Building a Smarter AI Powered Spam Classifier

### Problem Definition

In machine learning, spam filtering protocols use instance-based or memory- based learning methods to identify and classify incoming spam emails based on their resemblance to stored training examples of spam emails Spam is any unsolicited communication sent in bulk.Usually sent via email ,spam is also distributed through text messages(SMS),social media,or phone calls. Spam messages often come in the form of harmless

(though annoying)promotional emails .But sometimes spam is a fraudulent or malicious scam.

### Design Thinking

1. The First Component to Consider When Building the AISolution Is the Problem Identification:

Before developing a productor feature,it’s essential to focus on the user’s pain point and figure out the value proposition (value-prop) that users can get from your product.

A value proposition has to do with the value you promise to deliver to your customers should they choose to purchase your product.

1. Have the Right Data and Clean It:

Now,when you’ve framed the problem ,you need to pick the right data sources. It’s more critical to get high-quality data than to spend time on improving the AImodel itself. Data falls under two categories

1. Create Algorithms:

When telling the computer what to do, you also need to choose how it will do it. That’s where computer algorithms step in. Algorithms are mathematical instructions. It’s necessary to create prediction or classification machine learning algorithms so, the AI model can learn from the dataset.

1. Train the Algorithms

Moving forward with how to create an AI,you need to train the algorithm using the collected data. It would be best to optimize the algorithm to achieve an AI model with high accuracy during the training process. How ever ,you may need additional data to improve the accuracy of your model.

1. Opt for the Right Platform:

Apart from the data required to train your AI model, you need to pick the right platform for your needs. You can go for an in-house or cloud framework. What’s the main difference between these frameworks The cloud makes it easy for enterprises to experiment and grow as projects go into production and demand increases by allowing faster training and deployment of ML models.

**o** In-houseFrameworks

1. Choose a Programming Language:

There is more than one programming language ,including the classic C++,Java

,Python, and R. The latter two coding languages are more popular because they offer a robust set of tools such as extensive ML libraries. Make the right choice by considering your goals and needs.

**Algorithm**

**Step1** : E-mail Data Collection.The data set contained in a corpus plays a crucial role in assessing the performance of any spam filter.

**Step2**:Pre-processing of E-mail content **Step3**:Feature Extraction and Selection **Step 4**: Implementation **Step5**:Performance Analysis.

we load the dataset for the spam detection project.The dataset is stored in a CSV file located at'/content/spam.csv'. We use the pandas library to read the CSV file and do some preprocessing to the dataset like text Cleaning, Stemming and etc.

To perform natural language processing tasks,we'll first install the Natural Language Toolkit (NLTK) library.

**Analysis:**

We import the pandas library using import pandas as pd.

We use pd.read\_csv() to read the CSV file containing the dataset. The encoding='latin-1' argument is used to handle special characters.

We select only the relevant columns ('v1'forlabels,'v2'foremailcontent)using data[['v1', 'v2']]. Finally,we display the resulting DataFrame to inspect the loaded data.

###### Code:

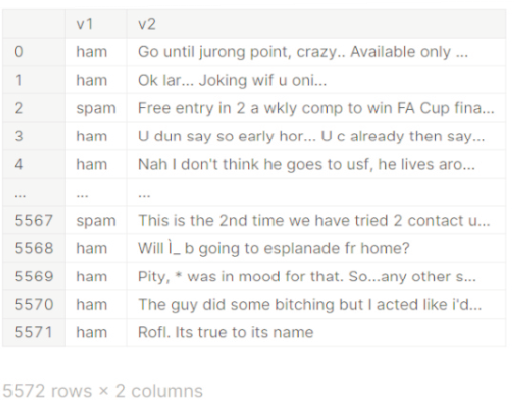
Import pandas as pd # Load the dataset

data=pd.read\_csv('/kaggle/input/sms-spam-collection-dataset/spam.csv', encoding='latin-1')

data=data[['v1','v2']]

#Selecting only the relevant columns data #printing

**Output**



**Data Preprocessing**

In this step,we perform datapreprocessing tasks,which include converting labels to binary values and removing duplicates from the dataset.

#### Explanation:

We use data['v1'].apply(lambdax:1ifx=='spam' else0) toconvertthelabels.'ham'is mapped to 0, and 'spam' is mapped to 1 in the 'v1' column.

We then remove duplicate rows from the dataset using data=data.drop\_duplicates().

