**BUILDING A SMARTER AI POWERED SPAM CLASSIFIER**

INTRODUCTION:

An AI spam classifier is a machine learning system designed to automatically identify and filter out unsolicited or unwanted messages, typically found in emails, text messages, or other forms of digital communication. It uses various techniques, such as natural language processing (NLP) and pattern recognition, to analyze the content of messages and determine whether they are spam or legitimate. The goal of such a classifier is to help users avoid clutter and potential security risks by directing spam messages to a separate folder or blocking them altogether, ensuring a cleaner and safer communication experience.



PROBLEM STATEMENT:

The challenge is to construct an AI-powered spam classifier capable of accurately discerning between spam and non-spam messages in email or text messages. The objective is to minimize the occurrences of both false positives and false negatives, while simultaneously achieving a high degree of accuracy.

DESIGN THINKING AND PHASES OF DEVELOPMENT:

**1. Data Collection**: We will amass a substantial and diverse dataset of labeled spam and non-spam messages to train and evaluate our model.

**2. Data Preprocessing**: Rigorous data preprocessing will involve text cleaning, tokenization, and feature extraction, utilizing techniques like TF-IDF and word embeddings to represent text data effectively.

**3. Machine Learning Model**: We will employ state-of-the-art machine learning models, potentially including deep learning approaches like neural networks, recurrent neural networks (RNNs), or transformer models like BERT.

**4. Evaluation Metrics**: We will gauge the model's performance using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. Aiming to strike the right balance between false positives and false negatives, we may fine-tune the model accordingly.

**5. Cross-Validation**: To ensure robustness, we will employ cross-validation techniques, preventing overfitting and providing a realistic assessment of the model's generalization performance.

**6. Model Interpretability**: Implementing techniques for model interpretability, we will strive to understand the factors influencing model predictions and improve transparency.

**7. Deployment:** We will create a user-friendly interface and deploy the classifier as a web-based API, making it accessible for real-time email or text message classification.

**8**. **Continuous Monitoring**: Post-deployment, continuous monitoring will be established to track model performance, with automatic retraining triggered by performance degradation.

DATASET USED:

Kaggle, a popular data science platform, often has datasets related to email spam classification. You can search for relevant datasets on Kaggle and find user-contributed datasets.

**UNDERSTANDING THE DATASET:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | V1 | V2 | Unnamed:2 | Unnamed:3 | Unnamed:4 |
| 0 | Ham | Go until jurong point, crazy.. | Nan | Nan | Nan |
| 1 | Ham | Ok lar… | Nan | Nan | Nan |
| 2 | Spam | Free entry in 2 weekly comp to win.. | Nan | Nan | Nan |
| 3 | Ham | U dun say so early hor.. | Nan | Nan | Nan |
| 4 | Ham | Mah I don’t think he goes to USF… | Nan | Nan | Nan |
| …. | …. | ….. |  |  |  |
| 5568 | Ham | Will I\_b going to esplanade. | Nan | Nan | Nan |
| 5569 | Ham | Pitty, \*was in mood for that | Nan | Nan | Nan |
| 5570 | Ham | The guy did some bitching  But I acted like I’d… | Nan | Nan | Nan |
| 5571 | Ham | Rofl. It’s true to it’s name.. | Nan | Nan | Nan |

The columns in the data set are currently not named and as you can see, there are 2 columns. The first column takes two values, ‘ham’ which signifies that the message is not spam, and ‘spam’ which signifies that the message is spam. The second column is the text content of the SMS message that is being classified.

DATA PREPROCESSING:

**1. Data Collection:** Make sure you have the SMS spam collection dataset. You can download it from various sources, such as the UCI Machine Learning Repository or Kaggle.

**2. Data Cleaning:**

- Remove any duplicates in the dataset.

- Handle missing values, if any.

- Remove irrelevant or unnecessary columns, if present.

**3. Text Preprocessing:**

- Convert all text to lowercase to ensure uniformity.

