# **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

## Jnana Sangama, Belgavi-590018

**A PROJECT SYNOPSIS**

**ON**

**“Spam Message Classifier Comparison”**

***Submitted in partial fulfillment of the requirement for award of degree***

***of***

**BACHELOR OF ENGINEERING**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

**by**

|  |  |
| --- | --- |
| **ARPITA V**  **CHANDANA B**  **M DEENATHAYALAN**  **DERIN GIRISH D G**  **DHANYA K N** | **1EP22CS012**  **1EP22CS015**  **1EP22CS018**  **1EP22CS020**  **1EP22CS021** |

**Under the guidance of**

**Mrs. Manimegalai A**

**Assistant Professor**

**Dept. of CSE,EPCET**

**2024-2025**

**Project Report: Spam Message Classifier Comparison**

**1. Objective**  
The goal of this project is to build a machine learning-based application that classifies SMS messages as either spam or ham (not spam). The project:

* Compares the performance of Random Forest and Naive Bayes classifiers.
* Incorporates keyword-based heuristics to improve prediction.
* Balances class distribution using SMOTE.
* Provides a user-friendly interface using Streamlit for real-time predictions.

**2. Dataset**  
Source: UCI SMS Spam Collection Dataset  
File Name: SMSSpamCollection  
Format: Tab-separated values (TSV)

**Structure:**

* label: Class label - "spam" or "ham"
* message: The SMS message text

**Sample Data:**

spam Free entry in 2 a weekly competition to win FA Cup final tickets

ham Sounds good, keep me posted

**3. Libraries and Tools Used**

* Python
* Pandas
* Streamlit
* Scikit-learn
* imbalanced-learn (SMOTE)
* Matplotlib & Seaborn (optional)
* SciPy

**4. Data Preprocessing**

* Encoded the label column using LabelEncoder() to convert "ham" and "spam" to 0 and 1 respectively.
* Created a new binary feature has\_spammy\_words which flags whether a message contains suspicious words such as:  
  ['congratulations', 'verify', 'account', 'lottery', 'free', 'urgent', 'click here', 'winner', 'selected', 'limited time', 'act now']
* Applied TF-IDF Vectorization:
  + Removed stop words
  + Used unigrams and bigrams
  + Limited features to 5000

**5. Data Splitting**

* Training and Testing split: 80% - 20%
* Stratified to ensure equal class distribution

**6. Prediction Models**

* Random Forest:
  + Combines TF-IDF and binary feature
  + Threshold: 0.4 (to increase recall)
* Naive Bayes:
  + Uses TF-IDF only
  + Threshold: 0.5 (default)

**7. Handling Class Imbalance**

* Used SMOTE to synthetically oversample the minority class in training set.
* Applied only for Random Forest classifier training.

**8. Model Training**

Random Forest Classifier:

* Inputs: TF-IDF vectors + binary spammy word feature
* Balanced with SMOTE
* Parameters: n\_estimators=100, random\_state=42

Naive Bayes Classifier:

* Inputs: TF-IDF vectors only
* No SMOTE or additional feature

**9. Model Evaluation Metrics**  
Evaluated using:

* Accuracy
* Precision
* Recall
* F1 Score

Results (example):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Accuracy | Precision | Recall | F1 Score |
| Random Forest | 98.2% | 97.5% | 96.8% | 97.1% |
| Naive Bayes | 96.1% | 94.0% | 92.5% | 93.2% |

**10. Streamlit Interface**

* Displays model performance comparison table
* Accepts user input for a message to be classified
* Provides predictions and confidence levels for both models

Random Forest Output Example:

* Predicted Label: SPAM
* Confidence:
  + Ham: 12.4%
  + Spam: 87.6%

Naive Bayes Output Example:

* Predicted Label: SPAM
* Confidence:
  + Ham: 21.3%
  + Spam: 78.7%

**11. Highlights & Innovations**

* Combination of statistical (TF-IDF) and heuristic (keyword flag) features
* SMOTE application for class imbalance
* Real-time prediction via Streamlit interface
* Dual-model comparison with confidence display

