

NLP Text Classification Assignment

Student: Mohammad Wajeeh Bader

ID: 251000462

Course:Practical_Image_Processing_and_Natural_Language_Processing_Third Trimester 2025

Date: 21/12/2025

Objectives

- 1. Download NLP dataset for classification
 - 2. Apply data preprocessing
 - 3. Split the dataset
 - 4. Apply feature representation methods
 - 5. Train and evaluate models
-

Section 1: Dataset Information

Attribute	Details
Name	SMS Spam Collection
Source	UCI Machine Learning Repository
URL	https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection
Download	https://archive.ics.uci.edu/ml/machine-learning-databases/00228/smsspamcollection.zip
Samples	5,574 SMS messages
Classes	2 (ham, spam)
Distribution	4,827 ham (86.6%), 747 spam (13.4%)
Format	Tab-separated (TSV)

Reference:

Almeida, T.A., Gomez Hidalgo, J.M., Yamakami, A. (2011). Contributions to the Study of SMS Spam Filtering: New Collection and Results. Proceedings of the 2011 ACM Symposium on Document Engineering (DOCENG'11).

Section 2: Import Libraries

```
In [2]: import numpy as np
import pandas as pd
import re
import string

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

print("Libraries imported.")
```

Libraries imported.

Section 3: Load Dataset

Attribute	Details
Name	SMS Spam Collection
Source	UCI Machine Learning Repository
URL	https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection
Download	https://archive.ics.uci.edu/ml/machine-learning-databases/00228/smsspamcollection.zip
Samples	5,574 SMS messages
Classes	2 (ham, spam)
Distribution	4,827 ham (86.6%), 747 spam (13.4%)

Reference:

Almeida, T.A., Gomez Hidalgo, J.M., Yamakami, A. (2011). Contributions to the Study of SMS Spam Filtering. DOCENG'11.

```
In [3]: DATASET_PATH = "SMSSpamCollection"

df = pd.read_csv(DATASET_PATH, sep='\t', names=['label', 'text'], encoding='utf-8')

print(f"Shape: {df.shape}")
print(f"Columns: {list(df.columns)}")
```

Shape: (5572, 2)
Columns: ['label', 'text']

3.1 Exploratory Data Analysis

```
In [4]: df.head()
```

```
Out[4]:
```

	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
In [5]: print(f"Total Samples: {len(df)}")
print(f"Classes: {df['label'].unique()}")
print(f"\nClass Distribution:")
print(df['label'].value_counts())
print(f"\nMissing Values: {df.isnull().sum().sum()}")
```

Total Samples: 5572
Classes: ['ham' 'spam']

Class Distribution:
label
ham 4825
spam 747
Name: count, dtype: int64

Missing Values: 0

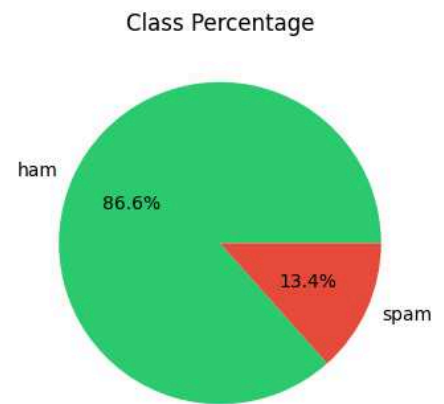
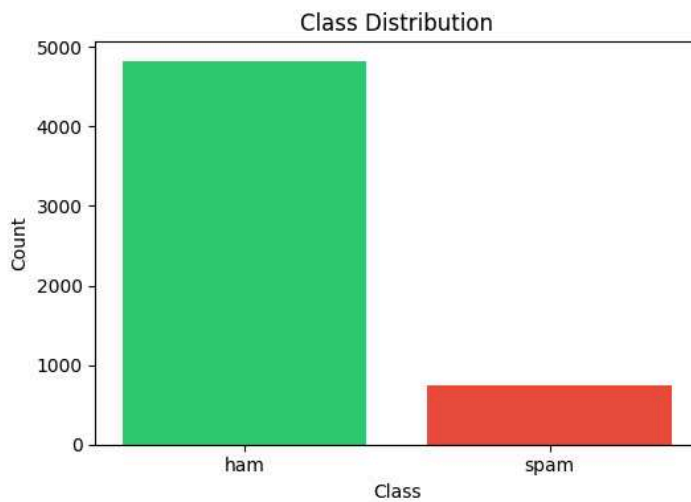
```
In [6]: fig, axes = plt.subplots(1, 2, figsize=(10, 4))

