SMS Spam Detection

A Data Science Project Using Machine Learning

Building an intelligent classifier to automatically detect spam messages and protect users from unwanted content

Project Overview & Objectives

Problem Statement

SMS spam is a **common issue** affecting users worldwide, causing inconvenience and potential security risks

Dataset

- SMS Spam Collection Dataset from Kaggle
- **5,572** SMS messages with labels
- Binary classification: spam vs ham (not spam)

Key Metrics

Performance Focus

Accuracy
Overall correctness

Precision
Minimize false positives

Project Objectives

- Build a **high-accuracy** model to classify SMS messages
- Achieve **high precision** to minimize false positives
- Compare multiple ML algorithms for best performance
- Deploy model via **Streamlit app** for practical use
- Analyze text patterns in spam vs ham messages

Data Overview

Dataset Details

SMS Spam Collection | 5,572 messages | Binary classification

5,169 after cleaning | 403 duplicates removed

Data Structure

Column	Description
v1 (target)	Message label (ham/spam)
v2 (text)	Message content

Sample Messages

Ham Messages:

"Go until jurong point, crazy.. Available only..."

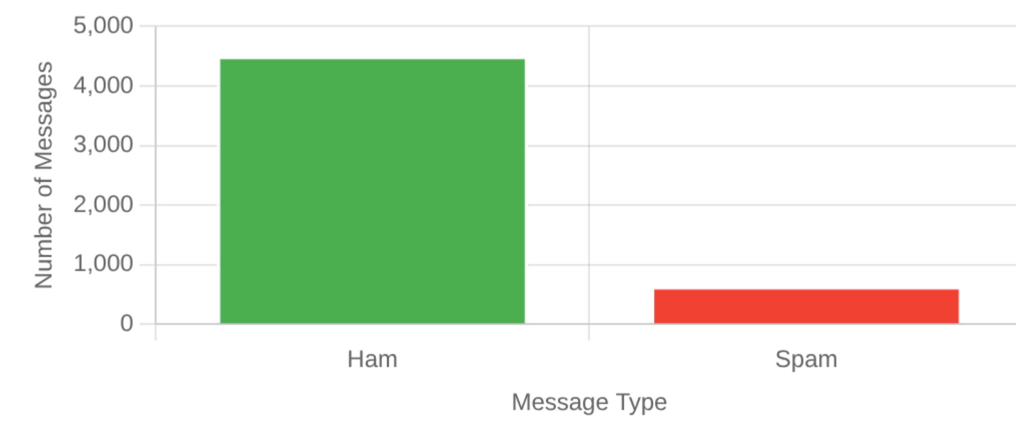
"Ok lar... Joking wif u oni..."

Spam Messages:

"Free entry in 2 a wkly comp to win FA Cup final..."

"URGENT! Your Mobile No was awarded a £2,000 prize..."

Class Distribution



87.4%
Ham Messages

12.6%
Spam Messages

Key Observations

Class imbalance: 87.4% ham vs 12.6% spam
Risk of misclassifying spam as ham

403 duplicates removed (7.2% of data)
Improves model generalization

No missing values in target and text columns

Addressing imbalance: Focus on precision metric

> **The dataset is clean and ready for modeling, but class imbalance poses a key challenge.**

Data Cleaning & Preprocessing

Data Cleaning Steps

1. **Remove Unnecessary Columns**
   Dropped columns: Unnamed: 2, 3, 4
2. **Rename Columns**
   v1 → target, v2 → text
3. **Label Encoding**
   ham → 0, spam → 1
4. **Handle Missing Values**
   No missing values in target and text columns
5. **Remove Duplicates**
   Removed 403 duplicate records

Code Implementation

```
# Drop unnecessary columns
df.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed:
4'], inplace=True)

# Rename columns
df.rename(columns={'v1':'target','v2':'text'},
inplace=True)

# Label encoding
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df['target'] = encoder.fit_transform(df['target'])

# Remove duplicates
df = df.drop_duplicates(keep='first')
```

Dataset Transformation

Original Dataset	After Cleaning
5,572	**5,169**
Total rows	Total rows

- 5 columns reduced to 2 columns
- 403 duplicates (7.2% of data) removed
- No missing values in target and text columns
- Final dataset shape: (5169, 2)

