FOUNDATIONS OF MACHINE LEARNING – ASSIGNMENT 3

SPAM–HAM CLASSIFIER FROM SCRATCH

ARYAN PRASAD
DA25M007
IIT MADRAS

Contents

Abstract

In this assignment I implemented a complete spam–ham email classifier pipeline from scratch. The pipeline uses the public Kaggle dataset (link below), and includes text cleaning, TF–IDF feature extraction, Top–K feature selection, and multiple classifiers implemented from scratch. Linear and RBF SVMs from scikit-learn are used for comparison. The final deployed model is Multinomial Naive Bayes because it combines accuracy with computational efficiency and stability.

Dataset

The dataset used is the **Spam Mails Dataset** from Kaggle: `https://www.kaggle.com/datasets/venky73/spam-mails-dataset/data`.

Before cleaning:

$$\textbf{Total rows} = 5572, \quad \text{Ham} = 4825, \quad \text{Spam} = 747.$$

After cleaning:

$$\textbf{Total rows} = 4993, \quad \text{Ham} = 4300, \quad \text{Spam} = 693.$$

579 rows were removed due to empty or corrupted text. The cleaned dataset contains mixed types of emails: replies, newsletters, marketing messages, banking alerts and common spam patterns.

Preprocessing Pipeline

All preprocessing logic is in `preprocess.py`. Same steps are used for both training and prediction.

Text Cleaning

I applied:

- lowercasing,
- removal of URLs, punctuation, digits, symbols,
- whitespace normalisation.

Tokenisation

Whitespace tokenisation. Tokens of length 1 are removed because they don't help in classification.

Bag-of-Words

Emails are converted into count vectors.

TF–IDF Features

Using:

$$\mathrm{TFIDF}(t,d) = \mathrm{TF}(t,d) \cdot \log\left(\frac{N+1}{\mathrm{DF}(t)+1}\right),$$

with L2 normalisation to make emails length-invariant.

Top–K Feature Selection

Using log-odds:

$$score(w) = \log\left(\frac{P(w|spam)}{P(w|ham)}\right),$$

and selecting the top 5000 highest-absolute-score tokens.

Pipeline Diagram

```
RAW EMAILS
      ↓
CLEANING
      ↓
TOKENISATION
      ↓
REMOVE SMALL TOKENS
      ↓
BAG-OF-WORDS + TF-IDF
      ↓
TOP-K FEATURES (K = 5000)
      ↓
TRAIN MODELS
      ↓
SAVE BEST MODEL (NB)
      ↓
predict.py CLASSIFIES test/ EMAILS
```

Models Implemented

Models coded from scratch: NB, Logistic Regression, Perceptron, k–NN. Linear SVM and RBF SVM used for comparison.

Multinomial Naive Bayes

$$P(w|y) = \frac{N_{w,y}+\alpha}{\sum_u (N_{u,y}+\alpha)}, \quad \alpha = 1.$$

Logistic Regression

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

Perceptron

$$w \leftarrow w + \eta(y - \hat{y})x.$$

k–NN

Uses Euclidean distance on TF–IDF vectors.

SVM

Linear SVM and RBF SVM used only for comparison.

Design Decisions in My Implementation

Laplace Smoothing

Prevents zero probabilities and keeps NB stable.

TF–IDF with Normalisation

Makes email vectors length-invariant and improves stability for LR and SVM.

Class Weighting

Balances LR against ham-heavy distribution.

Top–K Log-Odds

Reduces noise and drops meaningless tokens.

Deterministic Splits

Seed-based splitting ensures reproducible results.

High-Dimensional Behaviour of Models

Naive Bayes

High-dimensional sparse data works in NB's favour because only a few tokens influence the likelihood.

k–NN

Distances lose meaning in 5000 dimensions; neighbourhoods collapse.

