420-PIA-1D - Integration project

Assignment

Ashley Addingadoo
STUDENT ID: 650391073

Integration project

Table of Contents

Ashley Addingadoo
Student ID: 650391073

PROJECT:

Project: Build an email classifier Using Python

Create a project for an email classification technique for Ham and Spam filters. Email is one of the well-known communication services in which a message sends electronically. The lessening in the cost of email benefits by telecom organizations has prompted the expanded utilization of it. This ascent pulled in assailants/phishing/scam, which have brought about Email Spam problem. Spam messages include advertisements, free services, promotions, awards, etc.

People are using the ubiquity of emails are expanding day by day as they give a vast variety of services by reducing the cost of services. In any case, this has prompted an expansion in mobile phones attacks like Email Spam. This problem is further expanded using multiple background datasets.

The most prevalent service that standouts amongst all other things on Internet are the Emails. A gigantic amount of organizations, associations and people use email every single day for either their personal or professional use. The constant development of email clients has brought about the expanding of unsolicited messages otherwise called Spam.

Spammers have presented some successful traps comprising of installing spam substance into digital images, pdf and doc as an attachment which can make the current systems incapable which is based on examination of computerized message in the body and subject fields of email.

A significant number of strategies suggests an anti-spam filtering approach that depends on data mining techniques which classifies the spam and ham messages. The viability of these methodologies is evaluated on the basis of large text dataset as well as image dataset.

User Stories

Phase 1 - Development

1) Explore the Problem
2) Find the Motivation
3) Identify the dataset
   a) Make sure you use the maximum number of test & train data
   b) Remove the Noise
4) Find different Machine Learning Algorithms – Please Choose 3 Algorithms
   a) [Choose any of the following Algorithms]
      I. Supervised
      II. Semi-supervised
      III. Reinforcement
   b) Identify algorithms with the highest accuracy and lowest time complexity
5) At the end of training dataset, Plot the graph between ham & spam.

Phase 2 - Documentation

a) Create a Presentation

b) Create a Report

c) Present 420-P1A-ID Project

Bonus features

a. Increase accuracy > 95

b. Use > 1000 data for train and test data (if possible)

Criteria:

Criteria	Marks	Comments
You have chosen **three** appropriate algorithms for the solution type	15	
You have chosen an appropriate dataset	10	
You have applied the correct transformations for your chosen algorithms	30	
You have developed a fully functional ML pipeline in VSCode	30	Will be checked on Wednesday next week.
You have evidence of hyperparameter tuning on MLFlow	20	Will be checked on Wednesday next week.
You have used a modular approach	10	
You have a sphinx documentation	5	
You have a fully functional streamlit app.	30	You will choose a random spam and ham content, put it in a text field and predict whether it is ham or spam.
Report	50	4-5 pages report 1. **Introduction**: What is the objective of the project? That is what is the problem you want to solve with the ML algorithm. 2. **Methodology**: Here you will describe the ML workflow. Put a snapshot of MLflow when you discuss hyperparameter tuning. 3. **Results: Which model was the best. Comment on your metric** 4. **Personal reflection:** Identify a challenge and explain what you would do to adapt your solution.
Total	200	

Submission: Monday 20th of October

Report

Introduction:

What is the objective of the project? That is what is the problem you want to solve with the ML algorithm.

Answer:

The goal of this project is to build a machine learning model that can automatically classify emails as either spam (unwanted messages like ads, scams, or phishing) or ham (normal emails). Spam emails are a major issue, and an effective spam filter can help improve email security and user experience.

To tackle this problem, we are using a supervised learning approach. This means we train the model using a labeled dataset, where each email is already marked as spam or ham. The model will learn from this data and then be able to classify new emails.

We will use three machine learning models:

- **RandomForestClassifier:** *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html*
- **MultinomialNB:** *https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html*
- **LogisticRegression:** *https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html*

During training, we will perform hyperparameter tuning and use cross-validation (e.g. 5-fold cross-validation) to ensure the model generalizes well to unseen data. This helps avoid overfitting and provides a better estimate of model performance.

