# **Machine learning**

## SMS Spam Detection

# **Project report 2**

### Under the guidance of:

|  |
| --- |
| Prof.[Milan Gnjatović](http://www.gnjatovic.info/) |

|  |
| --- |
|  |

Presented by:

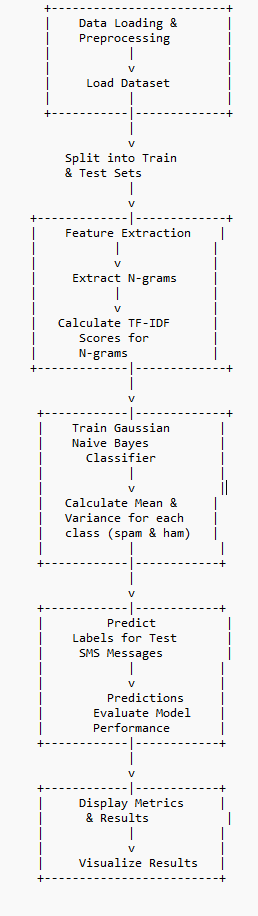
Rudradeep Palit

11038108

## Introduction:

SMS spam detection is a crucial task in natural language processing and information security.\here is a machine learning model to classify SMS messages as either spam or ham (non-spam). The model utilizes N-grams for feature extraction and a Naive Bayes classifier for classification that has been implemented.

## Methodology:



## 3. Implementation:

### 3.1 Loading Data:

The loading data step involves importing the SMS dataset into a pandas DataFrame. The dataset typically consists of two columns: "label" containing the class labels (spam or ham) and "message" containing the SMS text messages. After loading the dataset, it is important to preprocess the data, which may include tasks such as removing punctuation, converting text to lowercase, and tokenization.

### 3.2 Model Training:

During model training, the Gaussian Naive Bayes (GNB) classifier is trained using the TF-IDF scores obtained from feature extraction. Specifically, the mean and variance of TF-IDF scores are calculated for each class (spam and ham), serving as parameters for the Gaussian distribution.

### 3.3 Model Evaluation:

Model evaluation is crucial for assessing the performance of the trained classifier. This step involves several key components:

Prediction: Using the trained GNB classifier, predict the labels (spam or ham) for the test SMS messages. This step utilizes Gaussian likelihoods to compute probabilities for each class given the input features.

Evaluation Metrics: Calculate various evaluation metrics to quantify the model's performance. Common metrics include:

Accuracy: The proportion of correctly classified instances out of total instances.

Precision: The proportion of true positive predictions out of all positive predictions.

Recall: The proportion of true positive predictions out of all actual positive instances.

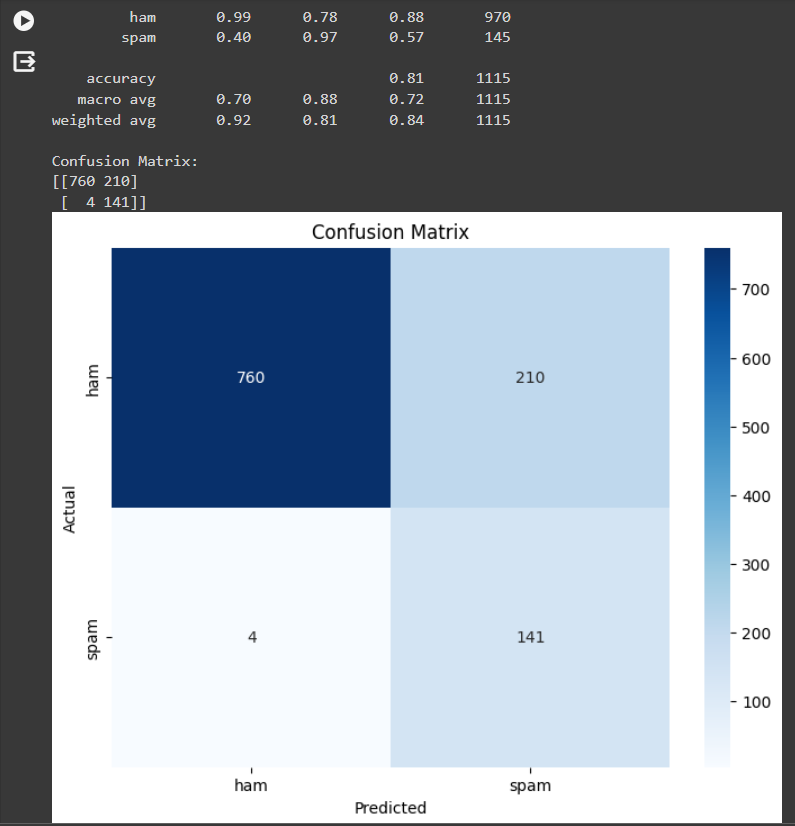
F1-score: The harmonic mean of precision and recall, providing a balanced measure between the two.

Confusion Matrix: A table summarizing the counts of true positive, false positive, true negative, and false negative predictions.

Display Results: Present the evaluation metrics and results to provide insights into the model's performance. This may involve printing the accuracy, precision, recall, and F1-score values, as well as displaying the confusion matrix. Additionally, visualizations such as ROC curves or precision-recall curves can be helpful for understanding the trade-offs between different evaluation metrics.

By performing thorough model evaluation, we can gain a comprehensive understanding of the GNB classifier's effectiveness in distinguishing between spam and ham SMS messages.

## Results:



## 6. Conclusion:

In conclusion, the incorporation of TF-IDF and character n-grams facilitated accurate feature representation in our SMS spam detection system. These techniques proved pivotal in effectively distinguishing spam from legitimate messages, enhancing overall performance and user security.