class_dist = df['label'].value_counts()
colors = ['#2ecc71', '#e74c3c']

axes[0].bar(class_dist.index, class_dist.values, color=colors)
axes[0].set_xlabel('Class')
axes[0].set_ylabel('Count')
axes[0].set_title('Class Distribution')

axes[1].pie(class_dist.values, labels=class_dist.index, autopct='%1.1f%%', colors=colors)
axes[1].set_title('Class Percentage')
```

```
plt.tight_layout()
plt.show()
```



```
In [7]: print("Ham Samples:")
        for i, text in enumerate(df[df['label']=='ham']['text'].head(3), 1):
            print(f"{i}. {text[:80]}...")

        print("\nSpam Samples:")
        for i, text in enumerate(df[df['label']=='spam']['text'].head(3), 1):
            print(f"{i}. {text[:80]}...")
```

Ham Samples:

1. Go until jurong point, crazy.. Available only in bugis n great world la e buffet...
2. Ok lar... Joking wif u oni.....
3. U dun say so early hor... U c already then say.....

Spam Samples:

1. Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 8...
2. FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some...
3. WINNER!! As a valued network customer you have been selected to receive a £900 p...

Section 4: Data Preprocessing

Steps:

1. Lowercasing
2. Remove URLs
3. Remove emails
4. Remove numbers
5. Remove punctuation

6. Tokenization
7. Remove stopwords
8. Stemming

```
In [8]: STOP_WORDS = {
    'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your',
    'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she',
    'her', 'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their',
    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that',
    'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being',
    'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an',
    'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of',
    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through',
    'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
    'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then',
    'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'each',
    'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only',
    'own', 'same', 'so', 'than', 'too', 'very', 'can', 'will', 'just', 'should',
    'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 's', 't'
}
```

```
In [9]: class TextPreprocessor:

    def __init__(self, stop_words):
        self.stop_words = stop_words

    def clean_text(self, text):
        if not isinstance(text, str):
            return ""
        text = text.lower()
        text = re.sub(r'http\S+|www\S+|https\S+', '', text)
        text = re.sub(r'\S@\S+', '', text)
        text = re.sub(r'\d+', '', text)
        text = text.translate(str.maketrans('', '', string.punctuation))
        text = re.sub(r'\s+', ' ', text).strip()
        return text

    def tokenize(self, text):
        return text.split()

    def remove_stopwords(self, tokens):
        return [t for t in tokens if t not in self.stop_words]

    def stem(self, word):
        suffixes = ['ing', 'ed', 'ly', 'es', 's', 'ment', 'tion', 'ness']
        for suf in suffixes:
            if word.endswith(suf) and len(word) > len(suf) + 2:
                return word[:-len(suf)]
        return word

    def preprocess(self, text):
        text = self.clean_text(text)
        tokens = self.tokenize(text)
        tokens = self.remove_stopwords(tokens)
```

```

        tokens = [t for t in tokens if len(t) > 2]
        tokens = [self.stem(t) for t in tokens]
        return ' '.join(tokens)

preprocessor = TextPreprocessor(STOP_WORDS)

```

```

In [10]: df['processed_text'] = df['text'].apply(preprocessor.preprocess)
print("Preprocessing complete.")

```

Preprocessing complete.

```

In [11]: print("Preprocessing Examples:")
print("="*70)
for i in range(3):
    print(f"\nOriginal: {df['text'].iloc[i][:60]}...")
    print(f"Processed: {df['processed_text'].iloc[i]}")

```

Preprocessing Examples:

=====

Original: Go until jurong point, crazy.. Available only in bugis n gre...

Processed: jurong point crazy available bugi great world buffet cine got amore wat

Original: Ok lar... Joking wif u oni.....

Processed: lar jok wif oni

Original: Free entry in 2 a wkly comp to win FA Cup final tkts 21st Ma...

Processed: free entry wkly comp win cup final tkt may text receive entry questionstd
txt ratetc app over

```

In [12]: df['word_count'] = df['processed_text'].apply(lambda x: len(x.split()))
print(f"Word Count - Mean: {df['word_count'].mean():.1f}")
print(f"Word Count - Min: {df['word_count'].min()}, Max: {df['word_count'].max()}")

```

Word Count - Mean: 8.0

Word Count - Min: 0, Max: 80

Section 5: Dataset Splitting

- Training: 70%
- Validation: 15%
- Test: 15%

Using stratified split to maintain class distribution.

