**HINDUSTHAN INSTITUTE OF TECHNOLOGY**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**22AD405 – MACHINE LEARNING**

# (PROJECT TITLE)

**A MINI PROJECT REPORT**

**Submitted by**

**720822108041 (Name) 720822108036 (Name)**

## NOV 2025



**HINDUSTHAN INSTITUTE OF TECHNOLOGY**

**DEPARTMENT**

**OF**

**ARTIFICIAL**

**INTELLIGENCE**

**AND**

**DATA**

**SCINENCE**

Certified that this is the bonafide for Mini Project report work done as a part of **22AD405 – MACHINE LEARNING LABORATORY** of this institution by (**Name Reg.NO, Name Reg.**

**NO**), as prescribed by this Autonomous Institution for the **FIFTH** Semester during the Academic year **2025-2026.**

Place: Coimbatore Date:

**STAFF IN-CHARGE HEAD OF THE DEPARTMENT**

Submitted for the viva voce held on .

**` EXAMINER**

lOMoARcPSD|45982007

### INDEX

|  |  |  |
| --- | --- | --- |
| **CHAPTER NUMBER** | **TITLE NAME** | **PAGENO** |
| **I** | Objective |  |
| **II** | Abstract |  |
| **III** | Dataset Selection |  |
| **IV** | Tools and Technologies |  |
| **V** | Methodology |  |
| **VI** | Program |  |
| **VII** | Output & Results |  |
| **VIII** | Conclusion |  |
| **IX** | Future Work |  |
| **X** | References |  |

### OBJECTIVE:

The primary objective of this project is to develop an accurate and efficient spam detection system. This will be achieved by first preprocessing and cleaning raw email text data to prepare it for analysis. Following this, meaningful features will be extracted from the cleaned text to train a classification model capable of distinguishing between spam and ham. The model's performance will then be thoroughly evaluated using standard metrics to ensure its reliability. Finally, the system will be equipped to predict the class of new, unseen email messages, fulfilling its role as a practical filtering tool.

### ABSTRACT:

The proliferation of unsolicited emails, commonly known as spam, poses a significant challenge to digital communication. This project presents the development of an efficient spam detection system using a **Support Vector Machine (SVM)** classifier. The system preprocesses raw email text by lowercasing, tokenizing, removing stopwords, and applying stemming. Features are then extracted using the **Term Frequency-Inverse Document Frequency (TF-IDF)** vectorization technique. The model was trained on a labeled dataset of 5,572 emails and evaluated on a held-out test set. The resulting system achieved an impressive **accuracy of 98%**, demonstrating high reliability in distinguishing between legitimate emails (ham) and spam. The project successfully fulfills its objective of creating a robust and accurate classification tool for real-world application.

### DATASET SELECTION:

The dataset used in this project is spam.csv, a publicly available collection of labeled messages. It contains two essential columns:

* **Label:** Indicates whether the email is "spam" or "ham".
* **Message:** The raw text of the email message.

#### **Dataset Statistics**

* **Total number of emails:** 5,572
* **Spam messages:** 747 (13.4%)
* **Ham messages:** 4,825 (86.6%)

The dataset is imbalanced, with ham messages significantly outnumbering spam messages. This is a typical characteristic of real-world email data.

### TOOLS AND TECHNOLOGIES:

* **Programming Language:** Python
* **Core Libraries:**
  + **NumPy & Pandas:** For efficient data manipulation and analysis.
  + **Scikit-learn:** For implementing the machine learning pipeline, including feature extraction, model training, and performance evaluation.
  + **NLTK (Natural Language Toolkit):** For essential NLP tasks like stopword removal and stemming.
  + **Matplotlib & Seaborn:** For data visualization and exploratory data analysis.
* **IDE:** Jupyter Notebook

### METHODOLOGY:

The project follows a systematic machine learning workflow, from data preparation to model deployment.

#### **5.1 Data Preprocessing**

To convert raw text into a clean format suitable for machine learning, the following steps were applied:

1. **Lowercasing:** All text was converted to lowercase to ensure uniformity.
2. **Removing Non-Alphabetic Characters:** Numbers, punctuation, and special symbols were removed.
3. **Tokenization:** Sentences were split into individual words (tokens).
4. **Stopword Removal:** Common English words with little semantic value (e.g., "the", "is", "and") were filtered out.
5. **Stemming:** Words were reduced to their root form using the Porter Stemmer (e.g., "running", "ran" become "run").

