Name: Mohan Sai Bandarupalli
UCID: mb2279
E-mail: mb2279@njit.edu
Professor: Yasser Abduallah
CS 634 104 Data Mining

FINAL TERM PROJECT REPORT

Abstract: This project focuses binary classification into spam and ham using machine learning and deep learning approaches. The goal is to create predictive models that can accurately identify between unwanted (spam) and acceptable (ham) messages, which were pre-processed by cleaning, tokenization and stemming to prepare for analysis. Random Forest, Naïve Bayes, and Long Short-Term Memory (LSTM) were evaluated providing each unique insights for classification task. Exploratory Data Analysis uncovered characteristics of data like distribution of message lengths and balance of classes. The used models in this project tested using metrics like accuracy, F1 score and area under ROC curve. Among the three Random Forest exhibited high accuracy and precision, while LSTM model uncovered strength in handling sequential data and robust predictions.

Introduction:
In this present era spam messages are most annoying they also present great risks to privacy and security making effective spam detection is important. This classification is useful for improving efficiency of spam filtering.

This project uses a dataset of SMS messages labelled spam or ham providing real world context for analysis. The initial challenge is processing and classification of unstructured data. To tackle this several machine learning and deep learning techniques are implemented and by each model performance evaluation is done based on its ability to correctly predict the class of new messages. The approaches like Randon Forest and Naïve Bayes and deep learning method such as LSTM networks are used which are particularly adept processing sequential nature of text.

The primary goal of this project is to explore the effectiveness of various modelling techniques in text classification. This report gives the overview of the steps taken from data processing to model evaluation.

Problem Statement:

Developing Binary Classification model to classify spam vs ham messages using SMS Dataset

Implementation / Work Flow Overview:

As mentioned above the project focuses on classifying text messages as 'spam' or 'ham' using various machine learning and deep learning techniques. The data used in this project is from the 'sms_spam_ham_dataset.csv', which includes examples of both spam and non-spam (ham) messages.

1. **Data Preprocessing**: By this preprocessing step the data is well prepared for subsequent stages of model building and analysis. The steps involved are:
   1.1. Load Data: The initial step is to load the dataset into pandas Data Frame for analysis and view the dataset to understand the things like structure and content.
   1.2. Data Cleaning: By performing this cleaning step, the null values, unwanted characters, white spaces are removed, and normalize the text to reduce the complexity and improve the consistency.
   1.3. Tokenization: In this the text is split into individual words or tokens this helps in analyzing test into manageable units for future process.
   1.4. Stop Words Removal: In this common English words and additional frequently occurring words are removed because they don't contribute to context.
   1.5. Stemming: Here the words are reduced to root form by removing suffixes which helps in decreasing the model complexity.
   1.6. Feature Engineering: Here the text length calculation is done it means calculate the length of each message before text cleaning. And then label encoding, encode target variables into binary format done by ML algorithms.

1.7. Splitting Data: Here, the data is divided into training and testing to ensure the model can be trained on one portion of data and validated on another this helps in evaluating the model's performance.

2. **Exploratory Data Analysis**: Generated pie charts to illustrate the ratio of ham and spam messages and created histograms to uncover the distribution of text lengths in dataset. In other words, by performing EDA, we can gain crucial insights into nature of data and explore the underlying patterns and identifying features which are important for distinguishing between spam and ham.

3. **Model Development**:

3.1. Random Forest Classifier: Trained a Random Forest model with cross-validation to predict whether a message is spam. Evaluated model performance using metrics like accuracy, precision, recall, F1-score, and AUC (Area Under the Curve). Generated a confusion matrix to visualize the classifier's performance.

3.2. Naive Bayes Classifier: Implemented a Gaussian Naive Bayes model and assessed its effectiveness in spam detection. Similarly, used metrics like accuracy, precision, recall, F1-score, and AUC for evaluation.