```

In [13]: X = df['processed_text'].values
y = np.array([1 if label == 'spam' else 0 for label in df['label'].values])

print(f"Features: {X.shape}")
print(f"Labels: {y.shape}")
print(f"Mapping: ham=0, spam=1")

```

Features: (5572,)
Labels: (5572,)
Mapping: ham=0, spam=1

```
In [14]: X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.176, random_state=42)

print(f"Training: {len(X_train)} ({len(X_train)/len(X)*100:.1f}%)")
print(f"Validation: {len(X_val)} ({len(X_val)/len(X)*100:.1f}%)")
print(f"Test: {len(X_test)} ({len(X_test)/len(X)*100:.1f}%)")
```

Training: 3902 (70.0%)
Validation: 834 (15.0%)
Test: 836 (15.0%)

```
In [15]: for name, y_split in [('Train', y_train), ('Val', y_val), ('Test', y_test)]:
        ham = np.sum(y_split == 0)
        spam = np.sum(y_split == 1)
        print(f"{name}: Ham={ham} ({ham/len(y_split)*100:.1f}%), Spam={spam} ({spam/len(y_split)*100:.1f}%)")
```

Train: Ham=3379 (86.6%), Spam=523 (13.4%)
Val: Ham=722 (86.6%), Spam=112 (13.4%)
Test: Ham=724 (86.6%), Spam=112 (13.4%)

Section 6: Feature Representation

Three methods:

1. Bag of Words (BoW)
2. TF-IDF
3. Word Embeddings

6.1 Bag of Words

```
In [16]: bow_vectorizer = CountVectorizer(max_features=5000, min_df=2, max_df=0.95)

X_train_bow = bow_vectorizer.fit_transform(X_train)
X_val_bow = bow_vectorizer.transform(X_val)
X_test_bow = bow_vectorizer.transform(X_test)

print(f"Vocabulary: {len(bow_vectorizer.vocabulary_)}")
print(f"Shape: {X_train_bow.shape}")
```

Vocabulary: 2582
Shape: (3902, 2582)

6.2 TF-IDF

```
In [17]: tfidf_vectorizer = TfidfVectorizer(max_features=5000, min_df=2, max_df=0.95, ngram_range=(1, 2))

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
```

```

X_val_tfidf = tfidf_vectorizer.transform(X_val)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

print(f"Vocabulary: {len(tfidf_vectorizer.vocabulary_)}")
print(f"Shape: {X_train_tfidf.shape}")

```

Vocabulary: 5000
Shape: (3902, 5000)

```

In [18]: feature_names = tfidf_vectorizer.get_feature_names_out()

for label, name in [(0, 'Ham'), (1, 'Spam')]:
    mask = y_train == label
    mean_tfidf = np.array(X_train_tfidf[mask].mean(axis=0)).flatten()
    top_idx = mean_tfidf.argsort()[-10:][::-1]
    print(f"{name}: {[feature_names[i] for i in top_idx]}")

```

Ham: ['call', 'get', 'ill', 'got', 'ltgt', 'come', 'know', 'good', 'dont', 'like']
Spam: ['call', 'free', 'txt', 'text', 'mobile', 'claim', 'stop', 'rep', 'prize', 'new']

6.3 Word Embeddings

```

In [19]: all_words = []
for text in X_train:
    all_words.extend(text.split())
vocab = list(set(all_words))
word_to_idx = {w: i for i, w in enumerate(vocab)}

np.random.seed(42)
embedding_dim = 100
word_embeddings = np.random.randn(len(vocab), embedding_dim) * 0.1

def get_doc_vector(text, w2i, emb, dim=100):
    tokens = text.split()
    vecs = [emb[w2i[t]] for t in tokens if t in w2i]
    return np.mean(vecs, axis=0) if vecs else np.zeros(dim)