- Tokenization: Split text into words or tokens.

- Remove punctuation and special characters.

- Remove stop words (common words like "and," "the," "is" that do not provide much information for classification).

- Stemming or Lemmatization to reduce words to their base or root form.

**4. Feature Extraction:**

- Use techniques like TF-IDF (Term Frequency-Inverse Document Frequency) to convert text data into numerical format. This helps in converting text into a format that Naive Bayes can work with.

**5. Data Splitting:**

- Split your dataset into training and testing sets. The common split ratio is 70-30 or 80-20.

**6. Training the Naive Bayes Model:**

- Use the training set to train a Naive Bayes classifier, such as Multinomial Naive Bayes for text classification.

**7. Model Evaluation:**

- Use the testing set to evaluate the model's performance using metrics like accuracy, precision, recall, and F1-score.

**8. Tuning Hyperparameters:**

- You can optimize the Naive Bayes classifier by tuning hyperparameters like smoothing (Laplace smoothing) to achieve better performance.

**9. Model Deployment:**

- Once satisfied with the model's performance, you can deploy it for spam detection.

python

import pandas as pd

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

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

# Load the dataset

data = pd.read\_csv('sms\_spam\_dataset.csv')

# Data cleaning

data.drop\_duplicates(inplace=True)

data.dropna(inplace=True)

# Text preprocessing

data['text'] = data['text'].str.lower()

data['text'] = data['text'].str.replace('[^a-zA-Z\s]', '') # Remove non-alphabetic characters

# Feature extraction

tfidf\_vectorizer = TfidfVectorizer(stop\_words='english')

X = tfidf\_vectorizer.fit\_transform(data['text'])

y = data['label']

# Data splitting

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the Naive Bayes model

naive\_bayes = MultinomialNB()

naive\_bayes.fit(X\_train, y\_train)

# Model evaluation

y\_pred = naive\_bayes.predict(X\_test)

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

print(f'Accuracy: {accuracy}')

print(classification\_report(y\_test, y\_pred))

**FEATURE EXTRACTION:**

**1. Bag of Words (BoW):**

- This technique represents text data as a matrix of word frequencies.

- Each row of the matrix represents a document, and each column represents a unique word in the corpus.

- The matrix is sparse, and the values represent word counts or frequencies (Term Frequency).

**2. Term Frequency-Inverse Document Frequency (TF-IDF):**

**-** TF-IDF is an improvement over BoW. It measures the importance of a word in a document relative to its importance in the entire corpus.

- It reduces the weight of common words and increases the weight of rare words.

- TF-IDF values are calculated for each word in each document.

**3. Word Embeddings:**

- Word embeddings like Word2Vec, GloVe, and FastText represent words as dense vectors in a continuous vector space.

- These embeddings capture semantic relationships between words and can be used to represent text documents by averaging or summing the word vectors within a document.

**4. N-grams:**

- N-grams are contiguous sequences of N words from a given text.

- They capture local word patterns and can be useful for capturing short phrases and expressions.

**5. Hashing Vectorizer:**

- Hashing Vectorization is a memory-efficient technique that hashes words to a fixed number of indices.

- It is useful when working with large vocabularies and memory limitations.

**6. Word Frequency:**

- You can use word frequencies (Term Frequency) directly as features. Each word's frequency in a document is used as a feature.

**7. Topic Modeling:**

- Topic modeling techniques like Latent Dirichlet Allocation (LDA) can be used to discover topics in a collection of documents. The topic distribution for each document can be used as features.

**8. Part-of-Speech (POS) Tags:**

- You can use the distribution of POS tags in a document as features. For example, the frequency of nouns, verbs, adjectives, etc.

**9. Sentiment Analysis Scores:**

- If sentiment analysis is relevant to your task, you can use sentiment scores (positive, negative, neutral) as features.