Linear SVM

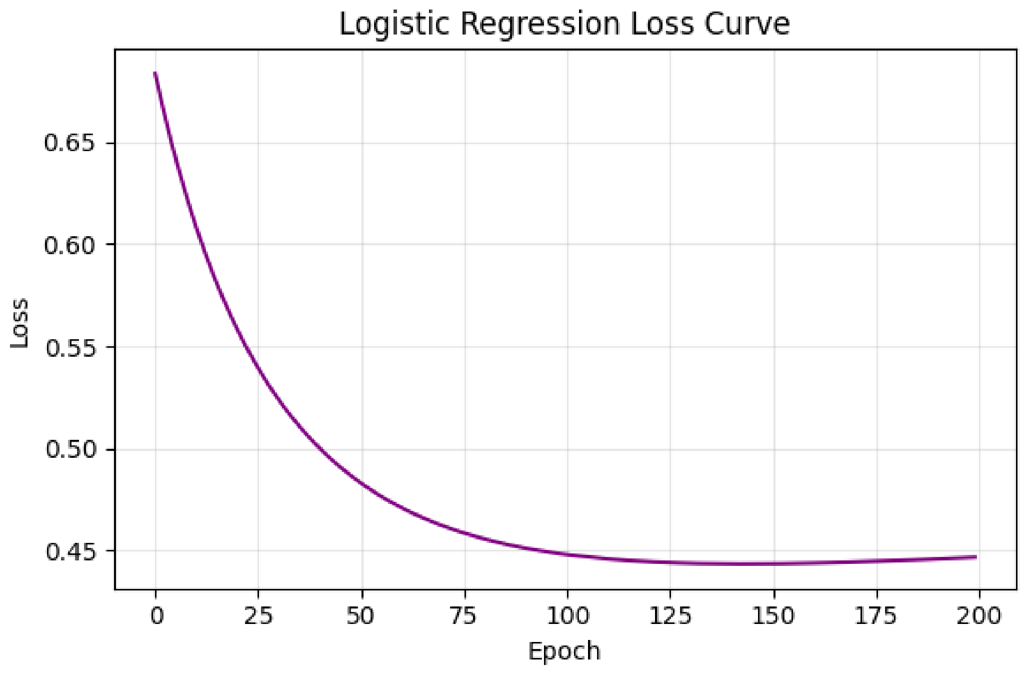
TF–IDF usually produces near-linear separability.

RBF SVM

Nonlinear boundaries do not add much because the dataset is already linearly separable.

