Building a Smarter AI-Powered Classifier

Phase 5: Project Documentation & Submission

Topic: In this section you will document the complete project and prepare it forsubmission.

Introduction:

- An SMS spam classifier is a powerful tool designed to sift through the deluge of text messages we receive on our mobile devices, separating genuine messages from the intrusive and often malicious ones.

- In an era where communication via SMS is an integral part of our daily lives, the need to combat unsolicited and potentially harmful text messages has become increasingly vital.

- This classifier leverages advanced machine learning techniques and algorithms to automatically identify and filter out unwanted SMS messages, ensuring that users are not inundated with spam, scams, or fraudulent content.

- Advanced SMS spam classifiers can analyze incoming messages in real-time, swiftly identifying and diverting potential spam messages before they even reach the user's inbox.

- Some SMS spam classifiers also raise awareness by informing users about common spam techniques and encouraging them to be cautious and report suspicious messages.

- These classifiers harness the power of artificial intelligence to make intelligent decisions about the nature of incoming messages. They can recognize patterns, anomalies, and language cues that signify spam, allowing for accurate classification.

SMS spam classifiers often extend their protection to multiple mobile platforms, ensuring that users on various devices and operating systems can benefit from a consistent and spam-free messaging experience.

Data Source:

Building an effective SMS spam classifier requires a labeled dataset of SMS messages, which are categorized as either spam or legitimate (ham) messages.

Dataset Link: (https://www.kaggle.com/datasets/uciml/sms-spam-collection- dataset)

Building a Smarter AI-Powered Classifier

Given Dataset:

	type	text
0	ham	Hope you are having a good week. Just checking in
1	ham	K..give back my thanks.
2	ham	Am also doing in cbe only. But have to pay.
3	spam	complimentary 4 STAR Ibiza Holiday or £10,000 ...
4	spam	okmail: Dear Dave this is your final notice to...
...	...	...
5554	ham	You are a great role model. You are giving so ...
5555	ham	Awesome, I remember the last time we got someb...
5556	spam	If you don't, your prize will go to another cu...
5557	spam	SMS. ac JSco: Energy is high, but u may not kn...
5558	ham	Shall call now dear having food

5559 rows × 2 columns

Here's a list of tools and software commonly used in the process:

1. **Programming Language:**

- Python is commonly used for natural language processing (NLP) tasks like SMS spam classification.

2. **Machine Learning Frameworks and Libraries:**

- Scikit-Learn offers a wide range of machine learning algorithms for classification tasks.
- NLTK provides tools and libraries for natural language processing.
- A simple NLP library for processing textual data.
- A more advanced NLP library for text preprocessing and analysis.

3. **Data Preprocessing and feature Extraction:**

- **Pandas:** For data manipulation and cleaning.
- **NumPy:** For numerical operations on data.
- **Regular Expressions (RegEx):** Useful for text pattern matching and extraction.
- **TF-IDF (Term Frequency -Inverse Document Frequency):** commonly Used for feature extraction in text classification tasks.

4. **Feature Extraction and Text Vectorization:**

- **Word Embeddings:** Techniques like Word2Vec, FastText, and GloVe can be used to convert words into numerical vectors.
- **Bag of Words:** A simple technique for text vectorization.
- **Count Vectorization:** Another technique for text vectorization.

5. **Machine Learning Algorithms:**

- **Naive Bayes:** A popular algorithm for text classification.

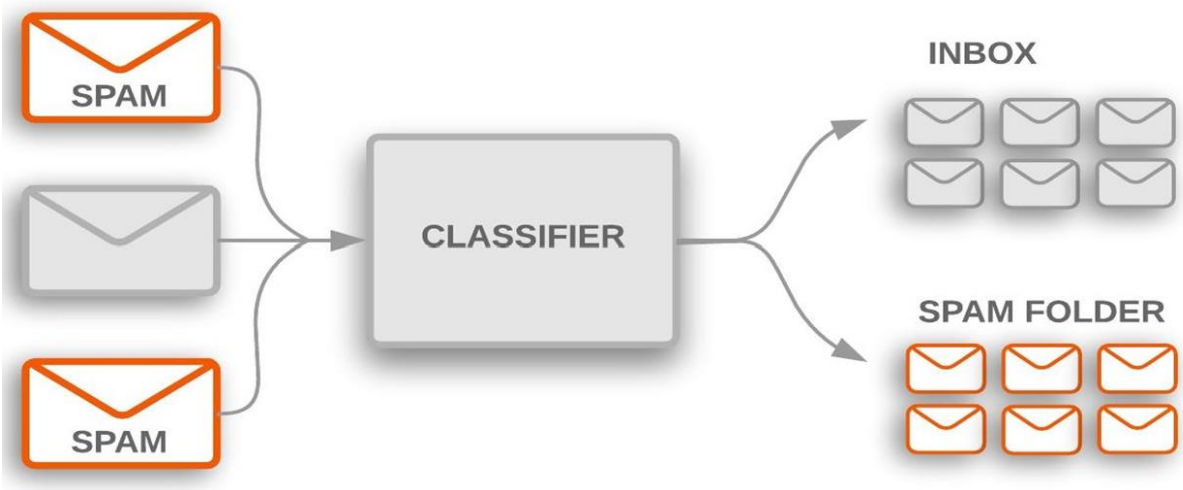
- **Support Vector Machine (SVM):** Effective for binary classification.
- **Random Forest:** Ensemble learning method that can be used for classification
- **Logistic Regression:** Simple and interpretable for text classification tasks.