## TF-IDF Vectorizer

***# Initialize TF-IDF Vectorizer***

feature\_extraction = TfidfVectorizer(min\_df=1, stop\_words="english", lowercase=True)

***# Feature extraction for training and testing data***

X\_train\_features = feature\_extraction.fit\_transform(X\_train)

X\_test\_features = feature\_extraction.transform(X\_test)

***# Convert Y\_train and Y\_test to integer type***

Y\_train = Y\_train.astype("int")

Y\_test = Y\_test.astype("int")

print(X\_train)

3075 Mum, hope you are having a great day. Hoping t...

1787 Yes:)sura in sun tv.:)lol.

1614 Me sef dey laugh you. Meanwhile how's my darli...

4304 Yo come over carlos will be here soon

3266 Ok then i come n pick u at engin?

...

789 Gud mrng dear hav a nice day

968 Are you willing to go for aptitude class.

1667 So now my dad is gonna call after he gets out ...

3321 Ok darlin i supose it was ok i just worry too ...

1688 Nan sonathaya soladha. Why boss?

Name: Message, Length: 4457, dtype: object

print(X\_train\_features)

(0, 741) 0.3219352588930141

(0, 3979) 0.2410582143632299

(0, 4296) 0.3891385935794867

(0, 6599) 0.20296878731699391

(0, 3386) 0.3219352588930141

(0, 2122) 0.38613577623520473

(0, 3136) 0.440116181574609

(0, 3262) 0.25877035357606315

(0, 3380) 0.21807195185332803

(0, 4513) 0.2909649098524696

(1, 4061) 0.380431198316959

(1, 6872) 0.4306015894277422

(1, 6417) 0.4769136859540388

(1, 6442) 0.5652509076654626

(1, 7443) 0.35056971070320353

(2, 933) 0.4917598465723273

(2, 2109) 0.42972812260098503

(2, 3917) 0.40088501350982736

(2, 2226) 0.413484525934624

(2, 5825) 0.4917598465723273

(3, 6140) 0.4903863168693604

(3, 1599) 0.5927091854194291

(3, 1842) 0.3708680641487708

(3, 7453) 0.5202633571003087

(4, 2531) 0.7419319091456392

: :

(4452, 2122) 0.31002103760284144

(4453, 999) 0.6760129013031282

(4453, 7273) 0.5787739591782677

(4453, 1762) 0.45610005640082985

(4454, 3029) 0.42618909997886

(4454, 2086) 0.3809693742808703

(4454, 3088) 0.34475593009514444

(4454, 2001) 0.4166919007849217

(4454, 1049) 0.31932060116006045

(4454, 7346) 0.31166263834107377

(4454, 5370) 0.42618909997886

(4455, 1148) 0.38998123077430413

(4455, 6433) 0.38998123077430413

(4455, 6361) 0.25697343671652706

(4455, 2764) 0.3226323745940581

(4455, 7358) 0.2915949626395065

(4455, 7407) 0.3028481995557642

(4455, 2108) 0.3136468384526087

(4455, 4251) 0.30616657078392584

(4455, 3763) 0.16807158405536876

(4455, 4773) 0.35860460546223444

(4456, 6117) 0.5304350313291551

(4456, 6133) 0.5304350313291551

(4456, 1386) 0.4460036316446079

(4456, 4557) 0.48821933148688146

**MACHINE LEARNING ALGORITHM-NAÏVE BAYES CLASSIFIER:**

**For building an SMS spam classifier using Naive Bayes, the Naive Bayes algorithm is a natural choice for text classification tasks. Here's how you can use Naive Bayes specifically for this purpose:**

**1. Multinomial Naive Bayes:**

Multinomial Naive Bayes is well-suited for text classification problems where the features (words in this case) are discrete and represent word counts or term frequencies. It's commonly used for spam classification.