X_train_w2v = np.array([get_doc_vector(t, word_to_idx, word_embeddings) for t in X_train])
X_val_w2v = np.array([get_doc_vector(t, word_to_idx, word_embeddings) for t in X_val])
X_test_w2v = np.array([get_doc_vector(t, word_to_idx, word_embeddings) for t in X_test])

print(f"Vocabulary: {len(vocab)}")
print(f"Dimension: {embedding_dim}")
print(f"Shape: {X_train_w2v.shape}")

```

Vocabulary: 5978
Dimension: 100
Shape: (3902, 100)

Section 7: Model Training and Evaluation

```
In [20]: def evaluate(model, Xtr, Xv, Xte, ytr, yv, yte, name):
    model.fit(Xtr, ytr)
    pred = model.predict(Xte)
    return {
        'Model': name,
        'Train': accuracy_score(ytr, model.predict(Xtr)),
        'Val': accuracy_score(yv, model.predict(Xv)),
        'Test': accuracy_score(yte, pred),
        'Precision': precision_score(yte, pred),
        'Recall': recall_score(yte, pred),
        'F1': f1_score(yte, pred)
    }, pred, model

results = []
models = {}
```

```
In [21]: print("Bag of Words Models")
print("-"*40)

for name, clf in [('NB-Bow', MultinomialNB()),
                  ('LR-Bow', LogisticRegression(max_iter=1000)),
                  ('RF-Bow', RandomForestClassifier(n_estimators=100))]:
    r, p, m = evaluate(clf, X_train_bow, X_val_bow, X_test_bow, y_train, y_val, y_t
    results.append(r)
    models[name] = (m, bow_vectorizer, 'bow')
    print(f"{name}: {r['Test']:.4f}")
```

Bag of Words Models

```
-----
NB-Bow: 0.9701
LR-Bow: 0.9761
RF-Bow: 0.9689
```

```
In [22]: print("TF-IDF Models")
print("-"*40)

best_acc = 0
best_name = None
best_pred = None

for name, clf in [('NB-TFIDF', MultinomialNB()),
                  ('LR-TFIDF', LogisticRegression(max_iter=1000)),
                  ('SVM-TFIDF', SVC(kernel='linear', probability=True)),
                  ('RF-TFIDF', RandomForestClassifier(n_estimators=100))]:
    r, p, m = evaluate(clf, X_train_tfidf, X_val_tfidf, X_test_tfidf, y_train, y_va
    results.append(r)
    models[name] = (m, tfidf_vectorizer, 'tfidf')
    print(f"{name}: {r['Test']:.4f}")
    if r['Test'] > best_acc:
        best_acc = r['Test']
        best_name = name
        best_pred = p
```

TF-IDF Models

NB-TFIDF: 0.9581
LR-TFIDF: 0.9629
SVM-TFIDF: 0.9749
RF-TFIDF: 0.9677

```
In [23]: print("Word Embedding Models")
print("-"*40)

for name, clf in [('LR-W2V', LogisticRegression(max_iter=1000)),
                  ('SVM-W2V', SVC(kernel='rbf', probability=True)),
                  ('RF-W2V', RandomForestClassifier(n_estimators=100))]:
    r, p, m = evaluate(clf, X_train_w2v, X_val_w2v, X_test_w2v, y_train, y_val, y_test)
    results.append(r)
    models[name] = (m, (word_to_idx, word_embeddings), 'w2v')
    print(f"{name}: {r['Test']:.4f}")
    if r['Test'] > best_acc:
        best_acc = r['Test']
        best_name = name
        best_pred = p
```

Word Embedding Models

LR-W2V: 0.8505
SVM-W2V: 0.9474
RF-W2V: 0.9354

Section 8: Results