3.3. Deep Learning Model (LSTM): Designed and trained an LSTM model using word embeddings. And set up the model with layers like Embedding, LSTM, and Dense. Compiled model with Adam optimizer and binary cross-entropy loss. Plotted loss and accuracy curves to monitor training and validation performance over epochs.

Steps to Run:

1. Import all necessary packages and modules.

```
import numpy as np
import pandas as pd
import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from keras.preprocessing.text import Tokenizer
from sklearn.ensemble import RandomForestClassifier
from keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
import seaborn as sns
from gensim.models import Word2Vec
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.naive_bayes import GaussianNB
from keras.optimizers import Adam
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding
```

2. Load the dataset by providing the path to the CSV file

```
df=pd.read_csv("sms_spam_ham_dataset.csv",encoding="latin-1")
```

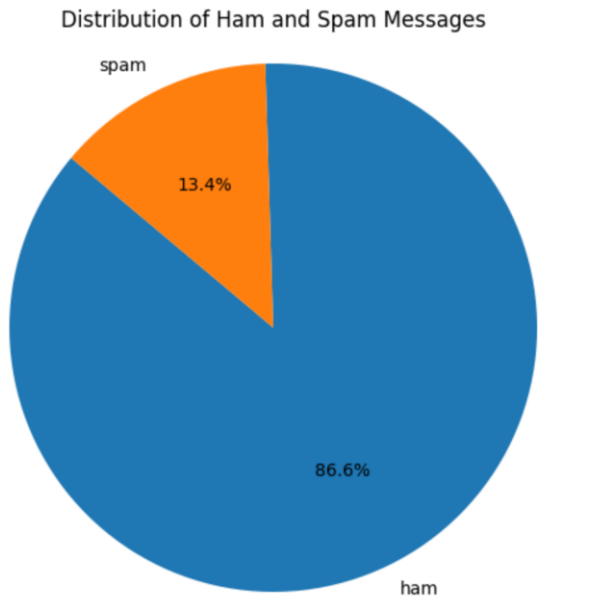
3. Run each cell one by one to see the results.

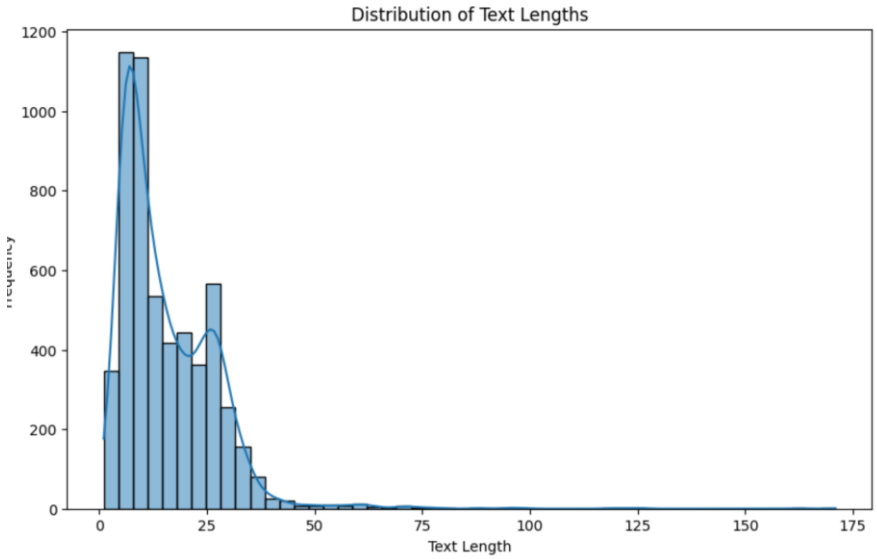
Dataset: The SMS Spam Collection dataset at the UCI Machine Learning Repository contains 5,574 SMS messages, labelled as spam or ham (not spam). It includes different sources such as manually extracted spam messages from a UK forum, and ham messages from the NUS SMS Corpus and other collections. This dataset is useful for research in mobile phone spam filtering and involves tasks like classification and clustering. Each record in the dataset is a single text file with the message's classification followed by the actual message text

Data set is downloaded from https://archive.ics.uci.edu/dataset/228/sms+spam+collection

Data set Description:

	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 aro...
5	spam	FreeMsg Hey there darling it's been 3 week's n...
6	ham	Even my brother is not like to speak with me. ...
7	ham	As per your request 'Melle Melle (Oru Minnamin...
8	spam	WINNER!! As a valued network customer you have...
9	spam	Had your mobile 11 months or more? U R entitle...





