements:**
  - ☑ Implement caching mechanisms for frequently accessed data.
  - ☑ Use AWS CloudFront (CDN) to improve frontend performance.
  - ☑ Optimize database queries to reduce API request latency.
  - ☑ Monitor system performance using AWS CloudWatch to detect performance bottlenecks early.

5. Security & Data Privacy

- Security and data privacy are critical aspects of cloud deployment,

ensuring that user data remains protected, API endpoints are secured, and system vulnerabilities are minimized.

- Security Measures Implemented:
- API Authentication: Implementing JWT-based authentication to secure API requests.
- Data Encryption: Ensuring all sensitive user data is encrypted before storage.
- Firewall & Access Controls: Configuring AWS Security Groups and firewall rules to prevent unauthorized access.
- Regular Security Audits: Conducting periodic security audits to identify vulnerabilities and patch them promptly.
- By implementing these strategies, the email spam classification system will be highly efficient, scalable, and secure, ensuring users can benefit from real-time spam detection with minimal risks.
- AWS Lambda in the future.
- Dock erization can be implemented to ensure smooth deployment across multiple cloud platforms.
- API rate-limiting and authentication measures can be added to enhance security and prevent misuse.



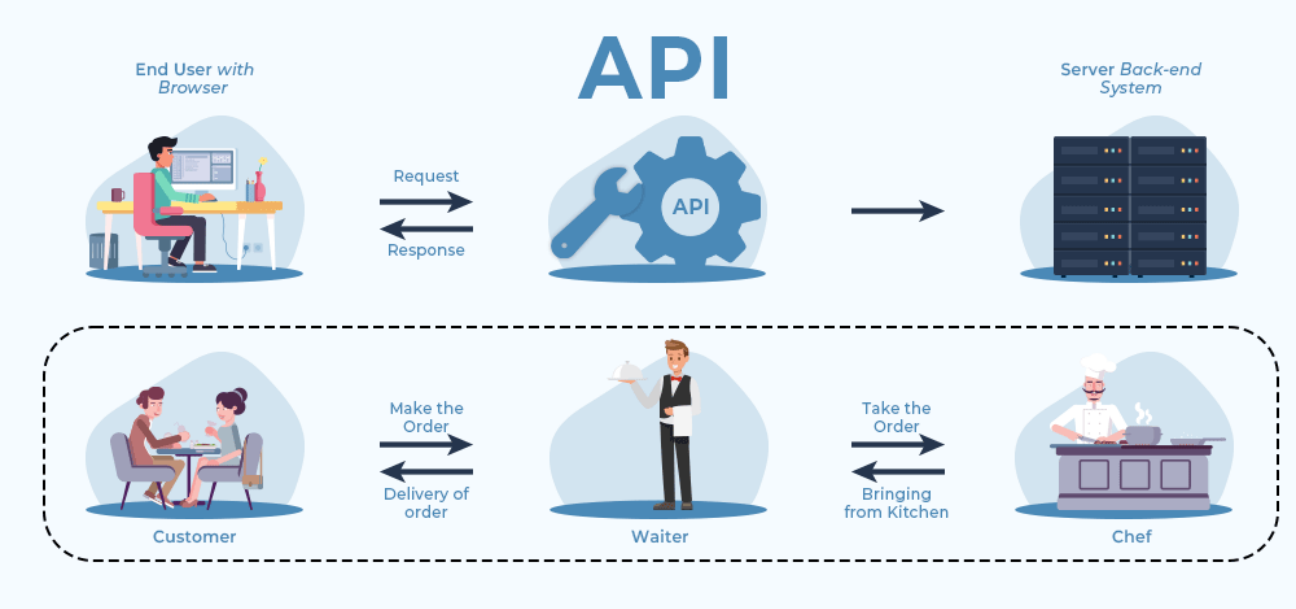
Why Use REST API for Integration?

☑ Scalability: API-based architecture allows multiple users to interact with the model simultaneously.

☑ Platform Independence: The backend can serve multiple frontends (React Web App, Mobile App, or third-party systems).

☑ Security: API requests are secured using authentication mechanisms, ensuring only authorized users can access predictions.

☑ Real-Time Processing: API calls allow predictions to be generated instantly, providing businesses with quick insights.



System Architecture Diagram & Explanation

The architecture of the Email Spam Classification System consists of multiple components that work together to ensure seamless spam detection. The system is designed for scalability, security, and efficiency in processing large volumes of email data.

System Components:

1. **User Interface (Frontend):**
   - The user enters email text via a simple web interface.
   - Developed using HTML, CSS, and JavaScript, hosted on AWS EC2 for high availability.
   - Sends requests to the backend via REST API calls.