6. **Evaluation Matrices:**

- **Precision, Recall, F1-Score:** Common metrics for evaluating the performance of text classification models.
- **Confusion Matrix:** Provides detailed information about model predictions.

7. **Collaboration and Documentation:**

- **Jupyter Notebook**: For interactive data exploration and documentation.
- **Confluence, Jira or other project Management tools:** For team collaboration and task tracking.



1. DESIGN THINKING AND IN FORM OF DOCUMENT

Design thinking is a human-centred, iterative approach to problem solving and product development. While it's often associated with physical products and user experiences, it can also be applied to the development of an SMS spam classifier. Here's how design thinking can be applied to the development of an SMS spam classifier:

1. Empathize:

- Begin by understanding the needs and pain points of the users. In this case, the users may include both end-users who receive SMS messages and administrators who manage the spam classifier.

2. Define the Problem:

- Clearly define the problem you are trying to solve. For instance, you might be looking to reduce the number of spam SMS messages that users receive while ensuring that important messages are not mistakenly classified as spam.

3. Ideate:

- Brainstorms Solutions:

Encourage cross-functional teams to brainstorm ideas for addressing the problem. Consider various approaches, from text analysis techniques to user interface design.

4. Prototype:

- Develop prototypes of the SMS spam classifier system. This can include mock-ups of user interfaces, algorithms for text analysis, and workflows for managing spam messages.

5. Test:

- Gather Feedback:

Test the prototypes with potential end-users and gather their feedback. Are they satisfied with the user interface? Are the spam classification results accurate? What improvements do they suggest?

6. Iterate:

- Refine the Solution:

Based on user feedback and testing results, iterate on the prototypes and make improvements to the SMS spam classifier system. This may involve refining the text analysis algorithms, improving the user interface, or enhancing system performance.

7. Implement:

- Develop the final Solution:

Once you have a refined prototype, proceed to develop the final SMS spam classifier system. This involves writing the code, implementing the algorithms, and setting up the infrastructure.

8. **Test and Validate:**

- Rigorously test the final system to ensure it meets the defined objectives. Use a diverse set of SMS messages, including both spam and legitimate messages, to validate the classifier's accuracy.

9. **Scale and Adapt:**

- As the system is in use, be prepared to scale it to handle a growing volume of SMS messages. Adapt the system to evolving spam tactics and user needs.

10. User Education:

- Educate end-users on how to use the spam classifier effectively. Provide clear instructions on what to do with messages that are flagged as spam or false positives.

2. DESIGN INTO INNOVATION

1. Data Collection:

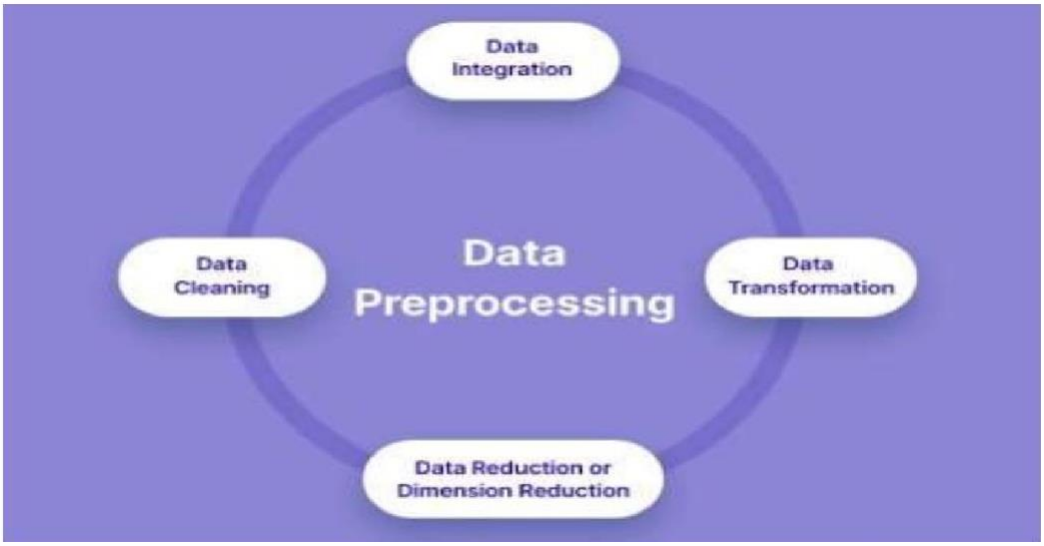
Depending on your data source, you may need to collect the SMS messages. If you are collecting your own data, consider these methods:

- Mobile app or SMS backup tools.
- Web scraping of publicly available SMS datasets.
- Collaborative efforts with users to contribute their SMS data.

2. Data Preprocessing:

Before using the data for training, you should preprocess it. This includes:

- Removing duplicates and irrelevant messages.
- Tokenization: Splitting messages into individual words or tokens.
- Removing stop words (common words like "and," "the," etc.).
- Normalizing text (converting to lowercase, handling abbreviations, etc.).
- Balancing the dataset to ensure there's an even distribution of spam and ham messages (if needed).