Models Used:

1. **Random Forest Classifier:**

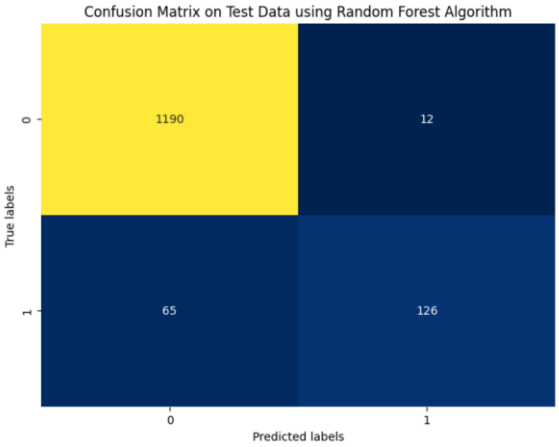
In [17]:
```
rfc = RandomForestClassifier(n_estimators=50, random_state=2)
rfc.fit(x_train_w2v, y_train)

# Predict on test data
y_pred = rfc.predict(x_test_w2v)
accuracy_score(y_test,y_pred)
```

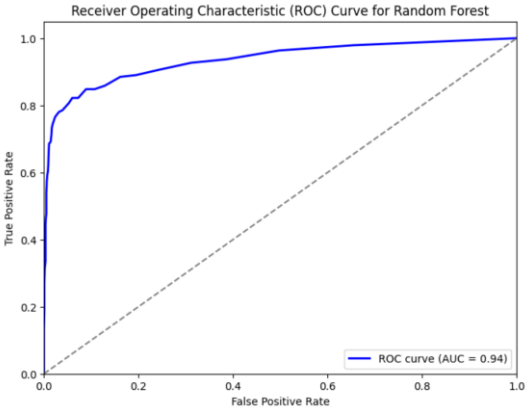
Out[17]: 0.9461593682699211

In [18]:
```
y_train=y_train.to_numpy()
```

In [19]:
```
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='cividis', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix on Test Data using Random Forest Algorithm')
plt.show()
```

Test Accuracy: 94%





10-Fold Cross Validation Results:

In [21]:

```
# Define the number of folds
num_folds = 10

# Initialize KFold cross-validation
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
# Lists to store evaluation metrics for each fold
fold_metrics = []

# Iterate over each fold
for fold, (train_index, test_index) in enumerate(kf.split(x_train_w2v), 1):
    X_train, X_test = x_train_w2v[train_index], x_train_w2v[test_index]
    y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]

    # Initialize and train the RandomForestClassifier
    rfc = RandomForestClassifier(n_estimators=50, random_state=2)
    rfc.fit(X_train, y_train_fold)

    # Predict on the test set
    y_pred_fold = rfc.predict(X_test)
    metrics=calculate_metrics(y_test_fold, y_pred_fold)

    # Store metrics for this fold
    fold_metrics.append(metrics)
# Create DataFrame from fold_metrics list
df_results_Random_Forest = pd.DataFrame(fold_metrics)

# Display the results
df_results_Random_Forest
```

