BUILDING A SMARTER AI-POWERED SPAM CLASSIFIER:

PRESENTED BY: SWATHI M

(422521104036)

PROBLEM STATEMENT:

The problem is to develop a spam classifier for SMS messages using the Naive Bayes classification algorithm. The goal is to automatically classify incoming SMS messages as either spam or not spam (ham) to help users filter out unwanted messages and improve the messaging experience.

DESIGN THINKING PROCESS:

Design thinking is a human-centered approach to problem-solving that emphasizes empathy, creativity, and collaboration. Here's how you can apply design thinking to build a spam classifier using the Naive Bayes algorithm:

1. Empathize:

Understand the user's perspective: Identify the pain points of receiving spam SMS messages and the impact on the user's experience.Gather user feedback and preferences to ensure the spam classifier is designed to meet their needs

.2. Define:

Clearly define the problem statement: Create a well-defined problem statement that outlines the need for a spam classifier for SMS messages.

Identify specific user requirements and expectations for the spam classifier.

3. Ideate:

Brainstorm solutions: Generate creative ideas for building a spam classifier.

Explore various approaches and technologies for spam detection, such as Naive Bayes, machine learning, and natural language processing (NLP).

4. Prototype:

Create a prototype of the spam classifier using the Naive Bayes algorithm. This involves developing a small-scale model to test the concept .Collect a sample SMS dataset for training and testing.

5. Test:

Evaluate the prototype: Use the collected dataset to train and test the Naive Bayes classifier. Measure its accuracy, precision, recall, and F1-score to assess its performance.

6. Feedback:

Collect feedback from users and stakeholders regarding the prototype's effectiveness and user satisfaction.

Identify any issues or areas for improvement.

7. Iterate:

Based on feedback and test results, refine the spam classifier's design and implementation. Iterate through the prototype, test, and feedback stages until the spam classifier meets user expectations.

PHASE OF DEVELOPMENT:

1. Data Collection:

Gather a labeled SMS dataset with examples of both spam and non-spam (ham) messages. The dataset should be representative and diverse.

2. Data Preprocessing:

Preprocess the SMS data, including text cleaning, tokenization, and removing stop words. Transform the text data into a numerical format suitable for the Naive Bayes algorithm, such as TF-IDF (Term Frequency-Inverse Document Frequency) or bag of words.

3. Model Development:

Implement the Naive Bayes algorithm (e.g., Multinomial Naive Bayes) to train the classifier on the preprocessed dataset.

Split the dataset into training and testing sets to evaluate the model's performance.

4. Model Evaluation:

Use metrics like accuracy, precision, recall, and F1-score to assess the classifier's performance on the testing dataset. Fine-tune the model parameters to improve performance if needed.

5. Deployment:

Once the model performs well, deploy it to a production environment, such as a mobile app or a server, to classify incoming SMS messages in real-time.

6. Maintenance and Updates:

Continuously monitor the spam classifier's performance and update the model as needed to adapt to evolving spam techniques.

Maintain a feedback loop with users to address any false positives or false negatives and make necessary improvements.

DATASET USED:

https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

The dataset used for this task typically consists of SMS (text) messages, each labeled as either "spam" or "ham" (non-spam). The dataset is essential for training and evaluating the Naive Bayes classifier.

SMS Content (Text):

The actual text content of the SMS messages.

Label:

Each message is associated with a label, which can be "spam" or "ham," indicating whether it's a spam message or not

Data Preprocessing:

Data preprocessing is crucial to clean and prepare the dataset for machine learning. Common data preprocessing steps include:

1.Text Cleaning:

Remove special characters, symbols, punctuation marks, and other noise from the SMS messages. This helps standardize the text data and make it more amenable to analysis.

2.Tokenization:

Split the text into individual words or tokens. Tokenization is crucial for transforming text data into a format suitable for machine learning.

3.Lowercasing:

Convert all text to lowercase to ensure consistency in text data, as Naive Bayes is case-sensitive.

4.Stop Word Removal:

Eliminate common stop words (e.g., "and," "the," "is") from the text, as they often carry little meaning in spam classification.

5.Label Encoding:

Encode the "spam" and "ham" labels as numerical values, such as 0 for ham and 1 for spam. This numeric encoding makes the labels suitable for machine learning.

