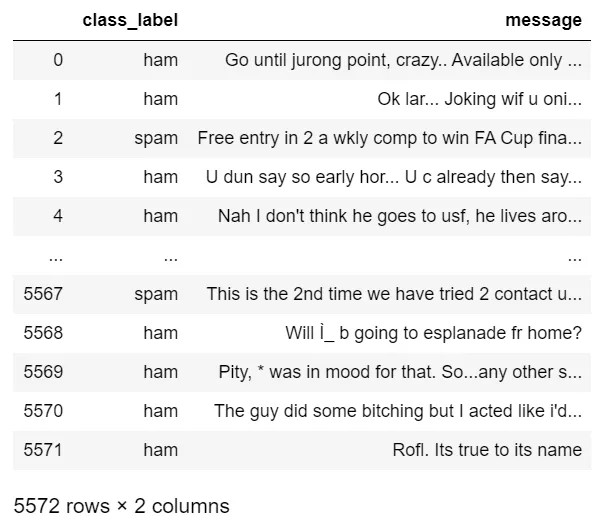
**Building a smarter AI-powered spam classifier**

**Introduction:**

Certainly! Building a smarter AI-powered spam classifier involves implementing advanced machine learning techniques to effectively differentiate between spam and legitimate messages. Key steps in this process include data preprocessing, feature engineering, model selection, training, and evaluation. Utilizing natural language processing (NLP) techniques and deep learning models can enhance the classifier’s accuracy. Regular updates and continuous monitoring are crucial to adapt to evolving spam patterns. If you need specific guidance or have any questions about this process, feel free to ask!

**Given data set :**



**Program**:

Import pandas as pd

Df =

pd.read\_csv(r’spam.csv’,encoding=’

ISO-8859-1’)

Df.rename(columns =

{‘v1’:’class\_label’, ‘v2’:’message’}, inplace = True)

Df.drop([‘Unnamed: 2’, ‘Unnamed: 3’, ‘Unnamed:

V 4’], axis = 1, inplace = True)

Df

**Data Collection:**

Gather a large and diverse dataset of emails or messages, labeled as spam or non-spam (ham). This dataset is crucial for training your AI model.

**Data Preprocessing**

Clean and preprocess the data. This may involve tasks like removing special characters, stemming, and tokenization.

Feature Extraction: Extract relevant features from the preprocessed data. Common techniques include Bag-of-Words, TF-IDF (Term Frequency-Inverse Document Frequency), or word embeddings like Word2Vec.

Choosing a Model: Select an appropriate machine learning algorithm or deep learning architecture for your task. Popular choices include Naïve Bayes, Support Vector Machines, or deep learning models like Recurrent Neural Networks (RNNs) or Transformers.

Training the Model: Train your chosen model using the preprocessed data. Use a portion of the dataset for training and another portion for validation to fine-tune the model.

**Evaluation**:

Evaluate the model’s performance using metrics like accuracy, precision, recall, and F1-score. This helps in understanding how well your model is performing.

**Fine-Tuning**:

Based on the evaluation results, fine-tune the model. This could involve adjusting hyperparameters, trying different algorithms, or exploring advanced techniques like ensemble methods.

**Testing and Deployment:**

Test the final model on a separate test dataset to ensure its generalizability. Once you’re confident in its performance, deploy the model into your application or system.

**Monitoring and Maintenance:**

Continuously monitor the model’s performance in real-world scenarios. Spam patterns can change over time, so periodic updates and retraining might be necessary to maintain the classifier’s accuracey.

**Program** .

# Import necessary libraries

Import pandas as pd

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

From sklearn.feature\_extraction.text import CountVectorizer

From sklearn.naive\_bayes import MultinomialNB

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

From sklearn.feature\_extraction.text import TfidfVectorizer

From sklearn.pipeline import Pipeline

From sklearn.externals import joblib

# Load the dataset (assuming it’s in CSV format with ‘text’ and ‘label’ columns)

Data = pd.read\_csv(‘spam\_dataset.csv’)

# Preprocessing: Cleaning and Tokenization

# Assuming you have a function clean\_text that cleans the text data

Data[‘cleaned\_text’] = data[‘text’].apply(clean\_text)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data[‘cleaned\_text’], data[‘label’], test\_size=0.2, random\_state=42)