2. **Backend (Flask API):**
   - Exposes API endpoints to receive email text and return predictions.
   - Hosted on AWS EC2 to ensure scalability and availability.
   - Loads and executes the pre-trained Naïve Bayes model to classify emails as spam or not spam.

3. **Machine Learning Model:**
   - Trained using TF-IDF vectorization and Naïve Bayes algorithm.
   - Stored in the backend and loaded dynamically for predictions.
   - Evaluates incoming emails based on learned spam patterns.

4. **Cloud Deployment:**

- Frontend: Hosted on AWS S3 for a scalable and cost-effective static website.
- Backend: Deployed on AWS EC2 for high availability.
- Database (if required): Amazon RDS or DynamoDB can be integrated for storing processed email data and logs.

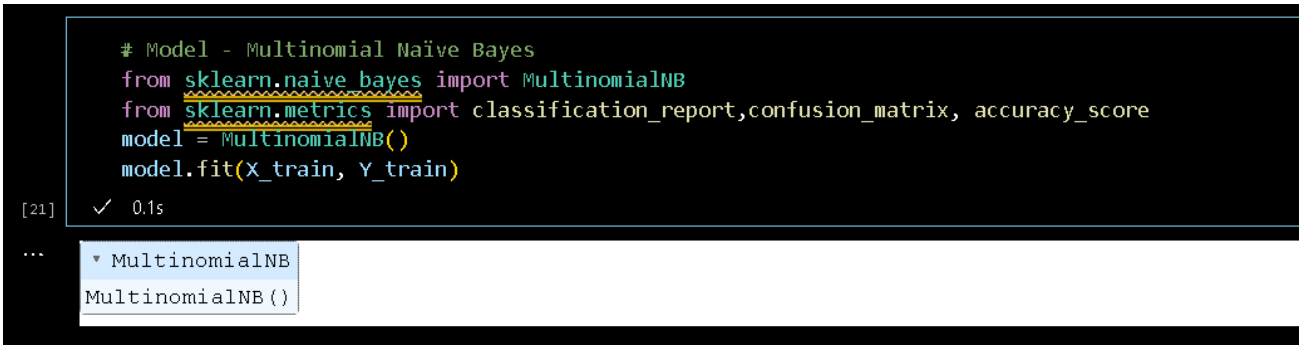
Workflow of the System:

1. **User Input:** The user enters email content in the web interface**.**
2. **API Request:** The frontend sends a request to the Flask API with the email text.
3. **Preprocessing:** The backend processes the text, removing stopwords and applying TF-IDF transformation**.**
4. **Prediction:** The trained Naïve Bayes model predicts if the email is spam or non-spam.
5. **Response:** The API sends the classification result back to the frontend.
6. **Display Output:** The user sees the result on the web interface.

This architecture ensures a fast, scalable, and accurate email spam classification system.

Machine Learning Model Prediction

- The Naïve Bayes model analyzes the input data and predicts the probability of the email being spam.
- The model categorizes the email into:
  - Spam (High Risk) – Likely to be unwanted or malicious content.
  - Not Spam (Low Risk) – Legitimate email content.
- The predicted classification is sent back to the frontend via an API response.



Natural Language Processing (NLP) in Spam Classification

To improve the accuracy of spam classification, Natural Language Processing (NLP) techniques are applied to preprocess email text before feeding it into the machine learning model.

1. Text Preprocessing:

- **Lowercasing:** Converts all text to lowercase for uniformity.
  Removing Special Characters: Eliminates unnecessary symbols and punctuation.
- **Tokenization:** Splits text into individual words (tokens).
- **Stopword Removal:** Removes common words (e.g., "the," "is," "and") that do not contribute to classification**.**
- **Stemming:** Reduces words to their root form (e.g., "running" → "run") using PorterStemmer.

2. Feature Extraction using CountVectorizer

After preprocessing, the text is transformed into numerical data using CountVectorizer, which converts words into a matrix of token counts. This allows the machine learning model to analyze text patterns effectively.