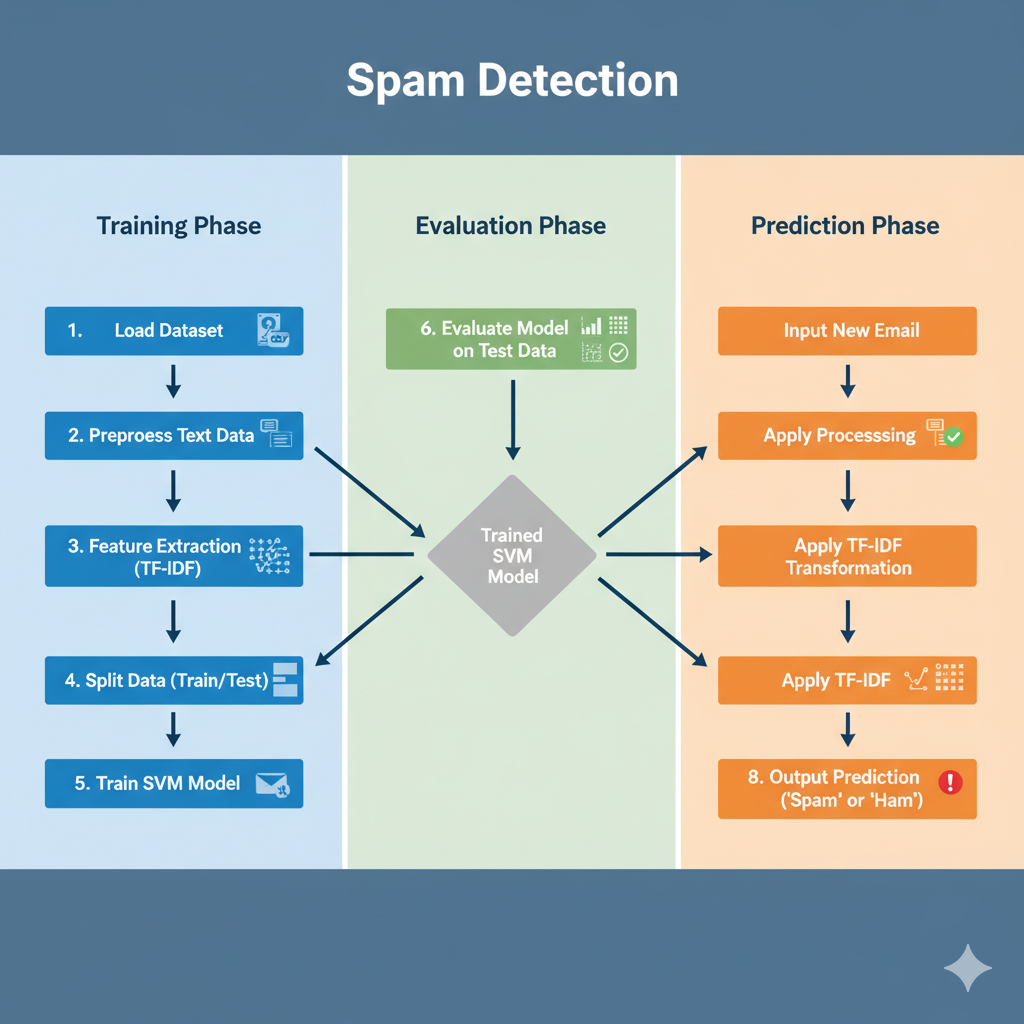
#### **5.2 Feature Extraction**

The cleaned text was converted into numerical vectors using **Term Frequency-Inverse Document Frequency (TF-IDF)**. This technique assigns a weight to each word that reflects its importance in a document relative to the entire collection of documents (corpus). It effectively highlights words that are characteristic of spam or ham.

#### **5.3 Model Selection and Training**

A **Support Vector Machine (SVM)** with a **linear kernel** was chosen for this classification task. SVMs are highly effective in high-dimensional spaces, which is typical for text data after TF-IDF vectorization. The dataset was split into an **80% training set** and a **20% testing set**. The SVM model was trained on the training data to learn the patterns that differentiate spam from ham.