Data Quality Checks

- **Missing values:** None in target and text columns
- **Duplicates:** Removed all 403 duplicate records
- **Data types:** All columns properly formatted
- **Class distribution:** Maintained after cleaning

✓ **Data Ready for EDA**
Clean dataset ready for exploratory analysis and feature engineering

Exploratory Data Analysis (EDA)

Text Feature Analysis

78.98
Avg Characters

18.46
Avg Words

1.97
Avg Sentences

Spam vs Ham Comparison

Feature	Ham	Spam
Avg Characters	70.46	137.89
Avg Words	17.12	27.67
Avg Sentences	1.82	2.97

Top N-grams

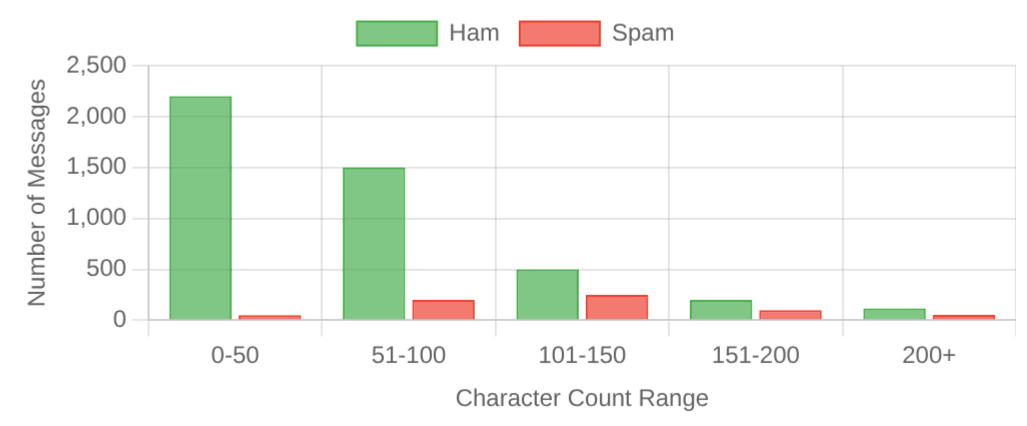
Spam Bigrams:

win prize · free entry · call now · text stop · claim prize

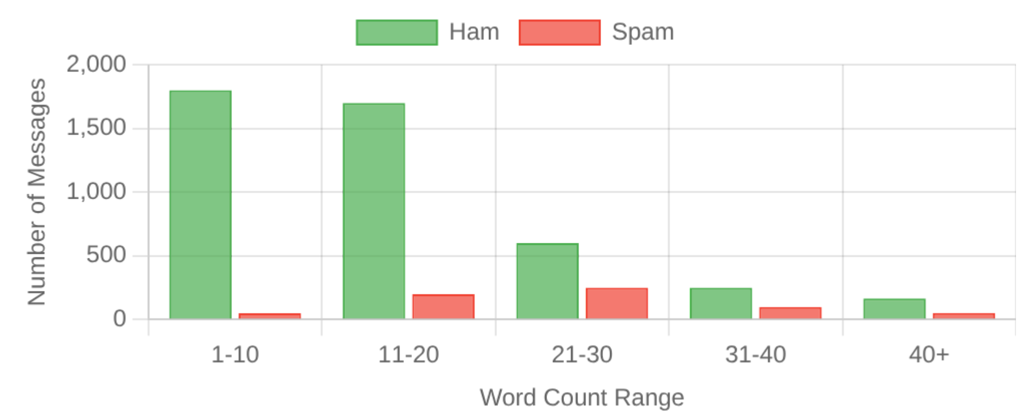
Ham Bigrams:

let know · go home · come home · good night · ill call

Character Count Distribution



Word Count Distribution



Key Insights

- **Spam messages are longer** than ham messages
- Strong correlation between characters and words
- Class imbalance: 87.4% ham vs 12.6% spam
- **Recall matters** - missing spam is a bigger risk