Python Program:

#import necessary libraries import numpy as np import pandas as pd import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import re import nltk

from nltk.corpus

import stopwords

```
from sklearn.feature_extraction.text
import TfidfVectorizer
from sklearn.feature_extraction.text
import CountVectorizer,TfidfTransformer
 from sklearn.model_selection
import train_test_split
from sklearn.naive_bayes
import MultinomialNB
from sklearn.metrics
import accuracy_score,classification_report, confusion_matrix
#Load the Dataset
df=pd.read_csv("C:/Users/ELCOT/Downloads/sms_spam.csv”)
# Display the first few rows of the dataset to get an overview
print("Dataset Preview:")
print(df.head())
#Data Preprocessing #text cleaning
def clean_text(text):
'''Do lowercase, remove text in square brackets, links, punctuation and words containing numbers.'''
text = str(text).lower()
text = re.sub(r'\[.*?\]', '', text)
text = re.sub(r'https?://\S+|www\.\S+', '', text) text = re.sub(r'+', '', text)
text = re.sub(r'[%s]' % re.escape(string.punctuation), '', text) text = re.sub(r'\n', '', text)
text = re.sub(r'\w*\d\w*', '', text) return text
def preprocessing(text): cleaned_text = clean_text(text)
# remove stopwords
cleaned_text = ' '.join(word for word in cleaned_text.split(' ') if word not in stop_words)
# do stem method
cleaned_text = ' '.join(stemmer.stem(word) for word in cleaned_text.split(' ')) return cleaned_text
```

```
df['Cleaned_Message'] = df['text'].apply(preprocessing)

# let's show new length after process

df['New_length'] = df['text'].apply(lambda x: len(x.split(' '))) df.head()

# Split data into training and testing sets

X_train ,X_test , y_train , y_test = train_test_split(train_data , y_label , test_size=0.2 , random_state=42)

# Display the pre-processed data

print("\nPreprocessed Data:")

print(X_train[:5])

print(y_train[:5])
```

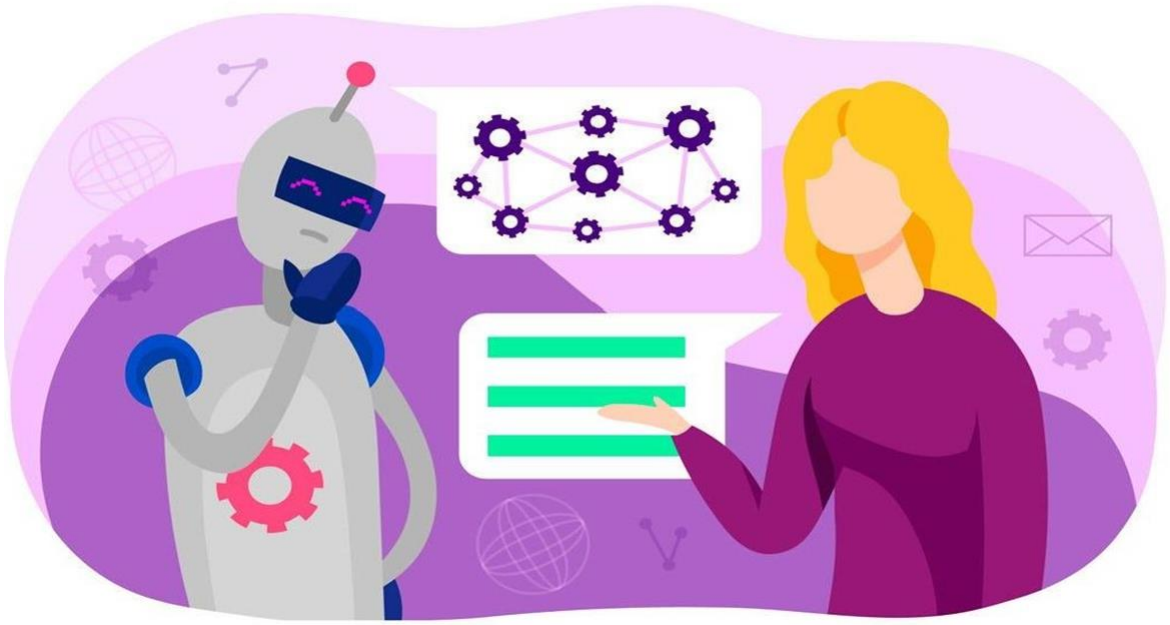
3. **Feature Engineering:**
   - Feature engineering is the initial step in building an efficient SMS Spam Classifier. It involves extracting meaningful information from the raw text data. Commonly used features include Text Preprocessing, TF-IDF, Word Embeddings.
4. **Model Selection:**
   - Choose from various machine learning and deep learning models for text classification include Naive bayes, Logistic Regression, Support Vector Machine, Decision Trees and Random Forest.
5. **Model Training:**
   - Train the selected model(s) on the training data.
   - Use the validation set to tune hyperparameters (e.g., learning rate, batch size) and assess model performance.
6. **Hyperparameter Tuning:**
   - Experiment with different models and hyperparameters to find the best- performing model.
   - Consider techniques like cross-validation and grid search for hyperparameter tuning.
7. **Evaluation:**
   - Evaluate the models on the test set using metrics like accuracy, precision, recall, F1 score, and ROC AUC.
8. **Model Interpretability:**
   - For some applications, it's essential to understand why a model makes certain predictions. Techniques like SHAP values and LIME can help interpret model decisions.
9. **Deployment:**
   - Once you've selected the best model, deploy it to your application or service for real-time or batch spam classification.

10. Maintenance and Monitoring:

- Continuously monitor your deployed model's performance, and be prepared to retrain it periodically with new data to maintain its effectiveness.

Remember that the choice of model can significantly impact the performance of your SMS spam classifier. It's important to consider the nature of your dataset, the available computational resources, and the desired trade-offs between model complexity and accuracy when making your decision.