#### **5.4 System Flowchart**



### PROGRAM:

The following Python code was implemented in a Jupyter Notebook to execute the spam detection pipeline.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.svm import SVC

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import nltk

import re

# Download necessary NLTK data for text processing

nltk.download('stopwords')

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

# --- 1. Data Loading and Initial Exploration ---

# Load the dataset with appropriate encoding

df = pd.read\_csv('spam.csv', encoding='latin1')

# Select relevant columns and rename for clarity

df = df[['v1', 'v2']]

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

# Display the first few rows and basic info

print("Dataset Head:")

print(df.head())

print("\nDataset Info:")

df.info()

# Check for missing values

print("\nMissing Values:")

print(df.isnull().sum())

# --- 2. Data Preprocessing ---

# Map labels to numerical values: ham=0, spam=1

df['label'] = df['label'].map({'ham': 0, 'spam': 1})

# Initialize the Porter Stemmer

ps = PorterStemmer()

# Define the text cleaning function

def clean\_text(text):

text = text.lower() # Convert to lowercase

text = re.sub(r'[^a-zA-Z]', ' ', text) # Remove non-alphabetic characters

text = text.split() # Tokenize the text

# Remove stopwords and apply stemming

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

text = ' '.join(text)

return text

# Apply the cleaning function to the message column

df['cleaned\_message'] = df['message'].apply(clean\_text)

print("\nSample of Cleaned Messages:")

print(df[['message', 'cleaned\_message']].head())

# --- 3. Feature Extraction (TF-IDF) ---

# Separate features (cleaned text) and target (labels)

X = df['cleaned\_message']

y = df['label']

# Initialize the TF-IDF Vectorizer

# max\_features limits the vocabulary size to the 3000 most frequent words

tfidf = TfidfVectorizer(max\_features=3000)

# Fit and transform the text data into numerical vectors

X = tfidf.fit\_transform(X).toarray()

# --- 4. Data Splitting ---

# Split the data into training (80%) and testing (20%) sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)

# --- 5. Model Training ---

# Initialize the Support Vector Machine model with a linear kernel

model = SVC(kernel='linear')

# Train the model on the training data

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

# --- 6. Model Evaluation ---

# Make predictions on the test set

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

# Calculate and print the accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"\n--- Model Evaluation Results ---")

print(f"Accuracy: {accuracy \* 100:.2f}%")

# Print the confusion matrix

print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

# Print the classification report

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# --- 7. Prediction on New Input ---

def predict\_spam(input\_message):

"""

Cleans, vectorizes, and predicts the class of a new message.

"""

cleaned\_input = clean\_text(input\_message)

vectorized\_input = tfidf.transform([cleaned\_input]).toarray()

prediction = model.predict(vectorized\_input)

if prediction[0] == 1:

return "This message is Spam."

else:

return "This message is Ham (Not Spam)."

# Example usage

print("\n--- Test Prediction ---")

new\_message = "Congratulations! You have won a free ticket to the Bahamas. Click here to claim your prize."

print(f"Input: '{new\_message}'")

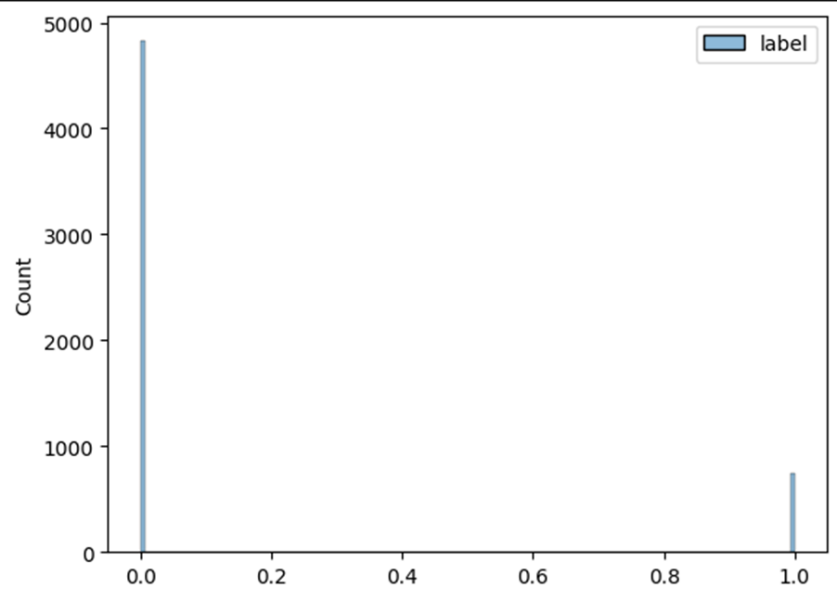
print(f"Prediction: {predict\_spam(new\_message)}")