Text Preprocessing & Feature Engineering

Text Transformation Steps

1. **Lowercase**
   Convert all text to lowercase
2. **Tokenization**
   Split text into individual words
3. **Remove Special Characters**
   Keep only alphanumeric characters
4. **Remove Stop Words & Punctuation**
   Filter out common words and punctuation
5. **Stemming**
   Reduce words to root form (e.g., dancing → danc)

Word Clouds

Spam Messages Word Cloud

Most Common Words

Spam: free, call, text, mobile, claim, prize, win, urgent

free · call · text · mobile · claim · prize · win · urgent

Ham: go, get, like, know, love, come, time, good

go · get · like · know · love · come · time · good

TF-IDF Vectorization

- Transformed text into numerical features
- Limited to **top 3,000 most frequent words**
- Created feature matrix of shape **(5169, 3000)**

5169	3000
Messages	Features

```
from sklearn.feature_extraction.text import
TfidfVectorizer
tfidf = TfidfVectorizer(max_features=3000)
X =
tfidf.fit_transform(df['transformed_text']).toarray()
```

Model Building & Evaluation

Models Tested

Naive Bayes
97.1% Accuracy
100% Precision

SVM
97.6% Accuracy
97.5% Precision

Random Forest
97.4% Accuracy
98.3% Precision

Logistic Regression
95.6% Accuracy
96.0% Precision

Key Findings

- **Multinomial Naive Bayes** achieved perfect precision
- **TF-IDF** with max_features=3000 performed best
- **Precision prioritized** over recall to minimize false positives
- Ensemble methods improved overall performance

Model Performance Comparison

Best Model: Multinomial Naive Bayes

100%
Precision
No false positives

78.3%
Recall
Some spam missed

	Predicted Spam
Actual Spam	108
30	

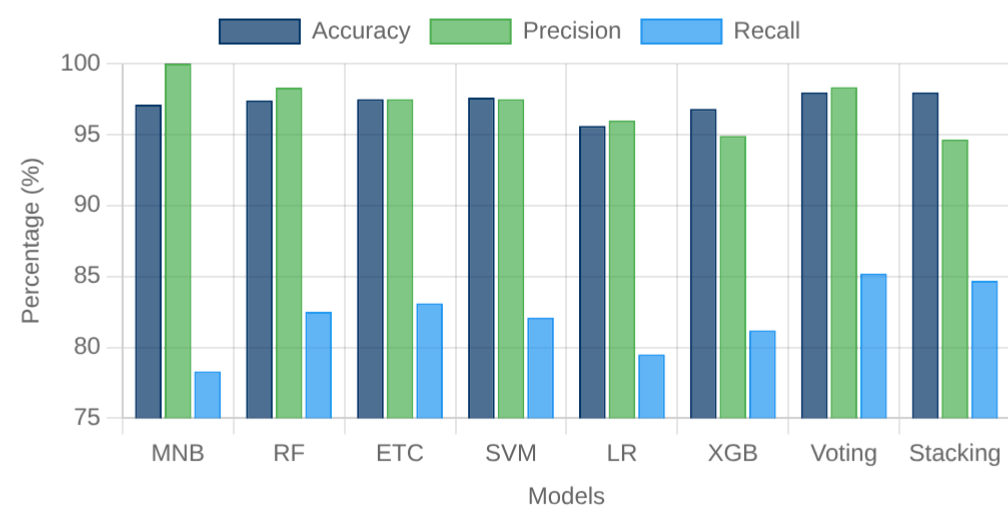
97.1%
Accuracy

100%
Precision

```
from sklearn.naive_bayes
import MultinomialNB
mnb = MultinomialNB()
mnb.fit(X_train, y_train)
y_pred = mnb.predict(X_test)
```

Model Comparison & Selection

Performance Comparison



Ensemble Methods

Voting Classifier

Combines SVM, Naive Bayes, and Extra Trees

97.97% Accuracy

98.35% Precision

Stacking Classifier

Uses base models with Random Forest meta-learner

97.97% Accuracy

94.66% Precision

Top Performing Models

Model	Accuracy	Precision	Recall
Multinomial NB	97.1%	100%	78.3%
Random Forest	97.4%	98.3%	82.5%
Extra Trees	97.5%	97.5%	83.1%
Voting Classifier	97.97%	98.35%	85.2%