Example:

```python

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

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

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

# Assuming you've already preprocessed your SMS data and have features (e.g., TF-IDF vectors)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.2, random\_state=42)

# Create and train a Multinomial Naive Bayes classifier

clf = MultinomialNB()

clf.fit(X\_train, y\_train)

# Make predictions

y\_pred = clf.predict(X\_test)

# Evaluate the model

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

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

print(f'Accuracy: {accuracy}')

print(report)

```

**2. Bernoulli Naive Bayes:**

Bernoulli Naive Bayes is another option for text classification, especially when you represent your data as binary features (e.g., presence or absence of words).

Example :

```python

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import BernoulliNB

# Assuming you've already preprocessed your SMS data and have binary features

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.2, random\_state=42)

# Create and train a Bernoulli Naive Bayes classifier

clf = BernoulliNB()

clf.fit(X\_train, y\_train)

**IMPLEMENTATION:**

import pandas as pd

import numpy as np

from matplotlib import pyplot as plt

import seaborn as sns

import nltk

import os

stop\_words = set**(**stopwords.words**(**'english'**))**

spam\_words = " ".join**(**df**[**df**[**'Category'**]** == 0**][**'Message'**])**.split**()**

ham\_words = " ".join**(**df**[**df**[**'Category'**]** == 1**][**'Message'**])**.split**()**

spam\_word\_freq = Counter**([**word.lower**()** for word in spam\_words if word.lower**()** not in stop\_words and word.isalpha**()])**

plt.figure**(**figsize=**(**10**,** 6**))**

plt.bar**(**\*zip**(**\*spam\_word\_freq.most\_common**(**10**)),** color='g'**)**

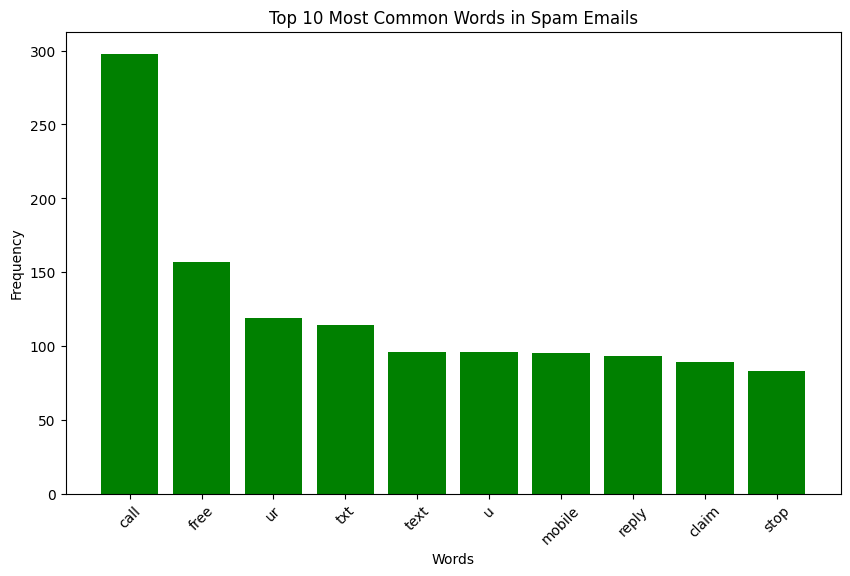
plt.xlabel**(**'Words'**)**

plt.ylabel**(**'Frequency'**)**

plt.title**(**'Top 10 Most Common Words in Spam Emails'**)**

plt.xticks**(**rotation=45**)**

plt.show**()**



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

sns.countplot(data=df, x='label')

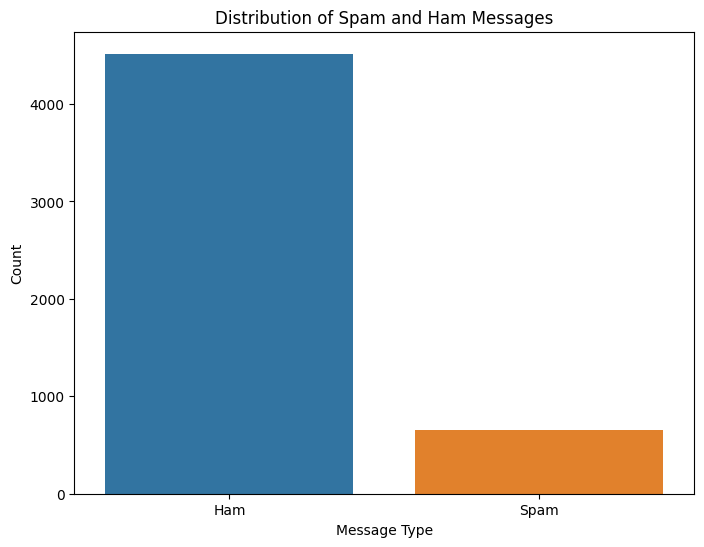
plt.xlabel('Message Type')

plt.ylabel('Count')

plt.title('Distribution of Spam and Ham Messages')

plt.xticks([**0**, **1**], ['Ham', 'Spam'])

plt.show()



**TRAINING MODEL:**

python

# Import necessary libraries

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

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

# Sample dataset (replace with your spam and non-spam messages)

messages = [

("Free entry to win a million dollars", "spam"),

("Hello, how are you today?", "not spam"),

# Add more data here

]

# Split messages into text and labels

texts, labels = zip(\*messages)

# Create a feature vector using CountVectorizer

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(texts)

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, labels, test\_size=0.2, random\_state=42)

# Initialize and train the Multinomial Naive Bayes classifier

classifier = MultinomialNB()

classifier.fit(X\_train, y\_train)

*#training our final Multinomial Naive Bayes model*

model=MultinomialNB()