	TP	TN	FP	FN	TSS	HSS	Accuracy	Precision	Recall	F1-Score
0	24	368	3	23	0.50	0.62	0.94	0.89	0.51	0.65
1	33	362	2	21	0.61	0.71	0.94	0.94	0.61	0.74
2	39	358	7	14	0.72	0.76	0.95	0.85	0.74	0.79
3	38	354	4	22	0.62	0.71	0.94	0.90	0.63	0.75
4	49	348	5	16	0.74	0.79	0.95	0.91	0.75	0.82
5	31	359	3	25	0.55	0.65	0.93	0.91	0.55	0.69
6	40	357	2	19	0.67	0.76	0.95	0.95	0.68	0.79
7	41	357	6	14	0.73	0.78	0.95	0.87	0.75	0.80
8	42	355	7	14	0.73	0.77	0.95	0.86	0.75	0.80
9	28	360	6	23	0.53	0.62	0.93	0.82	0.55	0.66

```
Average of 10 Folds
{'TP': 36.5, 'TN': 357.8, 'FP': 4.5, 'FN': 19.1, 'TSS': 0.64, 'HSS': 0.72, 'Accuracy': 0.94, 'Precision': 0.89, 'Recall': 0.65,
'F1-Score': 0.75}
```

Conclusion of Random Forest: Using Random Forest to classify SMS as Spam or Ham, accuracy obtained on the test data is 94.4%. ROC-AUC is around 0.94

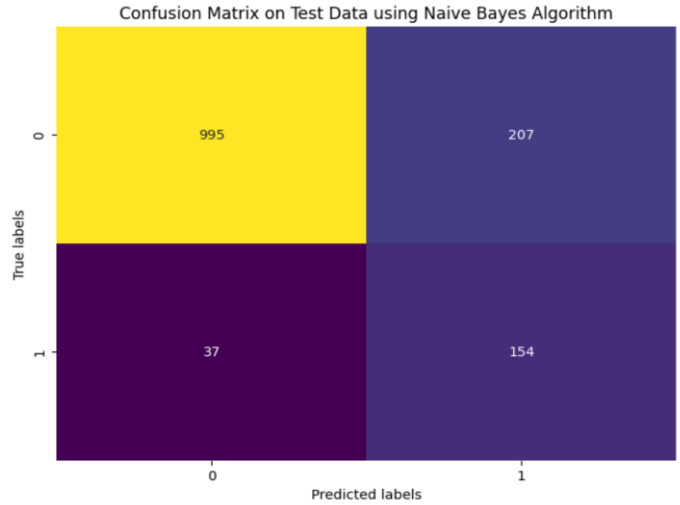
2. Naive Bayes:

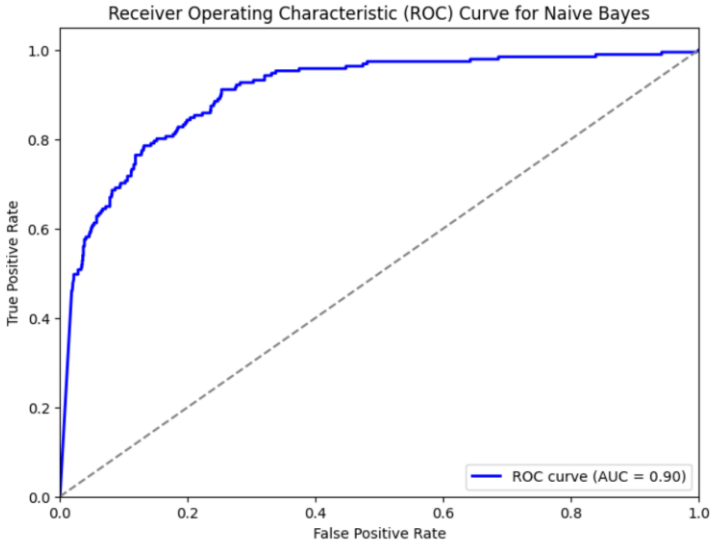
In [24]:
```
nb = GaussianNB()
nb.fit(x_train_w2v, y_train)
y_pred = nb.predict(x_test_w2v)
accuracy_score(y_test,y_pred)
```

Out[24]: 0.8650394831299354

In [25]:
```
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='viridis', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix on Test Data using Naive Bayes Algorithm')
plt.show()
```

Test accuracy: 86%





10-Fold Cross Validation Results:

In [27]:

```
num_folds = 10

kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
fold_metrics = []
for fold, (train_index, test_index) in enumerate(kf.split(x_train_w2v), 1):
    X_train, X_test = x_train_w2v[train_index], x_train_w2v[test_index]
    y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]
    nb = GaussianNB()
    nb.fit(X_train, y_train_fold)
    y_pred_fold = nb.predict(X_test)
    metrics=calculate_metrics(y_test_fold, y_pred_fold)
    fold_metrics.append(metrics)
# Create DataFrame from fold_metrics list
df_results_Naive_Bayes = pd.DataFrame(fold_metrics)

# Display the results
df_results_Naive_Bayes
```

Out[27]:

	TP	TN	FP	FN	TSS	HSS	Accuracy	Precision	Recall	F1-Score
0	32	319	52	15	0.54	0.40	0.84	0.38	0.68	0.49
1	40	308	56	14	0.59	0.44	0.83	0.42	0.74	0.53
2	44	310	55	9	0.68	0.50	0.85	0.44	0.83	0.58
3	47	315	43	13	0.66	0.55	0.87	0.52	0.78	0.63
4	58	308	45	7	0.76	0.62	0.88	0.56	0.89	0.69
5	46	321	41	10	0.71	0.57	0.88	0.53	0.82	0.64
6	47	306	53	12	0.65	0.50	0.84	0.47	0.80	0.59
7	44	309	54	11	0.65	0.49	0.84	0.45	0.80	0.58
8	49	299	63	7	0.70	0.49	0.83	0.44	0.88	0.58
9	35	317	49	16	0.55	0.43	0.84	0.42	0.69	0.52

```
Average of 10 Folds
{'TP': 44.2, 'TN': 311.2, 'FP': 51.1, 'FN': 11.4, 'TSS': 0.65, 'HSS': 0.5, 'Accuracy': 0.85, 'Precision': 0.46, 'Recall': 0.7
9, 'F1-Score': 0.58}
```

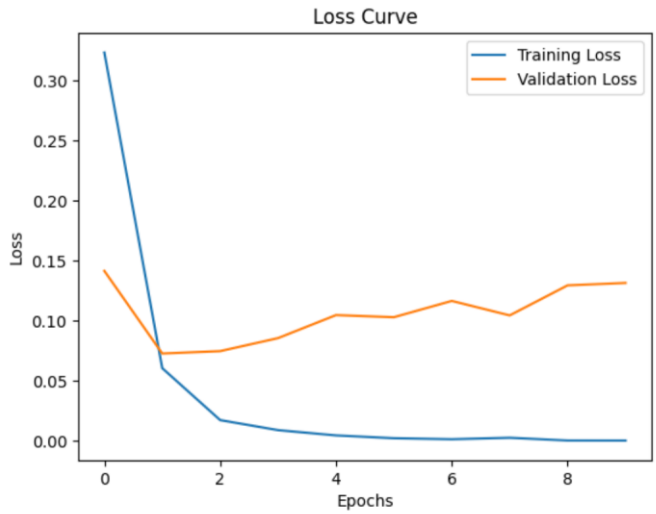
Conclusion of Naive Bayes: Using Naïve Bayes to classify SMS as Spam or Ham, accuracy obtained on the test data is 86.5%**.** ROC-AUC is around 0.90

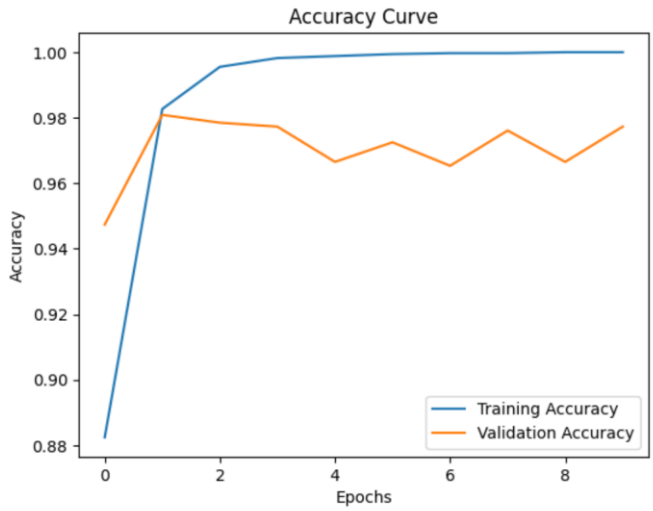
3. Deep Learning (LSTM):