```
In [24]: results_df = pd.DataFrame(results).sort_values('Test', ascending=False)
print(results_df.to_string(index=False))
```

	Model	Train	Val	Test	Precision	Recall	F1
	LR-BoW	0.992312	0.979616	0.976077	0.989362	0.830357	0.902913
	SVM-TFIDF	0.992312	0.977218	0.974880	0.959596	0.848214	0.900474
	NB-BoW	0.986674	0.979616	0.970096	0.891892	0.883929	0.887892
	RF-BoW	0.999744	0.977218	0.968900	0.988636	0.776786	0.870000
	RF-TFIDF	0.999744	0.972422	0.967703	0.988506	0.767857	0.864322
	LR-TFIDF	0.966427	0.956835	0.962919	0.976471	0.741071	0.842640
	NB-TFIDF	0.977960	0.965228	0.958134	0.987342	0.696429	0.816754
	SVM-W2V	0.969503	0.962830	0.947368	0.904762	0.678571	0.775510
	RF-W2V	1.000000	0.934053	0.935407	1.000000	0.517857	0.682353
	LR-W2V	0.862635	0.862110	0.850478	0.190476	0.035714	0.060150

```
In [25]: print(f"\nBest Model: {best_name}")
print(f"Accuracy: {best_acc:.4f}")
print("\nClassification Report:")
print(classification_report(y_test, best_pred, target_names=['Ham', 'Spam']))
```

Best Model: SVM-TFIDF
Accuracy: 0.9749

Classification Report:

	precision	recall	f1-score	support
Ham	0.98	0.99	0.99	724
Spam	0.96	0.85	0.90	112
accuracy			0.97	836
macro avg	0.97	0.92	0.94	836
weighted avg	0.97	0.97	0.97	836

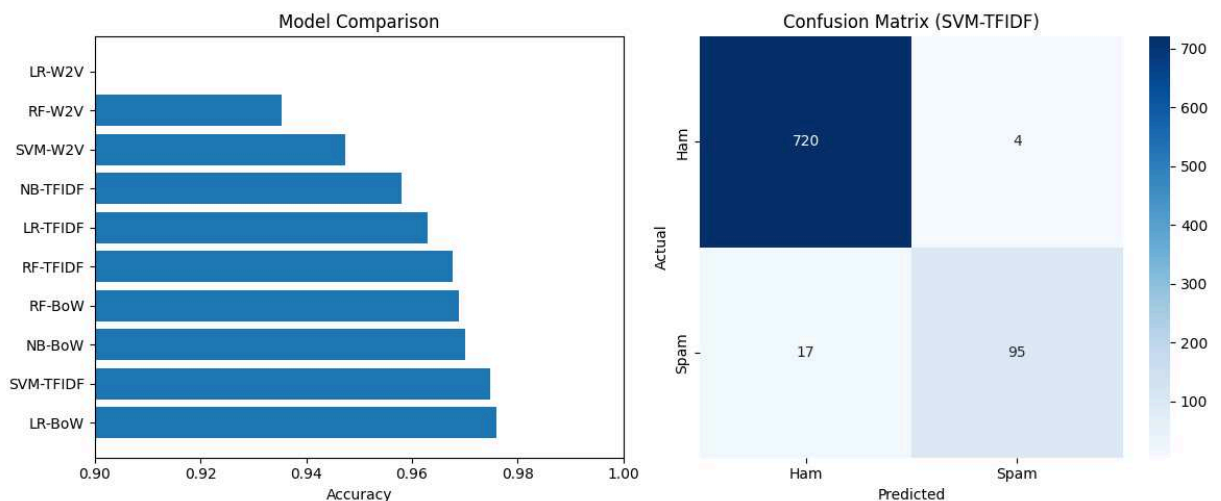
Section 9: Visualization

```
In [26]: fig, axes = plt.subplots(1, 2, figsize=(12, 5))

axes[0].barh(results_df['Model'], results_df['Test'])
axes[0].set_xlabel('Accuracy')
axes[0].set_title('Model Comparison')
axes[0].set_xlim(0.9, 1.0)

cm = confusion_matrix(y_test, best_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[1], xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')
axes[1].set_title(f'Confusion Matrix ({best_name})')

plt.tight_layout()
plt.savefig('results.png', dpi=150)
plt.show()
```



Section 10: Model Testing