The resulting DataFrame is displayed to show the cleaned dataset.

###### Code:

#Convert'ham'to0and'spam'to1directlyinthe'v1'column data['v1'] = data['v1'].apply(lambda x: 1 if x == 'spam' else 0)

# removing duplicates data=data.drop\_duplicates() data

**Output**



# Text Cleaning:

Text cleaning involves removing any unnecessary characters,symbols,or noise from the text data. This might include punctuation, special characters, and numbers.

Explanation:

We import the regular expression(re) module using import re.

The function clean\_text() takes a string text as input and uses a regular expression to remove all characters except alphabetic character.

The cleaned text is then returned.

We apply this function to the 'v2' column of the DataFrame using data['v2'].apply(lambdax:clean\_text(x)).Thiscleansthetextineachemail.

**Code:**

import re

def clean\_text(text):

cleaned\_text=re.sub(r'[^a-zA-Z]','',text) return cleaned\_text

data['v2']= data['v2'].apply(lambdax: clean\_text(x))

# Lower casing:

Converting all text to lowercase ensures that the model doesn't treat "Hello" and "hello" as different words.

#### Explanation:

We use the str.lower() method to convert all text in the 'v2' column to lowercase. This helps standardize the text data and ensure that the model is not case-sensitive.

data['v2']=data['v2'].str.lower()

# Tokenization:

Tokenization involves splitting the text into individual words or tokens.The NLTK library can be used for this.

# Explanation:

In this code cell ,we use nltk. download('punkt') to download the necessary resources for tokenization from the Natural Language Toolkit (NLTK). This resource includes pre-trained models for tokenizing text into words or sentences. This step is essential for further text processing.

##### Code:

import nltk nltk.download('punkt')

[nltk\_data] Downloading package punkt

to/usr/share/nltk\_data...

[nltk\_data] Package punkt is already up-to-date!

##### Output

True

**Stemming:**

Stemming reduce words to their base forms. This can help in reducing the dimensionality of the feature space.

**Explanation:**

We import the PorterStemmer class from the NLTK library. We initialize an instance of the PorterStemmer as stemmer.

We define a function stem\_words(words) that takes a list of words and applies stemming to each word using the stemmer.stem() method.

We apply this function to the 'v2' column of the DataFrame, effectively reducing words to their base forms through stemming. This step can help improve the model's performance by reducing the feature space.

**Code:**

From nltk.stem import PorterStemmer stemmer = PorterStemmer()

def stem\_words(words): return[stemmer.stem(word) forword inwords]

data['v2']=data['v2'].apply(stem\_word s) data

|  |  |  |
| --- | --- | --- |
| **Output**  **Preparations**  **First we have to create the python environment for themodel to run , get dataset from**  “/kaggle/input/sms-spam-collection-dataset/spam.csv “  **Then clean and preprocess it, Import the necessary libraries in the Notebook and then load the dataset and run the head() function and thedrop() function to ignore the Nan and undefined columns .**  CODE  *# This Python 3 environment comes with many helpful analytics libraries installed*  *# It is defined by the kaggle/python Docker image: https:*  *//github.com/kaggle/docker-python*  *# For example, here's several helpful packages to load*  import numpy as np *# linear algebra*  import pandas as pd *# d ta processing, CSV file I/O (e.g.pd.read\_csv)* | | |
|  | *# Input data files are vailable in the read-only "../input/" directory # For example, running this (by clicking run or pressin*  *Shift+Enter) will list all files under the input directory* |  |
|  |  |  |

*a*

*a*

*g*

import os

for dirname, \_, filenames **in** os.walk('/kaggle/input'):

for filename **in** filenames:

print(os.path.join(dirname, filename))

*# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when youcreate a version using "Save & Run All"*

*# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session*

Importing necessary libraries

*#necessary libraries*

import numpy as np *# linear algebra* import pandas as pd

*# data processing*

import nltk from nltk.corpus import stopwords from nltk.tokenize import word\_tokenize from nltk.stem import PorterStemmer

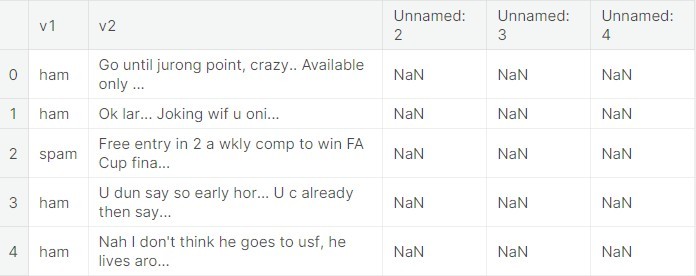
Loading the dataset

*# Load the dataset*

data = pd.read\_csv('/kaggle/input/sms-spam-collection-dataset/spam.csv', encoding='latin-1')

data.head()