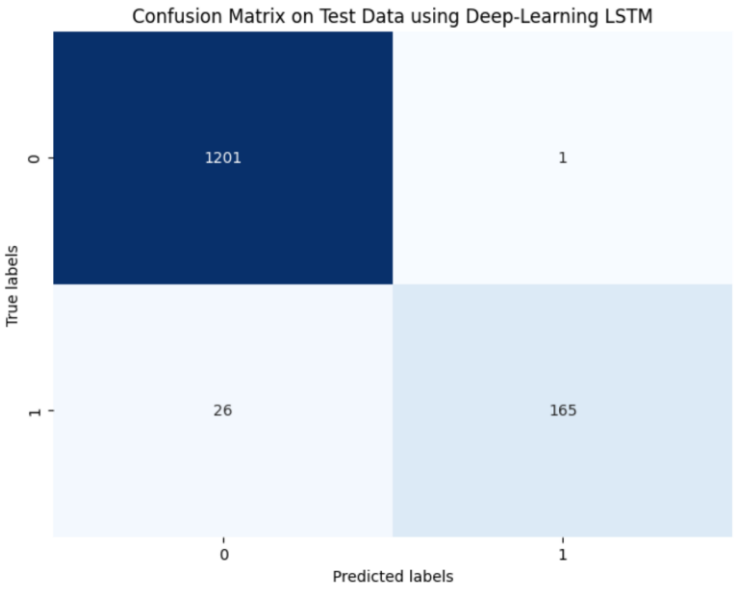
In [30]:

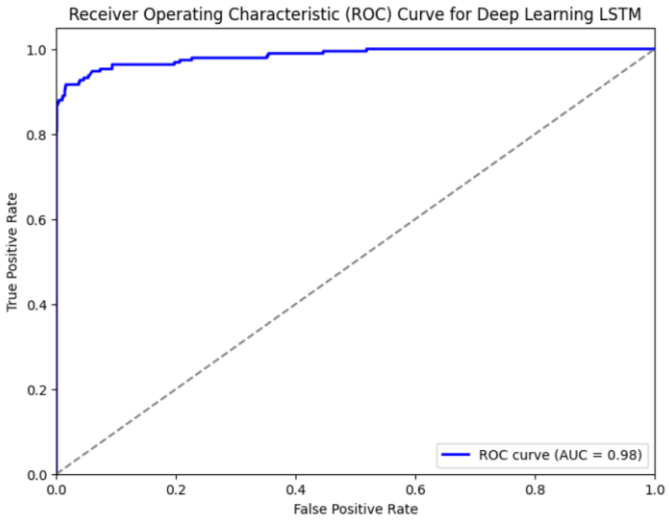
```
#Preprocessing Data
max_words = 10000
max_len = 100  # Maximum sequence length

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(x_train)

x_train_seq = tokenizer.texts_to_sequences(x_train)
x_test_seq = tokenizer.texts_to_sequences(x_test)

x_train_pad = pad_sequences(x_train_seq, maxlen=max_len)
x_test_pad = pad_sequences(x_test_seq, maxlen=max_len)

# Define LSTM model
embedding_dim = 100
lstm_units = 128

model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=embedding_dim, input_length=max_len))
model.add(LSTM(units=lstm_units))
model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(x_train_pad, np.array(y_train), epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model
loss, accuracy = model.evaluate(x_test_pad, np.array(y_test))
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