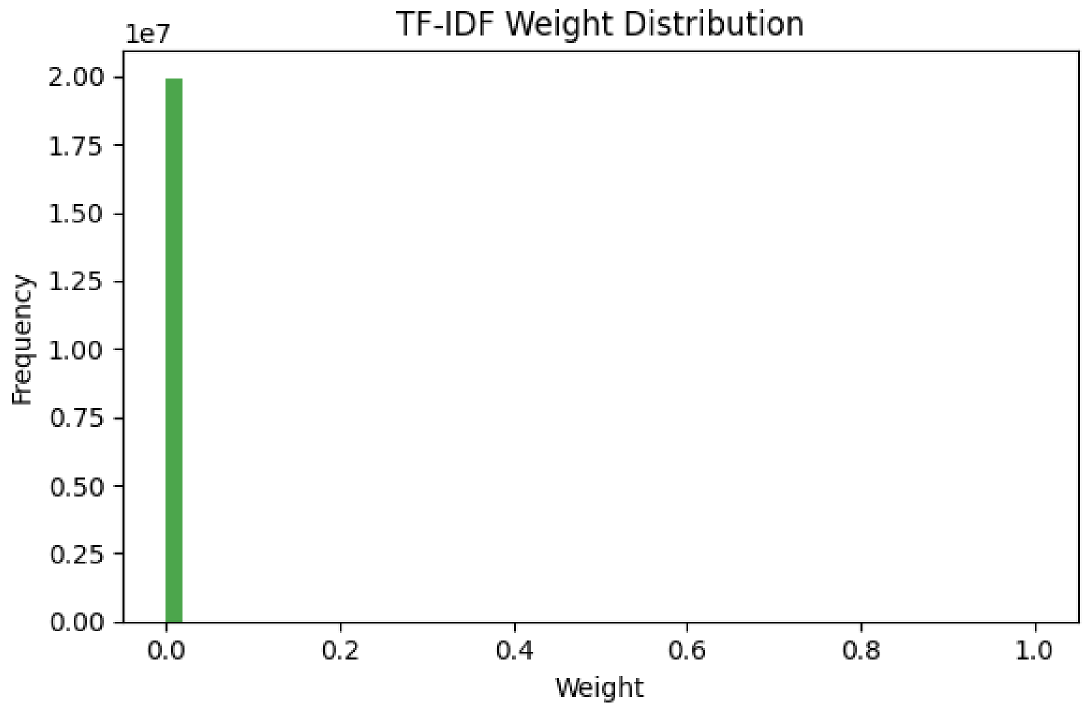
Training and Feature Engineering Plots

Logistic Regression Loss Curve



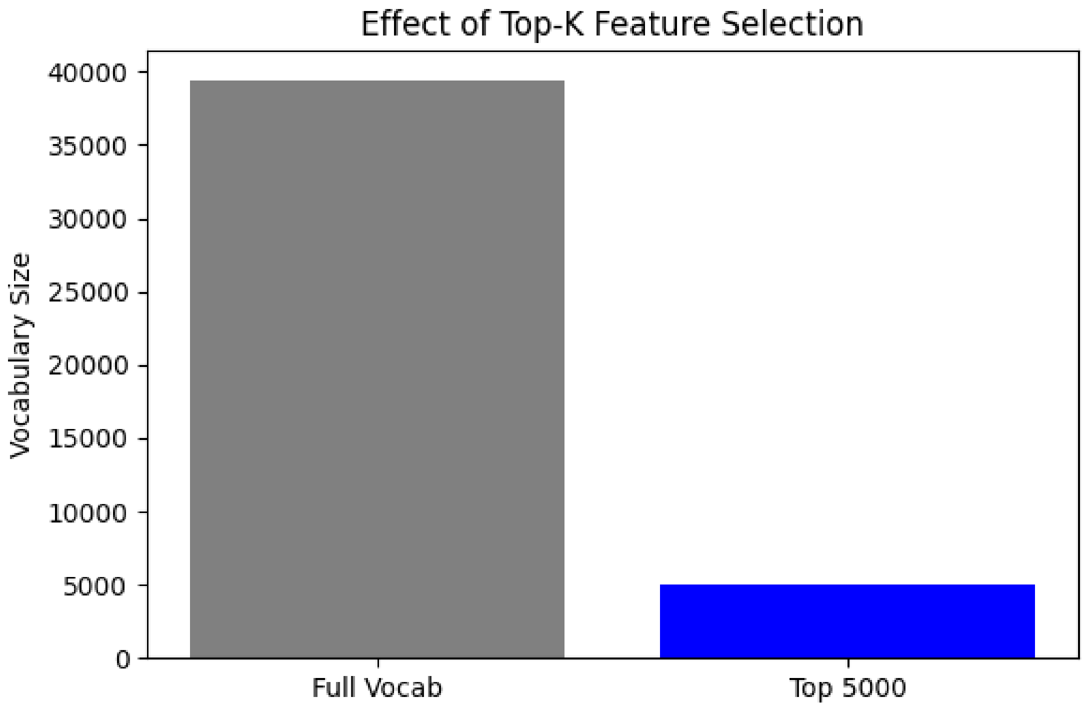
Loss curve for Logistic Regression.

TF–IDF Weight Distribution



TF–IDF distribution.

