Email Spam Detection with Machine Learning: A Comprehensive Guide

Azim Khan Follow 9 min read · Mar 22, 2024

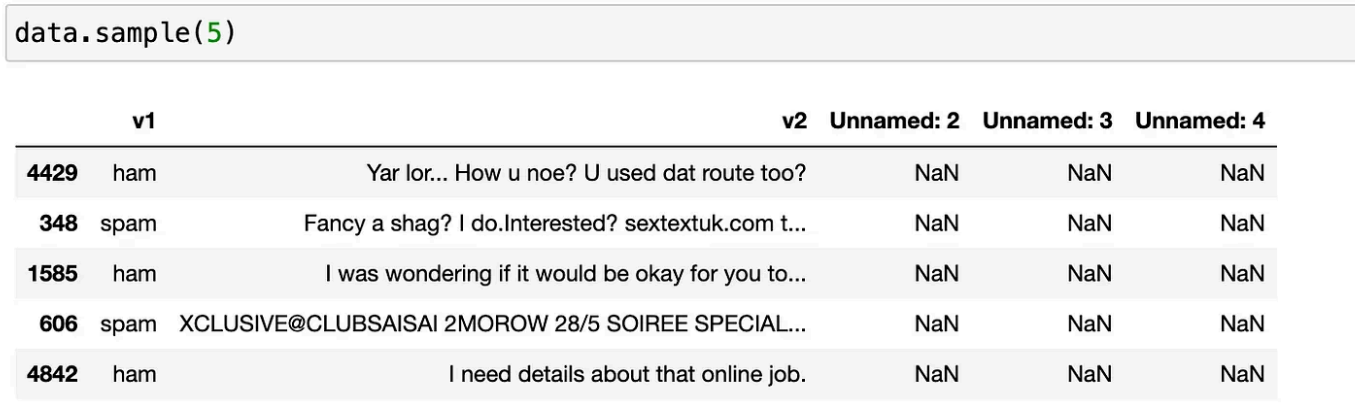
In today's world, email has become a crucial way for people to communicate. But along with the benefits of email, there's a big problem: spam. Spam is those annoying emails we didn't ask for, like ads or scams, that fill up our email inboxes. Thankfully, we can use machine learning to help us deal with this spam issue automatically. In this blog post, we'll learn how machine learning can help us find and block spam emails, using easy-to-understand Python code and popular machine learning tools.

STEP 1:

Importing the important libraries and dataset

So here, we start by importing necessary libraries like pandas for data handling and matplotlib/seaborn for visualization. Then, we perform data preprocessing tasks like removing unnecessary words, and punctuation and converting text to lowercase using NLTK. We also import the dataset containing email data from a CSV file. This dataset will be used to train our machine-learning models. Finally, we import various machine learning algorithms from the sklearn library for model building, including SVC (Support Vector Classifier), Random Forest Classifier, and Naive Bayes Classifier. These algorithms will help us classify emails as spam or non-spam.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# data preprocessing
import nltk
nltk.download('stopwords')
nltk.download('punkt')
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
import re
from collections import Counter
from wordcloud import WordCloud
from sklearn.preprocessing import LabelEncoder
# Model Building
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB

# ### Importing the Dataset
data = pd.read_csv("spam.csv",encoding='latin1')
data.sample(5)
```



```
data.sample(5)
```

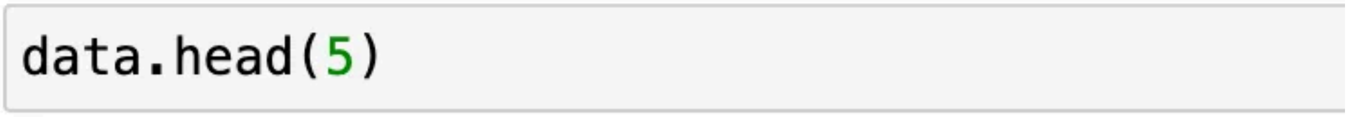
	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
4429	ham	Yar lor... How u noe? U used dat route too?	NaN	NaN	NaN
348	spam	Fancy a shag? I do.Interested? sextextuk.com t...	NaN	NaN	NaN
1585	ham	I was wondering if it would be okay for you to...	NaN	NaN	NaN
606	spam	XCLUSIVE@CLUBSAISAI 2MOROW 28/5 SOIREE SPECIAL...	NaN	NaN	NaN
4842	ham	I need details about that online job.	NaN	NaN	NaN

The code `data.sample(5)` will randomly pick 5 emails from the dataset to show us. Each email has two parts: a label (like "ham" for regular emails or "spam" for unwanted ones) and the actual content of the email. However, it seems like there are some extra columns in the dataset that don't have useful information, so they're labeled as "Unnamed" with missing values. By looking at this sample, we can get an idea of what kinds of emails are in the dataset and how they're organized, helping us understand the data better.

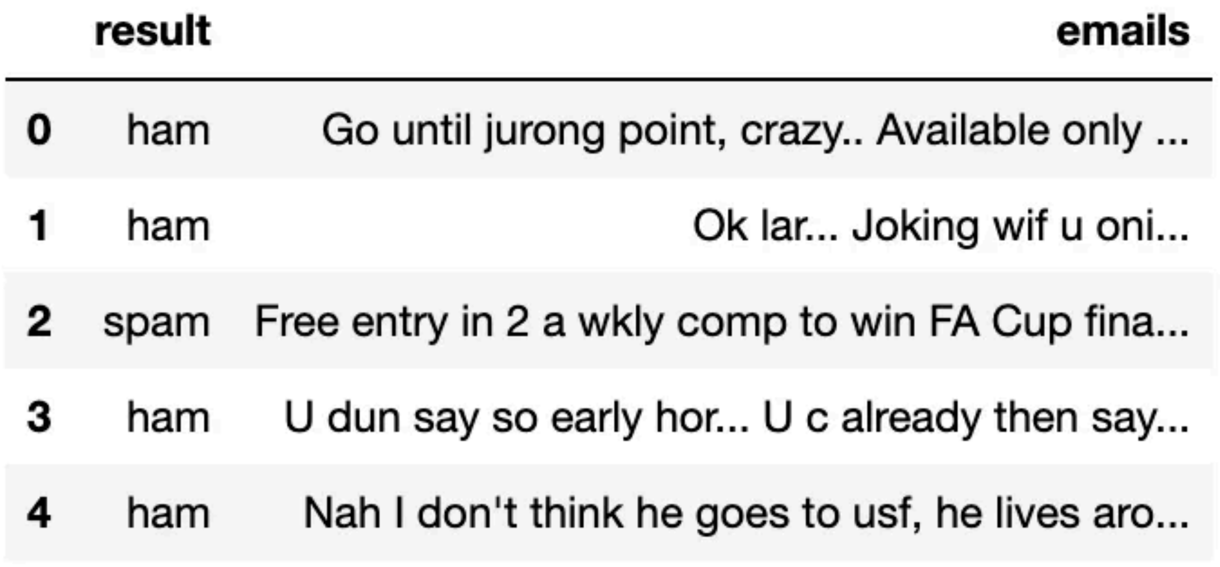
Step 2:

Initial Exploration And Data Cleaning

```
data.shape
data.drop(columns=["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], inplace=True)
data.rename(columns={'v1': 'result', 'v2': 'emails'}, inplace=True)
data
data.isnull().sum()
data.duplicated().sum()
data = data.drop_duplicates(keep='first')
data.shape
data.head(5)
```