**OUTPUT**



columns\_to\_drop = ['Unnamed: 2', 'Unnamed: 3',

'Unnamed: 4']

data = data.drop(columns\_to\_drop, axis=1, errors= 'ignore')

data.head()

## Tokenization and cleaning

Tokenization is the process of splitting the input and output texts into smaller units that can be processed by the LLM AI models. Tokens can be words , characters , subwords, or symbols, depending on the type and the size of the model

## CODE

*# Clean the "v2" column*

data['v2'] = data['v2'].str.lower()

*# Tokenization and cleaning of data*

def preprocess\_text(text): tokens = word\_tokenize(text)

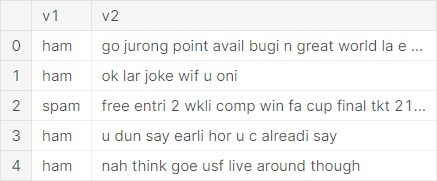
stop\_words = set(stopwords.words('english')) filtered\_tokens = [word for word **in** tokens if word.isa

lnum() **and** word **n ot in** stop\_words]stemmer = PorterStemmer()

stemmed\_tokens = [stemmer.stem(word) for word **in** filtered\_tokens] return ' '.join(stemmed\_tokens)

data['v2'] = data['v2'].apply(preprocess\_text)

data.head()

**OUTPUT**

from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.model\_selection import train\_test\_split

*# Load the dataset*

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_matrix = tfidf\_vectorizer.fit\_transform(data['v2']

)

*# Label Encoding*

data['v1'] = data['v1'].map({'ham': 0, 'spam': 1})

*# Split Data*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(tfidf\_matrix, data['v1'], test\_size=0.2, random\_state=42)

*# Check the shape of the TF-IDF matrix and the split data*print("TF-IDF Matrix Shape:", tfidf\_matrix.shape)

print("Training Data Shape:", X\_train.shape) print("Testing Data Shape:", X\_test.shape)

**OUTPUT**

TF-IDF Matrix Shape: (5572, 8672)

Training Data Shape: (4457, 8672)

Testing Data Shape: (1115, 8672)

Random Forest Classifier

Random forest is a commonly-used machine learning algorithm which combines the output of multiple decision trees to reach a singleresult.

##### CODE

from sklearn.ensemble import RandomForestClassifier

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

*# Create a Random Forest classifier*

rf\_classifier = RandomForestClassifier(random\_state=42)

*# Train the classifier on the training data*

rf\_classifier.fit(X\_train, y\_train)

*# Make predictions on the testing data*

y\_pred = rf\_classifier.predict(X\_test)

*# Evaluate the model's performance*

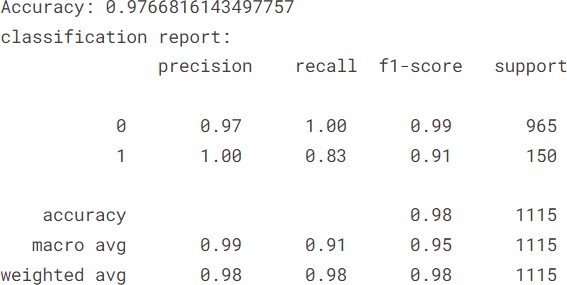
accuracy = accuracy\_score(y\_test, y\_pred) classification\_rep = classification\_report(y\_test, y\_pred)

*# Print the results*

print("Accuracy:", accuracy)

print("classification report:**\n**", classification\_rep)

##### OUTPUT



**Testing model**

##### Test1

input\_text = """**\a**pple Inc.Your iPhone 6 linked top\*\*\*zm".edu) has been used a few minutes

ago. To localize it,login now to your apple account ."""

*# Apply the same preprocessing as in your previous code*

input\_text = input\_text.lower()

*# Add more preprocessing steps if needed*

*# Transform the input text into a TF-IDF vector*

input\_tfidf = tfidf\_vectorizer.transform([input\_text])

*# Make a prediction using the trained Random Forest model*

prediction = rf\_classifier.predict(input\_tfidf)

*# predictions*

if prediction[0] == 1:

print("This message is predicted to be SPAM by trainedmodel.") else:

print("This message is predicted to be NOT SPAM by

trained model.")