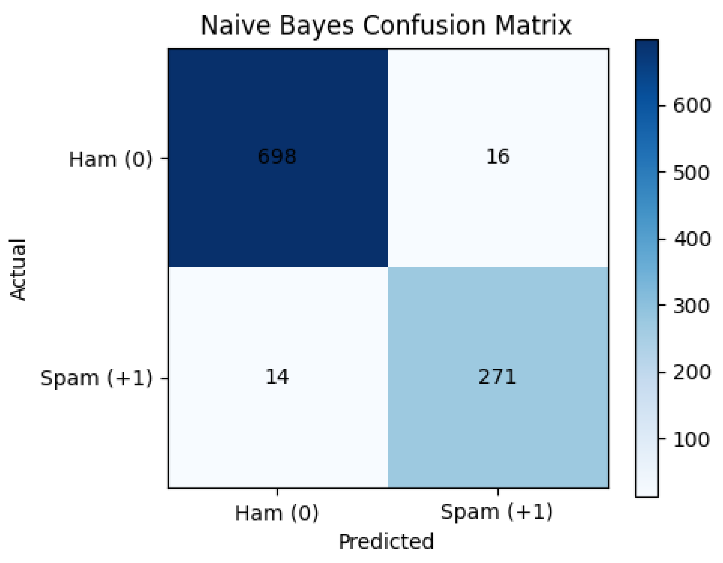
Top–K Features



Vocabulary reduced to 5000 tokens.

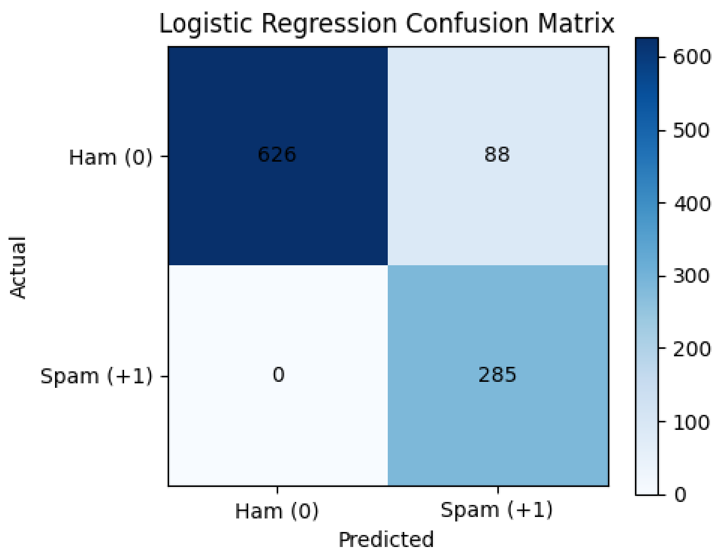
Confusion Matrices and Analysis

Naive Bayes



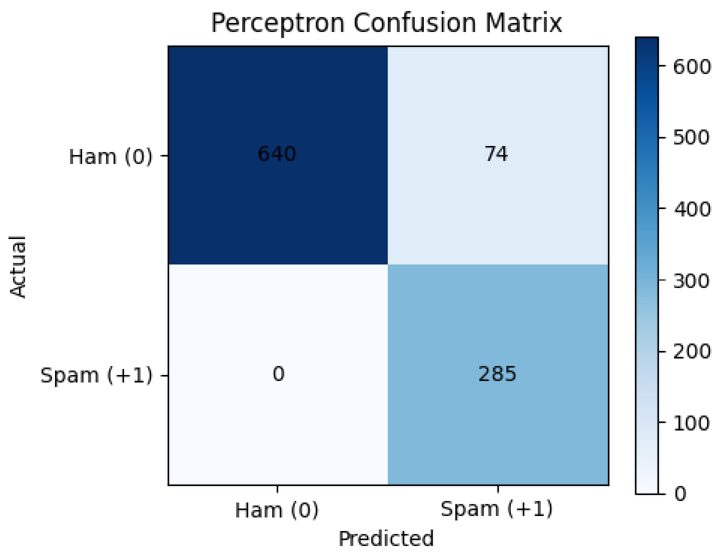
Naive Bayes Confusion Matrix.