```
data.head(5)
```



```
   result                                          emails
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...
```

In the data cleaning process, we're cleaning up a dataset to prepare it for analysis. First, we check its shape to understand its size. Then, we remove unnecessary columns named "Unnamed: 2," "Unnamed: 3," and "Unnamed: 4." We also rename two columns to make them more understandable: "v1" becomes "result," and "v2" becomes "emails." After cleaning, we check for any missing values and duplicates. The dataset's final shape is (5572, 5), meaning it has 5572 rows and 5 columns. There are no missing values, but 403 duplicate rows were removed. This cleaning ensures our data is ready for further analysis.

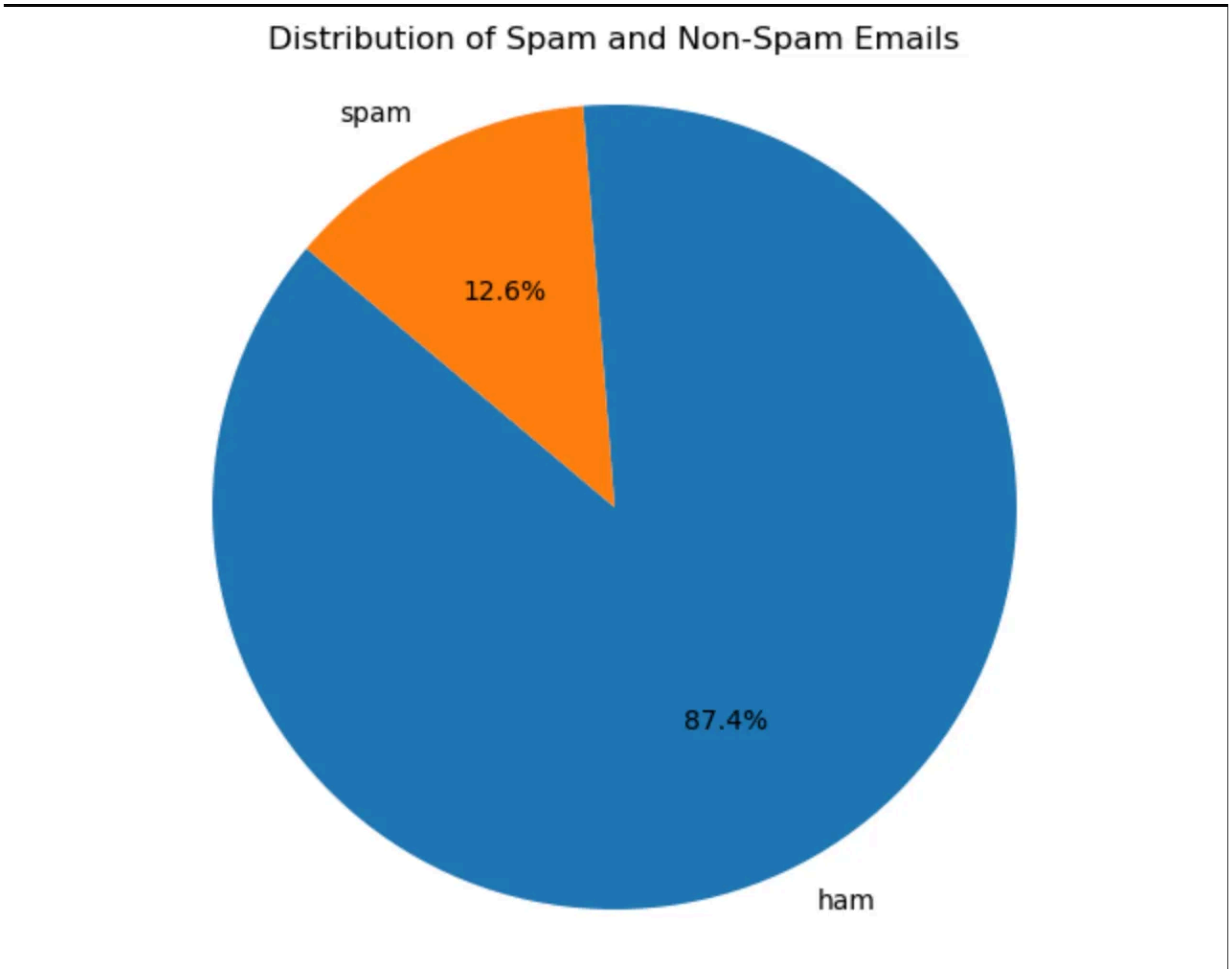
STEP 3-

Exploratory Data Analysis (EDA)

Here In our Exploratory Data Analysis (EDA), we dig into the dataset to understand important things about spam and regular (ham) emails. We look at how many emails are spam and how many are ham, using easy-to-read charts like pie charts. We also check the average length, word count, and sentence count of emails to see if there's a difference between spam and ham. Plus, we use special pictures called heatmaps to see if these things are related to each other. This helps us see patterns and understand the data better before we start working with it.

1- Distribution of Labels

```
data['result'].value_counts()

# Plotting
plt.figure(figsize=(8, 6))
plt.pie(data['result'].value_counts(), labels=data['result'].value_counts().inde
plt.title('Distribution of Spam and Non-Spam Emails')
plt.axis('equal')
plt.show()
```

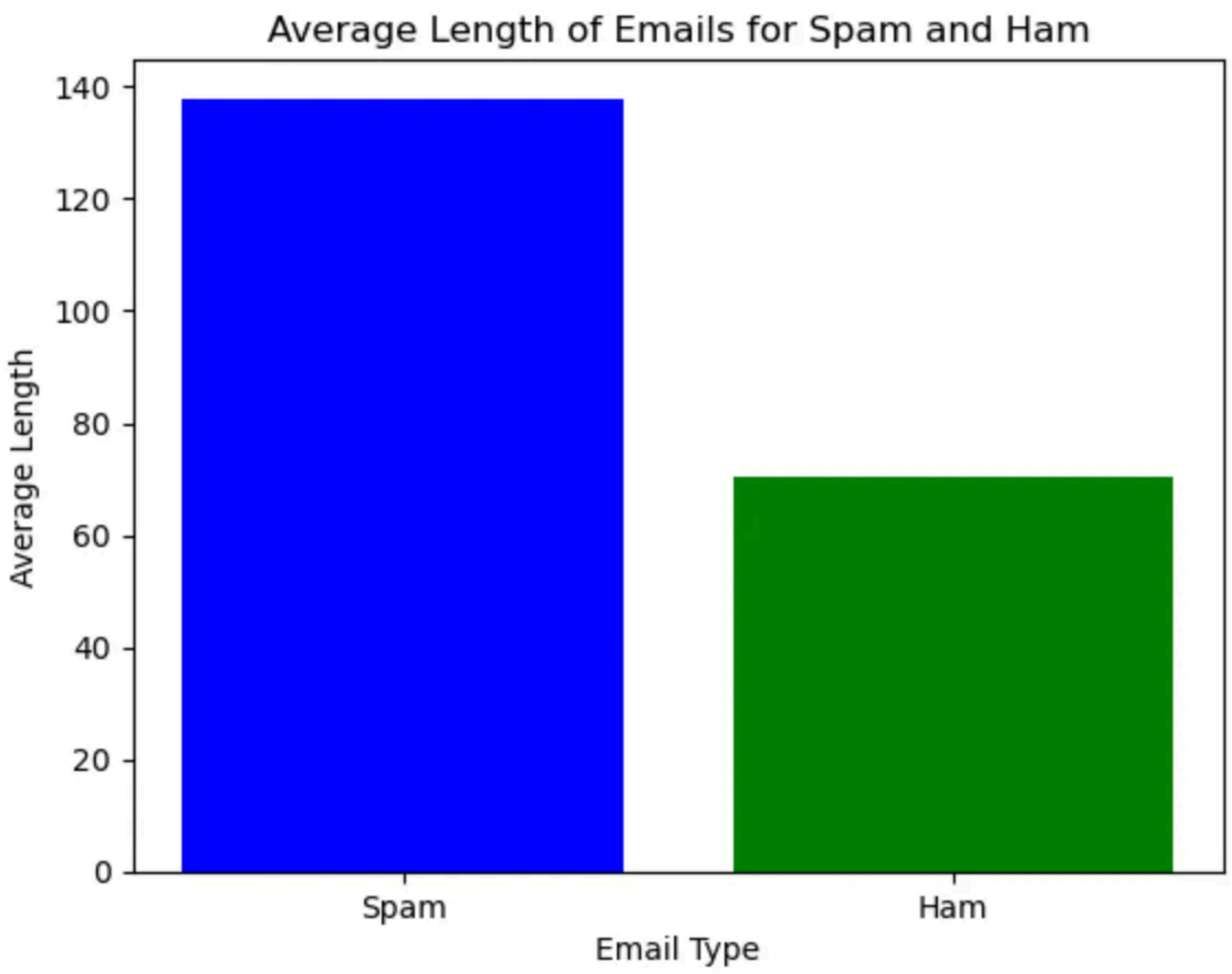


From the above graph we can see most emails in the dataset (87.4%) are non-spam (ham), while only a smaller portion (12.6%) are classified as spam. This difference is important because it affects how well our model can spot spam emails. Since there are many more non-spam emails, the model might become biased and miss some spam emails. To fix this, we need to use special techniques when training and testing our model. By doing this, we can make sure our model is good at finding both spam and non-spam emails, keeping our email inboxes safe and clutter-free.

2- Average Length of Emails for Spam and Ham