Additionally, ongoing monitoring and retraining may be necessary to maintain the model's performance as the spam landscape evolves.



3. BUILD LOADING AND PREPROCESSING THE DATASET

1**. Data Collection:**

Gather a labelled dataset of SMS messages, with each message categorized as either spam or non-spam (ham).

2. Load the Dataset:

Import relevant libraries, such as pandas for data manipulation and numpy for numerical operations.

- Load the dataset into a pandas Data Frame for easy data handling.
- You can use pd.read_csv() for CSV files or other appropriate functions for different file formats.

Building a Smarter AI-Powered Classifier

Program:

```
import numpy as np
import pandas as pd
import pandas as pd
import matplotlib.pyplot as plt
 import seaborn as sns
import re import nltk
from sklearn.ensemble
import RandomForestClassifier from sklearn.neighbors
import KNeighborsClassifier from sklearn.svm
import SVC
from sklearn.model_selection
from nltk.corpus
import stopwords
from sklearn.naive_bayes
import MultinomialNB
from sklearn.metrics
import accuracy_score,classification_report, confusion_matrix,
classification_report, accuracy_score, f1_score from sklearn
import metrics from sklearn.feature_extraction.text
import TfidfVectorizer
from sklearn.feature_extraction.text
import CountVectorizer,TfidfTransformer from sklearn.model_selection
import train_test_split
```

Loading Dataset:

```
df=pd.read_csv("C:/Users/ELCOT/Downloads/sms_spam.csv”)
```

	type	text
0	ham	Hope you are having a good week. Just checking in
1	ham	K..give back my thanks.
2	ham	Am also doing in cbe only. But have to pay.
3	spam	complimentary 4 STAR Ibiza Holiday or £10,000 ...
4	spam	okmail: Dear Dave this is your final notice to...
...	...	...
5554	ham	You are a great role model. You are giving so ...
5555	ham	Awesome, I remember the last time we got someb...
5556	spam	If you don't, your prize will go to another cu...
5557	spam	SMS. ac JSco: Energy is high, but u may not kn...
5558	ham	Shall call now dear having food

5559 rows × 2 columns

3. Data Exploration:

Explore the dataset to understand its structure and contents. Check for the presence of missing values, outliers, and data types of each feature.

4. Data Cleaning:

- Clean the text data by performing the following tasks:
- Remove any special characters, punctuation, and numbers.
- Convert text to lowercase for uniformity.
- Tokenize the text (split it into words)

5. Feature Selection:

Identify the most relevant information from your data that will be used to train the model. In text classification, the features are typically the words or phrases in the SMS messages.

6. Feature Engineering:

Feature engineering is the initial step in building an efficient SMS Spam Classifier. It involves extracting meaningful information from the raw text data. Commonly used features include Text Preprocessing, TF-IDF, Word Embeddings.

7. Data Encoding:

Encode the labels (spam or not spam) as binary values, where 0 represents not spam (ham), and 1 represents spam.

8. Data Splitting:

Split your dataset into training and testing sets. A common split is 80% for training and 20% for testing. This helps you evaluate the model's performance.

4. PERFORMING DIFFERENT ACTIVITIES LIKE FEATURE ENGINEERING, MODEL TRAINING, EVALUATION, etc…

1**. Feature engineering:**

As mentioned earlier, Feature engineering is the initial step in building an efficient SMS Spam Classifier. It involves extracting meaningful information from the raw text data. Commonly used features include Text Preprocessing, TF-IDF, Word Embeddings.

2**. Data Preprocessing & Visualization:**

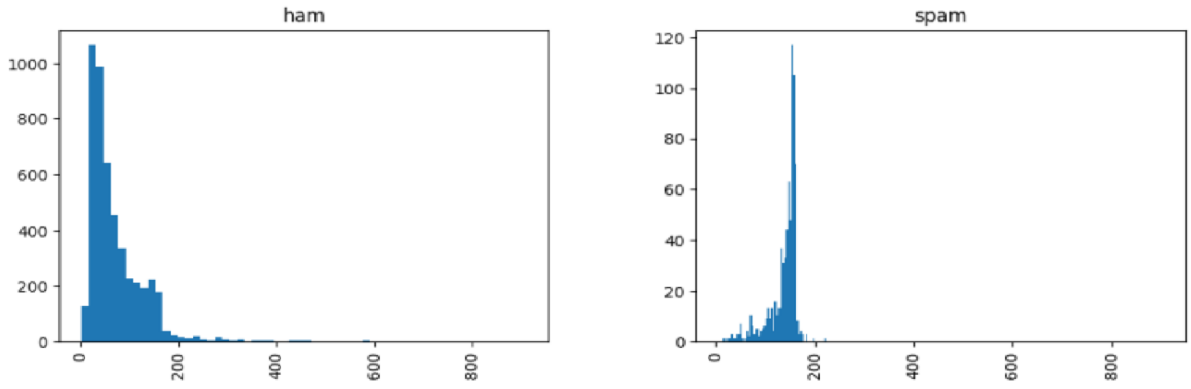
Once the dataset is loaded into the machine learning environment, you may need to preprocess it before you can start training and evaluating your model. This may involve cleaning the data, transforming the data into a suitable format, and splitting the data into training and test sets.

