

Experiment 1

Name: Jhanvi Parekh
Sap:60009210033

Aim: - Implement a Spam classifier using Naïve Bayes classifier

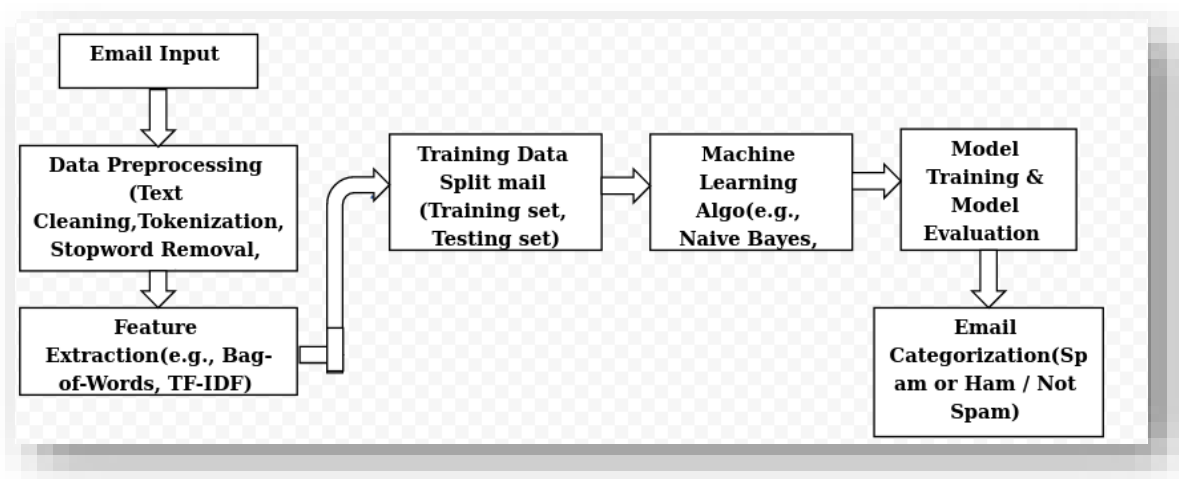
Theory:

Spam Classification

Spam classification, also known as email spam filtering or spam detection, is a classic problem in natural language processing and machine learning. It involves automatically identifying and categorizing incoming emails or messages as either "spam" (unsolicited and often irrelevant or inappropriate) or "ham" (legitimate and desired).

The primary goal of spam classification is to build a model that can accurately distinguish between spam and non-spam messages, thereby reducing the amount of unwanted and potentially harmful content that reaches users' inboxes. This is especially important considering the sheer volume of spam emails generated daily, as well as the potential security risks associated with phishing attempts and malicious content that may be contained in spam messages.

To solve the spam classification problem, machine learning algorithms are employed. Common approaches include the use of traditional algorithms like Naive Bayes, support vector machines (SVMs), decision trees, and more modern techniques like deep learning using neural networks.



Naïve Bayes classifier

The Naive Bayes classifier is a simple, yet effective probabilistic machine learning algorithm commonly used for classification tasks, particularly in natural language processing, spam filtering, sentiment analysis, and more. Despite its simplicity, Naive Bayes often performs surprisingly well and can be computationally efficient, making it a popular choice for certain applications.

The "Naive" in Naive Bayes comes from the assumption that the features (or attributes) used for classification are conditionally independent, given the class label. In other words, the model assumes that each feature contributes independently to the probability of a specific class. This is a simplification, and in reality, features might be correlated, but the assumption makes the algorithm computationally tractable.

How Naive Bayes works:

1. **Data and Labels:** The Naive Bayes classifier is trained on a labeled dataset containing features and their corresponding class labels. For example, in email spam classification, the features could be the words in the email, and the labels would be "spam" or "ham" (not spam).
2. **Prior Probability:** The first step in Naive Bayes is to calculate the prior probabilities of each class. The prior probability of a class is the probability of that class occurring without considering any features. For example, if you have 70% spam emails and 30% non-spam emails in your dataset, the prior probabilities would be $P(\text{spam}) = 0.7$ and $P(\text{ham}) = 0.3$.
3. **Likelihood Estimation:** Next, Naive Bayes calculates the likelihood probabilities for each feature given the class labels. The likelihood probability is the probability of observing a particular feature given the class. For example, $P(\text{word}=\text{'free'} \mid \text{spam})$ would represent the probability of the word "free" appearing in spam emails.
4. **Posterior Probability:** Using Bayes' theorem, the Naive Bayes classifier calculates the posterior probability of each class given the observed features. The posterior probability is the probability of a class given the evidence (features). The class with the highest posterior probability is the predicted class for the input.
5. **Prediction:** The Naive Bayes classifier then makes predictions based on the highest posterior probability. The class with the highest posterior probability for a given set of features is assigned as the predicted class.