```
data['Length'] = data['emails'].apply(len)
data['num_words'] = data['emails'].apply(word_tokenize).apply(len)
data['num_sentence'] = data['emails'].apply(sent_tokenize).apply(len)
data.head(2)
avg_length_spam = data[data['result'] == 'spam']['Length'].mean()
avg_length_ham = data[data['result'] == 'ham']['Length'].mean()
#plotting
print("Average Length of Spam Emails:", avg_length_spam)
print("Average Length of Ham Emails:", avg_length_ham)
plt.bar(['Spam', 'Ham'], [avg_length_spam, avg_length_ham], color=['Blue', 'gree
plt.title('Average Length of Emails for Spam and Ham')
plt.xlabel('Email Type')
plt.ylabel('Average Length')
plt.show()
```

```
Average Length of Spam Emails: 137.89127105666157
Average Length of Ham Emails: 70.45925597874225
```

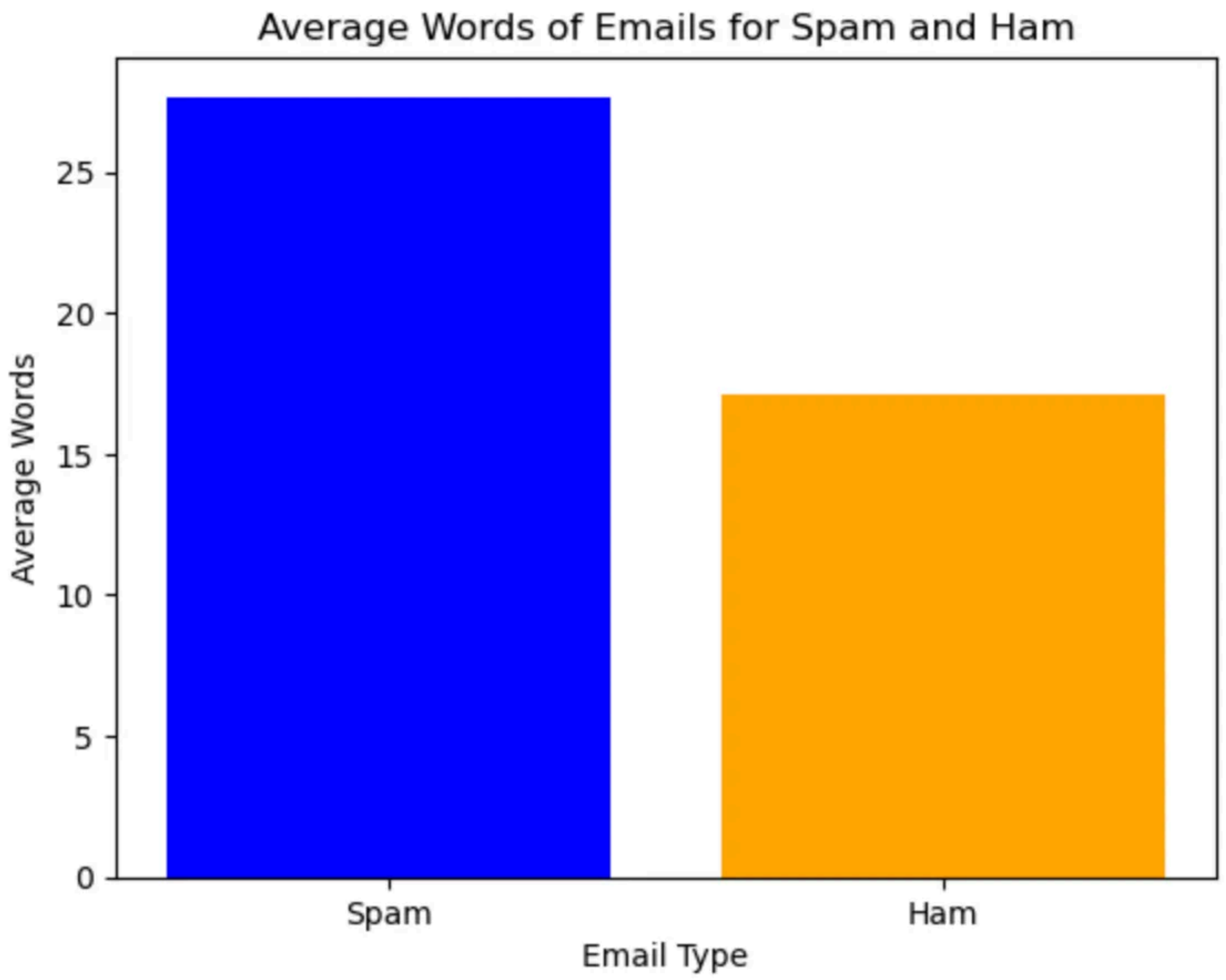


After looking at the lengths of spam and regular (ham) emails, we found that spam emails are much longer on average, around 137 characters. On the other hand, regular emails are much shorter, averaging about 70 characters. This means that spam emails tend to be more wordy and detailed, possibly because they're trying to grab your attention with lots of information. Regular emails, like the ones you get from friends or for work, are usually shorter and to the point. Understanding this helps us make better tools to filter out spam and keep our inboxes organized.

3- Average Word of Emails for Spam and Ham

```
avg_word_spam = data[data['result'] == 'spam']['num_words'].mean()
avg_word_ham = data[data['result'] == 'ham']['num_words'].mean()
print("Average Words of Spam Emails:", avg_word_spam)
print("Average Words of Ham Emails:", avg_word_ham)

# Plotting the graph
plt.bar(['Spam', 'Ham'], [avg_word_spam, avg_word_ham], color=['Blue', 'orange']
plt.title('Average Words of Emails for Spam and Ham')
plt.xlabel('Email Type')
plt.ylabel('Average Words')
plt.show()
```