Visualizing and Pre-Processing of data

In[1]:

```
df.hist(column='length',by='type',bins=60,figsize=(12,4)); plt.xlim(-40,950);
```

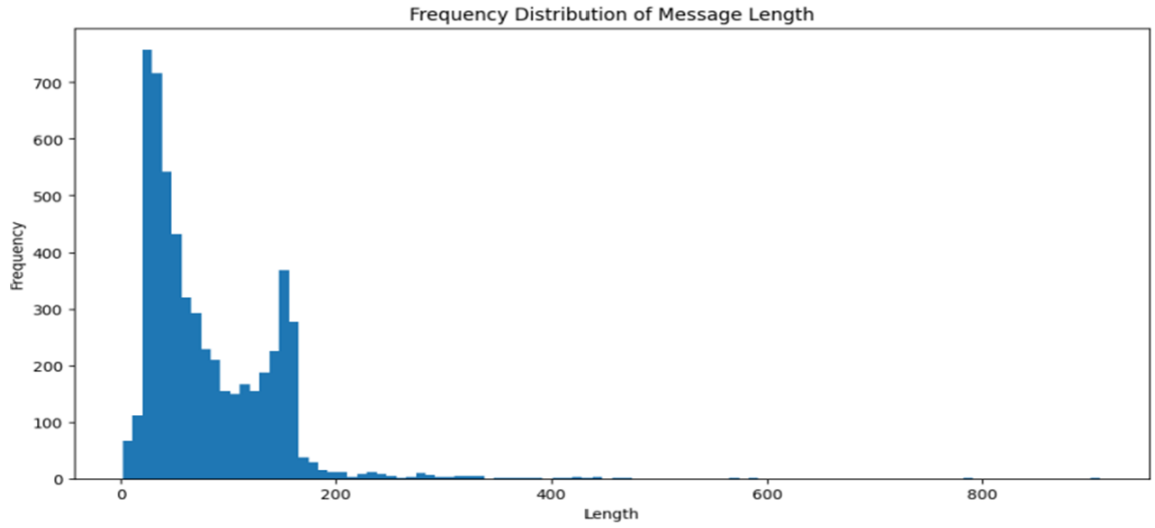
Out[1]:



In[2]:

```
plt.figure(figsize=(12,6))
df['length'].plot(bins=100, kind='hist')
# with 100 length bins (100 length intervals)
plt.title("Frequency Distribution of Message Length")
plt.xlabel("Length")
plt.ylabel("Frequency")
```

Out[2]:



Word cloud:

A word cloud is a visual representation of text data that displays the most frequently occurring words in a dataset. It can be used to identify the most common words in spam messages and help improve the accuracy of a spam classifier. You can use Python's word cloud library to create a word cloud.

In[3]:

```
data_ham = df[df['spam'] == 0].copy()
data_spam = df[df['spam'] == 1].copy()
```

In[4]:

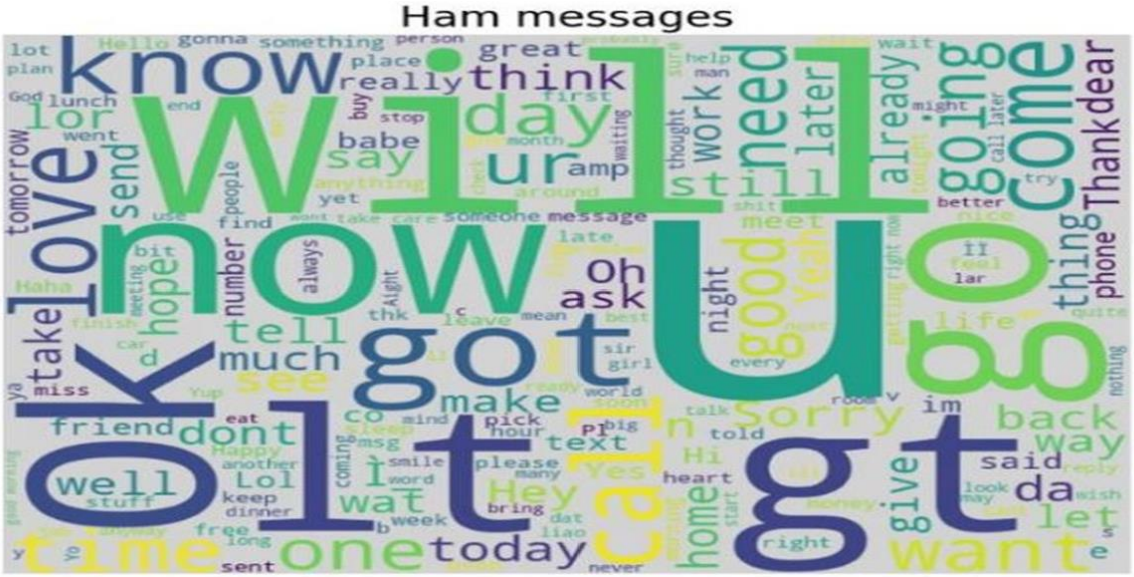
```
def show_wordcloud(data_spam_or_ham, title):
text = '
'.join(data_spam_or_ham['message'].astype(str).tolist()) stopwords =
set(wordcloud.STOPWORDS)
```

```
fig_wordcloud = wordcloud.WordCloud(stopwords=stopwords,background_color='lightgrey',
colormap='viridis', width=800, height=600).generate(text)

plt.figure(figsize=(10,7), frameon=True)

plt.imshow(fig_wordcloud)

plt.axis('off')

plt.title(title, fontsize=20 ) plt.show()
```

In[5]:

```
show_wordcloud(data_ham, "Ham messages")
```

Out[5]:



In[6]:

```
show_wordcloud(data_spam, "Spam messages")
```

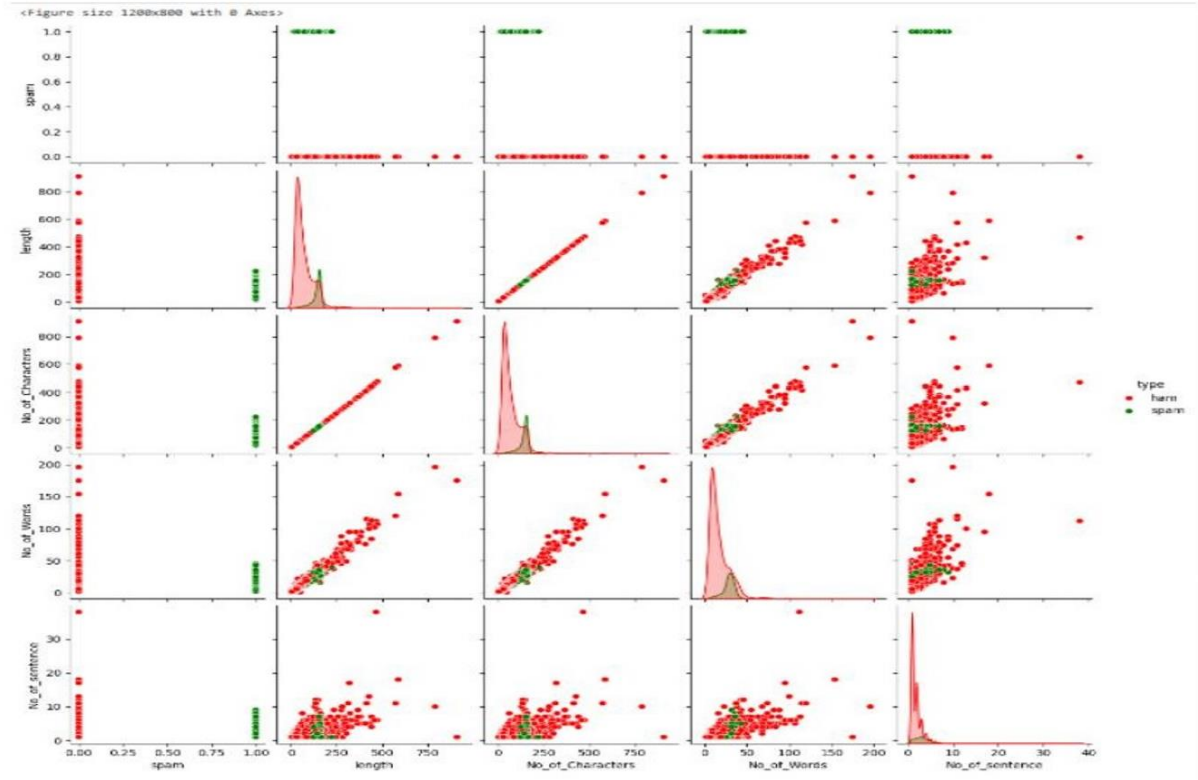
Out[6]:

In[7]:

Import seaborn as sns plt.figure(figsize=(12,8)) cols=["red","green"]

fg = sns.pairplot(data=df, hue="type",palette=cols) plt.show(fg)

Out[7]:



Model Training:

In[8]:

y = data["text"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

In[9]:

classifiers = [MultinomialNB(),

RandomForestClassifier(), KNeighborsClassifier(), SVC()]

for cls in classifiers: cls.fit(X_train, y_train)

pipe_dict = {0: "NaiveBayes", 1: "RandomForest", 2: "KNeighbours",3: "SVC"}

In[10]:

for i, model in enumerate(classifiers):

cv_score = cross_val_score(model, X_train,y_train,scoring="accuracy", cv=10) print("%s: %f " % (pipe_dict[i], cv_score.mean()))

Out[10]:

NaiveBayes: 0.967552

RandomForest: 0.974537

KNeighbours: 0.911450

SVC: 0.974086

In[11]:

creating lists of varios scores precision =[]

recall =[] f1_score = []

trainset_accuracy = [] testset_accuracy = [] for i in classifiers:

pred_train = i.predict(X_train) pred_test = i.predict(X_test)

prec = metrics.precision_score(y_test, pred_test) recal = metrics.recall_score(y_test, pred_test) f1_s = metrics.f1_score(y_test, pred_test) train_accuracy = model.score(X_train,y_train) test_accuracy = model.score(X_test,y_test) precision.append(prec)

recall.append(recal) f1_score.append(f1_s) trainset_accuracy.append(train_accuracy) testset_accuracy.append(test_accuracy)

In[12]:

data = {'Precision':precision,

'Recall':recall,'F1score':f1_score,'Accuracy on Testset':testset_accuracy,'Accuracy on Trainset':trainset_accuracy}

Results = pd.DataFrame(data, index =["NaiveBayes", "RandomForest", "KNeighbours","SVC"])

In[13]:

cmap2 = ListedColormap(["#E2CCFF","#E598D8"])
Results.style.background_gradient(cmap=cmap2)

Out[13]:

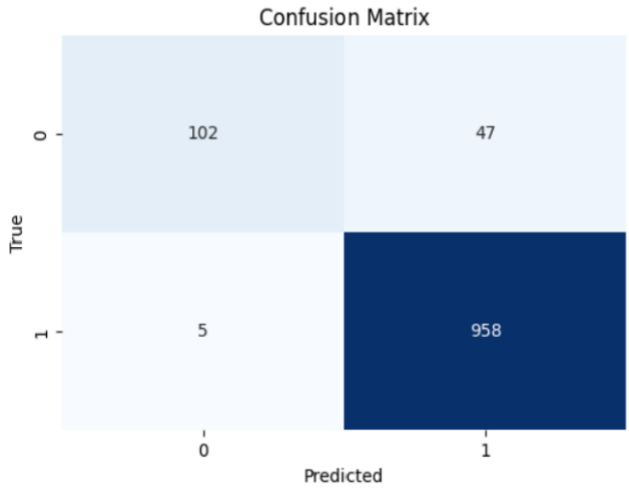
	Precision	Recall	F1score	Accuracy on Testset	Accuracy on Trainset
NaiveBayes	1.000000	0.705882	0.827586	0.974775	0.997521
RandomForest	1.000000	0.816176	0.898785	0.974775	0.997521
KNeighbours	0.977778	0.323529	0.486188	0.974775	0.997521
SVC	0.990909	0.801471	0.886179	0.974775	0.997521

Confusion Matrix In[14]:

Data visualization - Confusion Matrix

```
cm = confusion_matrix(Y_test, prediction_on_test_data)

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt="d", cmap='Blues', cbar=False)

plt.xlabel('Predicted')

plt.ylabel('True') plt.title('Confusion Matrix') plt.show()
```

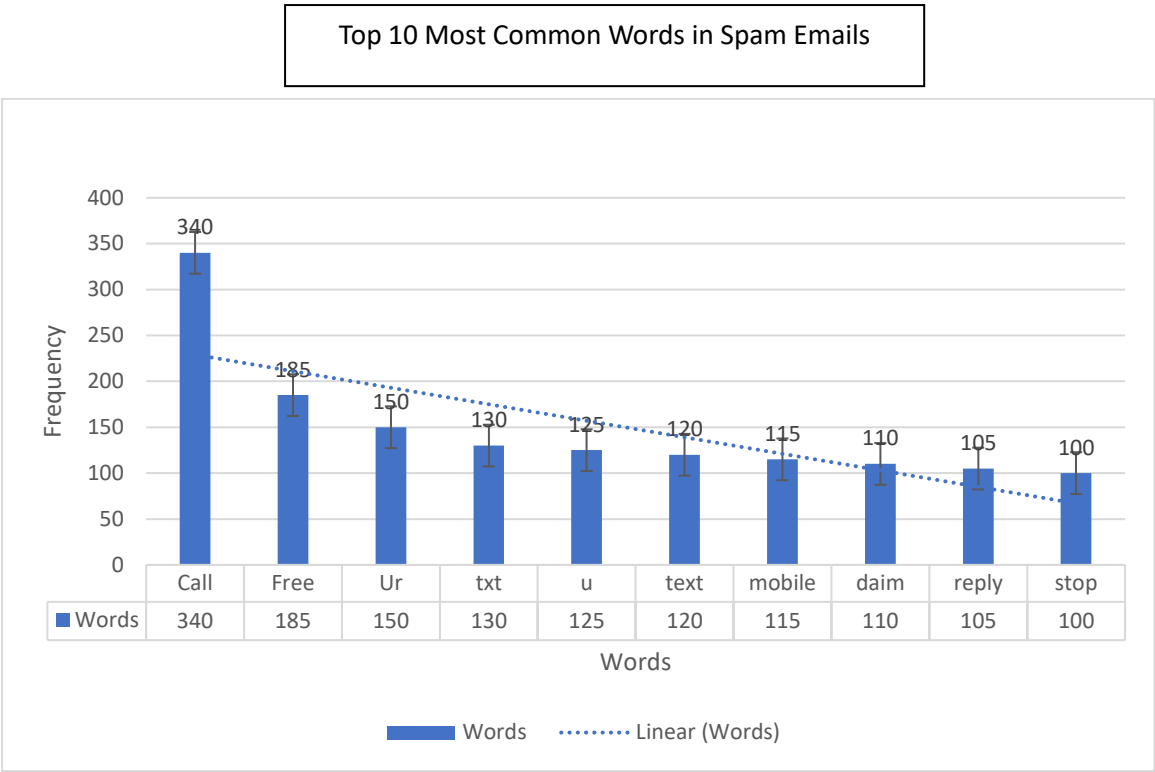
Out[14]:



In[25]:

```
stop_words = set(stopwords.words('english'))

spam_words = " ".join(df[df['type'] == 0]['text']).split()

ham_words = " ".join(df[df['type'] == 1]['text']).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()
```