Logistic Regression



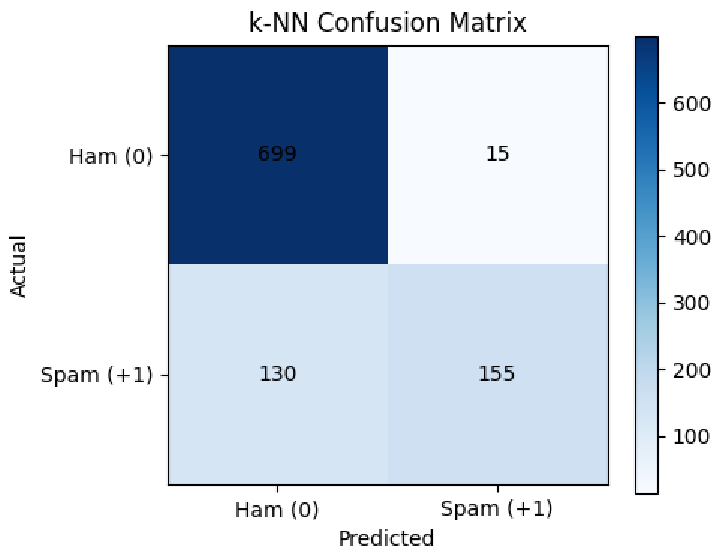
Logistic Regression Confusion Matrix.

Perceptron



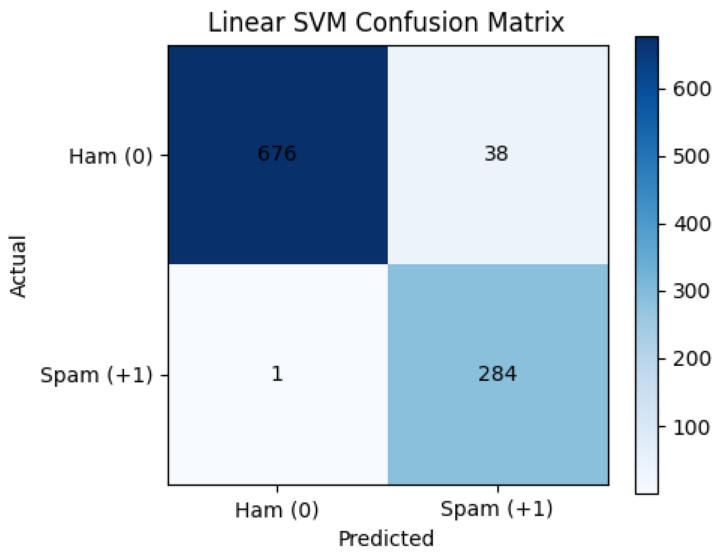
Perceptron Confusion Matrix.

k–NN



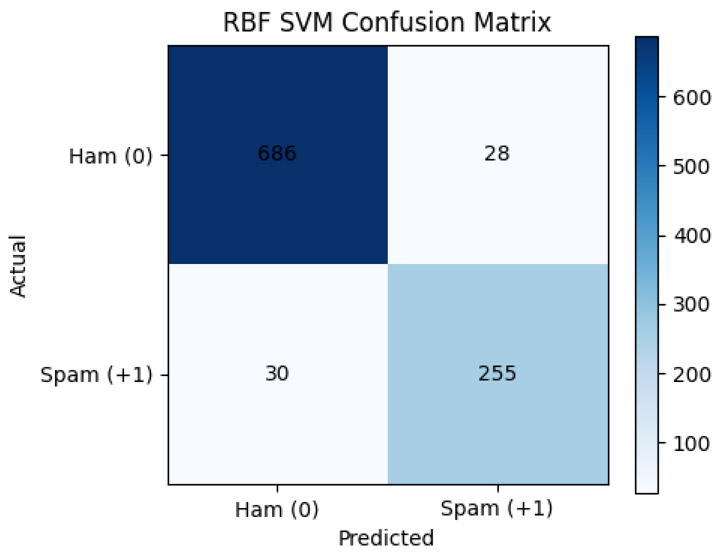
k–NN Confusion Matrix.

Linear SVM



Linear SVM Confusion Matrix.