```
Average Words of Spam Emails: 27.6676875957121
Average Words of Ham Emails: 17.12378210806023
```

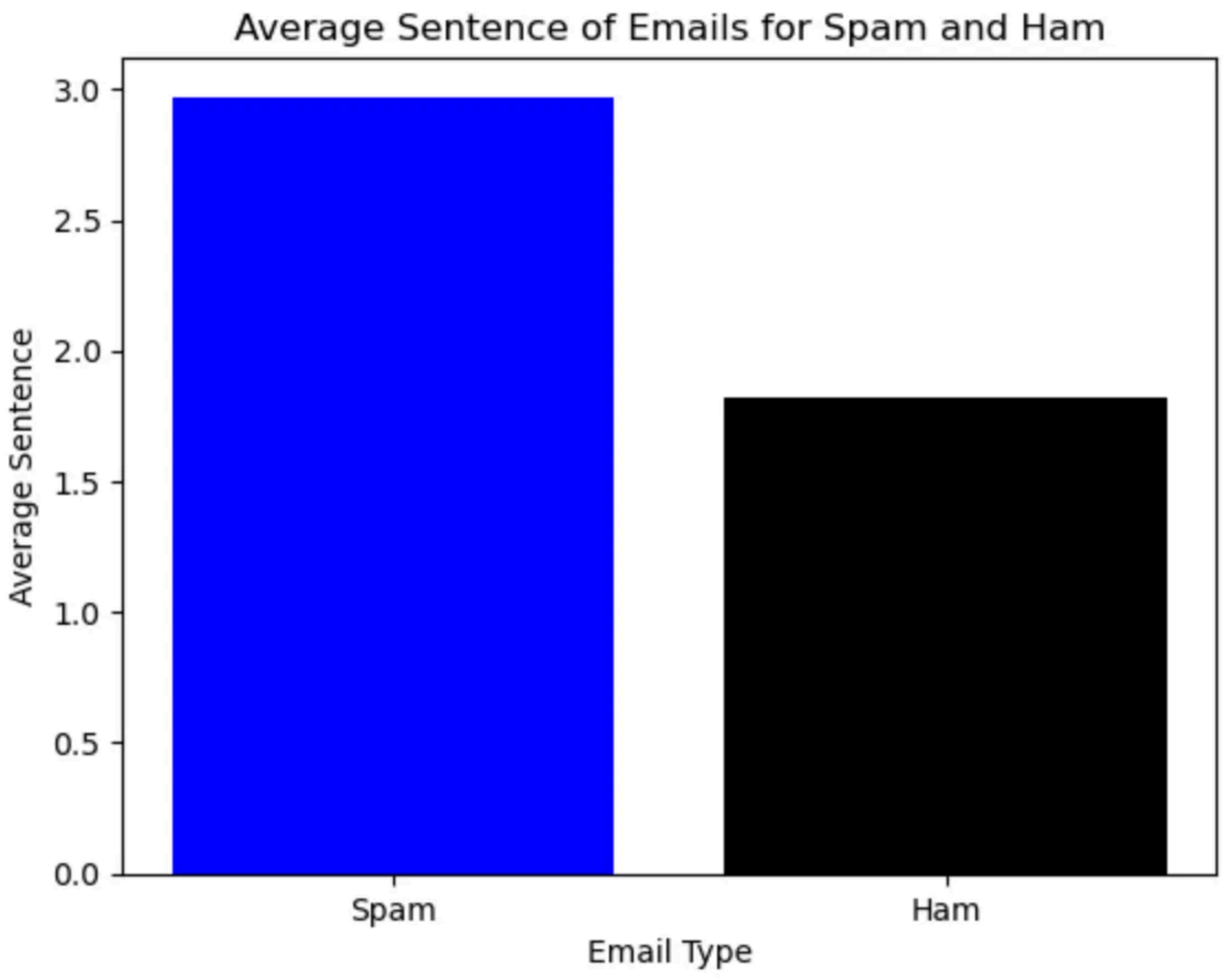


From the above graph, we can see that spam emails are longer, with an average of about 27 words per email. On the other hand, regular ham emails are shorter, averaging around 17 words per email. This means spam emails tend to be more wordy, maybe because they contain advertisements or misleading information. Meanwhile, regular emails are more straightforward and direct. Understanding this helps us create better filters to catch spam and keep our inboxes clean of unwanted messages, making it easier to find the emails that matter to us.

4- Average Sentence of Emails for Spam and Ham

```
avg_sentence_spam = data[data['result'] == 'spam']['num_sentence'].mean()
avg_sentence_ham = data[data['result'] == 'ham']['num_sentence'].mean()
print("Average Sentence of Spam Emails:", avg_sentence_spam)
print("Average Sentence of Ham Emails:", avg_sentence_ham)

# Plotting the graph
plt.bar(['Spam', 'Ham'], [avg_sentence_spam, avg_sentence_ham], color=['Blue', '
plt.title('Average Sentence of Emails for Spam and Ham')
plt.xlabel('Email Type')
plt.ylabel('Average Sentence')
plt.show()
```

```
Average Sentence of Spam Emails: 2.970903522205207
Average Sentence of Ham Emails: 1.8201948627103632
```

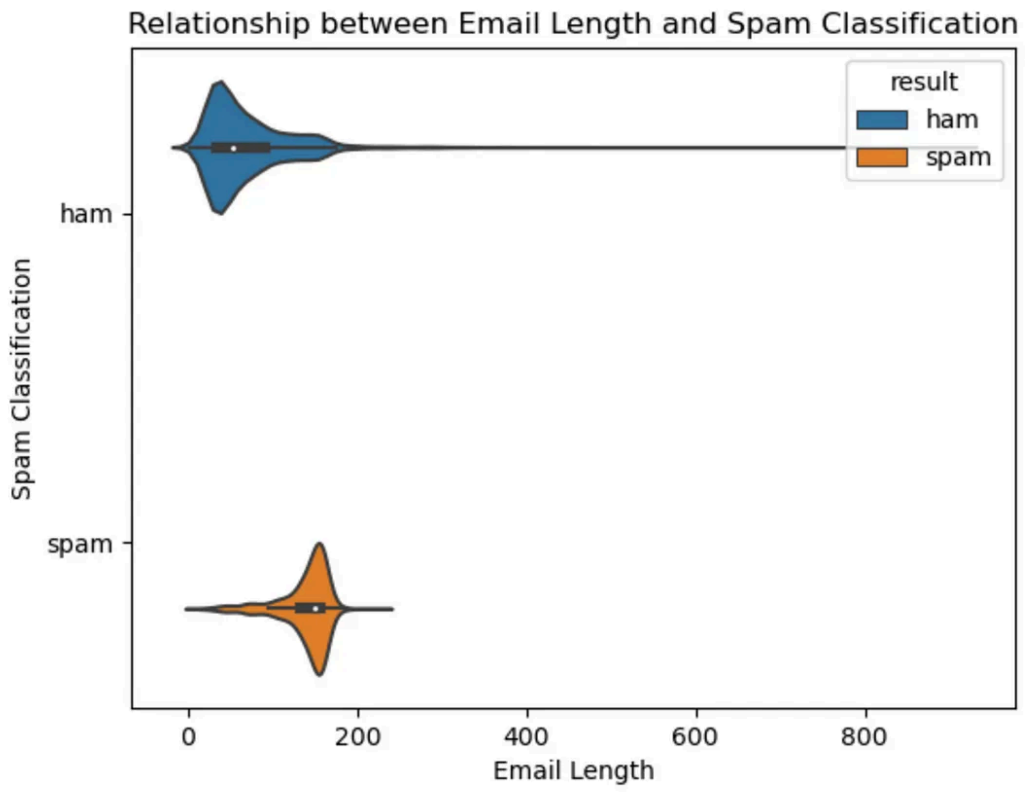


From the above graph, we can see that spam emails tend to have longer sentences compared to regular emails. On average, spam emails have about 3 sentences, while regular emails have about 2 sentences. This means that spam emails might be trying to say more or convince you of something, while regular emails are usually shorter and more straightforward. Understanding this difference helps us build better systems to detect and filter out spam emails, keeping our email inboxes cleaner and safer for everyone to use.

5- Relationship between Length and Spam

