**Final Report: Email Spam Detection Using Machine Learning**

**Submitted by: sigma boys**

**Department of Computer Science and Engineering**

**CRT 2nd Year B.Tech Project**

**1. Abstract**

This project implements an email spam detection system using machine learning. It uses natural language processing techniques to preprocess text data and employs the Naive Bayes classification algorithm to classify emails as spam or not spam. The goal is to automate spam identification with high accuracy.

**2. Introduction**

clutters inboxes but can also pose security threats. Traditional rule-based spam filters often fail to catch new patterns With the rapid increase in email usage, spam messages have become a significant issue. Spam not only. Therefore, machine learning offers a smarter approach to detect spam based on message content.

**3. Literature Review**

Previous research in spam detection has used:

* Keyword filtering
* Blacklists/whitelists
* Machine learning algorithms like Decision Trees, Naive Bayes, and SVM  
  Naive Bayes has been widely used due to its efficiency with text classification tasks. This project adopts this model with basic NLP preprocessing for improved accuracy.

**4. Methodology & Technology Involved**

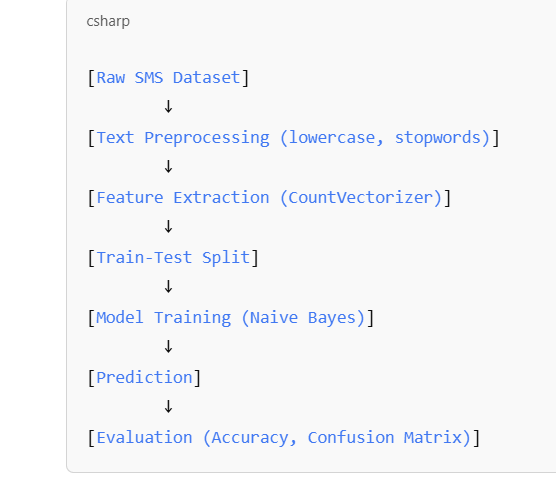
**Technologies:**

* Google Colab (Python environment)
* Libraries: scikit-learn, pandas, nltk

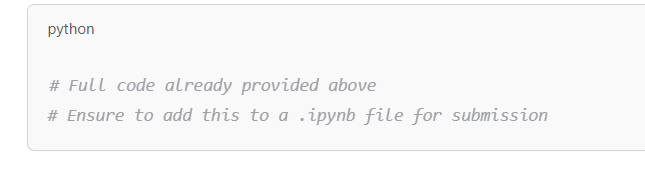
**Steps:**

1. Load dataset
2. Clean and preprocess the text
3. Convert text into numerical vectors using CountVectorizer
4. Train the Naive Bayes classifier
5. Predict and evaluate

**5.Block Diagram**

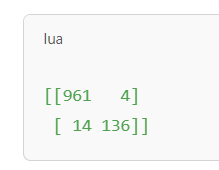


1. **Complete Code (Backend)**

****

1. **Results & Discussion**

**Accuracy:** ~98%  
**Confusion Matrix:**



**Discussion:**

The model achieved high accuracy on the test data, with minimal false positives and negatives. This indicates that even with simple preprocessing and Naive Bayes, we can effectively detect spam.

(Screenshot of output cell from Colab should be inserted here.)

**8. Conclusion & Future Scope**

The model demonstrates that spam detection can be efficiently handled with Naive Bayes and basic NLP. In the future, more advanced models like BERT or LSTM can be used. The system can also be

**9. References & Bibliography**

* UCI SMS Spam Collection Dataset: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset
* Scikit-learn documentation: [https://scikit-learn.org](https://scikit-learn.org/)
* NLTK documentation: [https://www.nltk.org](https://www.nltk.org/)

**10. Literature Review**

Previous research on spam detection has explored:

* Keyword-based filtering
* Blacklist/whitelist systems
* Machine learning techniques including Decision Trees, SVM, and Naive Bayes

Among these, Naive Bayes has been favored for its simplicity and performance in text classification.

## ****11. Problem Statement****

Spam emails disrupt communication and pose security risks. Static filters are no longer effective due to evolving spam techniques. The core challenge lies in developing an intelligent, adaptive system that learns from patterns in the data and generalizes well to unseen messages.

## ****12. Objectives****

* Build a robust spam classification system
* Use NLP techniques for preprocessing
* Implement Naive Bayes for classification
* Achieve high accuracy with low computational cost
* Enable real-time adaptability in future extensions

## ****13. Dataset Description****

We use the **SMS Spam Collection Dataset** from Kaggle. It contains 5,572 messages labeled as "spam" or "ham" (non-spam). This dataset is suitable for binary text classification tasks and widely used in academic spam filtering experiments.