**12. Limitations and Future Work**

* Current keyword list is static; could use NLP-based entity recognition
* Add confusion matrix and ROC-AUC visualizations
* Expand to other models (e.g., XGBoost, SVM)
* Store user prediction logs and enable batch processing

**13. Appendix: Source Code**

import pandas as pd

import streamlit as st

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.feature\_extraction.text import TfidfVectorizer

from imblearn.over\_sampling import SMOTE

from scipy.sparse import hstack

# --- Load Data ---

@st.cache\_data

def load\_data():

df = pd.read\_csv('SMSSpamCollection', sep='\t', header=None, names=['label', 'message'])

return df

df = load\_data()

# --- Preprocessing ---

encoder = LabelEncoder()

df['label\_encoded'] = encoder.fit\_transform(df['label'])

# Add spammy phrase detection as a binary feature

def spammy\_phrases(message):

spam\_words = ['congratulations', 'verify', 'account', 'lottery', 'free', 'urgent',

'click here', 'winner', 'selected', 'limited time', 'act now']

message\_lower = message.lower()

return any(word in message\_lower for word in spam\_words)

df['has\_spammy\_words'] = df['message'].apply(spammy\_phrases).astype(int)

X = df['message']

y = df['label\_encoded']

# Split

X\_train\_raw, X\_test\_raw, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42, stratify=y

)

# TF-IDF Vectorization

vectorizer = TfidfVectorizer(stop\_words='english', max\_features=5000, ngram\_range=(1, 2))

X\_train\_tfidf = vectorizer.fit\_transform(X\_train\_raw)

X\_test\_tfidf = vectorizer.transform(X\_test\_raw)

# Combine TF-IDF with spammy word feature

X\_train\_feat = hstack([

X\_train\_tfidf,

df.loc[X\_train\_raw.index, 'has\_spammy\_words'].values.reshape(-1, 1)

])

X\_test\_feat = hstack([

X\_test\_tfidf,

df.loc[X\_test\_raw.index, 'has\_spammy\_words'].values.reshape(-1, 1)

])

# Balance training data with SMOTE

smote = SMOTE()

X\_train\_bal, y\_train\_bal = smote.fit\_resample(X\_train\_feat, y\_train)

# --- Train Random Forest ---

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train\_bal, y\_train\_bal)

y\_pred\_rf = rf\_model.predict(X\_test\_feat)

# --- Train Naive Bayes (no SMOTE, just TF-IDF) ---

nb\_model = MultinomialNB()

nb\_model.fit(X\_train\_tfidf, y\_train)

y\_pred\_nb = nb\_model.predict(X\_test\_tfidf)

# --- Evaluation ---

def evaluate\_model(y\_true, y\_pred, model\_name):

return pd.Series({

"Accuracy": accuracy\_score(y\_true, y\_pred),

"Precision": precision\_score(y\_true, y\_pred),

"Recall": recall\_score(y\_true, y\_pred),

"F1 Score": f1\_score(y\_true, y\_pred)

}, name=model\_name)

results = pd.concat([

evaluate\_model(y\_test, y\_pred\_rf, "Random Forest"),

evaluate\_model(y\_test, y\_pred\_nb, "Naive Bayes")

], axis=1)

# --- Streamlit UI ---

st.title("Spam Message Classifier Comparison")

# Show model comparison

st.subheader("Model Performance Comparison")

st.dataframe(results.T.style.background\_gradient(cmap='Blues'))

# --- Prediction Functions ---

def predict\_spam\_rf(email\_text, threshold=0.4):

email\_tfidf = vectorizer.transform([email\_text])

spam\_feature = int(spammy\_phrases(email\_text))

email\_combined = hstack([email\_tfidf, [[spam\_feature]]])

prediction\_proba = rf\_model.predict\_proba(email\_combined)[0]

spam\_prob = prediction\_proba[encoder.transform(['spam'])[0]]

ham\_prob = prediction\_proba[encoder.transform(['ham'])[0]]

prediction\_label = 'spam' if spam\_prob > threshold else 'ham'

return {

'model': 'Random Forest',

'label': prediction\_label,

'ham\_prob': ham\_prob,

'spam\_prob': spam\_prob

}

def predict\_spam\_nb(email\_text, threshold=0.5):