Lab Experiments to be Performed in This Session: -

Data Set: - SMS Spam Collection

This dataset contains SMS messages labeled as spam or ham. It is commonly used for spam detection in text messages.

Exercise 1: - Perform Spam Classification With text preprocessing steps.

Step 1: Import Labeled with labels as "spam" or "not spam" (ham).

Step 2: Perform Data Preprocessing to convert it into a suitable format for training the Naive Bayes classifier. Common steps include (Removing punctuation and special characters, Tokenization, stop word Removal, converting to lowercase, Stemming or Lemmatization)

Step 3: Feature Extraction (bag-of-words model or term frequency-inverse document frequency (TF-IDF) representation.

Step 4: Training the Naive Bayes Classifier

Step 5: Evaluate the performance of the trained Naive Bayes classifier on the test dataset.

Step 6: Hyperparameter Tuning (Optional)

Step 7: Deployment Once you are satisfied with the model's performance, you can deploy it to classify new, unseen emails or messages as spam or ham.

Exercise 2: - Perform Spam Classification Without Using Any text preprocessing steps.

Step 1: - Import Labeled with labels as "spam" or "not spam" (ham).

Step 2: Feature Extraction (bag-of-words model or term frequency-inverse document frequency (TF-IDF) representation.

Step 3: Training the Naive Bayes Classifier

Step 4: Evaluate the performance of the trained Naive Bayes classifier on the test dataset.

Step 5: Hyperparameter Tuning (Optional)

Step 6: Deployment Once you are satisfied with the model's performance, you can deploy it to classify new, unseen emails or messages as spam or ham.

Exercise 3: -

- a. Compare The results of Exercise 1 and Exercise 2
- b. Show the Comparison using different visualization techniques.

```
import pandas as pd
import numpy as np
import nltk
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from nltk.stem import PorterStemmer

nltk.download('punkt')
nltk.download('stopwords')

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
True
```

```
df = pd.read_csv('/content/mail_data.csv')
print(df.head())
```

```
Category      Message
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...
```

```
# Convert the 'Message' column to lowercase
df['Message'] = df['Message'].str.lower()
print(df[['Message']].head())
```

```
Message
0  go until jurong point, crazy.. available only ...
1                ok lar... joking wif u oni...
2  free entry in 2 a wkly comp to win fa cup fina...
3  u dun say so early hor... u c already then say...
4  nah i don't think he goes to usf, he lives aro...
```

```
# Remove punctuation and special characters
df['Message'] = df['Message'].apply(lambda x: re.sub(r'\W', ' ', x))

# Display the first few rows to see the result
print(df[['Message']].head())
```

```
Message
0  go until jurong point  crazy   available only ...
1                ok lar    joking wif u oni
2  free entry in 2 a wkly comp to win fa cup fina...
3  u dun say so early hor    u c already then say
4  nah i don t think he goes to usf  he lives aro...
```

```
# Tokenize the text
df['tokenized_message'] = df['Message'].apply(word_tokenize)
```

```
# Display the first few rows to see the result
print(df[['tokenized_message']].head())
```

```
tokenized_message
0  [go, until, jurong, point, crazy, available, o...
1                [ok, lar, joking, wif, u, oni]
2  [free, entry, in, 2, a, wkly, comp, to, win, f...
3  [u, dun, say, so, early, hor, u, c, already, t...
4  [nah, i, don, t, think, he, goes, to, usf, he,...
```

```
# Remove stop words
stop_words = set(stopwords.words('english'))
df['filtered_message'] = df['tokenized_message'].apply(lambda x: [word for word in x if word not in stop_words])

# Display the first few rows to see the result
print(df[['filtered_message']].head())
```

```

5                                     filtered_message
0  [go, jurong, point, crazy, available, bugis, n...
1                                     [ok, lar, joking, wif, u, on]
2  [free, entry, 2, wkly, comp, win, fa, cup, fin...
3                                     [u, dun, say, early, hor, u, c, already, say]
4                                     [nah, think, goes, usf, lives, around, though]