```
C:\Users\mohan\anaconda3\Lib\site-packages\keras\src\layers\core\embedding.py:86: UserWarning: Argument `input_length` is
recated. Just remove it.
  warnings.warn(
Epoch 1/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 15s 194ms/step - accuracy: 0.8439 - loss: 0.4050 - val_accuracy: 0.9557 - val_loss: 0.1485
Epoch 2/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 9s 167ms/step - accuracy: 0.9766 - loss: 0.0969 - val_accuracy: 0.9749 - val_loss: 0.0874
Epoch 3/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 11s 205ms/step - accuracy: 0.9917 - loss: 0.0322 - val_accuracy: 0.9773 - val_loss: 0.0800
Epoch 4/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 10s 185ms/step - accuracy: 0.9971 - loss: 0.0148 - val_accuracy: 0.9689 - val_loss: 0.0954
Epoch 5/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 9s 162ms/step - accuracy: 0.9979 - loss: 0.0082 - val_accuracy: 0.9665 - val_loss: 0.1079
Epoch 6/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 9s 162ms/step - accuracy: 0.9975 - loss: 0.0057 - val_accuracy: 0.9761 - val_loss: 0.1012
Epoch 7/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 9s 164ms/step - accuracy: 0.9992 - loss: 0.0018 - val_accuracy: 0.9785 - val_loss: 0.1209
Epoch 8/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 9s 165ms/step - accuracy: 0.9996 - loss: 0.0015 - val_accuracy: 0.9785 - val_loss: 0.1082
Epoch 9/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 8s 148ms/step - accuracy: 1.0000 - loss: 5.3777e-04 - val_accuracy: 0.9713 - val_loss: 0.1245
Epoch 10/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 8s 142ms/step - accuracy: 1.0000 - loss: 5.4008e-04 - val_accuracy: 0.9761 - val_loss: 0.1321
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.9856 - loss: 0.0855
Test Loss: 0.11728667467832565
Test Accuracy: 0.9820531010627747
```






Confusion Matrix on Test Data using Deep-Learning LSTM
1201
1
26
165
0
1
True labels
0
1
Predicted labels


Receiver Operating Characteristic (ROC) Curve for Deep Learning LSTM
True Positive Rate
False Positive Rate
ROC curve (AUC = 0.98)
0.0
0.2
0.4
0.6
0.8
1.0

10-Fold Cross Validation Results:

In [34]:

```
# Perform cross-validation

# Initialize KFold cross-validation with 10 folds
kf = KFold(n_splits=10, shuffle=True, random_state=42)

# Lists to store metrics for each fold
all_metrics = []

for train_index, val_index in kf.split(x_train_pad, y_train):
    x_train_cv, x_val = x_train_pad[train_index], x_train_pad[val_index]
    y_train_cv, y_val = y_train[train_index], y_train[val_index]

    # Train the model
    model.fit(x_train_cv, y_train_cv, epochs=10, batch_size=64, verbose=0)

    # Evaluate the model on validation data
    loss, accuracy = model.evaluate(x_val, y_val, verbose=0)
    y_pred_val = model.predict(x_val)

    # Convert predicted probabilities to binary predictions
    y_pred_val_binary = (y_pred_val > 0.5).astype(int)
    metrics["loss"]=loss
    metrics = calculate_metrics(y_val, y_pred_val_binary)
    all_metrics.append(metrics)

# Calculate average metrics across all folds
df_results_Deep_Learning_LSTM = pd.DataFrame(all_metrics)
```

Out[36]:

	TP	TN	FP	FN	TSS	HSS	Accuracy	Precision	Recall	F1-Score	loss
0	46	370	1	1	0.98	0.98	1.0	0.98	0.98	0.98	3.571645e-04
1	54	364	0	0	1.00	1.00	1.0	1.00	1.00	1.00	2.218150e-05
2	53	365	0	0	1.00	1.00	1.0	1.00	1.00	1.00	6.131699e-05
3	60	358	0	0	1.00	1.00	1.0	1.00	1.00	1.00	1.225194e-05
4	65	353	0	0	1.00	1.00	1.0	1.00	1.00	1.00	1.817252e-06
5	56	362	0	0	1.00	1.00	1.0	1.00	1.00	1.00	2.685215e-06
6	59	359	0	0	1.00	1.00	1.0	1.00	1.00	1.00	9.056951e-07
7	55	363	0	0	1.00	1.00	1.0	1.00	1.00	1.00	4.915096e-07
8	56	362	0	0	1.00	1.00	1.0	1.00	1.00	1.00	4.236955e-07
9	51	366	0	0	1.00	1.00	1.0	1.00	1.00	1.00	NaN