RBF SVM



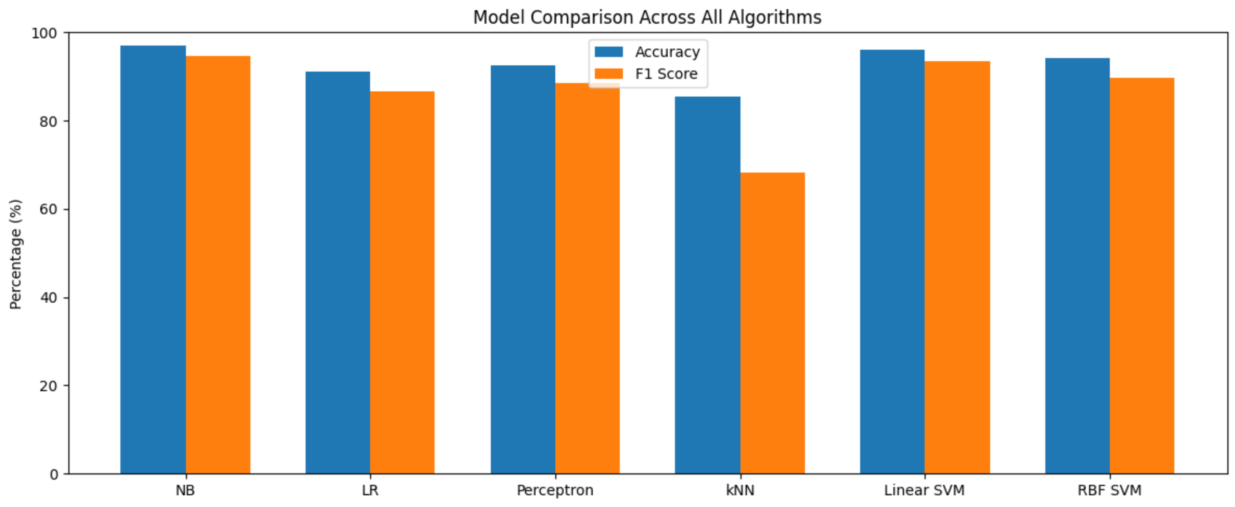
RBF SVM Confusion Matrix.

Model Accuracy Summary

Model	Accuracy (%)	F1 Score
Naive Bayes	97.00	0.9476
Logistic Regression	91.29	0.8676
Perceptron	92.99	0.8879
k–NN	84.38	0.6549
Linear SVM	96.10	0.9357
RBF SVM	94.69	0.9072

Model comparison summary.

Overall Comparison Plot



Accuracy and F1 across all models.

Training Time and Complexity Analysis

Naive Bayes

Training time: $O(NL)$ (very fast).

Logistic Regression

Training cost: $O(\text{epochs} \cdot N \cdot d)$ with batching.

Perceptron

Similar to LR but without sigmoid.

k–NN

No training cost but slow prediction: $O(Nd)$.

SVM

Linear SVM scales well; RBF kernel heavier.

Final Prediction System

`predict.py` loads the Naive Bayes model and predicts on every text file in the `test/` folder. It outputs 1 (spam) or 0 (ham).

Prediction Results

Email File	**Prediction**
email1.txt	0
email2.txt	0
email3.txt	0
email4.txt	0
email5.txt	1
email6.txt	0
email7.txt	0
email8.txt	0
email9.txt	0
email10.txt	0
email11.txt	1
email12.txt	1
email13.txt	0
email14.txt	1
email15.txt	0
email16.txt	1
email17.txt	1
email18.txt	1
email19.txt	1
email20.txt	0

Predictions from `predict.py` on 20 test emails.

Files Included in ZIP Submission

Report:

- Report.pdf

- Details.txt

Source Code:

- main.py, preprocess.py, naive_bayes.py, logistic_regression.py, perceptron.py, knn.py, metrics.py, plots.py, predict.py

Model + Dataset:

- spam_nb_model.npz
- spam_ham_dataset.csv

Figures:

- LR_loss_curve.png, tf_idf_weight.png, top_k.png
- NB_confusion.png, LR_Confusion.png
- Perceptron_Confusion.png, k_nn_confusion.png
- linear_svm_confusion.png, rbf_svm_confusion.png
- model_comparison.png

Conclusion

I created a full end-to-end spam classifier using a Kaggle dataset and scratch implementations of required algorithms. Naive Bayes performed the best among scratch models and was deployed. The whole pipeline, including feature engineering, model training, evaluation and prediction on unseen files, works correctly.