model.fit(x\_train,y\_train)

print("Model Training score : ",model.score(x\_train,y\_train))

Out[]:

Model Training score : 0.992261185006046

*#model performance*

evaluate\_model\_performance(model,x\_test,y\_test)

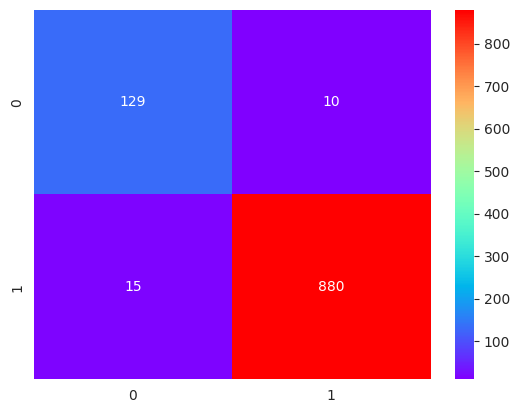
Accurary Score : 97.58

Out[]:

Score : 98.88

Recall Score : 98.32

F1 Score : 98.6



\*Classification Report\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

precision recall f1-score support

0 0.90 0.93 0.91 139

1 0.99 0.98 0.99 895

accuracy 0.98 1034

macro avg 0.94 0.96 0.95 1034

weighted avg 0.98 0.98 0.98 1034

spam\_wc=wc.generate(email\_df[email\_df["target"]==0]["transformed\_message"].str.cat(sep=" "))

plt.figure(figsize=(20,10))

plt.imshow(spam\_wc)

plt.show()



**EVALUATING METRICES:**

Accuracy measures how often the classifier makes the correct prediction. It’s the ratio of the number of correct predictions to the total number of predictions (the number of test data points).

Precision tells us what proportion of messages we classified as spam, actually were spam. It is a ratio of true positives(words classified as spam, and which are actually spam) to all positives(all words classified as spam, irrespective of whether that was the correct classification), in other words it is the ratio of

[True Positives/(True Positives + False Positives)]

Recall(sensitivity) tells us what proportion of messages that actually were spam were classified by us as spam. It is a ratio of true positives(words classified as spam, and which are actually spam) to all the words that were actually spam, in other words it is the ratio of

[True Positives/(True Positives + False Negatives)]

For classification problems that are skewed in their classification distributions like in our case, for example if we had a 100 text messages and only 2 were spam and the rest 98 weren't, accuracy by itself is not a very good metric. We could classify 90 messages as not spam(including the 2 that were spam but we classify them as not spam, hence they would be false negatives) and 10 as spam(all 10 false positives) and still get a reasonably good accuracy score. For such cases, precision and recall come in very handy. These two metrics can be combined to get the F1 score, which is weighted average of the precision and recall scores. This score can range from 0 to 1, with 1 being the best possible F1 score.

We will be using all 4 metrics to make sure our model does well. For all 4 metrics whose values can range from 0 to 1, having a score as close to 1 as possible is a good indicator of how well our model is doing.

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

f1\_score

print('Accuracy score: {}'.format(accuracy\_score(y\_test, predictions)))

print('Precision score: {}'.format(precision\_score(y\_test, predictions)))

print('Recall score: {}'.format(recall\_score(y\_test, predictions)))

print('F1 score: {}'.format(f1\_score(y\_test, predictions)))

Accuracy score: 0.9847533632286996

Precision score: 0.9420289855072463

Recall score: 0.935251798561151

F1 score: 0.9386281588447652