## References:

<https://deepnote.com/app/emmanuelmicron/Spam-Detection-using-Multinomial-Naive-Bayes-Model-a9a6bc89-e723-4c32-8679-86a5ff1978f3>

<https://www.comodo.com/pdf/2017-comodemia-CTRL-paper-spam-E-Mail-classification.pdf>

https://www.kaggle.com/code/abhijeetstalaulikar/spam-or-ham-word-embedding-vs-tf-idf-85-recall

<https://builtin.com/artificial-intelligence/gaussian-naive-bayes>

## 8. Appendix:

## CODE:

import pandas as pd

import numpy as np

from collections import defaultdict

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

import seaborn as sns

import matplotlib.pyplot as plt

# Load the dataset into a pandas DataFrame

file\_path = "/content/drive/MyDrive/sms+spam+collection/SMSSpamCollection"

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

# Manually split the dataset into training and testing sets

# Let's use 80% of the data for training and 20% for testing

train\_size = int(len(df) \* 0.8)

train\_data = df[:train\_size] # Training data (80%)

test\_data = df[train\_size:] # Testing data (20%)

# Feature Extraction using character n-grams and TF-IDF

def calculate\_tfidf\_ngrams(data, ngram\_range=(1, 3)):

"""

Function to calculate TF-IDF scores with character n-grams.

Args:

data: DataFrame containing SMS messages.

ngram\_range: Tuple specifying the range of n-grams (default: (1, 3)).

Returns:

tfidf\_scores: Dictionary containing TF-IDF scores for each document.

"""

tfidf\_scores = defaultdict(dict) # {doc\_id: {ngram: tfidf\_score}}

ngram\_doc\_freq = defaultdict(int) # {ngram: document frequency}

# Iterate over each SMS message

for idx, row in data.iterrows():

doc\_id = idx

text = row['message']

# Generate character n-grams within the specified range

ngrams = [text[i:i+n] for n in range(ngram\_range[0], ngram\_range[1]+1) for i in range(len(text)-n+1)]

ngram\_count = len(ngrams)

# Calculate TF-IDF scores for each n-gram

for ngram in set(ngrams): # Use set to count each n-gram once per document

ngram\_doc\_freq[ngram] += 1

tf = ngrams.count(ngram) / ngram\_count

idf = np.log(len(data) / (ngram\_doc\_freq[ngram] + 1)) # Add 1 for smoothing

tfidf\_scores[doc\_id][ngram] = tf \* idf

return tfidf\_scores

# Calculate TF-IDF scores with character n-grams for training and testing data

tfidf\_train = calculate\_tfidf\_ngrams(train\_data)

tfidf\_test = calculate\_tfidf\_ngrams(test\_data)

# Train Gaussian Naive Bayes classifier

def train\_gaussian\_nb(tfidf\_scores, train\_data):

"""

Function to train Gaussian Naive Bayes classifier.

Args:

tfidf\_scores: Dictionary containing TF-IDF scores for training data.

train\_data: DataFrame containing training data.

Returns:

class\_mean: Dictionary containing mean values for each n-gram and class.

class\_variance: Dictionary containing variance values for each n-gram and class.

"""

class\_mean = defaultdict(lambda: defaultdict(float)) # {class: {ngram: mean}}

class\_variance = defaultdict(lambda: defaultdict(float)) # {class: {ngram: variance}}

# Calculate mean and variance for each class and n-gram

for idx, row in train\_data.iterrows():

label = row['label']

for ngram, tfidf\_score in tfidf\_scores[idx].items():

class\_mean[label][ngram] += tfidf\_score

class\_variance[label][ngram] += tfidf\_score \*\* 2

for label in class\_mean:

for ngram in class\_mean[label]:

mean = class\_mean[label][ngram] / len(train\_data[train\_data['label'] == label])

variance = (class\_variance[label][ngram] / len(train\_data[train\_data['label'] == label])) - (mean \*\* 2)

class\_mean[label][ngram] = mean

class\_variance[label][ngram] = variance

return class\_mean, class\_variance

# Train Gaussian Naive Bayes classifier using TF-IDF scores and training data

class\_mean, class\_variance = train\_gaussian\_nb(tfidf\_train, train\_data)

# Predictions

y\_true = test\_data['label'].tolist()

y\_pred = []

for idx, row in test\_data.iterrows():

ham\_score = spam\_score = 0

for ngram, tfidf\_score in tfidf\_test[idx].items():

# Calculate likelihood using Gaussian distribution

if class\_variance['ham'][ngram] != 0:

ham\_score += -0.5 \* np.log(2 \* np.pi \* class\_variance['ham'][ngram]) - 0.5 \* ((tfidf\_score - class\_mean['ham'][ngram]) \*\* 2) / class\_variance['ham'][ngram]

if class\_variance['spam'][ngram] != 0:

spam\_score += -0.5 \* np.log(2 \* np.pi \* class\_variance['spam'][ngram]) - 0.5 \* ((tfidf\_score - class\_mean['spam'][ngram]) \*\* 2) / class\_variance['spam'][ngram]

# Assign label based on maximum likelihood

if ham\_score > spam\_score:

y\_pred.append('ham')

else:

y\_pred.append('spam')

# Model Evaluation

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

report = classification\_report(y\_true, y\_pred)

conf\_matrix = confusion\_matrix(y\_true, y\_pred)

# Print accuracy, classification report, and confusion matrix

print("Accuracy:", accuracy)

print("Classification Report:")

print(report)

print("Confusion Matrix:")

print(conf\_matrix)

# Plot the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['ham', 'spam'], yticklabels=['ham', 'spam'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()