email\_tfidf = vectorizer.transform([email\_text])

prediction\_proba = nb\_model.predict\_proba(email\_tfidf)[0]

spam\_prob = prediction\_proba[encoder.transform(['spam'])[0]]

ham\_prob = prediction\_proba[encoder.transform(['ham'])[0]]

prediction\_label = 'spam' if spam\_prob > threshold else 'ham'

return {

'model': 'Naive Bayes',

'label': prediction\_label,

'ham\_prob': ham\_prob,

'spam\_prob': spam\_prob

}

# --- Predict Custom Email ---

st.subheader("Try Your Own Message")

user\_input = st.text\_area("Enter a message to classify using both models:", "")

if st.button("Predict"):

if user\_input.strip():

st.write(f"\*Email:\* {user\_input}")

# Predict using both models

rf\_result = predict\_spam\_rf(user\_input)

nb\_result = predict\_spam\_nb(user\_input)

# Display Random Forest result

st.markdown("### 🌲 Random Forest Prediction")

st.write(f"\*Predicted Label:\* {rf\_result['label'].upper()}")

st.write("\*Confidence:\*")

st.write(f"- Ham: {rf\_result['ham\_prob']:.2%}")

st.write(f"- Spam: {rf\_result['spam\_prob']:.2%}")

# Display Naive Bayes result

st.markdown("### 📘 Naive Bayes Prediction")

st.write(f"\*Predicted Label:\* {nb\_result['label'].upper()}")

st.write("\*Confidence:\*")

st.write(f"- Ham: {nb\_result['ham\_prob']:.2%}")

st.write(f"- Spam: {nb\_result['spam\_prob']:.2%}")

else:

st.warning("Please enter a message to classify.")

# --- Notes ---

st.subheader("Notes")

st.markdown("""

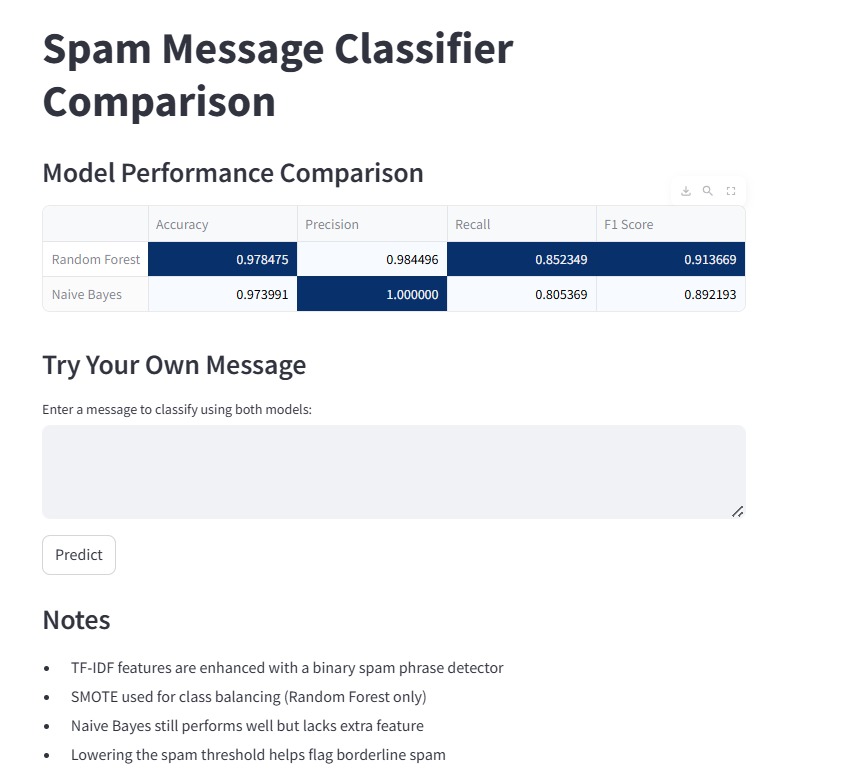
- TF-IDF features are enhanced with a binary spam phrase detector