```

```
# Apply Porter Stemmer to each word
stemmer = PorterStemmer()
df['stemmed_message'] = df['filtered_message'].apply(lambda x: [stemmer.stem(word) for word in x])
```

```
# Display the first few rows to see the result
print(df[['stemmed_message']].head())
```

```

5          stemmed_message
0  [go, jurong, point, crazi, avail, bugi, n, gre...
1          [ok, lar, joke, wif, u, onl]
2  [free, entri, 2, wkli, comp, win, fa, cup, fin...
3          [u, dun, say, earli, hor, u, c, alreadi, say]
4  [nah, think, goe, usf, live, around, though]

```

```
df['final_message'] = df['stemmed_message'].apply(lambda x: ' '.join(x))
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
# Initialize the TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
```

```
# Fit and transform the data to create the TF-IDF feature matrix
X_tfidf = tfidf_vectorizer.fit_transform(df['final_message'])
```

```
# Convert to DataFrame to inspect
tfidf_df = pd.DataFrame(X_tfidf.toarray(), columns=tfidf_vectorizer.get_feature_names_out())
```

```
# Display the first few rows of the TF-IDF DataFrame
print(tfidf_df.head())
```

	00	000	000pe	008704050406	0089	0121	01223585236	01223585334	\
0	0.0	0.0	0.0		0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0		0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0		0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0		0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0		0.0	0.0	0.0	0.0	0.0

	0125698789	02	...	zhong	zindgi	zoe	zogtoriu	zoom	zouk	zyada	èn	\
0		0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1		0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2		0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3		0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4		0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	ú1	≡ud
0	0.0	0.0
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0

```
[5 rows x 7244 columns]
```

```
X = X_tfidf # or X_bow, depending on what you used
y = df['Category'] # Labels: "spam" or "not spam"
```

```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```


```
print(f'Training set size: {X_train.shape[0]} samples')
print(f'Testing set size: {X_test.shape[0]} samples')
```

```
⇒ Training set size: 4457 samples
   Testing set size: 1115 samples
```

```
from sklearn.naive_bayes import MultinomialNB
```

```
# Initialize the Multinomial Naive Bayes classifier
nb_classifier = MultinomialNB()
```

```
# Train the model on the training data
nb_classifier.fit(X_train, y_train)
```


 MultinomialNB
 MultinomialNB()

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
# Make predictions on the testing data
```

```
y_pred = nb_classifier.predict(X_test)
```

```
# Calculate the accuracy
```

```
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f'Accuracy: {accuracy:.2f}')
```

```
# Print the classification report
```


```
print("\nClassification Report:")
```

```
print(classification_report(y_test, y_pred, target_names=['not spam', 'spam']))
```

```
# Print the confusion matrix
```

```
print("\nConfusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred, labels=['not spam', 'spam']))
```

 Accuracy: 0.97

Classification Report:

	precision	recall	f1-score	support
not spam	0.97	1.00	0.98	966
spam	1.00	0.77	0.87	149
accuracy			0.97	1115
macro avg	0.98	0.88	0.92	1115
weighted avg	0.97	0.97	0.97	1115

Confusion Matrix:


```
[[ 0  0]
 [ 0 114]]
```

```
index = 0 # Change the index to check different examples
```

```
print(f"Email: {df['Message'].iloc[X_test[index].indices[0]]}")
```

```
print(f"Actual Label: {y_test.iloc[index]}")
```

```
print(f"Predicted Label: {y_pred[index]}")
```

 Email: tbs persolvo been chasing us since sept for 38 definitely not paying now thanks to your information we will ignore them k
 Actual Label: ham
 Predicted Label: ham

Without Preprocessing

```
vectorizer = TfidfVectorizer()
```


```
# Fit and transform the data
```

```
X = vectorizer.fit_transform(df['Message'])
```

```
# Convert the result to a DataFrame for better readability
```

```
X_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())
```

```
print(X_df.head())
```



```

00 000 000pes 008704050406 0089 0121 01223585236 01223585334 \
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
2 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
3 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

0125698789 02 ... zhong zindgi zoe zogtorius zoom zouk zyada \
0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
2 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
3 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0

èn ú1 ðud
0 0.0 0.0 0.0
1 0.0 0.0 0.0
2 0.0 0.0 0.0
3 0.0 0.0 0.0
4 0.0 0.0 0.0

```

```
[5 rows x 8709 columns]
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Naive Bayes classifier
nb_classifier = MultinomialNB()

# Train the classifier
nb_classifier.fit(X_train, y_train)

# Make predictions
y_pred = nb_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:")
print(report)
```

→ Accuracy: 0.9650
Classification Report:

	precision	recall	f1-score	support
ham	0.96	1.00	0.98	966
spam	1.00	0.74	0.85	149
accuracy			0.97	1115
macro avg	0.98	0.87	0.91	1115
weighted avg	0.97	0.97	0.96	1115