We will evaluate the models using **confusion matrices, accuracy, precision, recall, and F1 score**. Based on these results, we will select the **best-performing model** that gives the highest accuracy and lowest error in classifying emails correctly.

The dataset utilized for this project is a publicly available collection from **Kaggle**, containing a large number of labeled emails. Additionally, other shared datasets can be used to ensure reproducibility of results and enable benchmarking against different models.

Dataset:

1) https://www.kaggle.com/datasets/meruvulikith/190k-spam-ham-email-dataset-for-classification
2) https://www.kaggle.com/datasets/meruvulikith/190k-spam-ham-email-dataset-for-classification

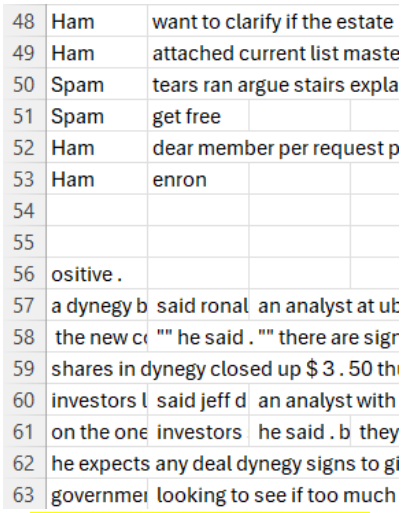
Methodology:

Here you will describe the ML workflow. Put a snapshot of MLflow when you discuss hyperparameter tuning.

Answer:

1) ETL: Extract Transform Load

When examining the dataset in Excel, we noticed that some of the columns were mixed together. Since there were no two distinct columns for the labels and messages, to resolve this, the dataset was exported as a text file for easier preprocessing.
Using the **Pandas** library in Python, the data was then parsed and split based on the keywords *"ham"* and *"spam"* or on the punctuation " , ". This process generated two separate columns: **label**, which contains the classification labels (*ham* or *spam*), and **message**, which contains the corresponding email content.



48	Ham	want to clarify if the estate
49	Ham	attached current list maste
50	Spam	tears ran argue stairs expla
51	Spam	get free
52	Ham	dear member per request p
53	Ham	enron
54		
55		
56	ositive .	
57	a dynegy b	said ronal an analyst at ub
58	the new c	"" he said . "" there are sigr
59	shares in dynegy closed up $ 3 . 50 th	
60	investors l	said jeff d an analyst with
61	on the one	investors he said . b they
62	he expects	any deal dynegy signs to gi
63	governmer	looking to see if too much

Fig: Mixed column using excel

```
df.head()
```

	v1,v2,,,
0	ham,"Go until jurong point, crazy.. Available ...
1	ham,Ok lar... Joking wif u oni...,,,
2	spam,Free entry in 2 a wkly comp to win FA Cup...
3	ham,U dun say so early hor... U c already then...
4	ham,"Nah I don't think he goes to usf, he live...

Fig: Mixed column using googlecolab

- Extracting data insights from the text file using pandas:

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5574 entries, 0 to 5573
Data columns (total 1 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   v1,v2,,,  5574 non-null   object
dtypes: object(1)
memory usage: 43.7+ KB
```

Comments: We find out there is no title for the column

- Assign a name to the column:

```
df.columns = ['text']
```

```
df.head()
```

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

- Split the text and create two new columns: “label” and “Message”

```
df[["label", "Message"]] = df["text"].str.split(',', n=1, expand=True)
```

```
df.head()
```

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

- After we drop the text column.

```
df.drop(columns=['text'], inplace=True)
```

- Examine the new dataset for missing values and additional details.