# Creating a pipeline with TfidfVectorizer and Multinomial Naïve Bayes Classifier

Text\_clf = Pipeline([

(‘tfidf’, TfidfVectorizer()), # Convert text to TF-IDF features

(‘clf’, MultinomialNB()) # Naïve Bayes classifier

])

# Train the model

Text\_clf.fit(X\_train, y\_train)

# Evaluate the model

Predicted = text\_clf.predict(X\_test)

Accuracy = accuracy\_score(y\_test, predicted)

Print(f’Accuracy: {accuracy:.2f}’)

# Print classification report for detailed evaluation

Print(classification\_report(y\_test, predicted))

# Save the trained model for future use

Joblib.dump(text\_clf, ‘spam\_classifier\_model.pkl’)

**Objective**

3. Dataset Splitting:

Split the dataset into training, validation, and testing subsets. Typically, a common split ratio is 70% for training, 15% for validation, and 15% for testing.

**Installing Librarie**

# split data into train and test

X = np.array(df.embedding)

y = np.array(df.class\_embeddings)

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

# train random forest classifier

clf = RandomForestClassifier(n\_estimators=100)

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

preds = clf.predict(X\_test.tolist())

# generate a classification report involving f1-score, recall, precision and accuracy

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

print(report)

**Importing Data**

%matplotlib inline

Import matplotlib.pyplot as plt

Import csv

Import sklearn

Import pickle

From wordcloud import WordCloud

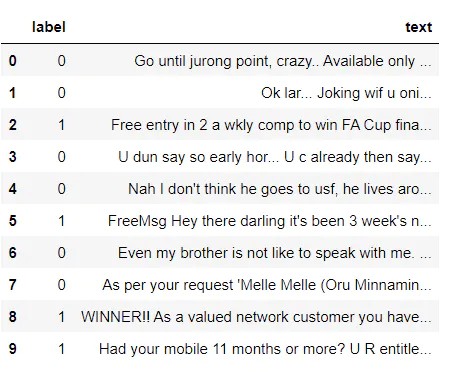
Import pandas as pd

Import numpy as np

Import nltk

From nltk.corpus import stopwords

From sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformerFrom sklearn.tree import DecisionTreeClassifier

From sklearn.model\_selection import GridSearchCV,train\_test\_split,StratifiedKFold,cross\_val\_score,learning\_curve

**Missing Value Analysis**

#import sklearn packages for building classifiers

From sklearn.linear\_model import LogisticRegression

From sklearn.svm import SVC

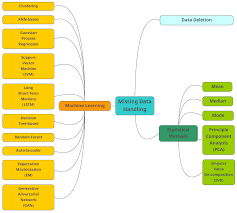
From sklearn.naive\_bayes import MultinomialNB

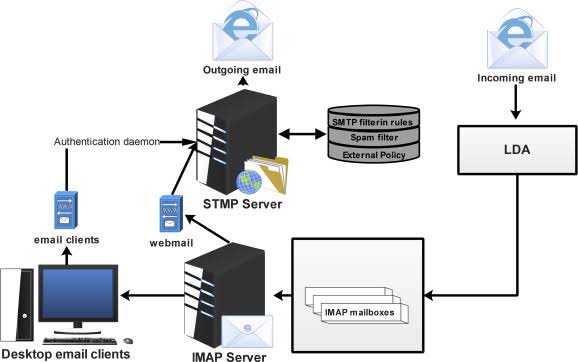
From sklearn.tree import DecisionTreeClassifier

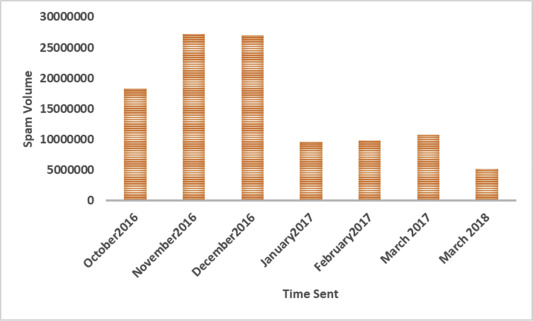
From sklearn.neighbors import KNeighborsClassifier