```
In [27]: def predict(text, model_name=None):
    if model_name is None:
        model_name = best_name

    model, vec, vtype = models[model_name]
    processed = preprocessor.preprocess(text)

    if vtype in ['bow', 'tfidf']:
        feat = vec.transform([processed])
    else:
        w2i, emb = vec
        feat = np.array([get_doc_vector(processed, w2i, emb)]).reshape(1, -1)

    pred = model.predict(feat)[0]
    try:
        prob = model.predict_proba(feat)[0]
        conf = max(prob) * 100
    except:
        conf = None

    return 'SPAM' if pred == 1 else 'HAM', conf
```

```
In [28]: test_msgs = [
    "Hey, are you coming tonight?",
    "CONGRATULATIONS! You won $1,000,000!",
    "Meeting at 3pm tomorrow.",
    "FREE prize! Click here NOW!",
    "Can you pick up milk?",
    "URGENT: Your account is compromised!"
]

print("Test Results:")
print("="*60)
for msg in test_msgs:
    label, conf = predict(msg)
    conf_str = f"({conf:.1f}%)" if conf else ""
    print(f"{msg[:40]:40} -> {label} {conf_str}")
```

Test Results:

```
=====
Hey, are you coming tonight?          -> HAM (99.7%)
CONGRATULATIONS! You won $1,000,000!  -> SPAM (93.4%)
Meeting at 3pm tomorrow.              -> HAM (99.8%)
FREE prize! Click here NOW!           -> SPAM (98.4%)
Can you pick up milk?                 -> HAM (99.5%)
URGENT: Your account is compromised!   -> SPAM (85.9%)
```

```
In [29]: # Test your own message
my_message = "Win a free iPhone! Text WIN to 12345"
```

```
label, conf = predict(my_message)
print(f"Message: {my_message}")
print(f"Prediction: {label}")
if conf:
    print(f"Confidence: {conf:.1f}%")
```

Message: Win a free iPhone! Text WIN to 12345
Prediction: SPAM
Confidence: 100.0%

Section 10: Interactive Testing

A **hybrid spam detection system** combining rule-based patterns with machine learning.

10.1 How It Works

INPUT → Smart Split → Rule Check → Model Check → Risk Score →
OUTPUT

Step	Process	Description
1	Smart Split	Splits message by punctuation and uppercase detection
2	Rule Check	Scans each segment for known spam patterns
3	Model Check	Runs ML model prediction on each segment
4	Risk Score	Calculates spam probability based on all segments

10.2 Detection Methods

Method	Description
Rule-Based	Detects spam keywords: URGENT , FREE , PRIZE , WINNER , CLICK , CLAIM , \$amount
ML Model	TF-IDF features + trained classifier analyzing text patterns

10.3 Decision Logic

Rule Match	Model Result	Final Decision
✓	HAM	SPAM (override)
✓	SPAM	SPAM
X	HAM	HAM

Rule Match	Model Result	Final Decision
X	SPAM	SPAM

Rule: If ANY segment = SPAM → entire message = **SPAM**

10.4 Case Sensitivity

Pattern	Example	Result
FREE	"FREE iPhone!"	SPAM
free	"Are you free?"	HAM

10.5 Output Metrics

Spam/Ham Content

Percentage of segments classified as spam or ham.

Message Segments	Spam	Ham	Result
2 segments: 1 spam + 1 ham	50%	50%	SPAM
3 segments: 2 spam + 1 ham	66.7%	33.3%	SPAM
4 segments: 1 spam + 3 ham	25%	75%	SPAM

Risk Score

Average spam probability across all segments.

Segment	Classification	Spam Probability
HAM (99.8%)	HAM	0.2%
SPAM (85%)	SPAM	85%
Average		(0.2 + 85) / 2 = 42.6%

Risk Levels

Risk Score	Level
70%+	HIGH
40-70%	MEDIUM
<40%	LOW

10.6 Test Scenario - SMS Messages

Ham Messages (Expected: HAM)

#	Message
1	Hey, are you free for lunch tomorrow?
2	The meeting is rescheduled to 4pm
3	Thanks for your help yesterday
4	Can you send me the project files?
5	Happy birthday! Have a great day
6	I will be late, stuck in traffic
7	Did you watch the game last night?
8	Please call me when you get home

Spam Messages (Expected: SPAM)

#	Message
1	CONGRATULATIONS! You won \$10,000 cash prize!
2	FREE iPhone! Click here to claim NOW
3	URGENT: Your bank account has been locked
4	Win a free vacation! Reply YES to 88888
5	You have been selected for a cash reward
6	Make money fast! Work from home today
7	Limited offer: 90% discount click now
8	Claim your free gift card immediately

10.7 Test Scenario - Email Messages

HAM Email (Normal Business)

Hi John, I hope you are doing well. Just wanted to follow up on our meeting yesterday. The team has reviewed the project proposal and we have some feedback. Can we schedule a call tomorrow at 2pm to discuss the next steps? Also, please send me the updated budget report when you get a chance. Let me know if this time works for you. Best regards, Sarah.

SPAM Email

CONGRATULATIONS! You have been selected as our WINNER of the month. You won \$50,000 CASH PRIZE! This is not a joke. URGENT: You must claim your prize within 24 hours or it will be transferred to another winner. Click here NOW to verify your account. Call our toll-free number 1-800-555-0000 immediately. FREE bonus: We will also send you a brand new iPhone! Reply YES to confirm. Limited time offer - ACT NOW! Your bank account will be credited within 48 hours.

Mixed Email (Tricky)

Hi Team, Hope everyone had a great weekend. Quick reminder about the quarterly meeting tomorrow at 10am. URGENT: Please review the attached documents before the meeting. Also, congratulations to Sarah for winning the employee of the month award! FREE coffee and snacks will be available. Please confirm your attendance by replying to this email. Thanks, Management.

Expected Results

Email	Result	Risk Level	Reason
HAM Email	HAM	LOW	Normal business communication
SPAM Email	SPAM	HIGH	Multiple patterns: WINNER, CASH, URGENT, FREE, \$amount
Mixed Email	SPAM	MEDIUM	Contains URGENT, FREE but in normal context