```
df.info()
```

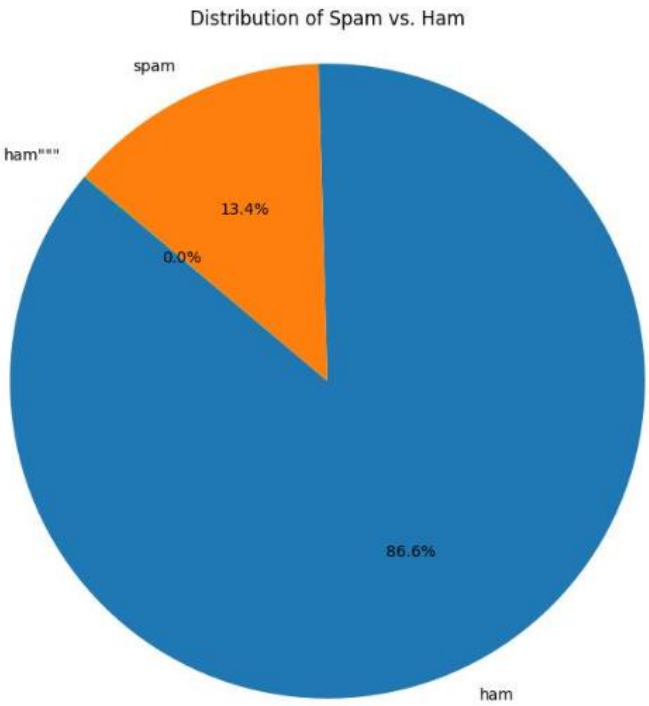
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5574 entries, 0 to 5573
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   label    5574 non-null   object
 1   Message  5574 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

- However, when counting the number of ham and spam messages in the dataset, the following observations were made

```
df["label"].value_counts()
```

label	count
ham	4825
spam	747
ham"""	2

dtype: int64



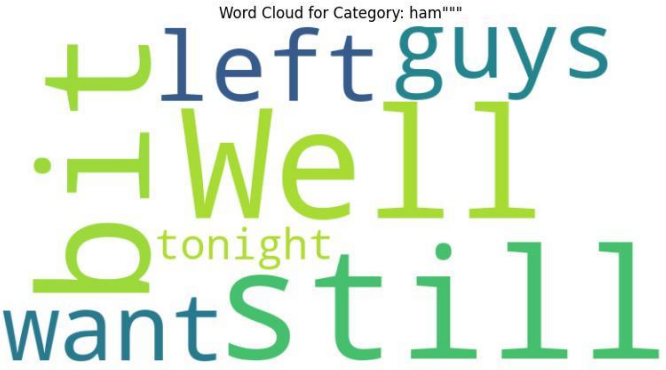
Comments: For the time being, this is not an issue as it will be resolved later through punctuation removal with the NLTK library

We also observe an imbalance between the ham and spam classes in the dataset. Despite this, we have chosen to proceed with developing the pipeline. Based on the confusion matrix results, we can deduce that imbalance can significantly affect the model's training.
To address this at a later stage, we can apply techniques such as merging datasets or augmenting the spam class with additional samples from other sources to mitigate the imbalance.

2) EDA: Exploratory Data Analysis

```
visulization:
visualizer = Visualization(df=df_processed,label_col="label",text_col="Message")
visualizer.visualize()
```

We make word clouds for spam and ham separately to see the different words that appear most often in each. This helps us quickly spot what kind of language or keywords are common in spam messages versus normal (ham) messages, making it easier to understand and detect spam.



3) Data Pre-processing

Natural Language Processing (**NLP**) techniques, supported by libraries such as **NLTK** (Natural Language Toolkit), are essential for transforming raw text into a structured format that machine learning models can understand. Text data often contains noise such as punctuation, stopwords, inconsistent casing, and irrelevant tokens that can reduce model accuracy.

By applying NLP methods like tokenization, stopword removal, and lemmatization, we can clean and normalize the text. This improves the quality of the features extracted (e.g., through TF-IDF) and ultimately leads to more accurate and reliable classification results.

Resources required from nltk are:

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer

# Required nltk ressources
nltk.download('punkt_tab')
nltk.download('punkt')
nltk.download('vader_lexicon')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download("stopwords")
```

Before training the model, several preprocessing steps need to be applied to clean and prepare the text data for analysis. Proper preprocessing ensures the data is consistent, structured, and suitable for feature extraction and model training. The steps include:

	label	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 ar...