Feature Extraction Techniques:

Feature extraction involves converting the preprocessed text data into numerical features that can be used for training the Naive Bayes classifier. Common feature extraction techniques for text data include:

1.Bag of Words (BoW):

Create a vocabulary of unique words in the dataset and represent each SMS message as a vector where each element corresponds to the frequency of word occurrences. This approach results in a high-dimensional, sparse feature matrix.

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

Calculate the TF-IDF score for each word in the dataset. TF-IDF measures word importance based on how often it appears in a document and how unique it is across the dataset.

3.Word Embeddings:

Utilize pre-trained word embeddings (e.g., Word2Vec, GloVe) to convert SMS messages into dense vector representations. These dense vectors capture the semantic meaning of words and can be used as features for Naive Bayes.

4.N-grams:

- Consider capturing sequences of words by using n-grams (e.g., bigrams or trigrams) as features. This can help capture contextual information in the text data.
- After data preprocessing and feature extraction, you can train a Naive Bayes classifier on the processed dataset. The choice of feature extraction technique may significantly impact the classifier's performance, and it might require experimentation to determine the most effective approach for your specific dataset. The goal is to classify incoming SMS messages as either spam or ham based on the extracted features.
- The choice of machine learning algorithm, model training, and evaluation metrics is crucial when building a spam classifier for SMS messages using the Naive Bayes algorithm. Here's an explanation of each of these aspects:

Machine Learning Algorithm:

The Naive Bayes algorithm is a popular choice for text classification tasks, including spam detection. There are several reasons to choose Naive Bayes for this task:

1.Simplicity:

Naive Bayes is relatively simple to implement and understand, making it a good choice for beginners and for quick prototyping.

2.Efficiency:

It is computationally efficient and can handle large datasets with high dimensionality, which is common in text classification.

3.Text Data:

Naive Bayes works well with text data, which is the primary data type in SMS spam classification. It's particularly suited for tasks involving discrete features, like word occurrences.

4.Reasonable Performance:

Despite its simplicity, Naive Bayes often provides competitive performance in text classification tasks, especially when the dataset is well-preprocessed.

Model Training:

The Naive Bayes algorithm, you need to train your model using the preprocessed and feature-extracted dataset. The training process involves:

Splitting the Dataset:

Divide the dataset into two parts: a training set and a testing set. The training set is used to train the model, and the testing set is used to evaluate its performance.

Training the Model:

Use the training data to estimate the parameters of the Naive Bayes model. For text data, this typically involves computing probabilities of word occurrences in spam and ham messages.

Evaluation Metrics:

Choosing the right evaluation metrics is essential to assess how well your SMS spam classifier is performing. Common evaluation metrics for binary classification tasks like spam detection include:

1.Accuracy:

It measures the overall correctness of the classifier's predictions. However, accuracy alone may not be sufficient, especially if the dataset is imbalanced (i.e., one class greatly outnumbers the other).

2.Precision:

It calculates the proportion of true positive predictions (correctly identified spam) out of all positive predictions. High precision is essential when false positives are costly.

3.Recall (Sensitivity or True Positive Rate):

It measures the proportion of true positives out of all actual positives (spam). High recall is crucial when missing a spam message (false negative) is costly.

4.F1-Score:

The F1-score is the harmonic mean of precision and recall, providing a balance between these two metrics. It's a good metric when you want to balance precision and recall.

5.Area Under the Receiver Operating Characteristic Curve (AUC-ROC):

ROC curves show the trade-off between true positive rate and false positive rate at different classification thresholds. AUC-ROC provides an aggregate measure of classifier performance across different threshold settings.

6.Confusion Matrix:

The confusion matrix provides a detailed breakdown of true positives, true negatives, false positives, and false negatives, which can be helpful for a more in-depth understanding of model performance.

Document innovative techniques or approaches used during the development.

The development of an SMS spam classifier using the Naive Bayes algorithm may involve various innovative techniques and approaches, depending on the specific goals and challenges of the project. Here are some potentially innovative techniques and approaches that can be used during development:

1. Feature Engineering:

N-grams: Beyond traditional bag-of-words (BoW) or TF-IDF features, you can experiment with using n-grams, which capture sequences of words. This can help the model understand the context and language patterns more effectively.

2. Advanced Text Preprocessing:

Handling Multiple Languages:

If your dataset includes SMS messages in multiple languages, consider incorporating language identification as a preprocessing step. This can help tailor the model's features and parameters to the specific language of each message.