From sklearn.ensemble import RandomForestClassifier

From sklearn.metrics import accuracy\_score



**Detection**



**Exploratory Data Analysis**

Def text\_clean(text, method, rm\_stop):

Text = re.sub(r”\n”,””,text) #remove line breaks

Text = text.lower() #convert to lowercase

Text = re.sub(r”\d+”,””,text) #remove digits and currencies

Text = re.sub(r’[\$\d+\d+\$]’, “”, text)

Text = re.sub(r’\d+[\.\/-]\d+[\.\/-]\d+’, ‘’, text) #remove dates

Text = re.sub(r’\d+[\.\/-]\d+[\.\/-]\d+’, ‘’, text)

Text = re.sub(r’\d+[\.\/-]\d+[\.\/-]\d+’, ‘’, text)

Text = re.sub(r’[^\x00-\x7f]’,r’ ‘,text) #remove non-ascii

Text = re.sub(r’[^\w\s]’,’’,text) #remove punctuation

Text = re.sub(r’https?:\/\/.\*[\r\n]\*’, ‘’, text) #remove hyperlinks

#remove stop words

If rm\_stop == True:

Filtered\_tokens = [word for word in word\_tokenize(text) if not word in set(stopwords.words(‘english’))]

Text = “ “.join(filtered\_tokens)

#lemmatization: typically preferred over stemming

If method == ‘L’:

Lemmer = WordNetLemmatizer()

Lemm\_tokens = [lemmer.lemmatize(word) for word in word\_tokenize(text)]

Return “ “.join(lemm\_tokens)

#stemming

If method == ‘S’:

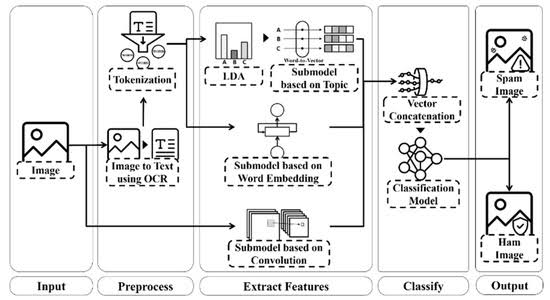
Porter = PorterStemmer()

Stem\_tokens = [porter.stem(word) for word in word\_tokenize(text)]

Return “ “.join(stem\_tokens)

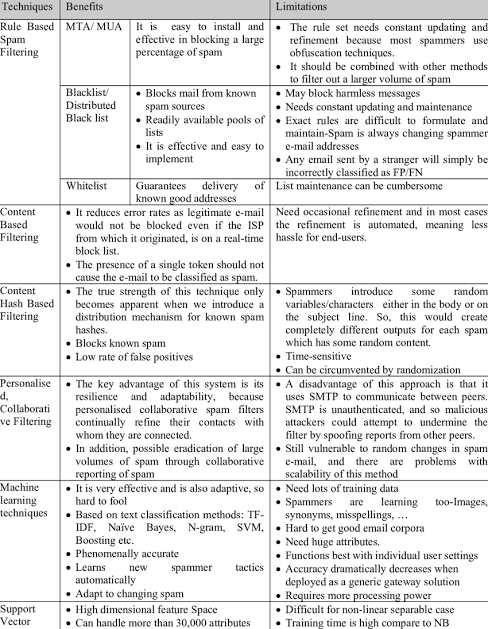
**Feature Enginnering:**

Till now, I explored the dataset, did missing value corrections and data visualization. Next, I have started feature engineering. Feature engineering is useful to improve the performance of machine learning algorithms and is often considered as applied machine learning. Selecting the important features and reducing the size of the feature set makes computation in machine learning and data analytic algorithms more feasible.



**Data pre- processing**

Data preprocessing, a component of data preparation, describes any type of processing performed on raw data to prepare it for another data processing procedure. It has traditionally been an important preliminary step for the data mining process.



**Conclusion**

Comprehensive Dataset:

A diverse and comprehensive dataset is crucial for training a robust spam classifier. It should cover various types of spam messages to ensure the model’s effectiveness in real-world scenarios.

Data Preprocessing:

Proper preprocessing techniques, including noise removal and standardization, are essential to prepare the data for feature extraction and model training. Clean data leads to more accurate results.