These steps enhance the model's ability to detect spam patterns by focusing on meaningful words and reducing noise in the dataset.

```
#NLP
ps = PorterStemmer()
corpus = []
for i in range(0, len(data)):
    review = re.sub('[^a-zA-Z]', ' ', data['Message'][i])
    review = review.lower()
    review = review.split()
    review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
    review = ' '.join(review)
    corpus.append(review)

# Printing the first 5 values in the corpus list
corpus[1:6]
```

```
['ok lar joke wif u oni',
 'free entri wkli comp win fa cup final tkt st may text fa receiv entri question std txt rate c appli',
 'u dun say earli hor u c alreadi say',
 'nah think goe usf live around though',
 'freemsg hey darl week word back like fun still tb ok xxx std chg send rcv']
```

```
[ ] cv = CountVectorizer(max_features = 4000)
    X = cv.fit_transform(corpus).toarray()
    Y = pd.get_dummies(data['Category'])
    Y = Y.iloc[:, 1].values
```

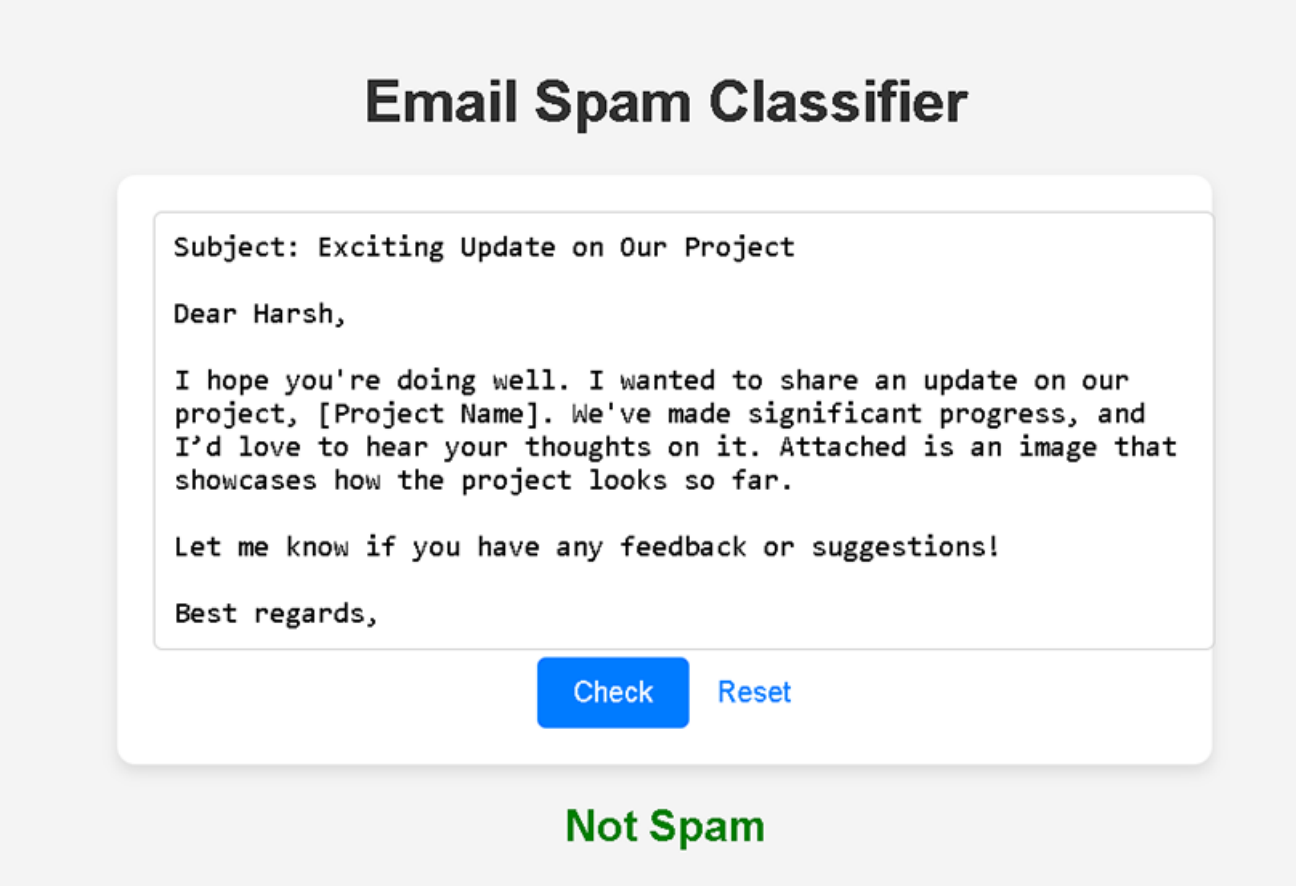
Frontend Input (User Interaction)

- The Email Spam Classifier is accessed through a web interface built with HTML, CSS, and JavaScript.
- The user (e.g., business analyst, HR recruiter, or customer service manager) enters email details (subject & body) into a text box to check whether it is spam or not.
- Upon clicking the "Check" button, the frontend sends an API request to the Flask backend for classification.