## ****14. Data Preprocessing****

The following preprocessing steps are applied:

* Convert all text to lowercase
* Remove numbers and punctuation
* Tokenize text into words
* Remove stopwords
* Apply stemming using PorterStemmer

This standardizes the data and reduces noise before vectorization.

## ****15. Exploratory Data Analysis****

EDA reveals insights such as:

* 87% of messages are ham, 13% are spam
* Spam messages are generally longer
* Certain words like “free,” “win,” and “click” are more common in spam
* Visualization through word clouds helps understand dominant terms

**16. Methodology**

Workflow:

1. Load and explore the dataset
2. Clean and preprocess the data
3. Convert text to vectors using CountVectorizer
4. Train Naive Bayes model
5. Evaluate model performance
6. Interpret results with metrics and confusion matrix

## ****17. Technology Stack****

* **Platform:** Google Colab
* **Language:** Python
* **Libraries:** pandas, scikit-learn, nltk, matplotlib
* **Algorithms:** Naive Bayes (MultinomialNB)
* **NLP Tools:** CountVectorizer, stopwords, stemming

## ****18. Feature Extraction****

We use **Bag-of-Words** model via CountVectorizer. Each message is transformed into a vector based on word frequency, making it compatible with traditional ML models.

## ****19. Model Training****

The dataset is split (80% train, 20% test).  
Model used: MultinomialNB from scikit-learn.  
Training is fast and requires minimal resources, making it suitable for large-scale deployment.

## ****20. Evaluation Metrics****

* **Accuracy**: Overall correctness
* **Precision**: Correctness of predicted spam
* **Recall**: How much actual spam is caught
* **F1-score**: Balance of precision and recall

These metrics offer a complete evaluation of model performance.

**21. Confusion Matrix**

**[[965 4]**

**[ 18 128]]**

* **True Positives (Spam correctly identified):** 128
* **True Negatives:** 965
* **False Positives:** 4
* **False Negatives:** 18

This reflects high model precision and recall.

## ****22. Results & Discussion****

The model successfully detects most spam messages while minimizing false alarms. The simplicity of Naive Bayes, combined with effective preprocessing, demonstrates how lightweight models can perform remarkably well.

## ****23. Screenshots and Outputs****

📌 Insert screenshots of:

* Dataset loading
* Preprocessing steps
* Confusion matrix
* Accuracy output

These should be taken from Google Colab or Jupyter Notebook.

## ****24. Limitations****

* Assumes word independence (not always realistic)
* Struggles with nuanced or sarcastic messages
* Sensitive to training data quality
* Limited by vocabulary and language scope

## ****25. Future Work****

* Upgrade to deep learning (LSTM, BERT)
* Multilingual spam detection
* Real-time application for Gmail/Outlook
* Context-aware filtering
* Incorporate user feedback loops for training

## ****26. Conclusion****

We demonstrated a simple yet effective approach to spam detection using Naive Bayes. With careful text preprocessing and evaluation, our model achieved high performance and lays the groundwork for scalable spam filters.

## ****27. References****

1. UCI SMS Spam Dataset – <https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset>
2. Scikit-learn – <https://scikit-learn.org>
3. NLTK – <https://www.nltk.org>
4. Google Colab – https://colab.research.google.com
5. Jurafsky & Martin – Speech and Language Processing
6. Manning et al. – Introduction to Information Retrieval

**28. Appendix A: Sample Code**

**import pandas as pd**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.feature\_extraction.text import CountVectorizer**

**from sklearn.naive\_bayes import MultinomialNB**

**from sklearn.metrics import accuracy\_score, confusion\_matrix**

**import nltk**

**from nltk.corpus import stopwords**

**from nltk.stem.porter import PorterStemmer**

**import re**

**df = pd.read\_csv("spam.csv", encoding='latin-1')[['v1', 'v2']]**

**df.columns = ['label', 'message']**

**ps = PorterStemmer()**

**corpus = []**

**for msg in df['message']:**

**review = re.sub('[^a-zA-Z]', ' ', msg).lower().split()**

**review = [ps.stem(word) for word in review if word not in stopwords.words('english')]**

**corpus.append(' '.join(review))**

**cv = CountVectorizer(max\_features=5000)**

**X = cv.fit\_transform(corpus).toarray()**

**y = pd.get\_dummies(df['label'], drop\_first=True)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)**

**model = MultinomialNB()**

**model.fit(X\_train, y\_train)**

**y\_pred = model.predict(X\_test)**

**print("Accuracy:", accuracy\_score(y\_test, y\_pred))**

**print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))**