Fig: Dataset before applying data pre-processing

A. *Lowercasing the Text*

All text in the Message column was converted to lowercase. This step ensures uniformity and helps prevent the model from treating the same word in different cases (e.g., “Free” and “free”) as separate terms.

```
def lower_case(df, column_name):
    df[column_name] = df[column_name].str.lower()
    return df
```

	label	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 ar...

B. *Removing Punctuation*

Punctuation marks were removed from all relevant columns, not just the Message column. This helps reduce noise and focuses the analysis on the actual words.

```
def remove_punc(df, column_name):
    punc = set(string.punctuation)
    df[column_name] = df[column_name].apply(lambda x: "".join(char for char in x if char not in punc))
    return df
```

	label	Message
0	ham	go until jurong point crazy available only in ...
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 dont think he goes to usf he lives aroun...

C. *Removing Extra Spaces*

Any unnecessary spaces within the text were stripped out to make the data cleaner and more consistent.

```
def remove_space(df, column_name):
    df[column_name] = df[column_name].apply(lambda x: " ".join(x.split()))
    return df
```

	label	Message
0	ham	go until jurong point crazy available only in ...
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 dont think he goes to usf he lives aroun...

D. *Text Normalization*

This step involved multiple sub-processes:

a. Tokenization: Splitting text into individual words (tokens).
b. Stopword Removal: Eliminating common but non-informative words (e.g., *"the"*, *"is"*, *"and"*).
c. Lemmatization: Reducing words to their root form (e.g., *"running"* → *"run"*).
d. Single Character Removal: Removing single-letter tokens that usually don't carry meaningful information.

```
def text_normalization(df, column_name):
  english_stop_words = stopwords.words("english")
  lemmatizer = WordNetLemmatizer()
  df[column_name] = df[column_name].apply(lambda x: [word for word in word_tokenize(str(x)) if word not in english_stop_words and len(word) > 1])
  df[column_name] = df[column_name].apply(lambda x: " ".join(lemmatizer.lemmatize(word) for word in x if len(lemmatizer.lemmatize(word)) > 1))
  return df
```

	label	Message
0	ham	go jurong point crazy available bugis great wo...
1	ham	ok lar joking wif oni
2	spam	free entry wkly comp win fa cup final tkts 21s...
3	ham	dun say early hor already say
4	ham	nah dont think go usf life around though

E. *Encoding the Labels*

The label column was encoded into numeric values to make it suitable for machine learning algorithms (e.g., *ham* = 0, *spam* = 1).

```
def encoding(df, column_name):
  le = LabelEncoder()
  df[column_name] = le.fit_transform(df[column_name])
  return df
```

	label	Message
0	0	go jurong point crazy available bugis great wo...
1	0	ok lar joking wif oni
2	1	free entry wkly comp win fa cup final tkts 21s...
3	0	dun say early hor already say
4	0	nah dont think go usf life around though

F. TF-IDF Transformation

Finally, the cleaned text data was transformed into numerical vectors using TF-IDF (Term Frequency–Inverse Document Frequency). This technique captures how important a word is to a message relative to the entire dataset, improving model performance.

```
def tfidf(df, column_name):
  tfidf_vectorizer = TfidfVectorizer(max_features=21000)
  X = tfidf_vectorizer.fit_transform(df[column_name])
  y = df["label"]
  return X, y, tfidf_vectorizer
```