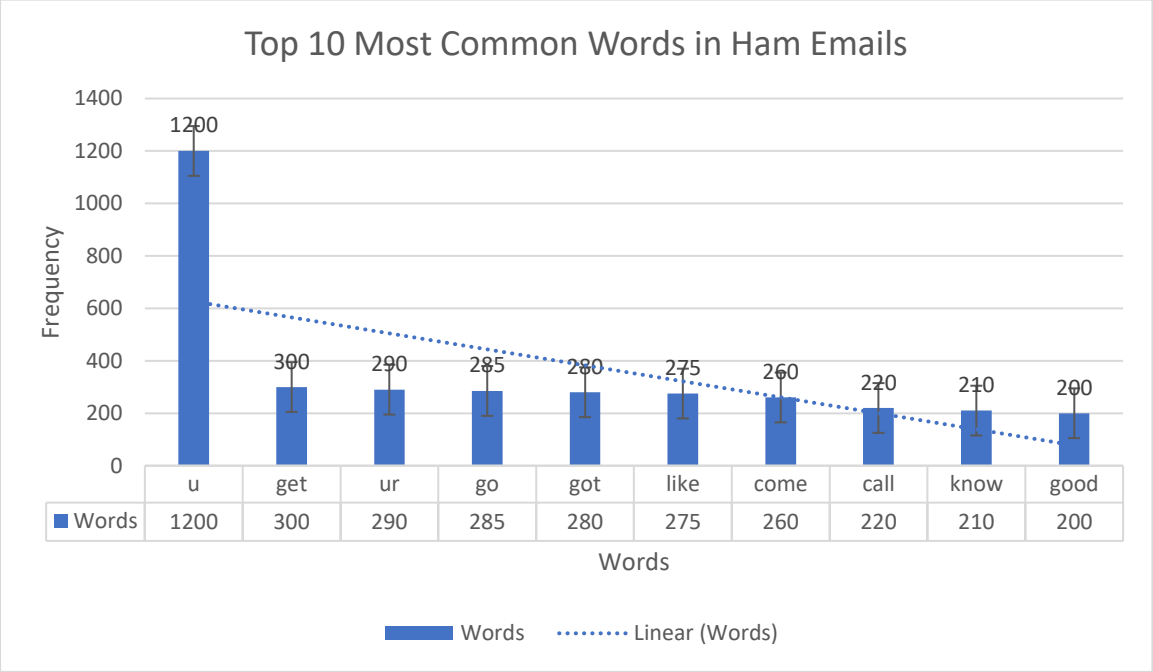
Out[25]:



In[26]:

```
ham_word_freq = Counter([word.lower() for word in ham_words if word.lower() not in stop_words and word.isalpha()])

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

plt.bar(*zip(*ham_word_freq.most_common(10)), color='maroon')

plt.xlabel('Words')

plt.ylabel('Frequency')

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

plt.xticks(rotation=45)

plt.show()
```

Out[26]:



Advantages:

1. **Reduction of Spam:**

significantly reduce the amount of spam messages that users receive. This leads to a cleaner and more efficient communication experience.

2. **Enhanced User Experience:**

By filtering out unwanted and potentially harmful messages, SMS spam classifiers improve the overall user experience. Users are less likely to be annoyed by spam, phishing attempts, or fraudulent schemes.

3. **Time and resource Savings:**

Users save time and resources that would otherwise be wasted on sifting through spam messages. This is particularly valuable for individuals and businesses with high volumes of incoming messages.

4. **Production from Scams:**

SMS spam classifiers can protect users from falling victim to scams, fraudulent offers, and phishing attacks. This contributes to enhanced online safety.

5. **Customization:**

Many SMS spam classifiers allow users to customize their filtering preferences, ensuring that important messages are not mistakenly classified as spam.

6. **Efficiency:**

SMS spam classifiers work quickly and efficiently, automatically categorizing messages, which is particularly important in real-time communication.

Disadvantages:

1. **False Positives:**

One of the primary drawbacks is the potential for false positives.

Legitimate messages may be incorrectly classified as spam, leading to important communications being missed. Users need to periodically check their spam folders for false positives.

2. **False Negatives:**

On the flip side, SMS spam classifiers can occasionally miss certain types of spam, allowing some unwanted messages to reach the inbox.

3. **Ongoing Maintenance:**

SMS spam classifiers require ongoing maintenance and updates to adapt to new spam tactics and variations. The arms race between spammers and classifiers are a constant challenge.

4. **Privacy Concerns:**

Some SMS classifiers may raise privacy concerns because they involve scanning the content of text messages. Users might be concerned about the storage and handling of their personal data.

5. **Resource Inventive:**

Implementing SMS spam classifiers on a large scale, such as at the network level, can be resource-intensive in terms of computational power and data storage.

6. **Cultural and Language Specificity:**

SMS classifiers may not perform as well in languages and cultures different from those in which they were trained. They might struggle with understanding context and nuances in various languages and dialects.

7. **Adaption Challenges:**

New spamming techniques and evolving tactics by spammers can pose challenges for SMS spam classifiers. Staying ahead of the spammers requires constant adaptation and development.

Conclusion:

- ✓ In conclusion, SMS spam classifiers represent a vital technology in the realm of text message communication, serving as a shield against the deluge of unwanted and potentially harmful messages that users face on a daily basis.
- ✓ These classifiers, driven by advanced machine learning and natural language processing techniques, offer both users and businesses a range of advantages, including the reduction of spam, enhanced user experiences, time and resource savings, and protection from scams.
- ✓ They efficiently sift through incoming messages and, when configured properly, provide users with a cleaner, safer, and more productive communication environment.

- ✓ However, the deployment of SMS spam classifiers is not without its challenges.
- ✓ False positives and false negatives can occasionally disrupt the flow of legitimate messages, and ongoing maintenance is essential to keep up with the ever-evolving tactics of spammers.
- ✓ Privacy concerns, resource-intensive implementations, and language and cultural specificity also require careful consideration.
- ✓ Nonetheless, the development and refinement of SMS spam classifiers continue to address these challenges.
- ✓ These technologies have played a pivotal role in safeguarding users from scams and reducing the annoyance caused by spam
- ✓ As the landscape of text message communication evolves, SMS spam classifiers will remain a valuable tool, adapting to new threats and ensuring that the benefits of text messaging can be enjoyed without the intrusion of spam.
- ✓ Through constant innovation and fine-tuning, these classifiers contribute to a more secure and efficient digital communication experience.