```
correlation = data['Length'].corr((data['result'] == 'spam').astype(int))
print("Correlation coefficient between email length and spam classification:", c


sns.violinplot(data=data, x='Length', y='result', hue='result')
plt.xlabel('Email Length')
plt.ylabel('Spam Classification')
plt.title('Relationship between Email Length and Spam Classification')
plt.show()
```

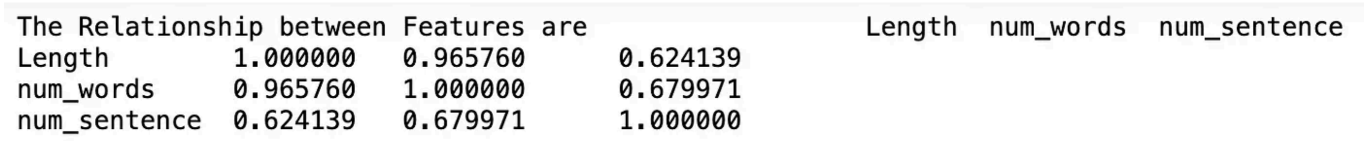
```
Correlation coefficient between email length and spam classification: 0.38471706671430644
```



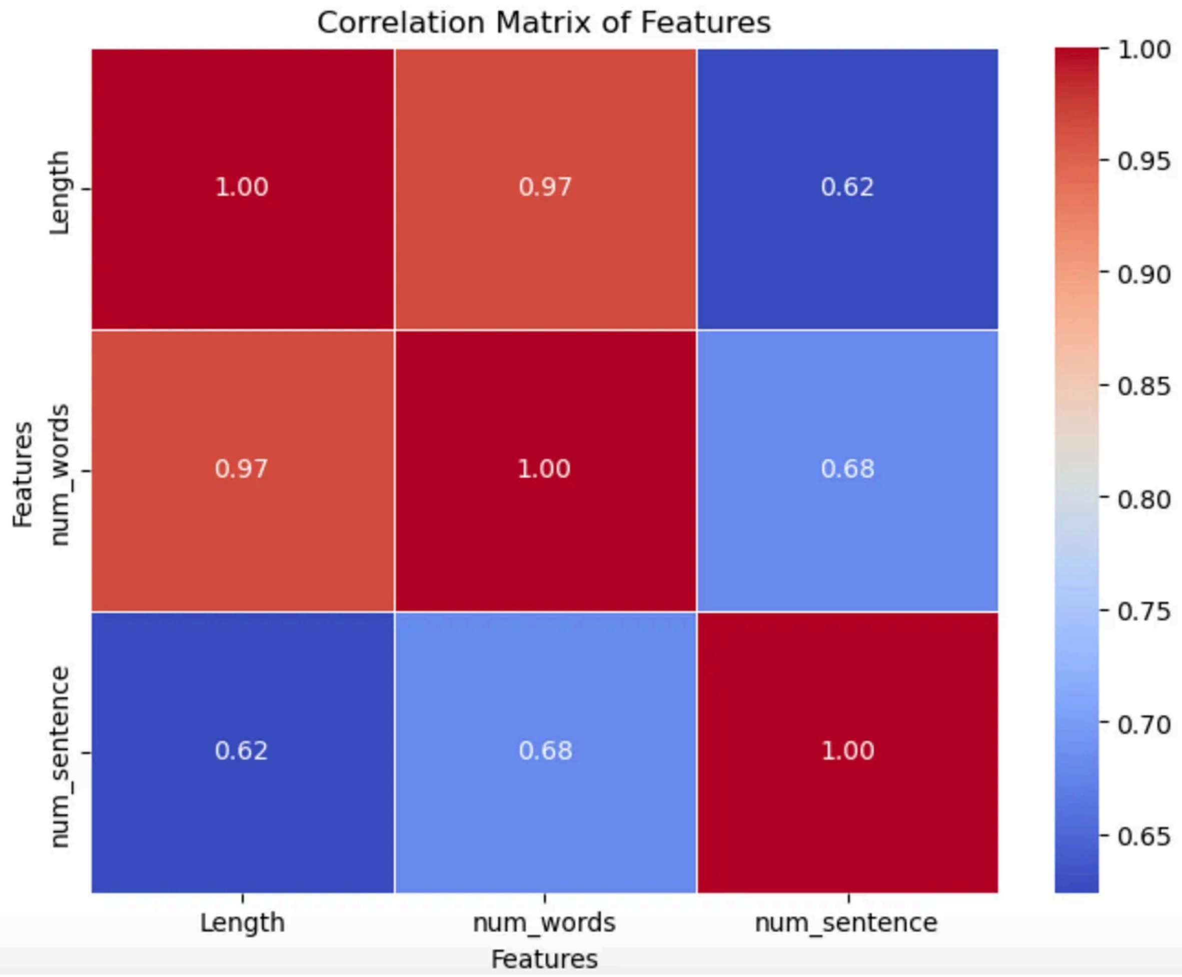
As we found that there is a positive correlation (correlation coefficient: 0.38) between email length and spam classification. This means that, on average, spam emails tend to be slightly longer than non-spam emails. However, it's important to note that the correlation is not very strong, indicating that other factors may also influence whether an email is classified as spam. Nonetheless, understanding this relationship can help improve spam detection algorithms by considering email length as one of the features in the classification process, alongside other relevant factors.

6- Relationship between Features

```
correlation_matrix = data[['Length', 'num_words', 'num_sentence']].corr()
print("The Relationship between Features are ",correlation_matrix )
# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidt
plt.title('Correlation Matrix of Features')
plt.xlabel('Features')
plt.ylabel('Features')
plt.show()
```



```
The Relationship between Features are                    Length  num_words  num_sentence
Length        1.000000  0.965760     0.624139
num_words     0.965760  1.000000     0.679971
num_sentence  0.624139  0.679971     1.000000
```



The correlation matrix indicates strong positive correlations between email length and the number of words (0.97) as well as between the number of words and the number of sentences (0.68). This suggests that longer emails tend to have more words, and emails with more words tend to have more sentences. However, the correlation between email length and the number of sentences is weaker (0.62). This implies that while longer emails may have more sentences, the relationship is not as strong as with the number of words. Understanding these relationships helps us grasp how different features contribute to the overall structure and content of emails, aiding in spam classification.

Step 4- Data Preprocessing

Before building machine learning models, we preprocess the email data to convert it into a suitable format for analysis. This involves tasks such as lowercasing, tokenization, removing special characters, stopwords, and punctuation, as well as stemming to reduce words to their root forms.

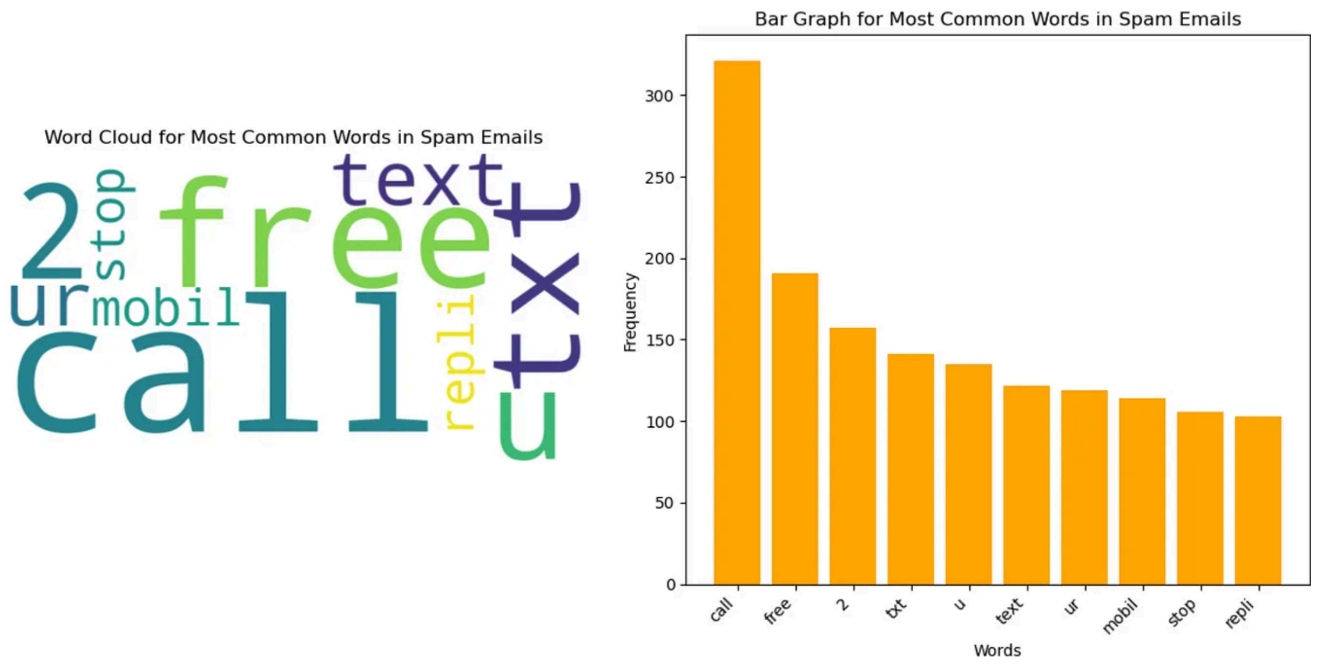
```
data['transform_text'] = data['emails'].str.lower()
# Tokenization
data['transform_text'] = data['transform_text'].apply(word_tokenize)