3. Ensemble Methods:

Ensembling multiple Naive Bayes classifiers with different feature representations (e.g., BoW and TF-IDF) can improve overall classification performance. Additionally, you can combine Naive Bayes with other algorithms, such as decision trees or random forests, in an ensemble to leverage the strengths of each model.

4. Active Learning:

Implement an active learning strategy to iteratively update the model as new data becomes available. This approach can help continuously improve the classifier's performance over time as it encounters new types of spam messages.

5. Adaptive Thresholds:

Instead of using a fixed classification threshold, you can explore adaptive thresholds that vary based on the specific needs and characteristics of the application. For example, you might adjust the threshold dynamically based on user feedback or the evolving spam landscape.

6. Deep Learning Models:

While Naive Bayes is a common choice for text classification, you can also consider using deep learning models, such as recurrent neural networks (RNNs) or convolutional neural networks (CNNs). These models can capture complex patterns and contextual information in text data.

7. Data Augmentation:

In cases where your dataset is limited, you can employ data augmentation techniques to artificially expand your dataset. Techniques like paraphrasing, synonym replacement, or adding noise to the text can help the model generalize better.

8. Anomaly Detection:

In addition to classifying messages as spam or ham, you can implement anomaly detection techniques to identify unusual patterns or novel types of spam messages that may not fit traditional spam characteristics.

9. Active Feedback Loop:

Implement a feedback loop with users to continuously update and improve the model based on their feedback on misclassified messages. Users can flag false positives and false negatives to refine the model.

10. Explainability:

Explore techniques for model explainability, such as LIME or SHAP values, to understand why the model makes specific predictions. This can help build trust and transparency in the classifier, which is important for user adoption.

11. Domain-Specific Rules:

Integrate domain-specific rules and heuristics into the classifier. For example, certain keywords or patterns may be indicative of spam in specific industries, and incorporating this knowledge can enhance the classifier's accuracy.

12. Real-time Learning:

Implement online or incremental learning, where the model continuously updates itself in real time as it encounters new data. This approach is valuable for adapting to emerging spam tactics.

Compile all the code files, including the data preprocessing, model training, and evaluation steps.

```
# 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 in


import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import nltk
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory


import os
print(os.listdir("../input"))
```

Output:

```
['spam.csv']
```

Checking the Length of SMS

```
import pandas
df_sms = pd.read_csv('../input/spam.csv',encoding='latin-1')
df_sms.head()
```

output:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

```
df_sms = df_sms.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
df_sms = df_sms.rename(columns={"v1":"label", "v2":"sms"})
df_sms.head()
```

output:

	label	sms
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
#Checking the maximum length of SMS
print (len(df_sms))
```

5572

df_sms.tail()

output:

	label	sms
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ì_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

#Number of observations in each label spam and ham

df_sms.label.value_counts()

output:

ham 4825

spam 747

Name: label, dtype: int64

df_sms.describe()

output:

	label	sms
count	5572	5572

	label	sms
unique	2	5169
top	ham	Sorry, I'll call later
freq	4825	30

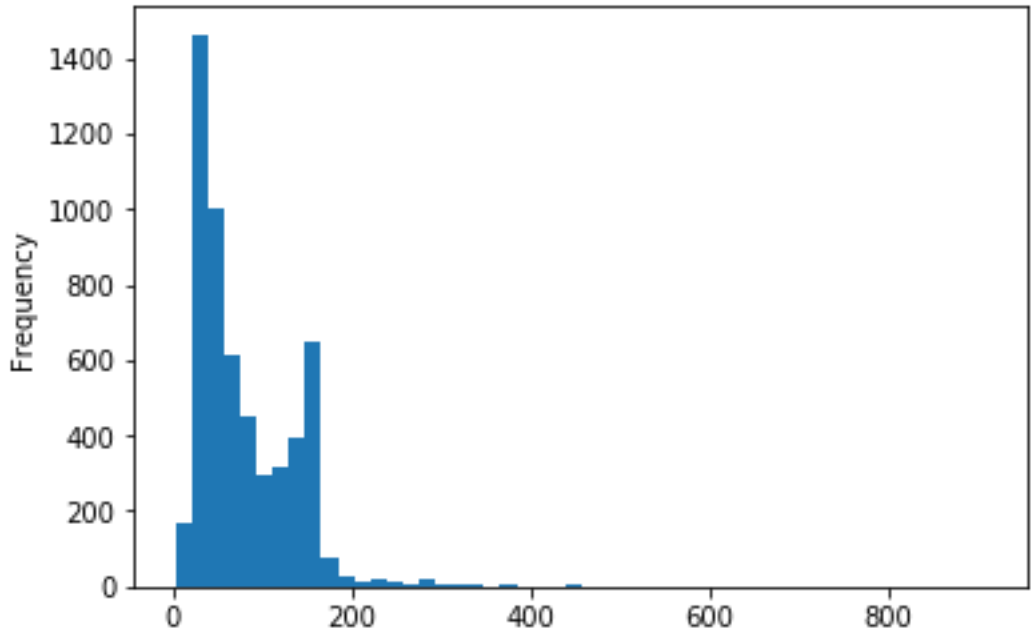
```
df_sms['length'] = df_sms['sms'].apply(len)
df_sms.head()
```

output:

	label	sms	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

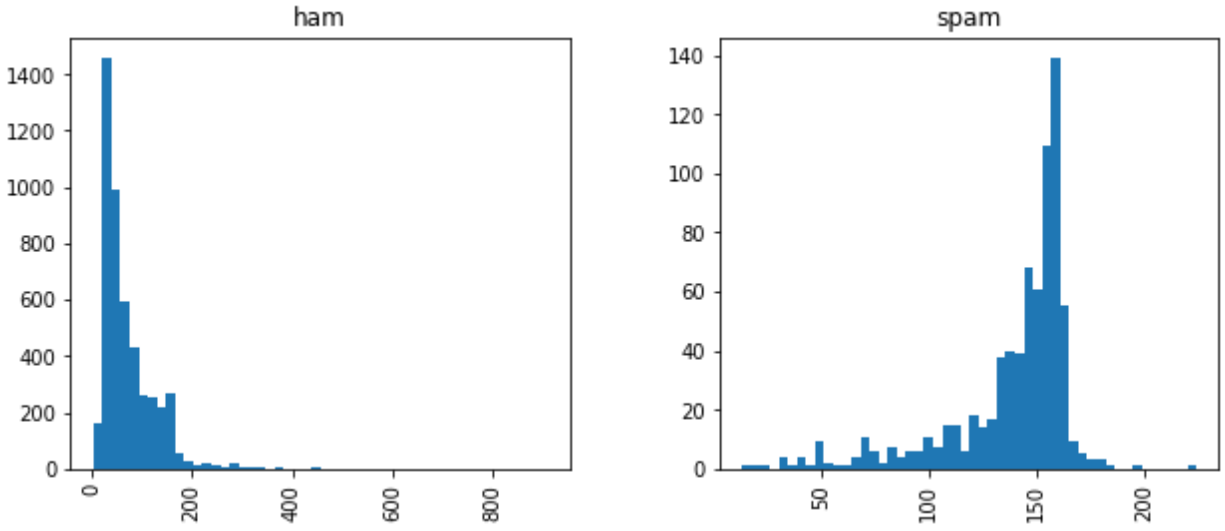
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
df_sms['length'].plot(bins=50, kind='hist')
<matplotlib.axes._subplots.AxesSubplot at 0x7a0877b84978>
```

output:




```
df_sms.hist(column='length', by='label', bins=50,figsize=(10,4))
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7a0877b0bb00>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7a0877a81d30>],
      dtype=object)
```

output:




```
df_sms.loc[:,'label'] = df_sms.label.map({'ham':0, 'spam':1})
```

```
print(df_sms.shape)
df_sms.head()
```

(5572, 3)

	label	sms	length
0	0	Go until jurong point, crazy.. Available only ...	111
1	0	Ok lar... Joking wif u oni...	29
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	0	U dun say so early hor... U c already then say...	49
4	0	Nah I don't think he goes to usf, he lives aro...	61

Implementation of Bag of Words Approach

Step 1: Convert all strings to their lower case form.

```
documents = ['Hello, how are you!',
        'Win money, win from home.',
        'Call me now.',
        'Hello, Call hello you tomorrow?']
lower_case_documents = []
lower_case_documents = [d.lower() for d in documents]
print(lower_case_documents)
```

output:

['hello, how are you!', 'win money, win from home.', 'call me now.', 'hello, call hello you tomorrow?']