Displaying the Results on the Frontend

- The frontend receives the API response and dynamically updates the UI.
- The classification result (Spam or Not Spam) is displayed on the screen.

- If needed, users can refine email filtering rules or improve email security settings based on the prediction results.



COST MANAGEMENT & FUTURE SCALABILITY

Cost Management Strategies:

- AWS EC2 Cost Optimization: Use Reserved Instances for predictable workloads and Spot Instances for non-critical tasks to reduce costs.
- Efficient Data Storage: Store logs in Amazon S3 Glacier for long-term storage and lower costs.
- Monitoring & Budget Alerts: Utilize AWS Cost Explorer and CloudWatch to track usage and set spending limits.

Future Scalability & Optimizations:

- Auto-Scaling & Load Balancing: Implement AWS Auto Scaling and Elastic Load Balancer for handling variable traffic loads.
- Serverless Transition: Consider AWS Lambda for event-driven processing to minimize EC2 dependency.
- Caching with CloudFront: Use AWS CloudFront to reduce backend load and

improve response times.

- Database Scaling: Switch to Amazon DynamoDB for efficient, scalable, and managed database solutions.

By adopting these strategies, the system ensures high availability, cost efficiency, and future-proof scalability.

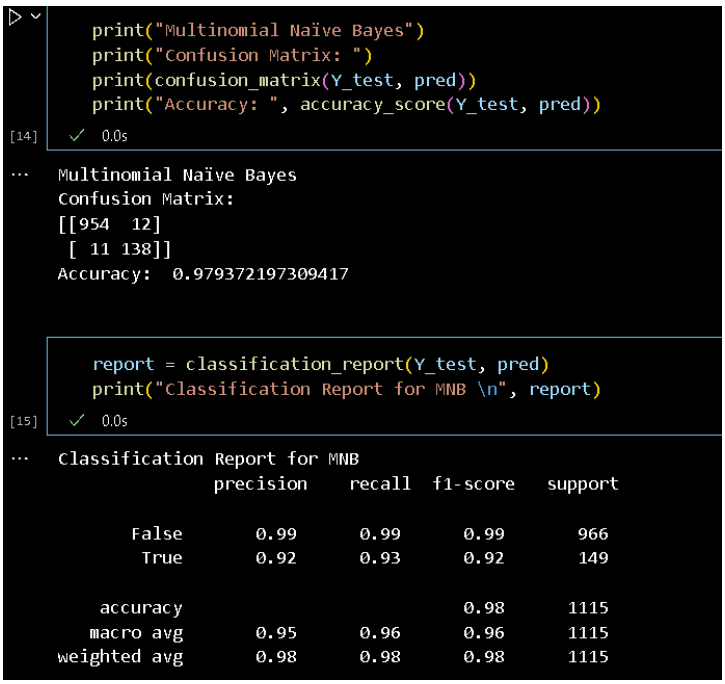
Testing & Evaluation Metrics

This section outlines the testing methods and performance evaluation metrics used to ensure the system's accuracy, efficiency, and reliability.

To ensure the reliability and performance of the spam classification model, several evaluation metrics are used:

- Accuracy: Measures the overall correctness of predictions.
- Precision: Assesses how many predicted spam emails are actually spam.
- Recall (Sensitivity): Evaluates how well the model captures actual spam emails.
- F1-Score: Balances precision and recall for an optimized metric.
- Confusion Matrix: Visualizes correct and incorrect predictions to identify improvement areas.
- ROC Curve & AUC Score: Analyse model performance at various thresholds.

These metrics provide a comprehensive evaluation of the model's effectiveness in distinguishing spam from non-spam emails.



```
    print("Multinomial Naïve Bayes")
    print("Confusion Matrix: ")
    print(confusion_matrix(Y_test, pred))
    print("Accuracy: ", accuracy_score(Y_test, pred))
[14] ✓ 0.0s
...
Multinomial Naïve Bayes
Confusion Matrix:
[[954  12]
 [ 11 138]]
Accuracy:  0.979372197309417

    report = classification_report(Y_test, pred)
    print("Classification Report for MNB \n", report)
[15] ✓ 0.0s
...
Classification Report for MNB
              precision    recall  f1-score   support

       False       0.99      0.99      0.99       966
        True       0.92      0.93      0.92       149

    accuracy                           0.98      1115
   macro avg       0.95      0.96      0.96      1115
weighted avg       0.98      0.98      0.98      1115
```

Challenges & Limitations

This section highlights the challenges faced during the development and deployment of the Customer Churn Prediction System and discusses current limitations that can be addressed in future improvements.