# Removing special characters
data['transform_text'] = data['transform_text'].apply(lambda x: [re.sub(r'[^a-zA

# Removing stop words and punctuation
stop_words = set(stopwords.words('english'))
data['transform_text'] = data['transform_text'].apply(lambda x: [word for word i

# Stemming
ps = PorterStemmer()
data['transform_text'] = data['transform_text'].apply(lambda x: [ps.stem(word) f

# Convert the preprocessed text back to string
data['transform_text'] = data['transform_text'].apply(lambda x: ' '.join(x))

# Display the preprocessed data
print(data[['emails', 'transform_text']].head())
```

In the data preprocessing step, we're getting our email data ready for analysis. First, we convert all text to lowercase so that uppercase and lowercase letters don't cause confusion. Then, we break down each email into smaller parts called tokens using a process called tokenization. After that, we remove any special characters like symbols or emojis that don't add useful information. Next, we take out common words like "the" or "and," as well as punctuation marks, because they're not helpful for identifying spam. Finally, we reduce words to their base form using a process called stemming, which helps in simplifying the data for analysis.

7- Most Common Words in Spam Emails

```
spam_emails = data[data['result'] == 'spam']['transform_text']
# Tokenize the text in spam emails
```

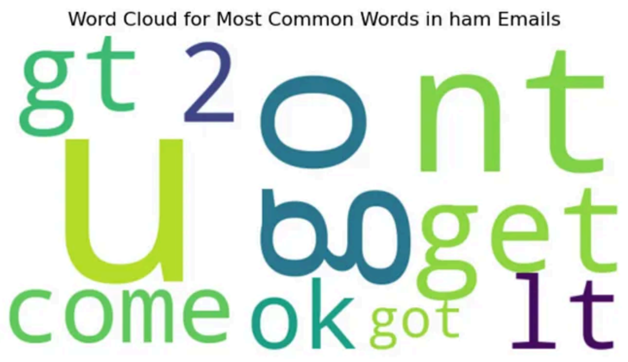
```
spam_words = ' '.join(spam_emails).split()
# Count occurrences of each word
word_counts = Counter(spam_words)
# Find the most common words
most_common_words = word_counts.most_common(10)
print("Top 10 Most Common Words in Spam Emails:")
for word, count in most_common_words:
    print(f"{word}: {count} occurrences")
    # Generate Word Cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate_
# Plot Word Cloud
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud for Most Common Words in Spam Emails')
plt.axis('off')
# Plot Bar Graph
plt.subplot(1, 2, 2)
words, counts = zip(*most_common_words)
plt.bar(words, counts, color='orange')
plt.title('Bar Graph for Most Common Words in Spam Emails')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

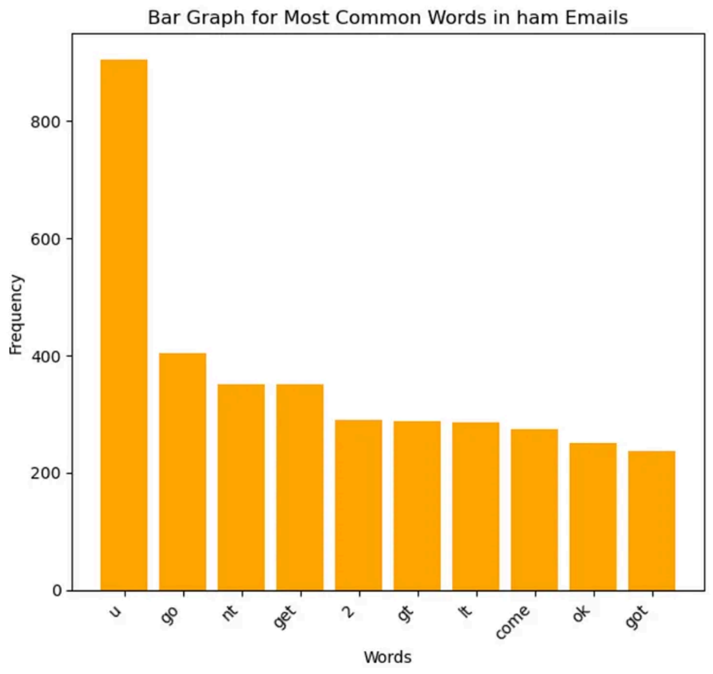


Looking at the most common words found in spam emails, we see patterns that spammers often use to catch our attention or convince us to act. Words like “call,” “free,” and “txt” show up frequently, suggesting offers or requests for action. This helps us understand what to watch out for in our emails to avoid falling for spam. By knowing these common tricks, we can be more careful about which emails we open or respond to, keeping our inboxes safer. Email filters also use this information to better recognize and block spam messages, making our email experience more secure

8- Most Common Words in Ham Emails

```
ham_emails = data[data['result'] == 'ham']['transform_text']
# Tokenize the text in spam emails
ham_words = ' '.join(ham_emails).split()
# Count occurrences of each word
word_counts = Counter(ham_words)
# Find the most common words
most_common_words = word_counts.most_common(10)

print("Top 10 Most Common Words in ham Emails:")
for word, count in most_common_words:
    print(f"{word}: {count} occurrences")
    # Generate Word Cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate_
# Plot Word Cloud
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud for Most Common Words in ham Emails')
plt.axis('off')
# Plot Bar Graph
plt.subplot(1, 2, 2)
words, counts = zip(*most_common_words)
plt.bar(words, counts, color='orange')
plt.title('Bar Graph for Most Common Words in ham Emails')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```





The top 10 most common words in non-spam emails are “u,” “go,” “nt,” “get,” “2,” “gt,” “lt,” “come,” “ok,” and “got.” These words show how people talk in emails, with shortcuts like “u” instead of “you” and “nt” for “not.” They also reveal common topics like going somewhere or confirming things with “ok.” Understanding these words helps in spotting normal emails. It tells us what to expect in regular messages, making it easier to spot unusual or suspicious ones, like spam.

STEP 5- “Preparing Data for Machine Learning: Label Encoding and Vectorization”

```
encoder = LabelEncoder()
data['result'] = encoder.fit_transform(data['result'])
data.sample(2)

#data spliting and vectorization
tfidf = TfidfVectorizer(max_features=3000)
X = tfidf.fit_transform(data['emails']).toarray()
y = data['result']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

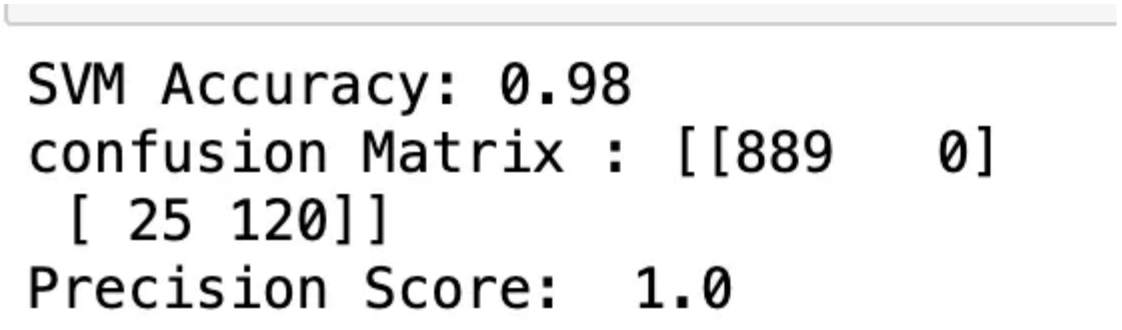
Here, First, we use a LabelEncoder to convert the ‘result’ column, which contains labels for spam (‘spam’) and non-spam (‘ham’) emails, into numerical values. This step is essential because machine learning algorithms typically work with numerical data. Then, we split our data into training and testing sets using train_test_split, where 80% of the data is used for training (X_train and y_train), and 20% is used for testing (X_test and y_test), which helps evaluate the performance of our model. Finally, we use TfidfVectorizer to convert our email text data into numerical vectors, making it suitable for machine learning algorithms to understand and analyze effectively.