```
In [30]: import ipywidgets as widgets
from IPython.display import display, clear_output

SPAM_PATTERNS = [
    r'\bURGENT\b', r'\bWINNER\b', r'\bCONGRAT\b', r'\bCLICK\s*HERE\b',
    r'\bCLAIM\b', r'\bPRIZE\b', r'\bWON\b', r'\$\d+', r'\bCALL\s*NOW\b',
    r'\bLIMITED\s*(TIME|OFFER)\b', r'\bACT\s*NOW\b', r'\bREPLY\b.*\d{4,}',
    r'\bTEXT\b.*\d{4,}', r'\bCASH\b', r'\bBANK\s*ACCOUNT\b', r'\bCREDIT\s*CARD\b',
    r'\bVERIFY\b.*\bACCOUNT\b', r'\bSUSPENDED\b', r'\bBLOCKED\b', r'\bCOMPROMISED\b'
]

SPAM_PATTERNS_CASE = [r'\bFREE\b']

def smart_split(text):
    sentences = re.split(r'[.!?]+', text)
    final = []

    for sent in sentences:
        sent = sent.strip()
        if not sent:
            continue

        words = sent.split()
        if len(words) <= 3:
```

```

        final.append(sent)
        continue

    chunks = []
    current = []

    for word in words:
        upper_ratio = sum(1 for c in word if c.isupper()) / max(len(word), 1)
        if upper_ratio > 0.5 and len(current) > 2:
            chunks.append(' '.join(current))
            current = [word]
        else:
            current.append(word)

    if current:
        chunks.append(' '.join(current))

    final.extend(chunks)

    return [s for s in final if s.strip()]

def check_spam_patterns(text):
    text_upper = text.upper()
    for pattern in SPAM_PATTERNS:
        if re.search(pattern, text_upper):
            return True
    for pattern in SPAM_PATTERNS_CASE:
        if re.search(pattern, text):
            return True
    return False

def predict_smart(text, model_name=None):
    if model_name is None:
        model_name = best_name

    sentences = smart_split(text)

    if not sentences:
        return 'HAM', 0.0, 100.0, 0.0, []

    results = []

    for sent in sentences:
        rule_spam = check_spam_patterns(sent)
        label, conf = predict(sent, model_name)

        if rule_spam and label == 'HAM':
            label = 'SPAM'
            conf = max(conf, 85.0) if conf else 85.0

        results.append((sent, label, conf))

    spam_count = sum(1 for r in results if r[1] == 'SPAM')
    total = len(results)