##### OUTPUT

This message is predicted to be NOT SPAM by trainedmodel.

input\_text1 = "Hey, I'm mark. How are you?."

*# Apply the same preprocessing as in your previous code*

input\_text1 = input\_text1.lower()

*# Transform the input text into a TF-IDF vector*

input\_tfidf = tfidf\_vectorizer.transform([input\_text1])

*# Make a prediction using the trained Random Forest model*

prediction = rf\_classifier.predict(input\_tfidf)

*# perdictions*

if prediction[0] == 1:

print("This message is predicted to be SPAM by trained model.")

else:

print("This message isained

model.")

predicted

to be NOT SPAM by tr

**Test2**

**OUTPUT**

This message is predicted not spam

# Wordcloud of ham category

#generate wordcloud plot for not-spam messages

**ham\_wc=wc.generate(email\_df[email\_df["target"]==1]["transformed\_message"].str**

**.cat(sep=" ")) plt.figure(figsize=(20,10)) plt.imshow(ham\_wc) plt.show()**



#used words in spam messages

**spam\_corpus=list()**

**for msg in email\_df[email\_df['target']==0]["transformed\_message"].to\_list(): for word in msg.split():**

**spam\_corpus.append(word) len(spam\_corpus)**

**OUTPUT**

**9883**

#print the most common 50 words from the spam category messages

**from collections import Counter spam\_top\_50\_common\_words=pd.DataFrame(Counter(spam\_corpus).most\_commo n(50))**

**print(spam\_top\_50\_common\_words)**

**OUTPUT**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | call | 320 |
| 1 | free | 189 |
| 2 | 2 | 155 |
| 3 | txt | 141 |
| 4 | text | 122 |
| 5 | u | 119 |
| 6 | ur | 119 |
| 7 | mobil | 114 |
| 8 | stop | 104 |
| 9 | repli | 103 |
| 10 | claim | 98 |
| 11 | prize | 82 |
| 12 | 4 | 76 |
| 13 | get | 74 |
| 14 | new | 64 |
| 15 | servic | 64 |
| 16 | tone | 63 |
| 17 | send | 60 |
| 18 | urgent | 57 |
| 19  **….** | nokia | 57 |

#used words in ham messages

**ham\_corpus=list()**

**for msg in email\_df[email\_df['target']==1]["transformed\_message"].to\_list(): for word in msg.split():**

**ham\_corpus.append(word)**

len(ham\_corpus)

**Out[24]: 34771**

#most commnaly used 50 words from ham category messages **ham\_top\_50\_common\_words=pd.DataFrame(Counter(ham\_corpus).most\_common( 50))**

**print(ham\_top\_50\_common\_words)**

**OUTPUT**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | u | 871 |
| 1 | go | 401 |
| 2 | get | 349 |
| 3 | gt | 288 |
| 4 | lt | 287 |
| 5 | 2 | 284 |
| 6 | come | 272 |
| 7 | got | 236 |
| 8 | like | 234 |
| 9 | know | 234 |
| 10 | call | 232 |
| 11 | time | 217 |
| 12 | good | 212 |
| 13 | want | 208 |
| 14  **…** | ok | 207 |

**Data Transformation**

Using Count Vectorization

**In [26]:**

**from sklearn.feature\_extraction.text import CountVectorizer cVector=CountVectorizer()** #CountVectorizer is used to convert text into numeric array

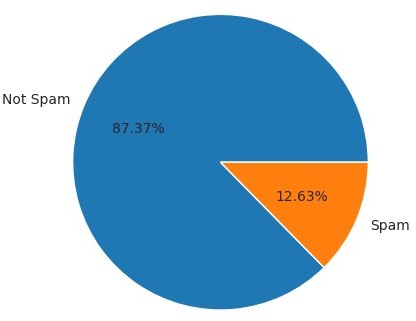
**x=cVector.fit\_transform(email\_df["transformed\_message"]).toarray() In [27]:**

#seperating target column

y=email\_df['target']

#check the distribution of target variable using Pie chart **plt.pie(y.value\_counts().values,labels=["Not Spam","Spam"],autopct="%0.2f%%") plt.show()**