STEP 6- Model Building

As we know Our Target variable is a classification type so to achieve this, we explore various machine learning models: Support Vector Machines (SVM), Random Forest, and Naive Bayes classifiers. These models learn from our preprocessed email data to distinguish between spam and legitimate messages. We evaluate each model's performance using metrics like accuracy and precision. By comparing their performance, we determine the most effective model for accurately identifying spam emails. This process aids in enhancing email security by ensuring that our detection system effectively filters out unwanted messages, providing users with a safer and more efficient email experience.

Model 1- SVC

```
svc_classifier = SVC()
svc_classifier.fit(X_train, y_train)
y_pred_svc = svc_classifier.predict(X_test)
accuracy_svc = accuracy_score(y_test, y_pred_svc)
print(f"SVM Accuracy: {accuracy_svc:.2f}")
print("confusion Matrix :",confusion_matrix(y_test,y_pred_svc))
print("Precision Score: ",precision_score(y_test,y_pred_svc))
```

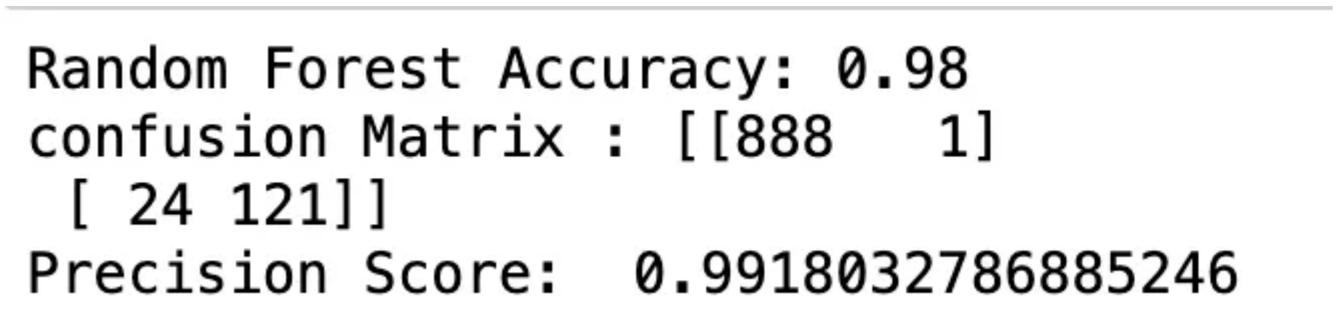


```
SVM Accuracy: 0.98
confusion Matrix : [[889   0]
 [ 25 120]]
Precision Score:  1.0
```

The Support Vector Machine (SVM) model accurately identified whether emails were spam or not 98% of the time. Out of all the emails it tested, it correctly labeled 889 non-spam emails and 120 spam emails. What's impressive is that it didn't mistakenly label any non-spam emails as spam. Also, every time it predicted an email was spam, it was indeed spam, showing perfect precision. In simpler terms, the SVM model did a great job at telling apart spam and non-spam emails, making it a dependable tool for keeping your inbox clean.

Model 2- Random Forest classifier

```
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train, y_train)
y_pred_rf = rf_classifier.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Random Forest Accuracy: {accuracy_rf:.2f}")
print("confusion Matrix :",confusion_matrix(y_test,y_pred_rf))
print("Precision Score: ",precision_score(y_test,y_pred_rf))
```

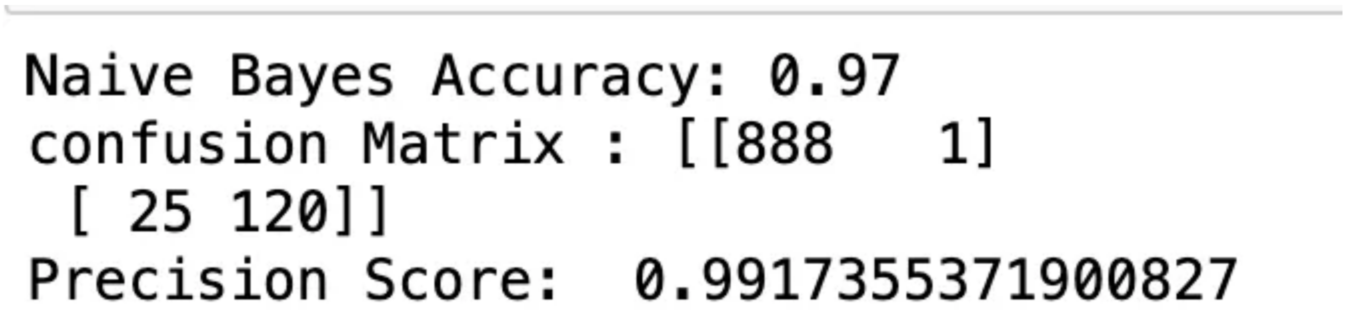


```
Random Forest Accuracy: 0.98
confusion Matrix : [[888   1]
 [ 24 121]]
Precision Score:  0.9918032786885246
```

The Random Forest model achieved an accuracy of 98%, indicating its ability to accurately classify emails as spam or ham. In the confusion matrix, out of 1015 emails, 888 were correctly classified as ham, while only 1 was incorrectly classified. For spam emails, 121 were correctly identified, with 24 being misclassified. The Precision Score, measuring the proportion of correctly classified spam emails among all emails classified as spam, was 99.18%.

Model 3- Naive Bayes classifier

```
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)
y_pred_nb = nb_classifier.predict(X_test)
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print(f"Naive Bayes Accuracy: {accuracy_nb:.2f}")
print("confusion Matrix :",confusion_matrix(y_test,y_pred_nb))
print("Precision Score: ",precision_score(y_test,y_pred_nb))
```



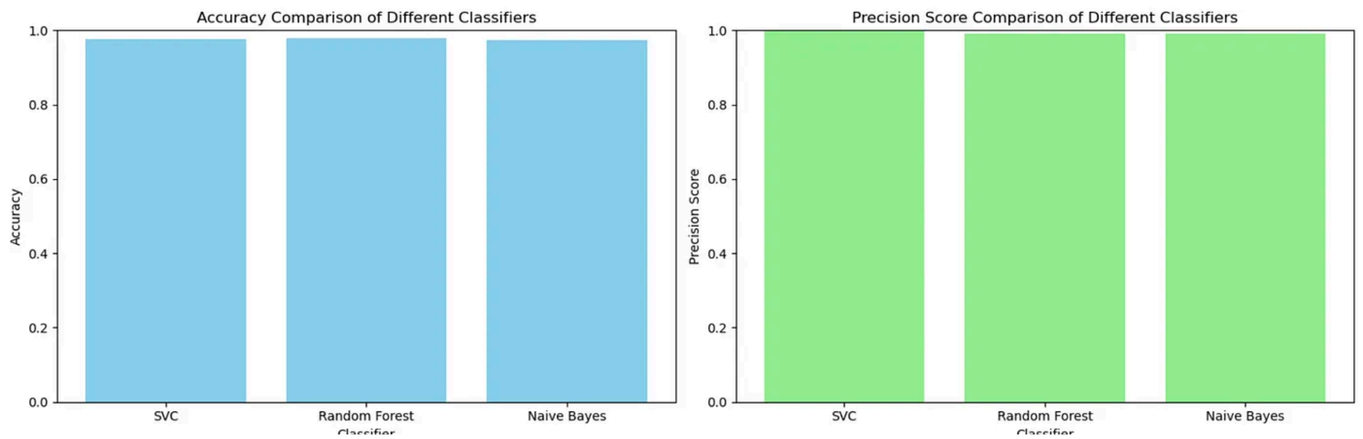
```
Naive Bayes Accuracy: 0.97
confusion Matrix : [[888   1]
 [ 25 120]]
Precision Score:  0.9917355371900827
```