Step 2: Removing all punctuations

```
sans_punctuation_documents = []
import string
for i in lower_case_documents:
    sans_punctuation_documents.append(i.translate(str.maketrans("","", string.punctuation)))


sans_punctuation_documents
```

output:

```
['hello how are you',
 'win money win from home',
 'call me now',
 'hello call hello you tomorrow']
```

Step 3: Tokenization

```
preprocessed_documents = [[w for w in d.split()] for d in sans_punctuation_documents]
preprocessed_documents
```

output:

```
[['hello', 'how', 'are', 'you'],
 ['win', 'money', 'win', 'from', 'home'],
 ['call', 'me', 'now'],
 ['hello', 'call', 'hello', 'you', 'tomorrow']]
```

Step 4: Count frequencies

```
frequency_list = []
import pprint
from collections import Counter
frequency_list = [Counter(d) for d in preprocessed_documents]
pprint.pprint(frequency_list)
```

output:

[Counter({'hello': 1, 'how': 1, 'are': 1, 'you': 1}),

Counter({'win': 2, 'money': 1, 'from': 1, 'home': 1}),

Counter({'call': 1, 'me': 1, 'now': 1}),

Counter({'hello': 2, 'call': 1, 'you': 1, 'tomorrow': 1})]

Data preprocessing with CountVectorizer():

In above step, we implemented a version of the CountVectorizer() method from scratch that entailed cleaning our data first. This cleaning involved converting all of our data to lower case and removing all punctuation marks. CountVectorizer() has certain parameters which take care of these steps for us. They are:

lowercase = True

The lowercase parameter has a default value of True which converts all of our text to its lower case form.

token_pattern = (?u)\b\w\w+\b

The token_pattern parameter has a default regular expression value of (?u)\b\w\w+\b which ignores all punctuation marks and treats them as delimiters, while accepting alphanumeric strings of length greater than or equal to 2, as individual tokens or words.

stop_words

The stop_words parameter, if set to english will remove all words from our document set that match a list of English stop words which is defined in scikit-learn. Considering the size of our dataset and the fact that we are dealing with SMS messages and not larger text sources like e-mail, we will not be setting this parameter value.

Code:

count_vector.fit(documents)

count_vector.get_feature_names()

output:

['are',

'call',

'from',

'hello',

'home',

'how',

'me',

'money',

'now',

'tomorrow',

'win',

'you']

```
doc_array = count_vector.transform(documents).toarray()
doc_array
```

output:

```
array([[1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1],
       [0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 2, 0],
       [0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
       [0, 1, 0, 2, 0, 0, 0, 0, 0, 1, 0, 1]])
```

```
frequency_matrix = pd.DataFrame(doc_array, columns = count_vector.get_feature_names())
frequency_matrix
```

output:

	are	call	from	hello	home	how	me	money	now	tomorrow	win	you
0	1	0	0	1	0	1	0	0	0	0	0	1
1	0	0	1	0	1	0	0	1	0	0	2	0
2	0	1	0	0	0	0	1	0	1	0	0	0

	are	call	from	hello	home	how	me	money	now	tomorrow	win	you
3	0	1	0	2	0	0	0	0	0	1	0	1

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_sms['sms'],
                                    df_sms['label'],test_size=0.20,
                                    random_state=1)
# Instantiate the CountVectorizer method
count_vector = CountVectorizer()


# Fit the training data and then return the matrix
training_data = count_vector.fit_transform(X_train)


# Transform testing data and return the matrix.
testing_data = count_vector.transform(X_test)
```

*Implementation of Naive Bayes Machine Learning Algorithm *

I will use sklearns sklearn.naive_bayes method to make predictions on our dataset.

Specifically, we will be using the multinomial Naive Bayes implementation. This particular classifier is suitable for classification with discrete features (such as in our case, word counts for text classification). It takes in integer word counts as its input. On the other hand Gaussian Naive Bayes is better suited for continuous data as it assumes that the input data has a Gaussian(normal) distribution.

```
from sklearn.naive_bayes import MultinomialNB
naive_bayes = MultinomialNB()
naive_bayes.fit(training_data,y_train)
```

output:

MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

Evaluating our model

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

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)))

output:

Accuracy score: 0.9847533632286996

Precision score: 0.9420289855072463

Recall score: 0.935251798561151

F1 score: 0.9386281588447652

CONCLUSION:

Building an SMS spam classifier using Naive Bayes involves preparing a dataset, preprocessing text data, training the model, and evaluating its performance. To enhance results, consider innovative techniques, real-time feedback, and iterative improvements. Deploy the model to classify incoming SMS messages as spam or legitimate. Maintenance is key to ensuring its ongoing accuracy.