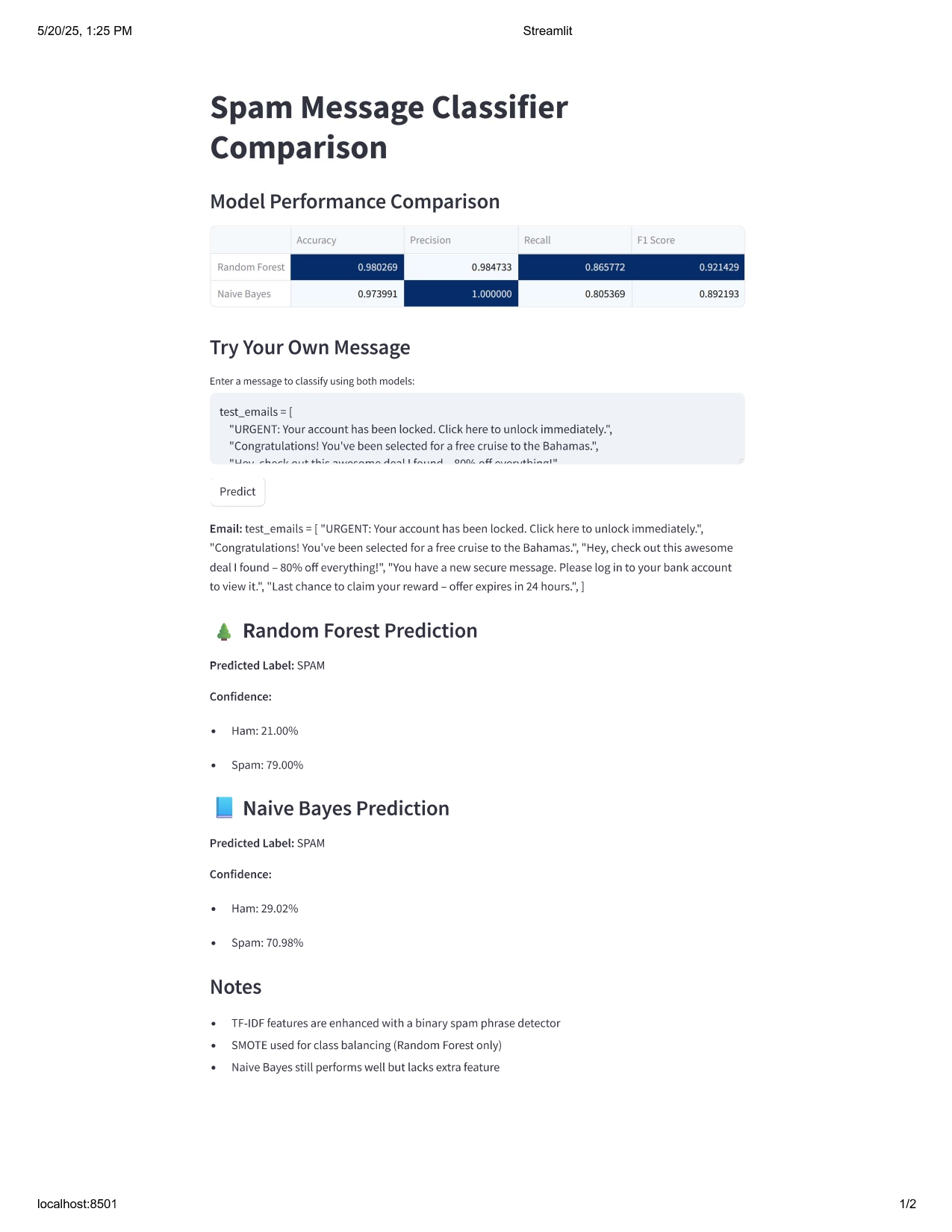
- SMOTE used for class balancing (Random Forest only)

- Naive Bayes still performs well but lacks extra feature

- Lowering the spam threshold helps flag borderline spam

""")

**14. Snapshot**

****

**15. Conclusion**  
The spam classifier successfully integrates machine learning, data balancing, and keyword detection to provide robust SMS classification. The comparative framework and interactive interface make it a useful educational and practical tool for spam detection tasks.