**OUTPUT**



**Conclusion** : as we can see our dataset is imbalanced.

# Spliting data into Training and Testing sets into 80/20 ratio

from sklearn.model\_selection import train\_test\_split x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=43) x\_train.shape,y\_train.shape,x\_test.shape,y\_test.shape

Out[29]:

((4135, 6629), (4135,), (1034, 6629), (1034,))

In [30]:

from sklearn.metrics import accuracy\_score,precision\_score,recall\_score,f1\_score,confusion\_matrix,classificatio n\_report

*#function to evaluate the performance of model*

def evaluate\_model\_performance(model,x\_test,y\_test): y\_pred=model.predict(x\_test)

print("Accurary Score :

**{}**".format(np.round(accuracy\_score(y\_test,y\_pred)\*100,decimals=2))) print("Precision Score :

**{}**".format(np.round(precision\_score(y\_test,y\_pred)\*100,decimals=2))) print("Recall Score :

**{}**".format(np.round(recall\_score(y\_test,y\_pred)\*100,decimals=2)))

print("F1 Score : **{}**".format(np.round(f1\_score(y\_test,y\_pred)\*100,decimals=2))) cm=confusion\_matrix(y\_test,y\_pred) sns.heatmap(cm,fmt="d",annot=True,cmap="rainbow")

plt.show()

print("\*Classification Report\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*") print(classification\_report(y\_test,y\_pred))

In [31]:

*#import models*

from sklearn.linear\_model import LogisticRegression from sklearn.naive\_bayes import MultinomialNB from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import GradientBoostingClassifier from lightgbm import LGBMClassifier

from xgboost import XGBClassifier

from sklearn.model\_selection import cross\_val\_score

In [32]:

from sklearn.model\_selection import StratifiedKFold

from imblearn.over\_sampling import RandomOverSampler

*# Define models*

models = { "lr":LogisticRegression(), "nb":MultinomialNB(), "svm":SVC(),

"knn":KNeighborsClassifier(), "cart":DecisionTreeClassifier(), "rf":RandomForestClassifier(), "ad":AdaBoostClassifier(), "gb":GradientBoostingClassifier(), "xgbc":XGBClassifier()

}

*# Define oversampler for dealing with imbalance*

oversampler = RandomOverSampler()

*# Define cross-validation strategy for imbalanced data*

cv = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

model\_scores=list()

*# Loop through each model and evaluate its performance*

for model\_name, model **in** models.items():

*# Apply oversampling to training data*

X\_resampled, y\_resampled = oversampler.fit\_resample(x, y)

*# Perform cross-validation*

scores = cross\_val\_score(model, X\_resampled[:500], y\_resampled[:500], cv=cv, scoring="f1\_micro")

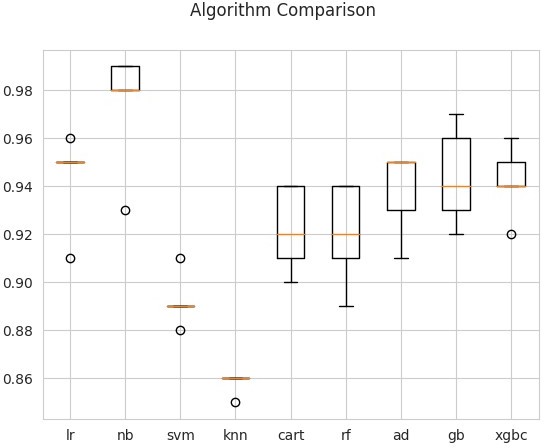
print(model\_name," : ",np.round(np.mean(scores)\*100,decimals=2)) model\_scores.append(scores)

*# boxplot algorithm comparison*

fig = plt.figure() fig.suptitle('Algorithm Comparison') ax = fig.add\_subplot(111) plt.boxplot(model\_scores) ax.set\_xticklabels(models.keys()) plt.show()

### OUTPUT

|  |  |
| --- | --- |
| lr : | 94.4 |
| nb : | 97.4 |
| svm : | 89.2 |
| knn : | 85.8 |
| cart : | 92.2 |
| rf : | 92.0 |
| ad : | 93.8 |
| gb : | 94.4 |
| xgbc : | 94.2 |



**CONCLUTION:**

Thus the SMS Spam Collection dataset is preprocessed , transformed and trained based on the algorithm ,Overall the Ai powered

spam classifier successfully classifies the dataset into ham and spam messages during the training of this model.