Final Model Selection

- Voting Classifier selected for deployment
- Highest precision (98.35%) to minimize false positives
- Better balance between precision and recall

97.97% Accuracy

98.35% Precision

Model Trade-offs

MNB
100% precision
Lower recall

Voting
98.35% precision
Better robustness

Model Deployment

Streamlit App Overview

Real-time Detection | User-friendly | No Installation | Web Accessible

App Functionality

Message Input

Text area for user messages

Preprocessing

Automatic text cleaning

Prediction

Model inference

Result Display

Clear classification

Technical Implementation

- Pickle for model saving/loading
- Replicated preprocessing in app
- TF-IDF for feature extraction

```
# Load model and vectorizer
tfidf = pickle.load(open('artifacts/vectorizer.pkl',
'rb'))
model = pickle.load(open('artifacts/model.pkl',
'rb'))
```

App Interface

SMS Spam Classifier

✍ Type your message here:

Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's

Predict

This message is SPAM.

Tech Stack

Python | Streamlit | Scikit-learn | Pickle | NLTK

Key Benefits

- Real-time spam detection
- No installation required
- Browser accessible
- Consistent preprocessing

Conclusion & Future Work

Project Achievements

High-Performance Model
Voting Classifier with 97.97% accuracy and 98.35% precision

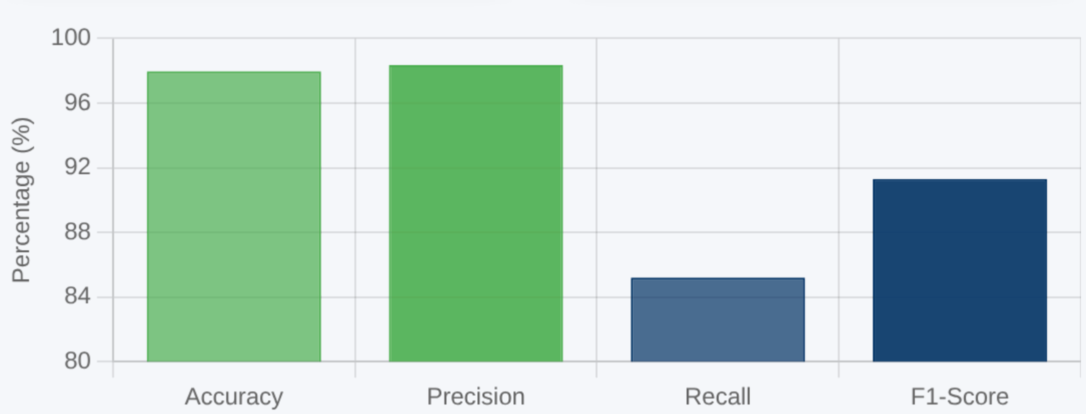
Key Insights
Spam messages are longer and contain specific keywords like "free", "win", "call"

Functional Deployment
Interactive Streamlit app for real-time spam detection

Model Performance

97.97%
Accuracy

98.35%
Precision



Future Improvements

Expand dataset
with more diverse spam examples
Improve model generalization across different spam types

Advanced NLP techniques
like BERT embeddings
Better semantic understanding of message context

Adversarial spam detection
(obfuscated text, emojis)
Handle evolving spam tactics and evasion techniques

Mobile integration
and scaling to millions of SMS
Real-time processing at scale with minimal latency

Key Takeaways

- **TF-IDF** with Multinomial Naive Bayes is highly effective
- **Ensemble methods** improve overall performance
- **Precision-focused** approach minimizes false positives
- **Streamlit** enables quick deployment and user interaction

Thank You!

Created by: Nisha Singla

Project Pipeline

End-to-End Workflow for SMS Spam Detection

Data Collection → Data Cleaning → EDA → Feature Engineering → Model Building → Evaluation → Deployment

Data & Cleaning

- 5,572SMS messages
- Remove duplicates
- Handle missing values
- Label encoding

EDA & Features

- Message length analysis
- Word frequency
- TF-IDFvectorization
- N-gram analysis

Modeling

- Multiple algorithms
- Voting Classifier
- Precision focus
- 98.35%precision

Deployment

- Streamlitapp
- Real-time prediction
- User-friendly interface
- Model persistence