To observe the changes, we can either print the shape of X and y or display the features generated by the tfidf_vectorizer.

```
print(X.shape)
```

output:

```
(5574, 8943)
```

```
print(y[:5])
```

output:

```
0    0
1    0
2    1
3    0
4    0
Name: label, dtype: int64
```

```
print(tfidf_vectorizer.get_feature_names_out()[:50])
```

```
['008704050406' '0089my' '0121' '01223585236' '01223585334' '0125698789'
 '02' '020603' '0207' '02070836089' '02072069400' '02073162414'
 '02085076972' '020903' '021' '050703' '0578' '06' '060505' '061104'
 '07008009200' '07046744435' '07090201529' '07090298926' '07099833605'
 '071104' '07123456789' '0721072' '07732584351' '07734396839'
 '07742676969' '07753741225' '0776xxxxxxx' '07786200117' '077xxx' '078'
 '07801543489' '07808' '07808247860' '07808726822' '07815296484'
 '07821230901' '0784987' '0789xxxxxxx' '079467462910788086786 7'
 '0796xxxxxx' '07973788240' '07xxxxxxxxx' '0800' '08000407165']
```

4) Train models

To build an effective classifier, we trained on the following models (**Random Forest**, **Multinomial Naive Bayes**, and **Logistic Regression)** on the training set. Each model was optimized using **GridSearchCV**, which performs an exhaustive search over specified hyperparameter values combined with 5-fold cross-validation.

For instance, we tuned parameters like the number of estimators and maximum depth for Random Forest, the smoothing parameter alpha for Naive Bayes, and the regularization strength C and solver for Logistic Regression. After training, the best-performing version of each model (based on cross-validation scores) was selected for evaluation.

To support future deployment such as integrating into a **Streamlit app,** all three tuned models were saved in **pickle (.pkl) format**, allowing them to be easily reloaded and used without retraining.

This approach ensures that each model is fine-tuned for optimal performance, reducing the risk of overfitting and improving generalization on unseen data.

Hyperparameter Tuning Result for each model:

RandomForestClassifier:

```
Training and tuning RandomForestClassifier...
Best parameters for RandomForestClassifier: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 200}
```

MultinimialNB:

```
Training and tuning MultinomialNB...
Best parameters for MultinomialNB: {'alpha': 0.5}
```

LogisticRegression:

```
Training and tuning LogisticRegression...
Best parameters for LogisticRegression: {'C': 10.0, 'solver': 'lbfgs'}
```

5) Evaluate each model

Evaluation of all 3 models on the test dataset

After training and tuning, each model was evaluated on the test set using key performance metrics: accuracy, precision, recall, and F1-score. These metrics provide a comprehensive view of how well each model performs in distinguishing between spam and ham messages.

The evaluation results show that all three models performed strongly, with Multinomial Naive Bayes achieving the highest F1-score of 0.978765, closely followed by Random Forest with 0.977742. While Logistic Regression also delivered solid performance, it slightly trailed with an F1-score of 0.972581.

This indicates that **MultinomialNB** may be slightly more effective in balancing precision and recall for this text classification task, making it a strong candidate for deployment, especially considering its simplicity and efficiency in handling text data.

