**Building a spam classifier involves several core components**

**1. Data Collection:**

- Gather a labeled dataset of emails or text messages, where each message is categorized as either spam or not spam (ham). You can find publicly available datasets or create your own.

**2. Data Preprocessing:**

- Clean the text data by removing any irrelevant information, such as HTML tags, special characters, and extra whitespace.

- Tokenize the text data by breaking it into individual words or tokens.

- Convert the text data into numerical features. Common techniques include TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings like Word2Vec or GloVe.

**3. Feature Engineering:**

- Create additional features based on the text data, such as message length, the frequency of specific words or phrases, and other relevant attributes that can help in distinguishing spam from non-spam.

**4. Split the Data:**

- Divide the dataset into training, validation, and testing sets. The training set is used to train the model, the validation set to tune hyperparameters, and the testing set to evaluate the model's performance.

**5. Model Selection:**

- Choose a machine learning or deep learning algorithm for classification. Common choices include:

- Naive Bayes

- Support Vector Machines

- Decision Trees

- Random Forest

- Neural Networks (e.g., deep learning models)

**6. Model Training:**

- Train the selected model using the training data. This involves fitting the model to the training data and optimizing its parameters.

**Deployment:**

- Once the spam classifier performs satisfactorily, you can deploy it in your email or messaging system to automatically filter out spam messages.

**CODE FOR DATACOLLECTION**

import requests

from bs4 import BeautifulSoup

url = "http://quotes.toscrape.com"

response = requests.get(url)

if response.status\_code == 200:

soup = BeautifulSoup(response.text, "html.pa

quotes = []

for quote in soup.find\_all("span", class\_="text"):

quotes.append(quote.get\_text()

for i, quote in enumerate(quotes, 1):

print(f"Quote {i}: {quote}")

else:

print("Failed to retrieve data. Status code:", response.status\_code)

**CODE :**

import numpy as np

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, confusion\_matrix

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

tfidf\_vectorizer = TfidfVectorizer(max\_features=5000) # You can adjust the max\_features

X\_train\_tfidf = tfidf\_vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = tfidf\_vectorizer.transform(X\_test)

naive\_bayes\_classifier = MultinomialNB()

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

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

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

print("Accuracy:", accuracy)

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

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

print("Confusion Matrix:\n", conf\_matrix)

**A Convolutional Neural Network (CNN) preprocessor is typically used to prepare image data for input to a CNN model. The preprocessor includes several steps such as data loading, resizing, normalization, and data augmentation.** **Below is a Python code example that demonstrates the typical preprocessing steps for image data using the popular deep learning library, TensorFlow and Keras. This code assumes you have a dataset of images and want to preprocess them before training a CNN model.**

**python**

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

batch\_size = 32

image\_size = (224, 224) # Adjust the size according to your model's requirements

data\_directory = 'path\_to\_image\_dataset\_directory'

data\_augmentation = tf.keras.Sequential([

tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal"),

tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),

tf.keras.layers.experimental.preprocessing.RandomZoom(0.1),

])

datagen = ImageDataGenerator(

rescale=1./255,

rotation\_range=20,

width\_shift\_range=0.2

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2

horizontal\_flip=True,

fill\_mode='nearest'

)

train\_data\_generator = datagen.flow\_from\_directory(

data\_directory,

target\_size=image\_size,

batch\_size=batch\_size,

class\_mode='binary'

)

validation\_data\_generator = datagen.flow\_from\_directory(

'path\_to\_validation\_dataset\_directory',

target\_size=image\_size,

batch\_size=batch\_size,

class\_mode='binary'

)

image = tf.image.resize(image, image\_size)

image /= 255.0 # Normalize pixel values to [0, 1]

return image

test\_data\_generator = tf.keras.preprocessing.image\_dataset\_from\_directory(

'path\_to\_test\_dataset\_directory',

image\_size=image\_size,

batch\_size=batch\_size,

label\_mode='binary' # You can adjust this based on your dataset

)

test\_data\_generator = test\_data\_generator.map(preprocess\_test\_data)

```

**This code includes data augmentation, which is beneficial for improving a CNN's generalization on the training data. It also provides data generators for training, validation, and testing data, ensuring that images are preprocessed consistently. Make sure to adjust the parameters and data paths according to your specific dataset and model requirements.**