```
Average of 10 Folds
{'TP': 55.5, 'TN': 362.2, 'FP': 0.1, 'FN': 0.1, 'TSS': 1.0, 'HSS': 1.0, 'Accuracy': 1.0, 'Precision': 1.0, 'Recall': 1.0, 'F1
-Score': 1.0, 'loss': 0.0}
```

Conclusion of Deep Learning (LSTM): Using Deep Learning LSTM Model, we obtained training accuracy nearly 99% and validation accuracy nearly 98.2%**.** ROC-AUC of 0.99 is obtained.

Comparison and Analysis of Random Forest, Naïve Bayes, Deep Learning LSTM:

```
# Assuming drf, dnb, and dlstm are defined somewhere in your code and hold different results
df_random_forest = pd.DataFrame([drf], index=['Random Forest'])
df_naive_bayes = pd.DataFrame([dnb], index=['Naive Bayes'])
df_lstm = pd.DataFrame([dlstm], index=['Deep Learning LSTM'])

# Concatenate all the dataframes into a single one for comparison
df_comparison = pd.concat([df_random_forest, df_naive_bayes, df_lstm])

# Display the comparison DataFrame
print("Comparison of Average Results Across 10 Folds:")
print(df_comparison)
```

```
Comparison of Average Results Across 10 Folds:
                    TP     TN    FP    FN   TSS   HSS  Accuracy  Precision  \
Random Forest     38.2  357.9   4.4  17.4  0.67  0.74      0.95       0.89
Naive Bayes       43.9  311.0  51.3  11.7  0.64  0.50      0.85       0.46
Deep Learning LSTM  55.4  362.2   0.1   0.2  0.99  1.00      1.00       1.00

                    Recall  F1-Score  loss
Random Forest         0.68      0.77   NaN
Naive Bayes           0.79      0.58   NaN
Deep Learning LSTM    1.00      1.00   0.0
```

Model	Accuracy	ROC-AUC
Random Forest	94.4	0.94
Naïve Bayes	86.5	0.9
Deep Learning (LSTM)	98.2	0.99

Random Forest uses an ensemble of decision trees to classify the embeddings of SMS as Spam or Ham. Naïve Bayes uses probabilistic models based on Bayes' theorem to classify data, making the assumption of independence among features. The LSTM (Long Short-Term Memory) architecture is specifically designed to handle sequential text data for NLP use cases; hence, the performance of LSTM is better than the above two machine learning models.

Accuracy: LSTM performed the best among three in terms of accuracy, followed by Random Forest and Naive Bayes.

Precision and Recall Trade-off: Random Forest had a better balance with reasonable precision and recall, while Naive Bayes had high recall but low precision. LSTM is assumed to perform well on both fronts given the high accuracy and AUC.

Model Suitability: Considering the spam detection, where false negatives are more critical, a higher recall is preferable. Naive Bayes, despite its low precision, excels in recall. However, LSTM offers the best overall performance considering all metrics.

Conclusion:

The LSTM model outperformed the other models in most metrics, making it the most suitable for this task based on the taken data and preprocessing steps. However, each model has its strengths and could be optimized further or used in ensemble methods to leverage their individual benefits.

Git Repository Link:

https://github.com/MohanSaiBandarupalli/Final_Project

The Source Code (.py), Jupyter notebook (.iypnb) and datasets (.csv) are attached in Zip file.