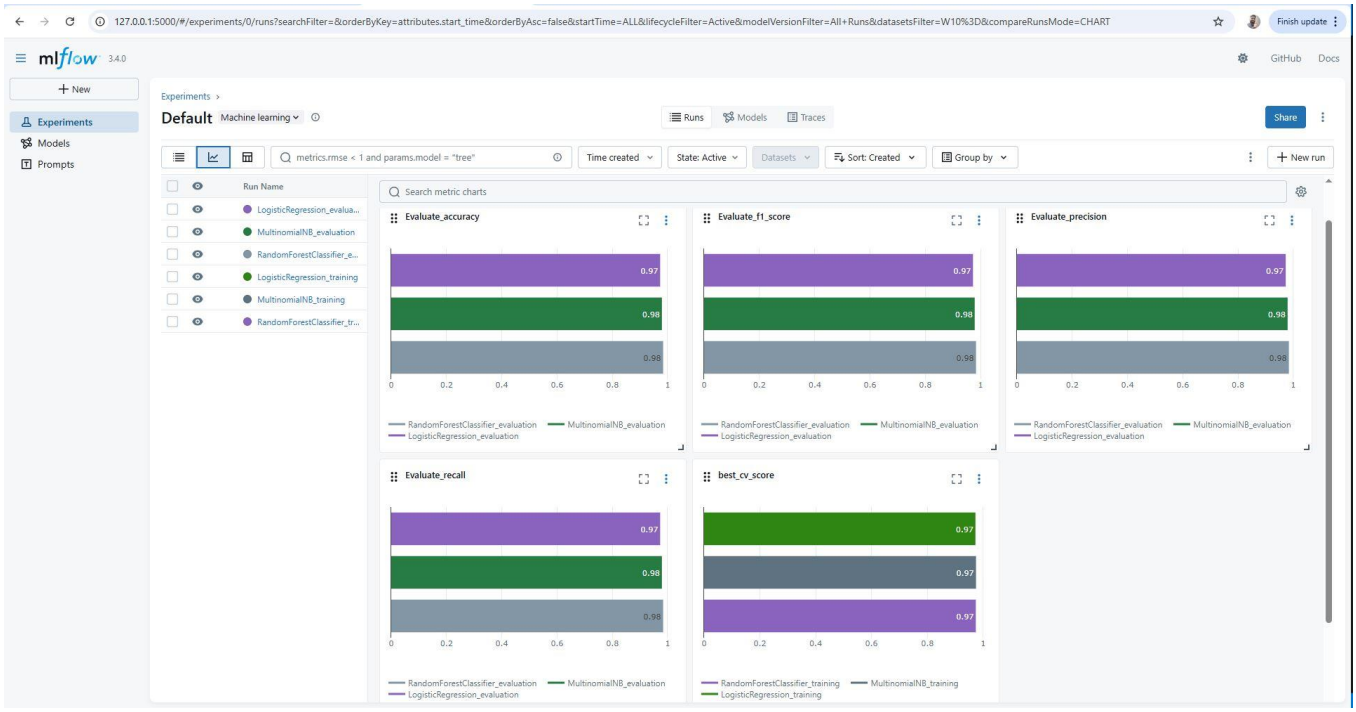
```
Final Results After Hyperparameter Tuning:

                    Model  Accuracy  Precision    Recall  F1-Score
0  RandomForestClassifier  0.978475   0.979004  0.978475  0.977742
1           MultinomialNB  0.979372   0.979663  0.979372  0.978765
2      LogisticRegression  0.973094   0.972696  0.973094  0.972581
```

Data logging with MLflow:

Experiment Tracking with MLflow
All training runs and model performance metrics were logged using MLflow.
This included:

- ❖ Parameters used during training and hyperparameter tuning
- ❖ Model artifacts such as trained model files
- ❖ Evaluation metrics (accuracy, precision, recall, F1-score)

By leveraging MLflow, we maintained a complete record of each experiment, which made it easy to compare models, track progress, and retrieve the best-performing configurations for future use or deployment.



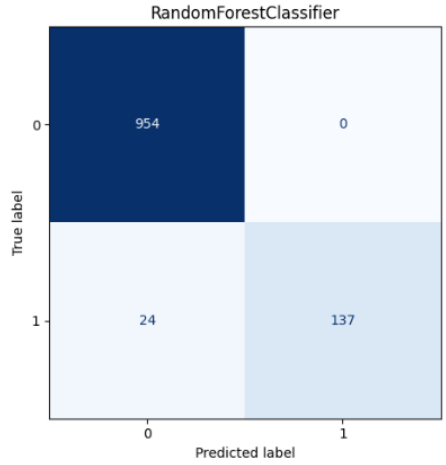
Results:

Which model was the best. Comment on your metric

Answer:

Confusion Matrix:

A confusion matrix is also useful for visualizing true positives, false positives, true negatives, and false negatives.