One of the major advantages that Naive Bayes has over other classification algorithms is its ability to handle an extremely large number of features. In our case, each word is treated as a feature and there are thousands of different words. Also, it performs well even with the presence of irrelevant features and is relatively unaffected by them.

The other major advantage it has is its relative simplicity. Naive Bayes' works well right out of the box and tuning it's parameters is rarely ever necessary, except usually in cases where the distribution of the data is known.

It rarely ever overfits the data.

Another important advantage is that its model training and prediction times are very fast for the amount of data it can handle.

**BENEFITS :**

**1. Simple and Efficient:**

Naive Bayes is a simple and computationally efficient classification algorithm, making it a good choice for text classification tasks like spam detection.

**2. Effective for Text Data:**

Naive Bayes works well with text data, as it's designed to handle high-dimensional, sparse data where the features (words) are mostly independent.

**3. Quick Training:**

Training a Naive Bayes model is typically fast, even with large datasets, making it suitable for real-time or near-real-time applications.

**4. Low Memory Usage:**

Naive Bayes models have a relatively small memory footprint, which is beneficial when resources are limited.

**5. Interpretability:**

The Naive Bayes algorithm is highly interpretable, and it's easy to understand why a particular classification decision was made based on the probability calculations.

**6. Applicability to Multiclass Problems:**

Naive Bayes can handle both binary and multiclass classification problems, which is useful in scenarios where you have more than two categories to classify.

**7. Robust to Irrelevant Features:**

Naive Bayes is generally robust to irrelevant features because it assumes that features are independent, meaning it doesn't consider the strength of relationships between features.

**8. Suitable for Small Datasets:**

It can work well even when you have limited training data, which is often the case in SMS spam classification.

**9. Works with Imbalanced Data:**

It can handle imbalanced datasets, where one class (e.g., spam) is significantly smaller than the other (e.g., non-spam).

**10. Baseline Model:**

Naive Bayes can serve as a useful baseline model for text classification tasks. You can start with Naive Bayes, and if needed, explore more complex models.

**CONCLUSION :**

In conclusion, our comprehensive solution aims to tackle the challenge of spam classification by

utilizing advanced AI techniques. The focus on minimizing both false positives and false negatives ensures a balanced and effective spam filter for email and text messages.

the Naive Bayes algorithm is a highly effective and popular choice for spam classification. Its simplicity, efficiency, and ability to handle high-dimensional data make it well-suited for this task. By utilizing probabilistic calculations and assuming independence between features (hence "naive"), Naive Bayes can accurately identify spam messages by comparing the likelihood of a message being spam or not. While it may not always achieve perfect results, it remains a robust and widely-used solution for spam classification, and its performance can be further enhanced with feature engineering and data preprocessing.