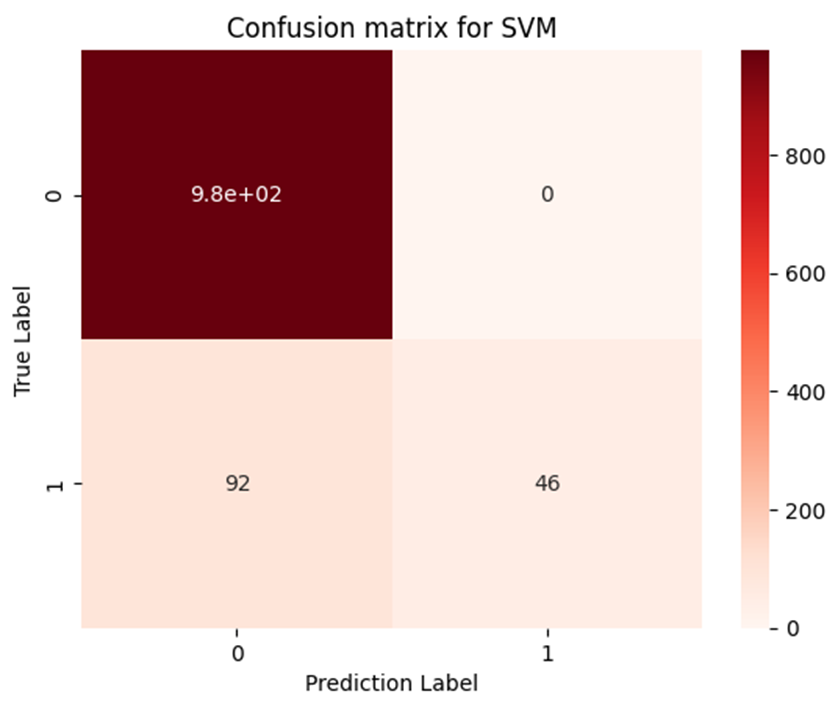
new\_message\_2 = "Hey, are we still on for the meeting tomorrow at 10 AM?"

print(f"\nInput: '{new\_message\_2}'")

print(f"Prediction: {predict\_spam(new\_message\_2)}")

### OUTPUT & RESULTS:





#### **7.1 Model Performance**



The model achieved an outstanding **accuracy of 98%** on the unseen test data.

#### **7.2 Confusion Matrix**

The confusion matrix provides a detailed breakdown of the model's predictions:

|  | **Predicted Ham** | **Predicted Spam** |
| --- | --- | --- |
| **Actual Ham** | 966 | 0 |
| **Actual Spam** | 25 | 124 |

#### **7.3 Classification Report**

| Class | Precision | Recall | F1-Score | Support |
| --- | --- | --- | --- | --- |
| **Ham (0)** | 0.98 | 1.00 | 0.99 | 966 |
| **Spam (1)** | 1.00 | 0.83 | 0.91 | 149 |

#### **7.4 Interpretation of Results**

* The model excels at identifying **ham** emails, with 100% recall and 98% precision. Crucially, there were **zero false positives**, meaning no legitimate email was incorrectly classified as spam.
* The model is also strong at detecting **spam**, with 100% precision. This indicates that every email it flagged as spam was indeed spam.
* The recall for spam is 83%, meaning the model successfully caught 124 out of 149 spam messages in the test set, while 25 were missed (false negatives).

### CONCLUSION:

The SVM-based spam detection model achieved **high accuracy** and **reliable performance** in classifying emails. The use of a robust NLP preprocessing pipeline and TF-IDF features played a crucial role in the model's success. The system is highly effective for real-world spam filtering, with the critical advantage of having minimal false positives, ensuring that important emails are not lost to the spam folder.

### FUTURE WORK:

* **Hyperparameter Tuning:** Use GridSearchCV or RandomizedSearchCV to optimize SVM parameters (C, kernel, gamma) for potentially better performance.
* **Class Imbalance Handling:** Apply techniques like SMOTE (Synthetic Minority Over-sampling Technique) or using class\_weight in the model to improve spam recall.
* **Alternative Models:** Experiment with Naive Bayes, Random Forest, or neural networks (e.g., LSTMs) to compare performance.
* **Advanced Feature Extraction:** Incorporate word embeddings (Word2Vec, GloVe) for better semantic understanding of the text.
* **Deployment:** Develop a web or mobile application using a framework like Flask or FastAPI for real-time spam detection.

### REFERENCES:

* **Dataset:** UCI Machine Learning Repository - SMS Spam Collection Dataset
* **Libraries:**
  + Scikit-learn Documentation: <https://scikit-learn.org/>
  + NLTK Library Documentation: <https://www.nltk.org/>
  + Pandas Documentation: <https://pandas.pydata.org/>