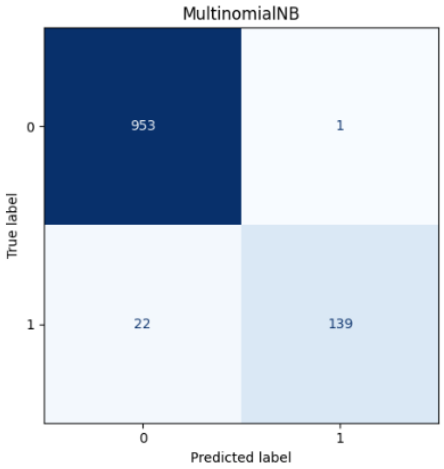


True Positives (TP): 137
Spam messages correctly identified as spam

True Negatives (TN): 954
Ham messages correctly identified as ham

False Positives (FP): 0
No ham messages were incorrectly classified as spam

False Negatives (FN): 24
24 spam messages were incorrectly classified as ham

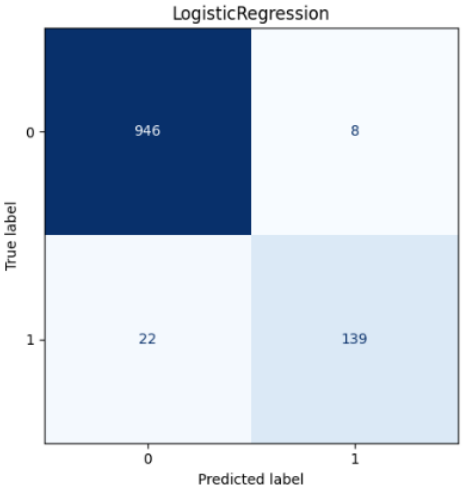


True Positives (TP): 139
Spam messages correctly predicted as spam

True Negatives (TN): 953
Ham messages correctly predicted as ham

False Positives (FP): 1
1 ham message was incorrectly predicted as spam

False Negatives (FN): 22
22 spam messages were incorrectly predicted as ham



True Positives (TP): 139
Spam messages correctly predicted as spam

True Negatives (TN): 946
Ham messages correctly predicted as ham

False Positives (FP): 8
1 ham message was incorrectly predicted as spam

False Negatives (FN): 22
22 spam messages were incorrectly predicted as ham

All three models, Random Forest, Multinomial Naive Bayes, and Logistic Regression show strong performance in classifying spam and ham. MultinomialNB achieves the best balance with the highest F1-score and low false positives and negatives. Random Forest has zero false positives but slightly more false negatives, while Logistic Regression has more false positives but similar recall to MultinomialNB.

Based on the evaluation results, **Multinomial Naive Bayes** is the most suitable model for email spam detection. It offers the best balance between precision and recall, with very few false positives and a relatively low number of false negatives. This makes it effective at identifying spam while minimizing the risk of misclassifying legitimate emails, which is crucial for maintaining user trust and system reliability.

Personal reflection:

Identify a challenge and explain what you would do to adapt your solution.

Answer:

One of the key challenges I encountered during this project was dealing with **imbalanced data**, where the number of ham messages significantly outweighed the spam messages. Although I initially identified this issue during the ETL phase, I chose to proceed without applying any balancing techniques. Surprisingly, the evaluation results and confusion matrices showed that the models, especially **Multinomial Naive Bayes**, still performed well, achieving high precision and recall. However, during testing in the **Streamlit app**, I tried a spam message from another dataset, and the model incorrectly classified it as ham. This revealed a limitation in how well the model generalizes to new, unseen spam formats. While performance on the original test set was strong, this result showed that the model may struggle with spam outside the structure and patterns of the training data.

To improve the solution, I would consider applying **resampling techniques**, **combining diverse datasets**, and exploring **more advanced models** or **richer preprocessing** to improve robustness and real-world reliability.