Challenges:

1. **High False Positives & Negatives** – Spam classifiers may incorrectly label legitimate emails as spam (false positives) or fail to detect actual spam (false negatives).
2. **Evolving Spam Techniques** – Spammers constantly modify their tactics, requiring frequent model updates to maintain accuracy.
3. **Dataset Bias –** Training data may not cover all variations of spam, leading to reduced effectiveness in real-world scenarios.
4. **Processing Overhead –** Large email datasets can slow down classification, requiring optimization in feature extraction and model efficiency.
5. **Deployment Complexity –** Ensuring seamless deployment and maintenance of the Flask API on AWS EC2 requires monitoring and resource management.

Limitations:

1. **Language Restrictions –** The model is trained on English emails and may struggle with spam detection in other languages.
2. **Lack of Contextual Understanding –** Traditional ML models like Naïve Bayes and SVM do not fully understand the semantics of an email compared to deep learning approaches.
3. **Limited Real-Time Adaptation –** The classifier does not continuously learn from new emails unless retrained periodically**.**
4. **No Image/Attachment Analysis –** The system only classifies text-based emails and does not analyse attachments or embedded images, which spammers often use to bypass filters.

Future Scope & Enhancements

1. **Deep Learning Models: Implement LSTMs or Transformers for better spam detection accuracy.**
2. **Real-Time Adaptation: Integrate reinforcement learning to continuously update the model.**

3. **Multi-Language Support: Extend classification to support emails in multiple languages.**
4. **Attachment & Image Analysis: Use computer vision models to analyze email attachments for phishing attempts.**
5. **Integration with Email Providers: Develop plugins for Gmail, Outlook, and Yahoo Mail for direct spam filtering.**

DISCUSSION

The Email Spam Classification System demonstrates the effectiveness of machine learning and NLP in automating spam detection. By leveraging TF-IDF vectorization and Naïve Bayes classification, the model successfully differentiates between spam and non-spam emails with high accuracy.

Key Findings:

- Efficient Text Processing: The use of stopword removal, stemming, and tokenization improves classification performance.
- Scalability through AWS: Deployment on AWS EC2 ensures real-time predictions with minimal latency.
- Model Performance: The classifier provides a high precision and recall balance, reducing false positives.

Challenges Identified:

- False Positives/Negatives: Some legitimate emails may be flagged as spam, requiring further optimization.
- Evolving Spam Patterns: Frequent model updates are needed to counter new spam tactics.
- Limited Real-Time Learning: The model does not continuously adapt without retraining.

Implications:

This system provides a cost-effective and scalable spam detection mechanism for businesses and individuals. Future improvements, such as deep learning-based models and multilingual support, can further enhance its performance and applicability.

CONCLUSION

This project successfully implements an **email spam classification system** using

machine learning and NLP techniques. The developed model effectively classifies spam and non-spam emails by leveraging **TF-IDF vectorization and Naïve Bayes classification**. The deployment of the system on **AWS EC2** via a **Flask API** ensures real-time accessibility and scalability.

The classifier demonstrates high accuracy and efficiency, making it a valuable tool for **automating spam detection and improving email security**. However, challenges such as **false positives, evolving spam techniques, and the need for periodic model retraining** highlight areas for future improvement.

In the future, enhancements like **deep learning-based spam filtering, real-time adaptive learning, and integration with email providers** can further improve accuracy and functionality. This project lays a strong foundation for advancing **automated email security systems** and contributes to the broader field of **cybersecurity and AI-driven content filtering**.

REFRENCES

- **McAfee, "Spam and Email Security," [Online]. Available: https://www.mcafee.com**
- **Google AI, "Machine Learning for Spam Detection," [Online]. Available: https://ai.google/research**
- **Scikit-Learn Documentation, "Naïve Bayes Classifiers," [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html**
- **AWS Documentation, "Deploying Machine Learning Models on EC2," [Online]. Available: https://aws.amazon.com/documentation**
- **K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of Documentation*, 1972. Available: https://www.staff.city.ac.uk/~sbrp622/idfpapers/ksj_orig.pdf**
- **M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, "A Bayesian Approach to Filtering Junk E-Mail," *Proceedings of AAAI Workshop on Learning for Text Categorization*, 1998. Available: https://www.researchgate.net/publication/2788064_A_Bayesian_Approach_to_Filtering_Junk_E-Mail**
- **Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT Press, 2016. Available: https://www.deeplearningbook.org/**
- **NLTK Documentation, "Natural Language Toolkit for Text Processing," [Online]. Available: https://www.nltk.org/**