The Naive Bayes classifier achieved an accuracy of 97%, indicating that it correctly classified 97 out of every 100 emails. In the confusion matrix, 888 emails were accurately classified as ham (non-spam), while only 1 ham email was incorrectly classified as spam. Similarly, 120 spam emails were correctly identified, with 25 spam emails being mistakenly classified as ham. The precision score, which measures the proportion of correctly predicted spam emails among all emails predicted as spam, is exceptionally high at 99.2%, indicating strong performance in identifying spam messages.

Choosing the Best Classifier for Email Spam Detection

```
# Calculate precision scores for each classifier
precision_svc = precision_score(y_test, y_pred_svc)
precision_rf = precision_score(y_test, y_pred_rf)
```

```
precision_nb = precision_score(y_test, y_pred_nb)

# Create lists to store accuracies and precision scores
classifiers = ['SVC', 'Random Forest', 'Naive Bayes']
accuracies = [accuracy_svc, accuracy_rf, accuracy_nb]
precision_scores = [precision_svc, precision_rf, precision_nb]

# Plot bar graph for accuracies and precision scores side by side
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# Plot bar graph for accuracies
axes[0].bar(classifiers, accuracies, color='skyblue')
axes[0].set_xlabel('Classifier')
axes[0].set_ylabel('Accuracy')
axes[0].set_title('Accuracy Comparison of Different Classifiers')
axes[0].set_ylim(0, 1)

# Plot bar graph for precision scores
axes[1].bar(classifiers, precision_scores, color='lightgreen')
axes[1].set_xlabel('Classifier')
axes[1].set_ylabel('Precision Score')
axes[1].set_title('Precision Score Comparison of Different Classifiers')
axes[1].set_ylim(0, 1)
plt.tight_layout()
plt.show()
```



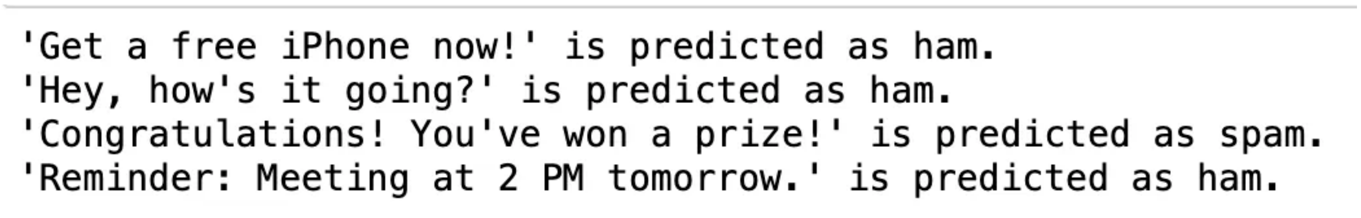
After sorting the model we find that SVC performs the best. So, we'll use SVC to make price predictions with SVC. predict(). This process helps us choose the most accurate model for predicting.

STEP 6- Model Prediction

Finally, we demonstrate how our trained SVC model can be used to predict whether new emails are spam or ham. We provide examples of predicting spam or ham status for sample email texts, showcasing the model's ability to classify emails in real-world scenarios.

Type 1- Predict with new data

```
new_emails = [
    "Get a free iPhone now!",
    "Hey, how's it going?",
    "Congratulations! You've won a prize!",
    "Reminder: Meeting at 2 PM tomorrow."
]

# Convert new data into numerical vectors using the trained tfidf_vectorizer
new_X = tfidf.transform(new_emails)
new_X_dense = new_X.toarray()

# Use the trained SVM model to make predictions
svm_predictions = svc_classifier.predict(new_X_dense)

# Print the predictions
for email, prediction in zip(new_emails, svm_predictions):
    if prediction == 1:
        print(f"'{email}' is predicted as spam.")
    else:
        print(f"'{email}' is predicted as ham.")
```



```
'Get a free iPhone now!' is predicted as ham.
'Hey, how's it going?' is predicted as ham.
'Congratulations! You've won a prize!' is predicted as spam.
'Reminder: Meeting at 2 PM tomorrow.' is predicted as ham.
```

In our email spam detection model, we tested it with four sample emails. The first email, "Get a free iPhone now!", was correctly predicted as non-spam (ham), meaning it's likely a legitimate message. Similarly, the third email, "Congratulations! You've won a prize!", was predicted as spam, which makes sense given its typical spam-like content. The second and fourth emails, "Hey, how's it going?" and "Reminder: Meeting at 2 PM tomorrow.", were also predicted as non-spam (ham), indicating they are likely genuine messages. Our model's predictions align well with our expectations, showcasing its effectiveness in distinguishing between spam and legitimate emails.

Type 2- User Input Data Prediction

```python
def predict_email(email):
    # Convert email into numerical vector using the trained TF-IDF vectorizer
    email_vector = tfidf.transform([email])

    # Convert sparse matrix to dense array
    email_vector_dense = email_vector.toarray()

    # Use the trained SVM model to make predictions
    prediction = svc_classifier.predict(email_vector_dense)

    # Print the prediction
    if prediction[0] == 1:
        print("The email is predicted as spam.")
    else:
        print("The email is predicted as ham.")

# Get user input for email
user_email = input("Enter the email text: ")

# Predict whether the input email is spam or ham
predict_email(user_email)
```

```
Enter the email text: Subject: Congratulations! You've Won a Prize! Hi there, Congratulations! You've been selected
as the lucky winner of our grand prize. To claim your reward, simply click on the link below and follow the instruc
tions. Don't miss out on this amazing opportunity! [Claim Your Prize Now]
The email is predicted as spam.
```

The `predict_email` function takes an email as input, converts it into a numerical vector using TF-IDF, and predicts whether it's spam or ham using a trained SVM model. For instance, if you enter an email claiming you've won a prize, it'll classify it as spam. This process ensures emails are accurately categorized to help users distinguish between legitimate and unwanted messages. By analyzing the email's content and comparing it to known patterns, the function helps maintain inbox cleanliness and security. It simplifies the complex task of spam detection, making email management more efficient and user-friendly.

Conclusion:

Email spam detection using machine learning provides a strong solution to the annoying issue of unwanted messages. By cleaning up and organizing the data, creating useful features, and building smart models, we can make effective filters that keep our emails safe. Since email is so important for communication, it's crucial to have good spam filters. These filters help us avoid clutter in our inboxes and make sure our digital conversations stay secure. With continued development, we can keep improving these systems to ensure our email experience is smooth and hassle-free.

DATASET AND SOURCECODE: https://drive.google.com/drive/folders/1UlvtzJREQn4LM187rF1Tv3-YnMDtqBNx?usp=drive_link

THANK YOU

Data Science Email Spam Detection Machine Learning NLP Data Visualization

Written by Azim Khan

32 Followers · 5 Following

Data Scientist

Follow

No responses yet

Write a response

What are your thoughts?