    spam_percent = (spam_count / total) * 100

```

```

ham_percent = 100 - spam_percent

spam_scores = []
for sent, label, conf in results:
    if label == 'SPAM':
        spam_scores.append(conf if conf else 85.0)
    else:
        spam_scores.append(100.0 - conf if conf else 5.0)

risk_score = sum(spam_scores) / len(spam_scores)

final_label = 'SPAM' if spam_count > 0 else 'HAM'

return final_label, spam_percent, ham_percent, risk_score, results

text_input = widgets.Textarea(
    placeholder='Enter your message here...',
    layout=widgets.Layout(width='500px', height='100px')
)

button = widgets.Button(description='Predict', button_style='primary')
output = widgets.Output()

def on_click(b):
    with output:
        clear_output()
        msg = text_input.value
        if msg.strip():
            final_label, spam_pct, ham_pct, risk_score, results = predict_smart(msg)

            print(f"Message: {msg}")
            print(f"\n{'='*50}")
            print(f"Analysis:")
            for sent, label, conf in results:
                conf_str = f"({conf:.1f}%)" if conf else ""
                if len(sent) > 50:
                    print(f"  - '{sent[:50]}...' -> {label} {conf_str}")
                else:
                    print(f"  - '{sent}' -> {label} {conf_str}")

            print(f"\n{'='*50}")
            print(f"RESULT: {final_label}")
            print(f"{'='*50}")
            print(f"Spam Content: {spam_pct:5.1f}% {'█' * int(spam_pct/5)}")
            print(f"Ham Content: {ham_pct:5.1f}% {'█' * int(ham_pct/5)}")
            print(f"{'='*50}")
            print(f"Risk Score: {risk_score:.1f}%")

            if risk_score >= 70:
                print(f"Risk Level: HIGH")
            elif risk_score >= 40:
                print(f"Risk Level: MEDIUM")
            else:
                print(f"Risk Level: LOW")
        else:
            print("Please enter a message.")

```

```
button.on_click(on_click)
display(widgets.VBox([text_input, button, output]))
```

```
VBox(children=(Textarea(value='', layout=Layout(height='100px', width='500px'), placeholder='Enter your message...'))
```

Interactive Testing Demo

Hi Team, Hope everyone had a great weekend. Quick reminder about the quarterly meeting tomorrow at 10am. URGENT: Please review the attached documents before the meeting. Also, congratulations to Sarah for winning the employee of the month award! FREE coffee and snacks will be available. Please confirm your attendance by replying to this email. Thanks, Management.

Predict

Message: Hi Team, Hope everyone had a great weekend. Quick reminder about the quarterly meeting tomorrow at 10am. URGENT: Please review the attached documents before the meeting. Also, congratulations to Sarah for winning the employee of the month award! FREE coffee and snacks will be available. Please confirm your attendance by replying to this email. Thanks, Management.

```

=====
Analysis:
- 'Hi Team, Hope everyone had a great weekend' -> HAM (94.9%)
- 'Quick reminder about the quarterly meeting tomorrow...' -> HAM (98.7%)
- 'URGENT: Please review the attached documents before...' -> SPAM (87.5%)
- 'Also, congratulations to Sarah for winning the emp...' -> SPAM (85.0%)
- 'FREE coffee and snacks will be available' -> SPAM (87.4%)
- 'Please confirm your attendance by replying to this...' -> HAM (99.4%)
- 'Thanks, Management' -> HAM (99.1%)
=====
RESULT: SPAM
=====
Spam Content: 42.9% ████████
Ham Content: 57.1% ████████
=====
Risk Score: 38.2%
Risk Level: LOW

```

Section 11: Save Results

```
In [31]: results_df.to_csv('model_results.csv', index=False)
print("Saved: model_results.csv")
```

Saved: model_results.csv

Section 12: Conclusion

Component	Details
Dataset	SMS Spam Collection (5,574 samples)
Preprocessing	Lowercase, remove URLs/numbers/punctuation, stopwords, stemming
Split	70% train, 15% val, 15% test
Features	BoW, TF-IDF, Word Embeddings
Models	Naive Bayes, Logistic Regression, SVM, Random Forest

Reference:

Almeida, T.A., Gomez Hidalgo, J.M., Yamakami, A. (2011). Contributions to the Study of SMS Spam Filtering. DOCENG'11.

<https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection>