

# vitz\_john\_finaltermproj

**Name:** John Vitz

**UCID:** jhv6

**Email Address:** [jhv6@njit.edu](mailto:jhv6@njit.edu)

**Class:** CS 634-101

**Professor:** Yasser Abdullah

**Date:** November 24, 2024

## 1. Final Project - Data Mining

### Github Repository Link:

<https://github.com/VitzJ/CS634-Data-Mining-Final-Project> The dataset for this project is comprised of questions from the website <https://www.quora.com/>. Quora is an internet forum that allows users to ask and answer each other's questions. In 2019, the website submitted a dataset comprised of 1.3+ million unique different questions along with an attached target label to a kaggle competition with the goal of leveraging the power of the kaggle community to come up with machine learning and natural language processing solutions to the following problem:

**How can we tell whether or not a question is sincere?**

For this project, I utilized 3 different machine learning algorithms in order to perform basic binary classification.

- - a. Random Forest (Required)
- - a. Bi-Directional LSTM (Additional Option: Deep Learning)
- - a. Naive Bayes (Additional Option: Algorithms)

I also included a Logistic Regression model for comparison. The goal of utilizing these models was to examine various classification task metrics across models utilizing 10 fold cross validation in order to learn about their implementation as well as how to interpret them in the context of machine learning models.

The zip file folder **readme.txt** / the github **readme.md** contain a detailed tutorial on how to run the python source file **vitz\_john\_finaltermproj.py** in the command prompt terminal.

### 1.1 Data Source, Data Description, Data Loading, and Imports

**Data Source:** <https://www.kaggle.com/c/quora-insincere-questions-classification/data>

## Important Data Pre-Processing Information:

The data was extremely large, so I was forced to cut the size down very substantially. I cut it down to a small subset utilizing the following code:

```
train = pd.read_csv('/content/train.csv')

# I want to get rid of questions that are below a choice groups of
# characters long
# because I don't think they will be valuable for training models, I
# want the
# limited data to contain at least 5 groups of characters.
min_chars = 5

num_train = 2500 # 2500 train values
num_test = int(num_train * 0.1) # 250 total test values

pos_neg_split = 0.5 # 50/50 class split

pos_train = int(num_train * pos_neg_split)
neg_train = num_train - pos_train

pos_test = int(num_test * pos_neg_split)
neg_test = num_test - pos_test

train['length'] = (train['question_text'].str.split().str.len())
train = train[train['length'] >= min_chars].copy()

test = pd.concat([train[train['target'] == 0][pos_train: pos_train +
pos_test].copy(),
                  train[train['target'] == 1][neg_train: neg_train +
neg_test].copy()],
                  axis=0)

train = pd.concat([train[train['target'] == 0][:pos_train].copy(),
                  train[train['target'] == 1][:neg_train].copy()],
                  axis=0)

train.to_csv('vitz_john_finaltermproj_train_set.csv', index=False)
test.to_csv('vitz_john_finaltermproj_test_set.csv', index=False)

train = pd.read_csv('vitz_john_finaltermproj_train_set.csv')
test = pd.read_csv('vitz_john_finaltermproj_test_set.csv')
```

The train.csv file from the previously mentioned source: <https://www.kaggle.com/c/quora-insincere-questions-classification/data?select=train.csv>

The data is filtered based on class membership, and then the number of training and testing points are selected from the subset. This is done in order to assure that the new subset will have balanced class membership.

## Data Class and Fields Description

### Dataset Overview

- **Number of questions in train:** 2500
- **Number of questions in test:** 250 (10% of amount of questions in the train data)

### Target Values

There are two possible values for the `target` field:

- **0:** Indicates that the given question **is not** insincere.
- **1:** Indicates that the given question **is** insincere.

### Data Fields

- **qid:** Unique question identifier
- **question\_text:** Quora question text
- **target:** A question labeled "insincere" has a value of 1, otherwise 0

## Required Imports

Package Name	Package Version	Website
pandas	2.2.2	<a href="https://pandas.pydata.org/">https://pandas.pydata.org/</a>
numpy	1.26.4	<a href="https://numpy.org/">https://numpy.org/</a>
matplotlib	3.8.0	<a href="https://matplotlib.org/">https://matplotlib.org/</a>
scikit-learn	1.5.2	<a href="https://scikit-learn.org/stable/">https://scikit-learn.org/stable/</a>
torch	2.5.1	<a href="https://pytorch.org/">https://pytorch.org/</a>
nltk	3.9.1	<a href="https://www.nltk.org/">https://www.nltk.org/</a>

```
# data structures/display methods imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.feature_extraction.text import TfidfVectorizer

# model imports
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
```

```

# neural network model imports
import torch
import torch.nn as nn
import torch.autograd as autograd
import torch.optim as optim
import torch.nn.functional as F

# metrics imports
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import auc

# string methods imports
import nltk # https://www.nltk.org/data.html
nltk.download('stopwords')
from nltk.corpus import stopwords
import re # regular expressions library builtin
import string # string library builtin

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

train = pd.read_csv(f'vitz_john_finaltermproj_train_set.csv')
test = pd.read_csv(f'vitz_john_finaltermproj_test_set.csv')

train.head()

{"summary": "{\n  \"name\": \"train\", \n  \"rows\": 2500, \n  \"fields\": [\n    {\n      \"column\": \"qid\", \n      \"properties\": {\n        \"dtype\": \"string\", \n        \"num_unique_values\": 2500, \n        \"samples\": [\n          \"0098e5b8335a62791abf\", \n          \"003ae8ef6c2b869b9a0c\", \n          \"00382fbffbe071469ce\" \n        ], \n        \"semantic_type\": \"\", \n        \"description\": \"\" \n      }, \n      \"column\": \"question_text\", \n      \"properties\": {\n        \"dtype\": \"string\", \n        \"num_unique_values\": 2500, \n        \"samples\": [\n          \"Is Dr. Steven Greer an extremely good fraudster with his claims of ufo' s and aliens?\", \n          \"Where in Abu Dhabi can I get all electronic components like resistors, transistors, breadboards, etc?\", \n          \"Has Nintendo ever made a game in which Pauline from Donkey Kong meets Princess Peach?\" \n        ], \n        \"semantic_type\": \"\", \n        \"description\": \"\" \n      }, \n      {\n        \"column\": \"target\", \n        \"properties\": {\n          \"dtype\": \"number\", \n          \"std\": 0, \n          \"min\": 0, \n          \"max\": 1, \n          \"num_unique_values\": 2, \n          \"samples\": [\n            1, \n            0 \n          ], \n          \"semantic_type\": \"\", \n          \"description\": \"\" \n        }, \n        {\n          \"column\": \"length\", \n          \"properties\": {\n            \"dtype\": \"number\", \n            \"std\": 8, \n            \"min\": 5, \n            \"max\": 51, \n            \"num_unique_values\": 47, \n          } \n        } \n      ] \n    } \n  ] \n}"

```

```

n        \"samples\": [\n                24,\n                49\n            ],\n\"semantic_type\": \"\", \n        \"description\": \"\"\n    }\n    ]\n}","type":"dataframe","variable_name":"train"}

grouped_lengths = train.groupby('target')['length'].agg(['count',
'min', 'max', 'median', 'mean', 'std'])
print(f'\nA comparison of the train set length of character groups
before cleaning: \n')
print(grouped_lengths, '\n')

```

A comparison of the train set length of character groups before cleaning:

	count	min	max	median	mean	std
target						
0	1250	5	50	11.0	12.6632	6.758514
1	1250	5	51	15.0	17.3864	9.341933

## 1.2 Data Handling and Cleaning Information

I utilized multiple techniques such as lowercasing, removing particular components of text, removing links, punctuation, and removing words containing numbers. I also removed stopwords through nltk and utilized a technique call stemming on the data. I applied both of these processes to the train set and the test set while maintaining their independence from each other. This was done with the purpose of standardizing the text to allow for more concentration into the vectorization.

```

# Special thanks to https://www.kaggle.com/tanulsingh077 for this
function

def clean_text(text):
    '''Make text lowercase, remove text in square brackets,remove
    links,remove punctuation
    and remove words containing numbers.'''

    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)

    return text

```

## Removal of Stopwords

Can try with and without. In an effort to reduce computational requirements for this project, I will try removing stopwords.

```
stop_words = list(set(stopwords.words('english'))) # uses nltk
#print(stop_words)

def remove_stopwords(text):
    text = ' '.join(word for word in text.split(' ') if word not in
stop_words)
    return text
```

## Stemming

Stemming - chopping off the ends of words in hopes to remove derivational affixes, such as trying to get words like 'adjustably' and 'adjustable' to be recognized under the same umbrella 'adjust' which is the root word by removing terms such as 'ably' and 'able' from the word.

```
stemmer = nltk.SnowballStemmer("english")

def stemm_text(text):
    text = ' '.join(stemmer.stem(word) for word in text.split(' '))
    return text
```

Clean Punctuation, Remove Stopwords, and Stem all words in the data.

```
def preprocess_data(text):
    # Clean punctuation, urls, and so on
    text = clean_text(text)
    # Remove stopwords
    text = ' '.join(word for word in text.split(' ') if word not in
stop_words)
    # Stem all the words in the sentence
    text = ' '.join(stemmer.stem(word) for word in text.split(' '))

    return text

# Apply the cleaning to the train set
train['question_text'] = train['question_text'].apply(preprocess_data)

# Apply the same cleaning to the test set (without looking at it in
any way)
test['question_text'] = test['question_text'].apply(preprocess_data)

train['new_length'] = (train['question_text'].str.split().str.len())
grouped_newlengths = train.groupby('target')
['new_length'].agg(['count', 'min', 'max', 'median', 'mean', 'std'])
```

```
print(f'\nA comparison of the length of character groups after
cleaning: \n')
print(grouped_newlengths, '\n')
avg_words_removed = sum(train['length'] - train['new_length']) /
train.shape[0]
print(f'\nAverage number of words removed after cleaning:
{avg_words_removed}\n\n')
```

A comparison of the length of character groups after cleaning:

	count	min	max	median	mean	std
target						
0	1250	1	26	5.0	6.2712	3.393474
1	1250	2	27	8.0	9.0912	4.863189

Average number of words removed after cleaning: 7.3436

## 1.3 Data Loading and Metrics Helper Functions

I utilize sklearn KFold to generate a 10 fold split for 10 fold cross validation. I also declare a few helper functions.

### K-Fold Cross Validation Splits

I have to make sure that when I use vectorizers / train models, I don't use the test fold to train the model, so I have to cut out the test data separately each time.

```
X_train = train['question_text'].values
y_train = train['target'].values

X_test = test['question_text'].values
y_test = test['target'].values

K_fold_CV = KFold(n_splits=10, shuffle=True, random_state=42)
K_fold_CV.get_n_splits(X_train)

10
```

## Helper Functions

### Get Metrics For Models

The **get\_metrics()** function processes y\_test (ground truth vector), prediction (model prediction vector), prediction probabilities (model prediction probabilities vector), and nn\_bool (boolean to handle neural network inputs). It returns a labeled pandas dataframe consisting of the various metrics computed through the function.

```

# Metrics from
https://njit.instructure.com/courses/42753/files/6730397?
module\_item\_id=1417132

def get_metrics(y_test, prediction, prediction_probabilities,
nn_bool=False):

    # TP: actual is 1, predicted is 1
    TP = sum(((np.array(y_test) == 1) & (np.array(prediction) == 1)) &
(np.array(y_test) == 1)) # True Positives

    # TN: actual is 0, predicted is 0
    TN = sum(((np.array(y_test) == 0) & (np.array(prediction) == 0)) &
(np.array(y_test) == 0)) # True Negatives

    # FP: actual is 0, predicted is 1, type II error
    FP = sum(((np.array(y_test) == 0) & (np.array(prediction) == 1)) &
(np.array(y_test) == 0)) # False Positives

    # FN: actual is 1, predicted is 0, type I error
    FN = sum(((np.array(y_test) == 1) & (np.array(prediction) == 0)) &
(np.array(y_test) == 1)) # False Negatives

    # Handle division by zero with 0 as default value

    TPR = 0 if (TP + FN) == 0 else TP / (TP + FN) # True Positive Rate
| Recall / Sensitivity

    TNR = 0 if (TN + FP) == 0 else TN / (TN + FP) # True Negative Rate
| Specificity

    FPR = 0 if (TN + FP) == 0 else FP / (TN + FP) # False Positive Rate

    FNR = 0 if (TP + FN) == 0 else FN / (TP + FN) # False Negative Rate

    Precision = 0 if (TP + FP) == 0 else TP / (TP + FP) # Precision

    F1_measure = 0 if (((2 * TP) + FP + FN) == 0) else (2 * TP) / ((2 *
TP) + FP + FN) # F1 Measure

    Accuracy = 0 if (TP + FP + TN + FN) == 0 else (TP + TN) / (TP + FP +
TN + FN) # Accuracy

    Error_rate = 0 if (TP + FP + TN + FN) == 0 else (FP + FN) / (TP + FP
+ TN + FN) # Error rate

    BACC = (TPR + TNR) / 2 # Balanced Accuracy

    TSS = TPR - FPR # True Skill Statistics

    HSS = (2 * ((TP * TN) - (FP * FN))) / (((TP + FN) * (FN + TN)) +

```



```

((TP + FP) * (FP + TN))) # Heidke Skill Score

if nn_bool:
    ### FOR LSTM ###
    # The values are all close to 0.5, which means that the model has
    likely not actually learned anything
    # from the data and is just adding in random noise for the
    predictions. Our classes are balanced for the chopped
    # data, so the naive prediction from BS_Naive is actually equal to
    the class proportions (50% 0, 50% 1). What this
    # means is that when comparing in BSS we get close to 1 because
    our model's output is very close to being Naive
    # this basically means the model isn't really learning much if at
    all.

    # Brier_Score = MSE between expected probabilities and predicted
    probabilities
    BS = (1 / y_test.shape[0]) * torch.sum((y_test -
    prediction_probabilities) ** 2)) # (1 / N) * sum((y_test -
    prediction_probabilities) ** 2)
    BS = BS.item()

    # Brier Skill Score = likelihood of an event to happen or BS / BS
    (Naive) where BS (Naive)
    # uses the average probability of class membership instead of
    prediction_probabilities
    BS_Naive = ((1 / y_test.shape[0])) * torch.sum((y_test -
    (torch.sum(y_test) / y_test.shape[0])) ** 2) # (1 / N) * (sum(y_test -
    mean(y_test) ** 2))
    BS_Naive = BS_Naive.item()

else:
    # Brier_Score = MSE between expected probabilities and predicted
    probabilities
    BS = (1 / len(y_test)) * sum(((y_test - prediction_probabilities)
    ** 2)) # (1 / N) * sum((y_test - prediction_probabilities) ** 2)

    # Brier Skill Score = likelihood of an event to happen or BS / BS
    (Naive) where BS (Naive)
    # uses the average probability of class membership instead of
    prediction_probabilities
    BS_Naive = ((1 / len(y_test))) * sum((y_test - (sum(y_test) /
    len(y_test))) ** 2) # (1 / N) * (sum(y_test - mean(y_test) ** 2))

    BSS = BS / BS_Naive

    NPV = 0 if (TN + FN) == 0 else TN / (TN + FN) # Negative Predictive
    Value

    FDR = 0 if (FP + TP) == 0 else FP / (FP + TP) # False Discovery Rate

```

```

ROC_AUC_SCORE = roc_auc_score(y_test, prediction_probabilities)

final_metrics = [TP, TN, FP, FN, TPR, TNR, FPR, FNR,
                  Precision, F1_measure, Accuracy, Error_rate,
                  BACC, TSS, HSS, BS, BSS, NPV, FDR, ROC_AUC_SCORE]

final_metrics_labels = ['TP', 'TN', 'FP', 'FN', 'TPR', 'TNR', 'FPR',
                        'FNR',
                        'Precision', 'F1_measure', 'Accuracy',
                        'Error_rate',
                        'BACC', 'TSS', 'HSS', 'BS', 'BSS', 'NPV',
                        'FDR',
                        'ROC_AUC_SCORE']

return pd.DataFrame(data=final_metrics, index=final_metrics_labels)

```

## Metrics Per Iteration and Average Metrics

The function **metrics\_per\_iteration** reshapes the metrics stored in the pandas dataframes from **get\_metrics()** into a format that arranges the metrics by model and then by fold.

```

def metrics_per_iteration(input_metrics, num_folds=10, verbose=True):
    #iterations = list(range(num_folds))
    columns = input_metrics[0].columns.to_list()

    new_dfs = []

    for model_col in columns:
        temp_df = pd.DataFrame()

        for k in range(num_folds):
            #print(input_metrics[k])
            temp_df[f'Fold{k + 1}'] = input_metrics[k][model_col]

        new_dfs.append(temp_df)

        if verbose:
            print(f'Metrics for {model_col}:')
            print(new_dfs[-1], '\n')

    return new_dfs

```

The function **average\_metrics** simply returns the metrics for each model averaged across all of the folds.

```

def average_metrics(input_metrics, num_folds=10):
    avg_metrics = sum(input_metrics) / num_folds
    print('Average Metrics by Model')

```

```
print(avg_metrics)
return avg_metrics
```

## 1.4 Machine Learning Algorithms

For this project, I utilized 3 different machine learning algorithms in order to perform basic binary classification.

- - a. Random Forest (Required)
- - a. Bi-Directional LSTM (Additional Option: Deep Learning)
- - a. Naive Bayes (Additional Option: Algorithms)

I also included a Logistic Regression model for comparison.

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

### Bi-Directional LSTM using pytorch

In this section, I construct a Bi-Directional LSTM using pytorch. The input is expanded tf-idf vectors. The model itself is extremely simple, because the training time is very long as is, and as I will explain in the final section, the input doesn't exactly return desirable results.

```
# Define a Bi-directional LSTM model for classification
class BiLSTMModel(nn.Module):

    def __init__(self, input_size, hidden_size, output_size):
        super(BiLSTMModel, self).__init__()
        # Bidirectional LSTM: set bidirectional=True
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True,
                             bidirectional=True)

        # Adjust the output size of the fully connected layer to
        # account for the doubled hidden size
        self.fc = nn.Linear(hidden_size * 2, output_size) #
        # hidden_size * 2 because of bidirectional

    def forward(self, x):
        # LSTM layer
        lstm_out, (h_n, c_n) = self.lstm(x) # lstm_out has shape
        # (batch_size, seq_len, hidden_size * 2)

        # We use the output from the last timestep of the
        # bidirectional LSTM
        out = lstm_out[:, -1, :] # Get the output from the last
        # timestep (for classification)
```

```

        # Apply fully connected layer
        out = self.fc(out)

        # Apply sigmoid for binary classification
        return torch.sigmoid(out) # Sigmoid for binary
classification, outputs range from [0, 1]

def fit_lstm(model, X_train_tensor, y_train_tensor,
prediction_threshold=0.5, verbose=False):

    # Loss function and optimizer
    criterion = nn.BCELoss() # Binary Cross-Entropy loss for binary
classification
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    # Training loop
    num_epochs = 5

    for epoch in range(num_epochs):
        model.train() # would be for if we had dropout/batch normalization
layers
        optimizer.zero_grad() # clears gradients from previous epoch

        # Forward pass
        outputs = model(X_train_tensor) # Forward pass through the model
        loss = criterion(outputs, y_train_tensor) # Computes the loss
between actual y and pred y

        # Backward pass and optimization
        loss.backward() # calculates the gradients (backprop)
        optimizer.step() # utilizes loss through the optimizer to update
weights

        # Print loss every few epochs
        if verbose & ((epoch == 0) | ((epoch + 1) % 2 == 0) | ((epoch + 1)
== num_epochs)):
            print(f"Epoch [{epoch+1}/{num_epochs}], Loss:
{loss.item():.4f}")

    return model

```

## 1.5 TFIDF Vectorization output with All Machine Learning Models

The **all\_models\_tfidf\_vectorize\_10FoldCV()** function utilizes KFold splits to generate tf-idf vector representations of the train set from each respective split. It then passes the tfidf vector to each of the models, trains them, generates predictions, and then passes them to the **get\_metrics** function.

**all\_models\_tfidf\_vectorize\_10FoldCV()** then records the metrics for each model for each fold, prints the metrics out for all models for the current fold, and repeats for each split until it finally returns a list of pandas dataframes (1 for each fold) on completion.

```
def all_models_tfidf_vectorize_10FoldCV():  
    metrics = []  
  
    for i, (train_index, test_index) in  
enumerate(K_fold_CV.split(X=X_train, y=y_train)):  
        # declare models  
        print(f"Fold {i + 1}:")  
  
        # get fold train subsets (Note that X and y are global variables)  
        temp_X_train = X_train[train_index]  
        temp_y_train = y_train[train_index]  
  
        # get fold test subsets (Note that X and y are global variables)  
        temp_X_test = X_train[test_index]  
        temp_y_test = y_train[test_index]  
  
        # instantiate and fit the tf-idf vectorizer  
        tfidf_vector = TfidfVectorizer(stop_words='english',  
                                       ngram_range=(1, 2),  
                                       norm="l2",  
                                       smooth_idf=True # prevents 0 divisions  
                                       #min_df=30, # lowest number of occurrences  
                                       required for words to be recorded in dtm  
                                       #max_df=0.7, # highest percentage  
                                       threshold before words stop being recorded in dtm  
                                       #max_features=100  
                                       )  
  
        tfidf_vector.fit(temp_X_train)  
  
        # Transform the X_train/X_test folds  
        X_train_tfidf_vector = tfidf_vector.transform(temp_X_train)  
        X_test_tfidf_vector = tfidf_vector.transform(temp_X_test)  
  
        # Transform for LSTM input  
        x_train_dtm_l_LSTM =  
np.expand_dims(X_train_tfidf_vector.toarray(), axis=1) # Shape  
becomes (n_samples, 1, n_features)  
        x_test_dtm_l_LSTM = np.expand_dims(X_test_tfidf_vector.toarray(),  
axis=1)  
  
        # Convert data to PyTorch tensors for LSTM  
        X_train_tensor = torch.tensor(x_train_dtm_l_LSTM,  
dtype=torch.float32)  
        X_test_tensor = torch.tensor(x_test_dtm_l_LSTM,
```

```

dtype=torch.float32)
    y_train_tensor = torch.tensor(temp_y_train,
dtype=torch.float32).view(-1, 1) # Reshaping for binary
classification
    y_test_tensor = torch.tensor(temp_y_test,
dtype=torch.float32).view(-1, 1) # Reshaping for binary classification

    # Get various models
    log_reg_model = LogisticRegression()
    rfc_model = RandomForestClassifier()
    nb_model = MultinomialNB()
    biLSTM_model = BiLSTMModel(input_size=X_train_tensor.shape[2], #
This is the number of features (i.e., the TF-IDF size)
                                hidden_size=64, # Hidden units in LSTM
                                output_size=1 # Binary output (0 or 1)
                                )

    # arrange models for input into a loop
    models = [log_reg_model, rfc_model, nb_model, biLSTM_model]
    model_names = ['LogisticRegression', 'RandomForestClassifier',
'Naive_Bayes', 'Bi-LSTM']

    iteration_i = []

    for k, temp_model in enumerate(models):

        # test if the model is a pytorch neural network
        if isinstance(temp_model, nn.Module):

            # fit model and predict for X_test
            temp_model = fit_lstm(temp_model, X_train_tensor,
y_train_tensor)

            # Evaluate the model on the test data
            temp_model.eval() # Set the model to evaluation mode

            # Don't calculate gradients during prediction
            with torch.no_grad():

                # output is a probability of positive (1) class membership
from range [0, 1]
                temp_model_proba_prediction = temp_model(X_test_tensor)

                # Mask the model output with a threshold value to get binary
predictions
                temp_model_prediction = (temp_model_proba_prediction >=
0.5).float()

                # get metrics for current iteration for each neural network
model

```

```

        temp_model_metrics = get_metrics(y_test_tensor.view(-1),
                                         temp_model_prediction.view(-
1),
temp_model_proba_prediction.view(-1),
                                         nn_bool=True)

    else:

        # fit the model and predict for X_test
        temp_model.fit(X_train_tfidf_vector, temp_y_train) #
X_train_tfidf_vector is the transformed count vectorized X_train
matrix
        temp_model_prediction =
temp_model.predict(X_test_tfidf_vector) # X_test_tfidf_vector is the
transformed count vectorized X_test matrix

        # output is a probability of positive (1) class membership
from range [0, 1]
        temp_model_proba_prediction =
temp_model.predict_proba(X_test_tfidf_vector)[: , 1]

        # get metrics for current iteration for each non-neural
network model
        temp_model_metrics = get_metrics(temp_y_test,
                                         temp_model_prediction,
                                         temp_model_proba_prediction,
                                         nn_bool=False)

        temp_model_metrics.rename(columns={0: model_names[k]},
inplace=True)
        iteration_i.append(temp_model_metrics)

        # get a single table with metrics for current fold for every model
metrics.append(pd.concat(iteration_i, axis=1))

    print(metrics[i], '\n')

    return metrics

print('Displaying Model Metrics by fold for 10 folds: \n')
final_metrics = all_models_tfidf_vectorize_10FoldCV()

```

Displaying Model Metrics by fold for 10 folds:

Fold 1:

	Logistic_Regression	Random_Forest_Classifier
Naive_Bayes \		
TP	104.000000	88.000000
112.000000		
TN	95.000000	108.000000

79.000000		
FP	29.000000	16.000000
45.000000		
FN	22.000000	38.000000
14.000000		
TPR	0.825397	0.698413
0.888889		
TNR	0.766129	0.870968
0.637097		
FPR	0.233871	0.129032
0.362903		
FNR	0.174603	0.301587
0.111111		
Precision	0.781955	0.846154
0.713376		
F1_measure	0.803089	0.765217
0.791519		
Accuracy	0.796000	0.784000
0.764000		
Error_rate	0.204000	0.216000
0.236000		
BACC	0.795763	0.784690
0.762993		
TSS	0.591526	0.569380
0.525986		
HSS	0.591791	0.568580
0.527031		
BS	0.171899	0.162701
0.171762		
BSS	0.687642	0.650846
0.687091		
NPV	0.811966	0.739726
0.849462		
FDR	0.218045	0.153846
0.286624		
ROC_AUC_SCORE	0.872056	0.852375
0.864183		

	Bi-LSTM
TP	102.000000
TN	56.000000
FP	68.000000
FN	24.000000
TPR	0.809524
TNR	0.451613
FPR	0.548387
FNR	0.190476
Precision	0.600000
F1_measure	0.689189



Accuracy	0.632000
Error_rate	0.368000
BACC	0.630568
TSS	0.261137
HSS	0.261874
BS	0.248779
BSS	0.995178
NPV	0.700000
FDR	0.400000
ROC_AUC_SCORE	0.681324

Fold 2:

	Logistic_Regression	Random_Forest_Classifier
Naive_Bayes \		
TP	112.000000	89.000000
128.000000		
TN	87.000000	101.000000
80.000000		
FP	25.000000	11.000000
32.000000		
FN	26.000000	49.000000
10.000000		
TPR	0.811594	0.644928
0.927536		
TNR	0.776786	0.901786
0.714286		
FPR	0.223214	0.098214
0.285714		
FNR	0.188406	0.355072
0.072464		
Precision	0.817518	0.890000
0.800000		
F1_measure	0.814545	0.747899
0.859060		
Accuracy	0.796000	0.760000
0.832000		
Error_rate	0.204000	0.240000
0.168000		
BACC	0.794190	0.773357
0.820911		
TSS	0.588380	0.546713
0.641822		
HSS	0.587885	0.529781
0.653922		
BS	0.166021	0.175644
0.158891		
BSS	0.671347	0.710257
0.642512		
NPV	0.769912	0.673333

0.888889		
FDR	0.182482	0.110000
0.200000		
ROC_AUC_SCORE	0.884252	0.868498
0.888975		

	Bi-LSTM
TP	8.000000
TN	111.000000
FP	1.000000
FN	130.000000
TPR	0.057971
TNR	0.991071
FPR	0.008929
FNR	0.942029
Precision	0.888889
F1_measure	0.108844
Accuracy	0.476000
Error_rate	0.524000
BACC	0.524521
TSS	0.049042
HSS	0.044242
BS	0.249290
BSS	1.008063
NPV	0.460581
FDR	0.111111
ROC_AUC_SCORE	0.722438

Fold 3:

	Logistic_Regression	Random_Forest_Classifier
Naive_Bayes \		
TP	96.000000	83.000000
100.000000		
TN	102.000000	116.000000
84.000000		
FP	31.000000	17.000000
49.000000		
FN	21.000000	34.000000
17.000000		
TPR	0.820513	0.709402
0.854701		
TNR	0.766917	0.872180
0.631579		
FPR	0.233083	0.127820
0.368421		
FNR	0.179487	0.290598
0.145299		
Precision	0.755906	0.830000
0.671141		

F1_measure	0.786885	0.764977
0.751880		
Accuracy	0.792000	0.796000
0.736000		
Error_rate	0.208000	0.204000
0.264000		
BACC	0.793715	0.790791
0.743140		
TSS	0.587430	0.581582
0.486280		
HSS	0.584426	0.586710
0.478409		
BS	0.171648	0.142990
0.173728		
BSS	0.689416	0.574311
0.697770		
NPV	0.829268	0.773333
0.831683		
FDR	0.244094	0.170000
0.328859		
ROC_AUC_SCORE	0.873401	0.885740
0.869481		

	Bi-LSTM
TP	0.000000
TN	133.000000
FP	0.000000
FN	117.000000
TPR	0.000000
TNR	1.000000
FPR	0.000000
FNR	1.000000
Precision	0.000000
F1_measure	0.000000
Accuracy	0.532000
Error_rate	0.468000
BACC	0.500000
TSS	0.000000
HSS	0.000000
BS	0.247177
BSS	0.992773
NPV	0.532000
FDR	0.000000
ROC_AUC_SCORE	0.780348

Fold 4:

	Logistic_Regression	Random_Forest_Classifier
Naive_Bayes \		
TP	103.000000	92.000000

111.000000		
TN	99.000000	106.000000
88.000000		
FP	28.000000	21.000000
39.000000		
FN	20.000000	31.000000
12.000000		
TPR	0.837398	0.747967
0.902439		
TNR	0.779528	0.834646
0.692913		
FPR	0.220472	0.165354
0.307087		
FNR	0.162602	0.252033
0.097561		
Precision	0.786260	0.814159
0.740000		
F1_measure	0.811024	0.779661
0.813187		
Accuracy	0.808000	0.792000
0.796000		
Error_rate	0.192000	0.208000
0.204000		
BACC	0.808463	0.791307
0.797676		
TSS	0.616926	0.582613
0.595352		
HSS	0.616295	0.583360
0.593301		
BS	0.178376	0.157385
0.169174		
BSS	0.713686	0.629702
0.676871		
NPV	0.831933	0.773723
0.880000		
FDR	0.213740	0.185841
0.260000		
ROC_AUC_SCORE	0.846745	0.855995
0.866590		

	Bi-LSTM
TP	86.000000
TN	94.000000
FP	33.000000
FN	37.000000
TPR	0.699187
TNR	0.740157
FPR	0.259843
FNR	0.300813

Precision	0.722689
F1_measure	0.710744
Accuracy	0.720000
Error_rate	0.280000
BACC	0.719672
TSS	0.439344
HSS	0.439570
BS	0.248200
BSS	0.993052
NPV	0.717557
FDR	0.277311
ROC_AUC_SCORE	0.779271

Fold 5:

	Logistic_Regression	Random_Forest_Classifier
Naive_Bayes \		
TP	103.000000	95.000000
109.000000		
TN	101.000000	112.000000
84.000000		
FP	26.000000	15.000000
43.000000		
FN	20.000000	28.000000
14.000000		
TPR	0.837398	0.772358
0.886179		
TNR	0.795276	0.881890
0.661417		
FPR	0.204724	0.118110
0.338583		
FNR	0.162602	0.227642
0.113821		
Precision	0.798450	0.863636
0.717105		
F1_measure	0.817460	0.815451
0.792727		
Accuracy	0.816000	0.828000
0.772000		
Error_rate	0.184000	0.172000
0.228000		
BACC	0.816337	0.827124
0.773798		
TSS	0.632674	0.654247
0.547596		
HSS	0.632188	0.655338
0.545571		
BS	0.166142	0.134012
0.168322		
BSS	0.664739	0.536185

0.673461		
NPV	0.834711	0.800000
0.857143		
FDR	0.201550	0.136364
0.282895		
ROC_AUC_SCORE	0.884386	0.896870
0.878113		

	Bi-LSTM
TP	78.000000
TN	82.000000
FP	45.000000
FN	45.000000
TPR	0.634146
TNR	0.645669
FPR	0.354331
FNR	0.365854
Precision	0.634146
F1_measure	0.634146
Accuracy	0.640000
Error_rate	0.360000
BACC	0.639908
TSS	0.279816
HSS	0.279816
BS	0.248426
BSS	0.993960
NPV	0.645669
FDR	0.365854
ROC_AUC_SCORE	0.740862

Fold 6:

	Logistic_Regression	Random_Forest_Classifier
Naive_Bayes \		
TP	106.000000	87.000000
111.000000		
TN	101.000000	110.000000
87.000000		
FP	23.000000	14.000000
37.000000		
FN	20.000000	39.000000
15.000000		
TPR	0.841270	0.690476
0.880952		
TNR	0.814516	0.887097
0.701613		
FPR	0.185484	0.112903
0.298387		
FNR	0.158730	0.309524
0.119048		

Precision	0.821705	0.861386
0.750000		
F1_measure	0.831373	0.766520
0.810219		
Accuracy	0.828000	0.788000
0.792000		
Error_rate	0.172000	0.212000
0.208000		
BACC	0.827893	0.788786
0.791283		
TSS	0.655786	0.577573
0.582565		
HSS	0.655912	0.576650
0.583387		
BS	0.168722	0.158018
0.162347		
BSS	0.674930	0.632112
0.649429		
NPV	0.834711	0.738255
0.852941		
FDR	0.178295	0.138614
0.250000		
ROC_AUC_SCORE	0.877752	0.864855
0.891897		

	Bi-LSTM
TP	120.000000
TN	32.000000
FP	92.000000
FN	6.000000
TPR	0.952381
TNR	0.258065
FPR	0.741935
FNR	0.047619
Precision	0.566038
F1_measure	0.710059
Accuracy	0.608000
Error_rate	0.392000
BACC	0.605223
TSS	0.210445
HSS	0.211610
BS	0.248585
BSS	0.994405
NPV	0.842105
FDR	0.433962
ROC_AUC_SCORE	0.727407

Fold 7:

Logistic_Regression	Random_Forest_Classifier
---------------------	--------------------------

Naive_Bayes \		
TP	105.000000	85.000000
115.000000		
TN	102.000000	111.000000
90.000000		
FP	23.000000	14.000000
35.000000		
FN	20.000000	40.000000
10.000000		
TPR	0.840000	0.680000
0.920000		
TNR	0.816000	0.888000
0.720000		
FPR	0.184000	0.112000
0.280000		
FNR	0.160000	0.320000
0.080000		
Precision	0.820312	0.858586
0.766667		
F1_measure	0.830040	0.758929
0.836364		
Accuracy	0.828000	0.784000
0.820000		
Error_rate	0.172000	0.216000
0.180000		
BACC	0.828000	0.784000
0.820000		
TSS	0.656000	0.568000
0.640000		
HSS	0.656000	0.568000
0.640000		
BS	0.163457	0.151408
0.161681		
BSS	0.653827	0.605634
0.646723		
NPV	0.836066	0.735099
0.900000		
FDR	0.179688	0.141414
0.233333		
ROC_AUC_SCORE	0.888064	0.876256
0.884992		

	Bi-LSTM
TP	125.000000
TN	4.000000
FP	121.000000
FN	0.000000
TPR	1.000000
TNR	0.032000



FPR	0.968000
FNR	0.000000
Precision	0.508130
F1_measure	0.673854
Accuracy	0.516000
Error_rate	0.484000
BACC	0.516000
TSS	0.032000
HSS	0.032000
BS	0.248941
BSS	0.995764
NPV	1.000000
FDR	0.491870
ROC_AUC_SCORE	0.693504

Fold 8:

	Logistic_Regression	Random_Forest_Classifier
Naive_Bayes \		
TP	100.000000	86.000000
113.000000		
TN	104.000000	112.000000
87.000000		
FP	21.000000	13.000000
38.000000		
FN	25.000000	39.000000
12.000000		
TPR	0.800000	0.688000
0.904000		
TNR	0.832000	0.896000
0.696000		
FPR	0.168000	0.104000
0.304000		
FNR	0.200000	0.312000
0.096000		
Precision	0.826446	0.868687
0.748344		
F1_measure	0.813008	0.767857
0.818841		
Accuracy	0.816000	0.792000
0.800000		
Error_rate	0.184000	0.208000
0.200000		
BACC	0.816000	0.792000
0.800000		
TSS	0.632000	0.584000
0.600000		
HSS	0.632000	0.584000
0.600000		
BS	0.165513	0.145247
0.159401		

BSS	0.662053	0.580987
0.637604		
NPV	0.806202	0.741722
0.878788		
FDR	0.173554	0.131313
0.251656		
ROC_AUC_SCORE	0.894144	0.898336
0.899840		

	Bi-LSTM
TP	53.000000
TN	105.000000
FP	20.000000
FN	72.000000
TPR	0.424000
TNR	0.840000
FPR	0.160000
FNR	0.576000
Precision	0.726027
F1_measure	0.535354
Accuracy	0.632000
Error_rate	0.368000
BACC	0.632000
TSS	0.264000
HSS	0.264000
BS	0.248614
BSS	0.994457
NPV	0.593220
FDR	0.273973
ROC_AUC_SCORE	0.733760

Fold 9:

	Logistic_Regression	Random_Forest_Classifier
Naive_Bayes \		
TP	86.000000	67.000000
94.000000		
TN	113.000000	132.000000
99.000000		
FP	33.000000	14.000000
47.000000		
FN	18.000000	37.000000
10.000000		
TPR	0.826923	0.644231
0.903846		
TNR	0.773973	0.904110
0.678082		
FPR	0.226027	0.095890
0.321918		
FNR	0.173077	0.355769

0.096154		
Precision	0.722689	0.827160
0.666667		
F1_measure	0.771300	0.724324
0.767347		
Accuracy	0.796000	0.796000
0.772000		
Error_rate	0.204000	0.204000
0.228000		
BACC	0.800448	0.774170
0.790964		
TSS	0.600896	0.548340
0.581928		
HSS	0.588683	0.566356
0.553599		
BS	0.172035	0.136518
0.172657		
BSS	0.708127	0.561930
0.710687		
NPV	0.862595	0.781065
0.908257		
FDR	0.277311	0.172840
0.333333		
ROC_AUC_SCORE	0.888567	0.890477
0.891596		

	Bi-LSTM
TP	0.000000
TN	146.000000
FP	0.000000
FN	104.000000
TPR	0.000000
TNR	1.000000
FPR	0.000000
FNR	1.000000
Precision	0.000000
F1_measure	0.000000
Accuracy	0.584000
Error_rate	0.416000
BACC	0.500000
TSS	0.000000
HSS	0.000000
BS	0.246830
BSS	1.015997
NPV	0.584000
FDR	0.000000
ROC_AUC_SCORE	0.745061

Fold 10:

Naive_Bayes \	Logistic_Regression	Random_Forest_Classifier
TP	120.000000	104.000000
127.000000		
TN	76.000000	88.000000
63.000000		
FP	31.000000	19.000000
44.000000		
FN	23.000000	39.000000
16.000000		
TPR	0.839161	0.727273
0.888112		
TNR	0.710280	0.822430
0.588785		
FPR	0.289720	0.177570
0.411215		
FNR	0.160839	0.272727
0.111888		
Precision	0.794702	0.845528
0.742690		
F1_measure	0.816327	0.781955
0.808917		
Accuracy	0.784000	0.768000
0.760000		
Error_rate	0.216000	0.232000
0.240000		
BACC	0.774721	0.774851
0.738448		
TSS	0.549441	0.549703
0.476897		
HSS	0.554661	0.537067
0.493141		
BS	0.174873	0.163189
0.166050		
BSS	0.714305	0.666577
0.678265		
NPV	0.767677	0.692913
0.797468		
FDR	0.205298	0.154472
0.257310		
ROC_AUC_SCORE	0.859029	0.858016
0.869290		
	Bi-LSTM	
TP	54.000000	
TN	90.000000	
FP	17.000000	
FN	89.000000	
TPR	0.377622	

TNR	0.841121
FPR	0.158879
FNR	0.622378
Precision	0.760563
F1_measure	0.504673
Accuracy	0.576000
Error_rate	0.424000
BACC	0.609372
TSS	0.218744
HSS	0.201663
BS	0.249195
BSS	1.017886
NPV	0.502793
FDR	0.239437
ROC_AUC_SCORE	0.683485

```
print('Displaying Model Metrics by model for 10 folds: \n')
final_metrics_per_iter = metrics_per_iteration(final_metrics)
```

Displaying Model Metrics by model for 10 folds:

Metrics for Logistic Regression:

	Fold1	Fold2	Fold3	Fold4
Fold5 \				
TP	104.000000	112.000000	96.000000	103.000000
103.000000				
TN	95.000000	87.000000	102.000000	99.000000
101.000000				
FP	29.000000	25.000000	31.000000	28.000000
26.000000				
FN	22.000000	26.000000	21.000000	20.000000
20.000000				
TPR	0.825397	0.811594	0.820513	0.837398
0.837398				
TNR	0.766129	0.776786	0.766917	0.779528
0.795276				
FPR	0.233871	0.223214	0.233083	0.220472
0.204724				
FNR	0.174603	0.188406	0.179487	0.162602
0.162602				
Precision	0.781955	0.817518	0.755906	0.786260
0.798450				
F1_measure	0.803089	0.814545	0.786885	0.811024
0.817460				
Accuracy	0.796000	0.796000	0.792000	0.808000
0.816000				
Error_rate	0.204000	0.204000	0.208000	0.192000
0.184000				
BACC	0.795763	0.794190	0.793715	0.808463

0.816337				
TSS	0.591526	0.588380	0.587430	0.616926
0.632674				
HSS	0.591791	0.587885	0.584426	0.616295
0.632188				
BS	0.171899	0.166021	0.171648	0.178376
0.166142				
BSS	0.687642	0.671347	0.689416	0.713686
0.664739				
NPV	0.811966	0.769912	0.829268	0.831933
0.834711				
FDR	0.218045	0.182482	0.244094	0.213740
0.201550				
ROC_AUC_SCORE	0.872056	0.884252	0.873401	0.846745
0.884386				

	Fold6	Fold7	Fold8	Fold9
Fold10				
TP	106.000000	105.000000	100.000000	86.000000
120.000000				
TN	101.000000	102.000000	104.000000	113.000000
76.000000				
FP	23.000000	23.000000	21.000000	33.000000
31.000000				
FN	20.000000	20.000000	25.000000	18.000000
23.000000				
TPR	0.841270	0.840000	0.800000	0.826923
0.839161				
TNR	0.814516	0.816000	0.832000	0.773973
0.710280				
FPR	0.185484	0.184000	0.168000	0.226027
0.289720				
FNR	0.158730	0.160000	0.200000	0.173077
0.160839				
Precision	0.821705	0.820312	0.826446	0.722689
0.794702				
F1_measure	0.831373	0.830040	0.813008	0.771300
0.816327				
Accuracy	0.828000	0.828000	0.816000	0.796000
0.784000				
Error_rate	0.172000	0.172000	0.184000	0.204000
0.216000				
BACC	0.827893	0.828000	0.816000	0.800448
0.774721				
TSS	0.655786	0.656000	0.632000	0.600896
0.549441				
HSS	0.655912	0.656000	0.632000	0.588683
0.554661				
BS	0.168722	0.163457	0.165513	0.172035

0.174873				
BSS	0.674930	0.653827	0.662053	0.708127
0.714305				
NPV	0.834711	0.836066	0.806202	0.862595
0.767677				
FDR	0.178295	0.179688	0.173554	0.277311
0.205298				
ROC_AUC_SCORE	0.877752	0.888064	0.894144	0.888567
0.859029				

Metrics for Random\_Forest\_Classifier:

	Fold1	Fold2	Fold3	Fold4
Fold5 \				
TP	88.000000	89.000000	83.000000	92.000000
95.000000				
TN	108.000000	101.000000	116.000000	106.000000
112.000000				
FP	16.000000	11.000000	17.000000	21.000000
15.000000				
FN	38.000000	49.000000	34.000000	31.000000
28.000000				
TPR	0.698413	0.644928	0.709402	0.747967
0.772358				
TNR	0.870968	0.901786	0.872180	0.834646
0.881890				
FPR	0.129032	0.098214	0.127820	0.165354
0.118110				
FNR	0.301587	0.355072	0.290598	0.252033
0.227642				
Precision	0.846154	0.890000	0.830000	0.814159
0.863636				
F1_measure	0.765217	0.747899	0.764977	0.779661
0.815451				
Accuracy	0.784000	0.760000	0.796000	0.792000
0.828000				
Error_rate	0.216000	0.240000	0.204000	0.208000
0.172000				
BACC	0.784690	0.773357	0.790791	0.791307
0.827124				
TSS	0.569380	0.546713	0.581582	0.582613
0.654247				
HSS	0.568580	0.529781	0.586710	0.583360
0.655338				
BS	0.162701	0.175644	0.142990	0.157385
0.134012				
BSS	0.650846	0.710257	0.574311	0.629702
0.536185				
NPV	0.739726	0.673333	0.773333	0.773723
0.800000				

FDR	0.153846	0.110000	0.170000	0.185841
0.136364				
ROC_AUC_SCORE	0.852375	0.868498	0.885740	0.855995
0.896870				
	Fold6	Fold7	Fold8	Fold9
Fold10				
TP	87.000000	85.000000	86.000000	67.000000
104.000000				
TN	110.000000	111.000000	112.000000	132.000000
88.000000				
FP	14.000000	14.000000	13.000000	14.000000
19.000000				
FN	39.000000	40.000000	39.000000	37.000000
39.000000				
TPR	0.690476	0.680000	0.688000	0.644231
0.727273				
TNR	0.887097	0.888000	0.896000	0.904110
0.822430				
FPR	0.112903	0.112000	0.104000	0.095890
0.177570				
FNR	0.309524	0.320000	0.312000	0.355769
0.272727				
Precision	0.861386	0.858586	0.868687	0.827160
0.845528				
F1_measure	0.766520	0.758929	0.767857	0.724324
0.781955				
Accuracy	0.788000	0.784000	0.792000	0.796000
0.768000				
Error_rate	0.212000	0.216000	0.208000	0.204000
0.232000				
BACC	0.788786	0.784000	0.792000	0.774170
0.774851				
TSS	0.577573	0.568000	0.584000	0.548340
0.549703				
HSS	0.576650	0.568000	0.584000	0.566356
0.537067				
BS	0.158018	0.151408	0.145247	0.136518
0.163189				
BSS	0.632112	0.605634	0.580987	0.561930
0.666577				
NPV	0.738255	0.735099	0.741722	0.781065
0.692913				
FDR	0.138614	0.141414	0.131313	0.172840
0.154472				
ROC_AUC_SCORE	0.864855	0.876256	0.898336	0.890477
0.858016				

Metrics for Naive\_Bayes:



Fold5 \	Fold1	Fold2	Fold3	Fold4
TP	112.000000	128.000000	100.000000	111.000000
109.000000				
TN	79.000000	80.000000	84.000000	88.000000
84.000000				
FP	45.000000	32.000000	49.000000	39.000000
43.000000				
FN	14.000000	10.000000	17.000000	12.000000
14.000000				
TPR	0.888889	0.927536	0.854701	0.902439
0.886179				
TNR	0.637097	0.714286	0.631579	0.692913
0.661417				
FPR	0.362903	0.285714	0.368421	0.307087
0.338583				
FNR	0.111111	0.072464	0.145299	0.097561
0.113821				
Precision	0.713376	0.800000	0.671141	0.740000
0.717105				
F1_measure	0.791519	0.859060	0.751880	0.813187
0.792727				
Accuracy	0.764000	0.832000	0.736000	0.796000
0.772000				
Error_rate	0.236000	0.168000	0.264000	0.204000
0.228000				
BACC	0.762993	0.820911	0.743140	0.797676
0.773798				
TSS	0.525986	0.641822	0.486280	0.595352
0.547596				
HSS	0.527031	0.653922	0.478409	0.593301
0.545571				
BS	0.171762	0.158891	0.173728	0.169174
0.168322				
BSS	0.687091	0.642512	0.697770	0.676871
0.673461				
NPV	0.849462	0.888889	0.831683	0.880000
0.857143				
FDR	0.286624	0.200000	0.328859	0.260000
0.282895				
ROC_AUC_SCORE	0.864183	0.888975	0.869481	0.866590
0.878113				
	Fold6	Fold7	Fold8	Fold9
Fold10				
TP	111.000000	115.000000	113.000000	94.000000
127.000000				
TN	87.000000	90.000000	87.000000	99.000000
63.000000				

FP	37.000000	35.000000	38.000000	47.000000
44.000000				
FN	15.000000	10.000000	12.000000	10.000000
16.000000				
TPR	0.880952	0.920000	0.904000	0.903846
0.888112				
TNR	0.701613	0.720000	0.696000	0.678082
0.588785				
FPR	0.298387	0.280000	0.304000	0.321918
0.411215				
FNR	0.119048	0.080000	0.096000	0.096154
0.111888				
Precision	0.750000	0.766667	0.748344	0.666667
0.742690				
F1_measure	0.810219	0.836364	0.818841	0.767347
0.808917				
Accuracy	0.792000	0.820000	0.800000	0.772000
0.760000				
Error_rate	0.208000	0.180000	0.200000	0.228000
0.240000				
BACC	0.791283	0.820000	0.800000	0.790964
0.738448				
TSS	0.582565	0.640000	0.600000	0.581928
0.476897				
HSS	0.583387	0.640000	0.600000	0.553599
0.493141				
BS	0.162347	0.161681	0.159401	0.172657
0.166050				
BSS	0.649429	0.646723	0.637604	0.710687
0.678265				
NPV	0.852941	0.900000	0.878788	0.908257
0.797468				
FDR	0.250000	0.233333	0.251656	0.333333
0.257310				
ROC_AUC_SCORE	0.891897	0.884992	0.899840	0.891596
0.869290				
Metrics for Bi-LSTM:				
	Fold1	Fold2	Fold3	Fold4
Fold5 \				
TP	102.000000	8.000000	0.000000	86.000000
78.000000				
TN	56.000000	111.000000	133.000000	94.000000
82.000000				
FP	68.000000	1.000000	0.000000	33.000000
45.000000				
FN	24.000000	130.000000	117.000000	37.000000
45.000000				
TPR	0.809524	0.057971	0.000000	0.699187

0.634146				
TNR	0.451613	0.991071	1.000000	0.740157
0.645669				
FPR	0.548387	0.008929	0.000000	0.259843
0.354331				
FNR	0.190476	0.942029	1.000000	0.300813
0.365854				
Precision	0.600000	0.888889	0.000000	0.722689
0.634146				
F1_measure	0.689189	0.108844	0.000000	0.710744
0.634146				
Accuracy	0.632000	0.476000	0.532000	0.720000
0.640000				
Error_rate	0.368000	0.524000	0.468000	0.280000
0.360000				
BACC	0.630568	0.524521	0.500000	0.719672
0.639908				
TSS	0.261137	0.049042	0.000000	0.439344
0.279816				
HSS	0.261874	0.044242	0.000000	0.439570
0.279816				
BS	0.248779	0.249290	0.247177	0.248200
0.248426				
BSS	0.995178	1.008063	0.992773	0.993052
0.993960				
NPV	0.700000	0.460581	0.532000	0.717557
0.645669				
FDR	0.400000	0.111111	0.000000	0.277311
0.365854				
ROC_AUC_SCORE	0.681324	0.722438	0.780348	0.779271
0.740862				

	Fold6	Fold7	Fold8	Fold9
Fold10				
TP	120.000000	125.000000	53.000000	0.000000
54.000000				
TN	32.000000	4.000000	105.000000	146.000000
90.000000				
FP	92.000000	121.000000	20.000000	0.000000
17.000000				
FN	6.000000	0.000000	72.000000	104.000000
89.000000				
TPR	0.952381	1.000000	0.424000	0.000000
0.377622				
TNR	0.258065	0.032000	0.840000	1.000000
0.841121				
FPR	0.741935	0.968000	0.160000	0.000000
0.158879				
FNR	0.047619	0.000000	0.576000	1.000000

0.622378				
Precision	0.566038	0.508130	0.726027	0.000000
0.760563				
F1_measure	0.710059	0.673854	0.535354	0.000000
0.504673				
Accuracy	0.608000	0.516000	0.632000	0.584000
0.576000				
Error_rate	0.392000	0.484000	0.368000	0.416000
0.424000				
BACC	0.605223	0.516000	0.632000	0.500000
0.609372				
TSS	0.210445	0.032000	0.264000	0.000000
0.218744				
HSS	0.211610	0.032000	0.264000	0.000000
0.201663				
BS	0.248585	0.248941	0.248614	0.246830
0.249195				
BSS	0.994405	0.995764	0.994457	1.015997
1.017886				
NPV	0.842105	1.000000	0.593220	0.584000
0.502793				
FDR	0.433962	0.491870	0.273973	0.000000
0.239437				
ROC_AUC_SCORE	0.727407	0.693504	0.733760	0.745061
0.683485				

```
print('Displaying Averaged Model Metrics: \n')
final_average_metrics = average_metrics(final_metrics)
```

Displaying Averaged Model Metrics:

Average Metrics by Model		Logistic_Regression	Random_Forest_Classifier
Naive_Bayes \			
TP		103.500000	87.600000
112.000000			
TN		98.000000	109.600000
84.100000			
FP		27.000000	15.400000
40.900000			
FN		21.500000	37.400000
13.000000			
TPR		0.827965	0.700305
0.895665			
TNR		0.783140	0.875911
0.672177			
FPR		0.216860	0.124089
0.327823			
FNR		0.172035	0.299695

0.104335		
Precision	0.792594	0.850530
0.731599		
F1_measure	0.809505	0.767279
0.805006		
Accuracy	0.806000	0.788800
0.784400		
Error_rate	0.194000	0.211200
0.215600		
BACC	0.805553	0.788108
0.783921		
TSS	0.611106	0.576215
0.567843		
HSS	0.609984	0.575584
0.566836		
BS	0.169869	0.152711
0.166401		
BSS	0.684007	0.614854
0.670041		
NPV	0.818504	0.744917
0.864463		
FDR	0.207406	0.149470
0.268401		
ROC_AUC_SCORE	0.876840	0.874742
0.880496		

	Bi-LSTM
TP	62.600000
TN	85.300000
FP	39.700000
FN	62.400000
TPR	0.495483
TNR	0.679970
FPR	0.320030
FNR	0.504517
Precision	0.540648
F1_measure	0.456686
Accuracy	0.591600
Error_rate	0.408400
BACC	0.587726
TSS	0.175453
HSS	0.173477
BS	0.248404
BSS	1.000154
NPV	0.657793
FDR	0.259352
ROC_AUC_SCORE	0.728746

## 1.6 ROC Curves and AUC scores on **test** dataset

We now take each model and train them on the full **train** dataset so that we can evaluate them on the untouched **test** dataset. The training and evaluation process is the same as was in the **all\_models\_tfidf\_vectorize\_10FoldCV()** function.

We do this in order to generate plots of the ROC/AUC curves for each model.

```
# instantiate and fit the tf-idf vectorizer
tfidf_vector = TfidfVectorizer(stop_words='english',
                               ngram_range=(1, 2),
                               norm="l2",
                               smooth_idf=True # prevents 0 divisions
                               #min_df=30, # lowest number of occurrences
                               #max_df=0.7, # highest percentage threshold
                               #max_features=100
                               )

tfidf_vector.fit(X_train)

# Transform the X_train/X_test folds
X_train_tfidf_vector = tfidf_vector.transform(X_train)
X_test_tfidf_vector = tfidf_vector.transform(X_test)

# Transform for LSTM input
x_train_dtm_l_LSTM = np.expand_dims(X_train_tfidf_vector.toarray(),
axis=1) # Shape becomes (n_samples, 1, n_features)
x_test_dtm_l_LSTM = np.expand_dims(X_test_tfidf_vector.toarray(),
axis=1)

# Convert data to PyTorch tensors for LSTM
X_train_tensor = torch.tensor(x_train_dtm_l_LSTM, dtype=torch.float32)
X_test_tensor = torch.tensor(x_test_dtm_l_LSTM, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).view(-1,
1) # Reshaping for binary classification
y_test_tensor = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)
# Reshaping for binary classification

def get_AUC_ROC_curve(temp_model, model_name=''):

    # if the input model is a torch.nn module (Neural Network)
    if isinstance(temp_model, nn.Module):
        # fit model and predict for X_test
        temp_model = fit_lstm(temp_model, X_train_tensor, y_train_tensor)

    # Evaluate the model on the test data
    temp_model.eval() # Set the model to evaluation mode
```

```

with torch.no_grad():
    temp_model_proba_prediction = temp_model(X_test_tensor) #
    outputs ranging between [0, 1]

    # our outputs are on a range from [0, 1] and in our case represent
    relative probabilities
    # (this is our model's estimation) that each test question is
    either a member of the
    # class (1) or a not a member of the class (0). So that means
    these results are comparable
    # to what we get from predict_proba, and can be used in the AUC /
    ROC plot.

else:
    temp_model.fit(X_train_tfidf_vector, y_train)
    temp_model_proba_prediction =
temp_model.predict_proba(X_test_tfidf_vector)[:, 1]

#return (X_test_tfidf_vector, temp_model_proba_prediction)
# Compute ROC curve and ROC area
fpr, tpr, _ = roc_curve(y_true=y_test,
                        y_score=temp_model_proba_prediction)

roc_auc = auc(fpr, tpr)

print('Please close the graphic to continue')

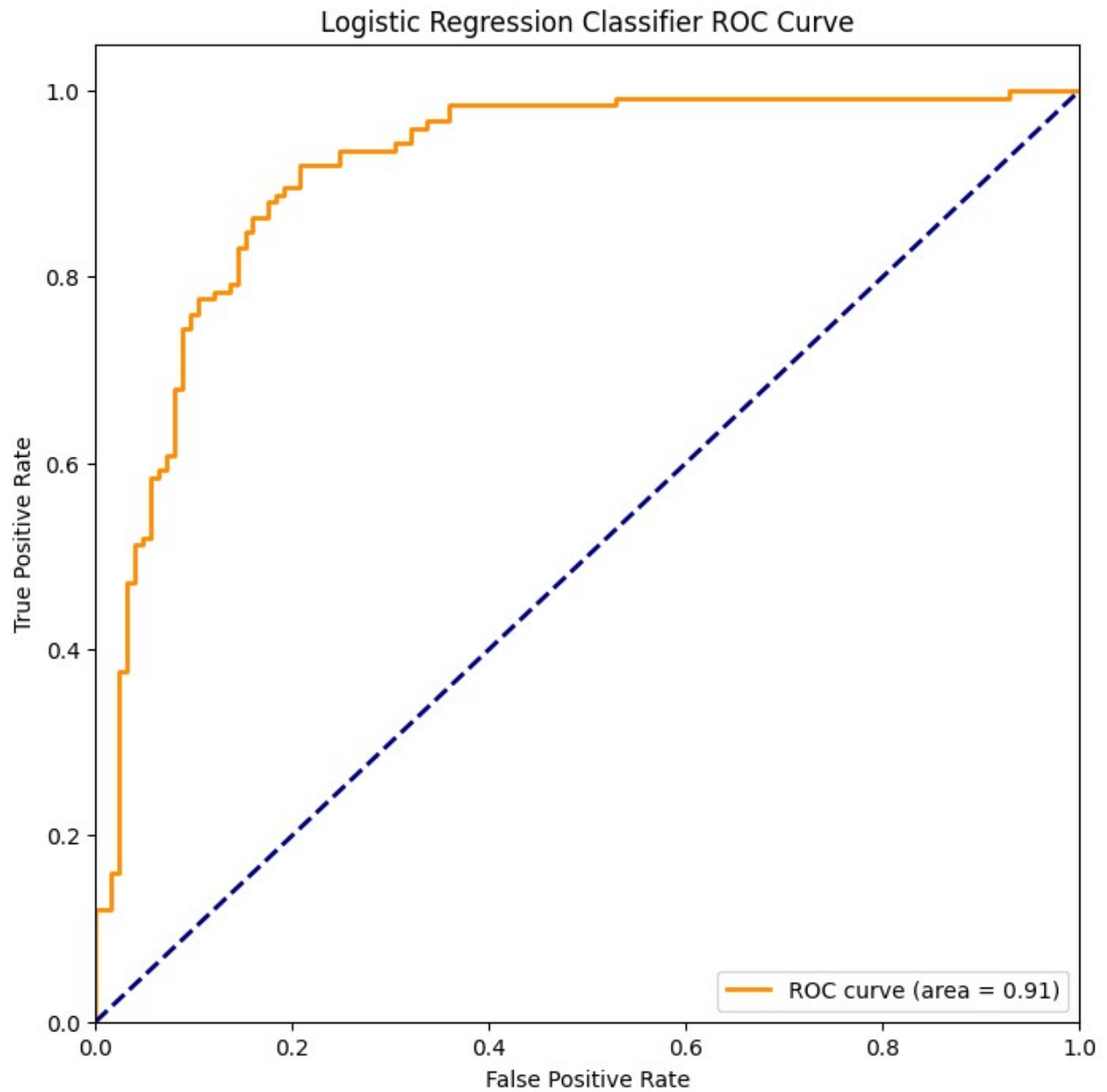
# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area
= {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'{model_name} ROC Curve')
plt.legend(loc='lower right')
plt.show()

return # (X_test_tfidf_vector, temp_model_proba_prediction)

get_AUC_ROC_curve(LogisticRegression(), model_name='Logistic
Regression Classifier')

Please close the graphic to continue

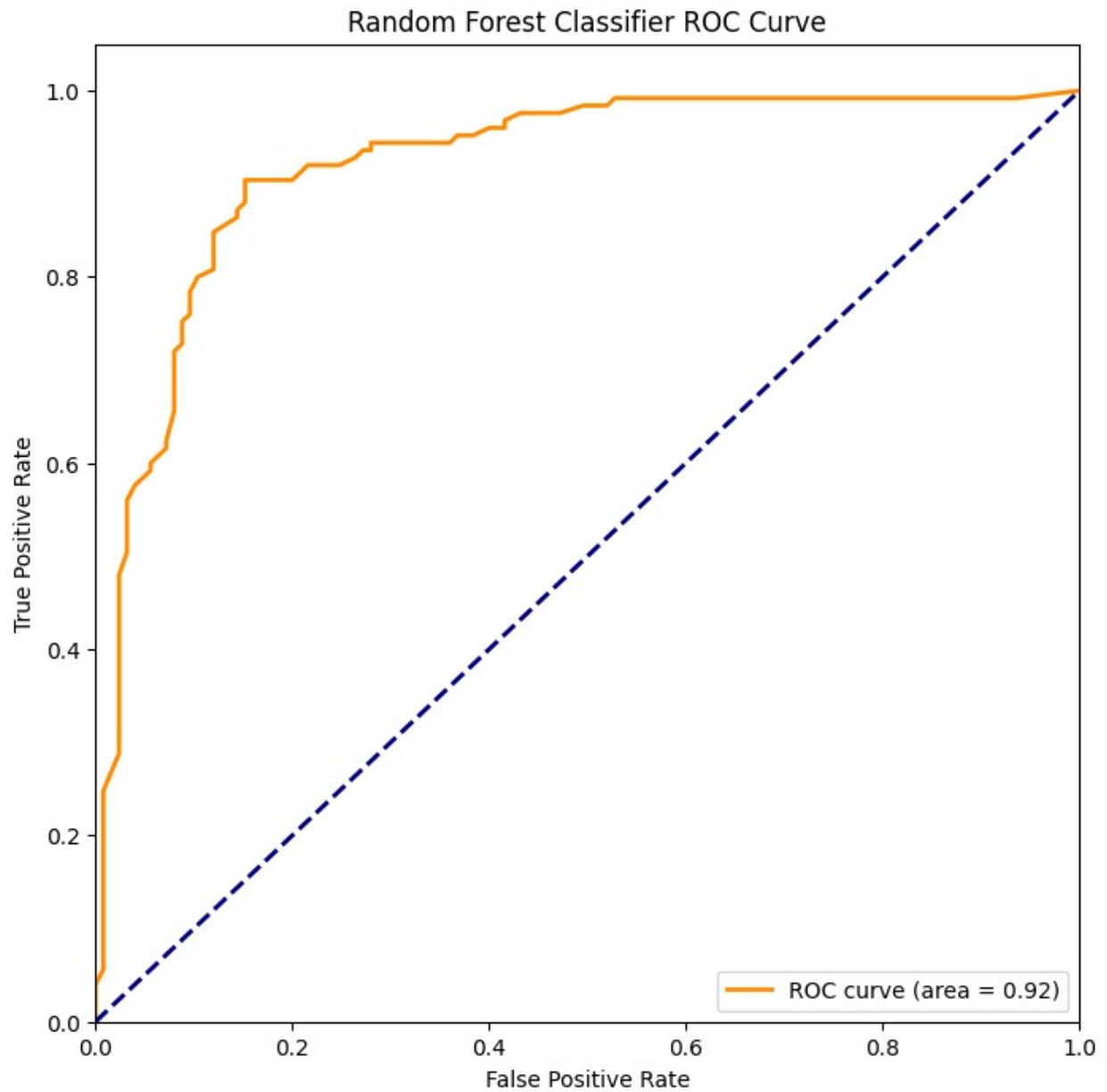
```



```
get_AUC_ROC_curve(RandomForestClassifier(), model_name='Random Forest Classifier')
```

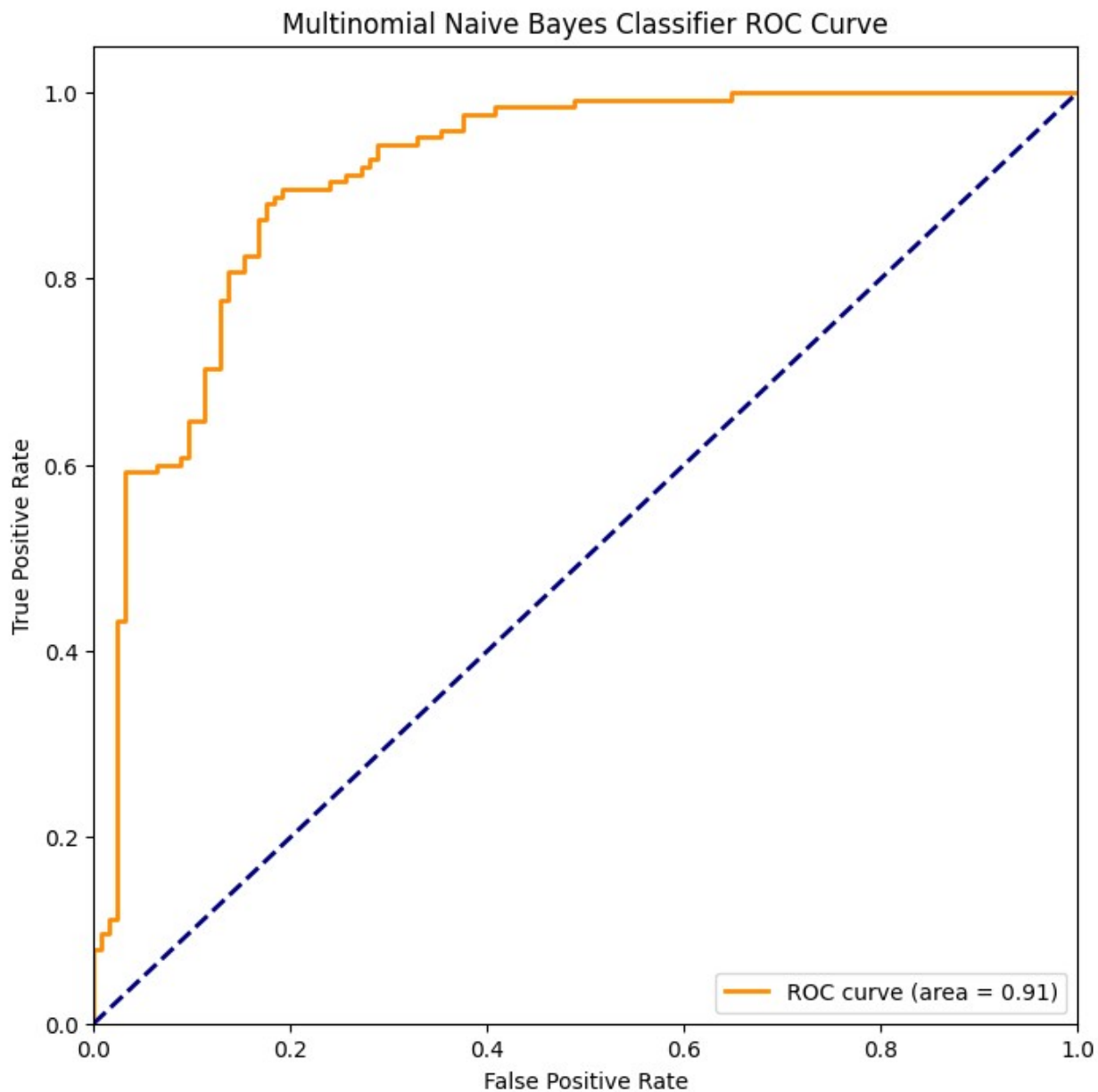
Please close the graphic to continue





```
get_AUC_ROC_curve(MultinomialNB(), model_name='Multinomial Naive Bayes Classifier')
```

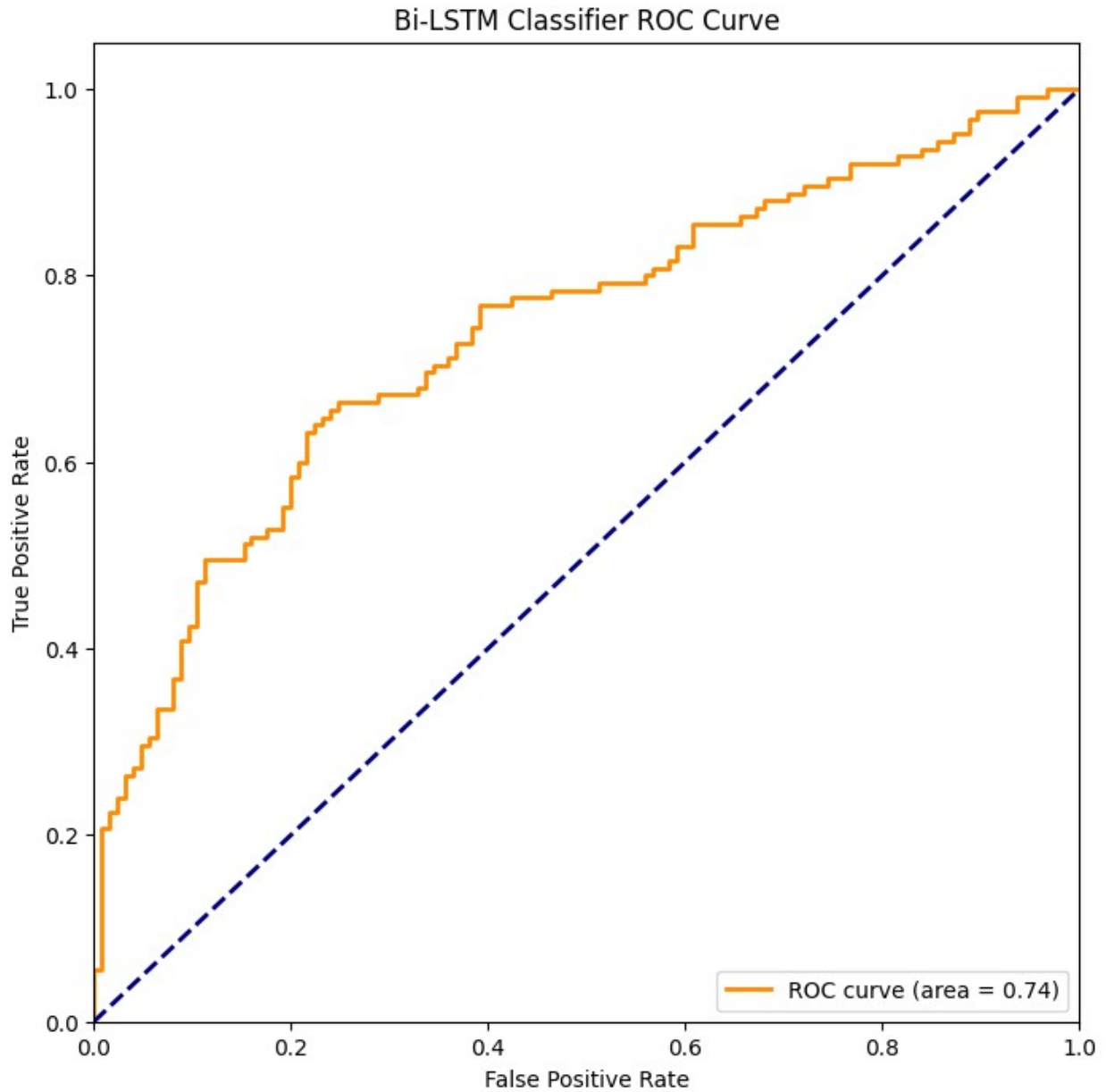
Please close the graphic to continue



```
biLSTM_model = BiLSTMModel(input_size=X_train_tensor.shape[2], # This
                             is the number of features (i.e., the TF-IDF size)
                             hidden_size=64, # Hidden units in LSTM
                             output_size=1 # Binary output (0 or 1)
                             )
```

```
get_AUC_ROC_curve(biLSTM_model, model_name='Bi-LSTM Classifier')
```

Please close the graphic to continue



## 1.7 Conclusion

For the purpose of the Conclusion, I think it is useful to visualize the average final metrics again. In particular I want to describe how this project helped me learn about the limitations and potential pitfalls of using neural networks, and why certain techniques are better than others regarding methods of vectorization.

```
print('Displaying Averaged Model Metrics: \n')  
print(final_average_metrics)
```

Displaying Averaged Model Metrics:

Naive_Bayes \	Logistic_Regression	Random_Forest_Classifier
TP	103.500000	87.600000
112.000000		
TN	98.000000	109.600000
84.100000		
FP	27.000000	15.400000
40.900000		
FN	21.500000	37.400000
13.000000		
TPR	0.827965	0.700305
0.895665		
TNR	0.783140	0.875911
0.672177		
FPR	0.216860	0.124089
0.327823		
FNR	0.172035	0.299695
0.104335		
Precision	0.792594	0.850530
0.731599		
F1_measure	0.809505	0.767279
0.805006		
Accuracy	0.806000	0.788800
0.784400		
Error_rate	0.194000	0.211200
0.215600		
BACC	0.805553	0.788108
0.783921		
TSS	0.611106	0.576215
0.567843		
HSS	0.609984	0.575584
0.566836		
BS	0.169869	0.152711
0.166401		
BSS	0.684007	0.614854
0.670041		
NPV	0.818504	0.744917
0.864463		
FDR	0.207406	0.149470
0.268401		
ROC_AUC_SCORE	0.876840	0.874742
0.880496		
	Bi-LSTM	
TP	62.600000	
TN	85.300000	
FP	39.700000	
FN	62.400000	
TPR	0.495483	

TNR	0.679970
FPR	0.320030
FNR	0.504517
Precision	0.540648
F1_measure	0.456686
Accuracy	0.591600
Error_rate	0.408400
BACC	0.587726
TSS	0.175453
HSS	0.173477
BS	0.248404
BSS	1.000154
NPV	0.657793
FDR	0.259352
ROC_AUC_SCORE	0.728746

My Classes are balanced, so accuracy is not necessarily a bad measure here. Of the three models: Random\_Forest\_Classifier, Naive\_Bayes, and Bi-LSTM, the non-neural network methods are much closer to ideal than the Bi-LSTM model tends to be.

## Summary of Model Performance Metrics

Metric	Logistic Regression	Random Forest	Naive Bayes	Bi-LSTM
<b>True Positives (TP)</b>	103.5	87.6	112.0	62.6
<b>True Negatives (TN)</b>	98.0	109.6	84.1	85.3
<b>False Positives (FP)</b>	27.0	15.4	40.9	39.7
<b>False Negatives (FN)</b>	21.5	37.4	13.0	62.4
<b>True Positive Rate (TPR)</b>	0.828	0.700	0.896	0.495
<b>True Negative Rate (TNR)</b>	0.783	0.876	0.672	0.680
<b>False Positive Rate (FPR)</b>	0.217	0.124	0.328	0.320
<b>False Negative Rate (FNR)</b>	0.172	0.300	0.104	0.505
<b>Precision</b>	0.793	0.851	0.732	0.541
<b>F1 Measure</b>	0.810	0.767	0.805	0.457
<b>Accuracy</b>	0.806	0.789	0.784	0.592
<b>Error Rate</b>	0.194	0.211	0.216	0.408
<b>Balanced Accuracy (BACC)</b>	0.806	0.788	0.784	0.588
<b>True Skill Statistic (TSS)</b>	0.611	0.576	0.568	0.175
<b>Heidke Skill Score (HSS)</b>	0.610	0.576	0.567	0.173

Metric	Logistic Regression	Random Forest	Naive Bayes	Bi-LSTM
<b>Brier Score (BS)</b>	0.170	0.153	0.166	0.248
<b>Brier Skill Score (BSS)</b>	0.684	0.615	0.670	1.000
<b>Negative Predictive Value (NPV)</b>	0.819	0.745	0.864	0.658
<b>False Discovery Rate (FDR)</b>	0.207	0.149	0.268	0.259
<b>ROC AUC Score</b>	0.877	0.875	0.880	0.729

I think that false positives and false negatives are equally important in this data, because as a subset with equally distributed classes, we don't want to over-predict either, as it could result in missing a lot of insincere comments or potentially the deletion/removal of a lot of sincere questions. Neither is ideal, at least within the confines of the subset.

When relating it to the original dataset, however, there is a massive class imbalance. I believe it is something like over 90% of the original data consists of sincere questions. This presents quite a problem, because overpredicting False Positives would lead to many more unjustifiable question removals, and might end up hurting site engagement more than helping it (which is presumably the aim of trying to detect/remove insincere questions).

That means that in my opinion, a good classifier has to be balanced, if not slightly properly overpredicting True Negatives at the cost of underpredicting False Negatives.

Basically, a good classifier for this problem should prioritize allowing as many sincere questions through as it can whilst progressively seeking to discover insincere questions. This should be done without an marked increase in False Positives so that there is no sweeping mass auto-ban of people's valid, sincere questions.

With that in mind, I will look at a few key metrics:

Maximize TP: Naive Bayes

Minimize FP: Random Forest

Maximize TN: Random Forest

Minimize FN: Naive Bayes

TPR: Random Forest

TNR: Random Forest

Precision: Random Forest

Accuracy / BACC: Random Forest, Naive Bayes Tie (basically)

I think that in this case, because we want to minimize FP while gaining as much TP as we can, that means that we need to choose the classifier that performs the best at minimizing FP without costing TN, which means that in this case I would choose Random Forest.

Random Forest and Naive Bayes may basically tie on accuracy metrics, however they differ with respect to what they are able to detect. Random Forest is better at minimizing FP, which means

that it more conservative with how it passes judgement on whether or not a question is insincere. Naive Bayes seems to be decent at maximizing the amount of positive classifications, which means that it is more aggressive than Random Forest. This might be okay if the original data was not so imbalanced, but given the imbalance this will be the worse choice, because in the context of the real world problem, this approach will result in many more detrimental total FP than it will advantageous total TP.

As a note; the reason that I did not even include the LSTM in this model is simple. My approach to this project utilizes tf-idf vectorization. This technique utilizes a generated vocabulary of sparse representations of word appearances to transform input sentences for model training.

This sparse representation works with models such as Naive Bayes and Random Forest, however it does not work with neural networks. This is because each of the vectors representing the sparse vocabulary term data has to be expanded into a dense vector before it can be utilized as input to a neural network. But this is the problem: when you expand a sparse vector/matrix, you make most of the values in the matrix/vector equal to zero. So I did this, and it basically results in a total loss of any discernable information for the input sentences. This obviously results in a total loss of ability for the neural network to learn any patterns, which is exactly what happens above. Upon closer examination of the neural network when computing the brier score / brier skill score, it becomes obvious that the model is not learning, because the probabilities that it outputs are basically the equivalent of a coin flip with added noise for every fold of training. I appreciate learning this here rather than in a professional setting.

## 1.8 END OF PROJECT

# TERMINAL OUTPUT:

```
[nltk_data] Downloading package stopwords to  
[nltk_data]   C:\Users\vitzj\AppData\Roaming\nltk_data...  
[nltk_data]   Package stopwords is already up-to-date!
```

A comparison of the train set length of character groups before cleaning:

	count	min	max	median	mean	std
target						
0	1250	5	50	11.0	12.6632	6.758514
1	1250	5	51	15.0	17.3864	9.341933

A comparison of the length of character groups after cleaning:

	count	min	max	median	mean	std
target						
0	1250	1	26	5.0	6.2712	3.393474
1	1250	2	27	8.0	9.0912	4.863189

Average number of words removed after cleaning: 7.3436



Displaying Model Metrics by fold for 10 folds:

Fold 1:

	Logistic_Regression	Random_Forest_Classifier	Naive_Bayes	Bi-LSTM
TP	104.000000	83.000000	112.000000	111.000000
TN	95.000000	109.000000	79.000000	47.000000
FP	29.000000	15.000000	45.000000	77.000000
FN	22.000000	43.000000	14.000000	15.000000
TPR	0.825397	0.658730	0.888889	0.880952
TNR	0.766129	0.879032	0.637097	0.379032
FPR	0.233871	0.120968	0.362903	0.620968
FNR	0.174603	0.341270	0.111111	0.119048
Precision	0.781955	0.846939	0.713376	0.590426
F1_measure	0.803089	0.741071	0.791519	0.707006
Accuracy	0.796000	0.768000	0.764000	0.632000
Error_rate	0.204000	0.232000	0.236000	0.368000
BACC	0.795763	0.768881	0.762993	0.629992
TSS	0.591526	0.537762	0.525986	0.259985
HSS	0.591791	0.536800	0.527031	0.261020
BS	0.171899	0.162068	0.171762	0.248684
BSS	0.687642	0.648313	0.687091	0.994799
NPV	0.811966	0.717105	0.849462	0.758065
FDR	0.218045	0.153061	0.286624	0.409574
ROC_AUC_SCORE	0.872056	0.854455	0.864183	0.693996

Fold 2:

	Logistic_Regression	Random_Forest_Classifier	Naive_Bayes	Bi-LSTM
TP	112.000000	92.000000	128.000000	29.000000
TN	87.000000	103.000000	80.000000	99.000000
FP	25.000000	9.000000	32.000000	13.000000
FN	26.000000	46.000000	10.000000	109.000000
TPR	0.811594	0.666667	0.927536	0.210145
TNR	0.776786	0.919643	0.714286	0.883929
FPR	0.223214	0.080357	0.285714	0.116071
FNR	0.188406	0.333333	0.072464	0.789855
Precision	0.817518	0.910891	0.800000	0.690476
F1_measure	0.814545	0.769874	0.859060	0.322222
Accuracy	0.796000	0.780000	0.832000	0.512000
Error_rate	0.204000	0.220000	0.168000	0.488000
BACC	0.794190	0.793155	0.820911	0.547037
TSS	0.588380	0.586310	0.641822	0.094073
HSS	0.587885	0.568614	0.653922	0.087045
BS	0.166021	0.169317	0.158891	0.249343
BSS	0.671347	0.684673	0.642512	1.008276
NPV	0.769912	0.691275	0.888889	0.475962
FDR	0.182482	0.089109	0.200000	0.309524
ROC_AUC_SCORE	0.884252	0.875291	0.888975	0.661297

Fold 3:

	Logistic_Regression	Random_Forest_Classifier	Naive_Bayes	Bi-LSTM
TP	96.000000	82.000000	100.000000	110.000000
TN	102.000000	116.000000	84.000000	39.000000
FP	31.000000	17.000000	49.000000	94.000000
FN	21.000000	35.000000	17.000000	7.000000
TPR	0.820513	0.700855	0.854701	0.940171
TNR	0.766917	0.872180	0.631579	0.293233
FPR	0.233083	0.127820	0.368421	0.706767
FNR	0.179487	0.299145	0.145299	0.059829
Precision	0.755906	0.828283	0.671141	0.539216
F1_measure	0.786885	0.759259	0.751880	0.685358
Accuracy	0.792000	0.792000	0.736000	0.596000
Error_rate	0.208000	0.208000	0.264000	0.404000
BACC	0.793715	0.786518	0.743140	0.616702
TSS	0.587430	0.573035	0.486280	0.233404
HSS	0.584426	0.578387	0.478409	0.223411
BS	0.171648	0.139592	0.173728	0.248753
BSS	0.689416	0.560664	0.697770	0.999105
NPV	0.829268	0.768212	0.831683	0.847826
FDR	0.244094	0.171717	0.328859	0.460784
ROC_AUC_SCORE	0.873401	0.889146	0.869481	0.721547

Fold 4:					
	Logistic_Regression	Random_Forest_Classifier	Naive_Bayes	Bi-LSTM	
TP	103.000000	89.000000	111.000000	102.000000	
TN	99.000000	105.000000	88.000000	76.000000	
FP	28.000000	22.000000	39.000000	51.000000	
FN	20.000000	34.000000	12.000000	21.000000	
TPR	0.837398	0.723577	0.902439	0.829268	
TNR	0.779528	0.826772	0.692913	0.598425	
FPR	0.220472	0.173228	0.307087	0.401575	
FNR	0.162602	0.276423	0.097561	0.170732	
Precision	0.786260	0.801802	0.740000	0.666667	
F1_measure	0.811024	0.760684	0.813187	0.739130	
Accuracy	0.808000	0.776000	0.796000	0.712000	
Error_rate	0.192000	0.224000	0.204000	0.288000	
BACC	0.808463	0.775174	0.797676	0.713847	
TSS	0.616926	0.550349	0.595352	0.427693	
HSS	0.616295	0.551196	0.593301	0.426057	
BS	0.178376	0.160756	0.169174	0.248282	
BSS	0.713686	0.643187	0.676871	0.993381	
NPV	0.831933	0.755396	0.880000	0.783505	
FDR	0.213740	0.198198	0.260000	0.333333	
ROC_AUC_SCORE	0.846745	0.847193	0.866590	0.770117	
Fold 5:					
	Logistic_Regression	Random_Forest_Classifier	Naive_Bayes	Bi-LSTM	
TP	103.000000	99.000000	109.000000	120.000000	
TN	101.000000	111.000000	84.000000	18.000000	
FP	26.000000	16.000000	43.000000	109.000000	
FN	20.000000	24.000000	14.000000	3.000000	
TPR	0.837398	0.804878	0.886179	0.975610	
TNR	0.795276	0.874016	0.661417	0.141732	
FPR	0.204724	0.125984	0.338583	0.858268	
FNR	0.162602	0.195122	0.113821	0.024390	
Precision	0.798450	0.860870	0.717105	0.524017	
F1_measure	0.817460	0.831933	0.792727	0.681818	
Accuracy	0.816000	0.840000	0.772000	0.552000	
Error_rate	0.184000	0.160000	0.228000	0.448000	
BACC	0.816337	0.839447	0.773798	0.558671	
TSS	0.632674	0.678894	0.547596	0.117342	
HSS	0.632188	0.679590	0.545571	0.115771	
BS	0.166142	0.125703	0.168322	0.249023	
BSS	0.664739	0.502942	0.673461	0.996348	
NPV	0.834711	0.822222	0.857143	0.857143	
FDR	0.201550	0.139130	0.282895	0.475983	
ROC_AUC_SCORE	0.884386	0.909993	0.878113	0.687344	
Fold 6:					
	Logistic_Regression	Random_Forest_Classifier	Naive_Bayes	Bi-LSTM	
TP	106.000000	92.000000	111.000000	2.000000	
TN	101.000000	106.000000	87.000000	123.000000	
FP	23.000000	18.000000	37.000000	1.000000	
FN	20.000000	34.000000	15.000000	124.000000	
TPR	0.841270	0.730159	0.880952	0.015873	
TNR	0.814516	0.854839	0.701613	0.991935	
FPR	0.185484	0.145161	0.298387	0.008065	
FNR	0.158730	0.269841	0.119048	0.984127	
Precision	0.821705	0.836364	0.750000	0.666667	
F1_measure	0.831373	0.779661	0.810219	0.031008	
Accuracy	0.828000	0.792000	0.792000	0.500000	
Error_rate	0.172000	0.208000	0.208000	0.500000	
BACC	0.827893	0.792499	0.791283	0.503904	
TSS	0.655786	0.584997	0.582565	0.007808	
HSS	0.655912	0.584399	0.583387	0.007748	
BS	0.168722	0.160055	0.162347	0.248908	
BSS	0.674930	0.640260	0.649429	0.995695	
NPV	0.834711	0.757143	0.852941	0.497976	
FDR	0.178295	0.163636	0.250000	0.333333	
ROC_AUC_SCORE	0.877752	0.858871	0.891897	0.713198	

## Fold 7:

	Logistic_Regression	Random_Forest_Classifier	Naive_Bayes	Bi-LSTM
TP	105.000000	91.000000	115.000000	112.000000
TN	102.000000	112.000000	90.000000	46.000000
FP	23.000000	13.000000	35.000000	79.000000
FN	20.000000	34.000000	10.000000	13.000000
TPR	0.840000	0.728000	0.920000	0.896000
TNR	0.816000	0.896000	0.720000	0.368000
FPR	0.184000	0.104000	0.280000	0.632000
FNR	0.160000	0.272000	0.080000	0.104000
Precision	0.820312	0.875000	0.766667	0.586387
F1_measure	0.830040	0.794760	0.836364	0.708861
Accuracy	0.828000	0.812000	0.820000	0.632000
Error_rate	0.172000	0.188000	0.180000	0.368000
BACC	0.828000	0.812000	0.820000	0.632000
TSS	0.656000	0.624000	0.640000	0.264000
HSS	0.656000	0.624000	0.640000	0.264000
BS	0.163457	0.144383	0.161681	0.248458
BSS	0.653827	0.577531	0.646723	0.993834
NPV	0.836066	0.767123	0.900000	0.779661
FDR	0.179688	0.125000	0.233333	0.413613
ROC_AUC_SCORE	0.888064	0.883648	0.884992	0.728192

## Fold 8:

	Logistic_Regression	Random_Forest_Classifier	Naive_Bayes	Bi-LSTM
TP	100.000000	85.000000	113.000000	14.000000
TN	104.000000	114.000000	87.000000	124.000000
FP	21.000000	11.000000	38.000000	1.000000
FN	25.000000	40.000000	12.000000	111.000000
TPR	0.800000	0.680000	0.904000	0.112000
TNR	0.832000	0.912000	0.696000	0.992000
FPR	0.168000	0.088000	0.304000	0.008000
FNR	0.200000	0.320000	0.096000	0.888000
Precision	0.826446	0.885417	0.748344	0.933333
F1_measure	0.813008	0.769231	0.818841	0.200000
Accuracy	0.816000	0.796000	0.800000	0.552000
Error_rate	0.184000	0.204000	0.200000	0.448000
BACC	0.816000	0.796000	0.800000	0.552000
TSS	0.632000	0.592000	0.600000	0.104000
HSS	0.632000	0.592000	0.600000	0.104000
BS	0.165513	0.142711	0.159401	0.248558
BSS	0.662053	0.570845	0.637604	0.994230
NPV	0.806202	0.740260	0.878788	0.527660
FDR	0.173554	0.114583	0.251656	0.066667
ROC_AUC_SCORE	0.894144	0.905536	0.899840	0.714432

## Fold 9:

	Logistic_Regression	Random_Forest_Classifier	Naive_Bayes	Bi-LSTM
TP	86.000000	70.000000	94.000000	104.000000
TN	113.000000	133.000000	99.000000	0.000000
FP	33.000000	13.000000	47.000000	146.000000
FN	18.000000	34.000000	10.000000	0.000000
TPR	0.826923	0.673077	0.903846	1.000000
TNR	0.773973	0.910959	0.678082	0.000000
FPR	0.226027	0.089041	0.321918	1.000000
FNR	0.173077	0.326923	0.096154	0.000000
Precision	0.722689	0.843373	0.666667	0.416000
F1_measure	0.771300	0.748663	0.767347	0.587571
Accuracy	0.796000	0.812000	0.772000	0.416000
Error_rate	0.204000	0.188000	0.228000	0.584000
BACC	0.800448	0.792018	0.790964	0.500000
TSS	0.600896	0.584036	0.581928	0.000000
HSS	0.588683	0.601506	0.553599	0.000000
BS	0.172035	0.134333	0.172657	0.250559
BSS	0.708127	0.552937	0.710687	1.031343
NPV	0.862595	0.796407	0.908257	0.000000
FDR	0.277311	0.156627	0.333333	0.584000
ROC_AUC_SCORE	0.888567	0.893967	0.891596	0.784444

```

Fold 10:
      Logistic_Regression Random_Forest_Classifier Naive_Bayes Bi-LSTM
TP      120.000000      101.000000      127.000000      133.000000
TN      76.000000      89.000000      63.000000      44.000000
FP      31.000000      18.000000      44.000000      63.000000
FN      23.000000      42.000000      16.000000      10.000000
TPR      0.839161      0.706294      0.888112      0.930070
TNR      0.710280      0.831776      0.588785      0.411215
FPR      0.289720      0.168224      0.411215      0.588785
FNR      0.160839      0.293706      0.111888      0.069930
Precision 0.794702      0.848739      0.742690      0.678571
F1_measure 0.816327      0.770992      0.808917      0.784661
Accuracy 0.784000      0.760000      0.760000      0.708000
Error_rate 0.216000      0.240000      0.240000      0.292000
BACC      0.774721      0.769035      0.738448      0.670642
TSS      0.549441      0.538069      0.476897      0.341285
HSS      0.554661      0.523295      0.493141      0.363979
BS      0.174873      0.167358      0.166050      0.248210
BSS      0.714305      0.683606      0.678265      1.013863
NPV      0.767677      0.679389      0.797468      0.814815
FDR      0.205298      0.151261      0.257310      0.321429
ROC_AUC_SCORE 0.859029      0.851055      0.869290      0.739265

Displaying Model Metrics by model for 10 folds:

Metrics for Logistic_Regression:
      Fold1      Fold2      Fold3      Fold4      Fold5      Fold6      Fold7      Fold8      Fold9      Fold10
TP      104.000000      112.000000      96.000000      103.000000      103.000000      106.000000      105.000000      100.000000      86.000000      120.000000
TN      95.000000      87.000000      102.000000      99.000000      101.000000      101.000000      102.000000      104.000000      113.000000      76.000000
FP      29.000000      25.000000      31.000000      28.000000      26.000000      23.000000      23.000000      21.000000      33.000000      31.000000
FN      22.000000      26.000000      21.000000      20.000000      20.000000      20.000000      20.000000      25.000000      18.000000      23.000000
TPR      0.825397      0.811594      0.820513      0.837398      0.837398      0.841270      0.840000      0.800000      0.826923      0.839161
TNR      0.766129      0.776786      0.766917      0.779528      0.795276      0.814516      0.816000      0.832000      0.773973      0.710280
FPR      0.233871      0.223214      0.233083      0.220472      0.204724      0.185484      0.184000      0.168000      0.226027      0.289720
FNR      0.174603      0.188406      0.179487      0.162602      0.162602      0.158730      0.160000      0.200000      0.173077      0.160839
Precision 0.781955      0.817518      0.755906      0.786260      0.798450      0.821705      0.820312      0.826446      0.722689      0.794702
F1_measure 0.803089      0.814545      0.786885      0.811024      0.817460      0.831373      0.830040      0.813008      0.771300      0.816327
Accuracy 0.796000      0.796000      0.792000      0.808000      0.816000      0.828000      0.828000      0.816000      0.796000      0.784000
Error_rate 0.204000      0.204000      0.208000      0.192000      0.184000      0.172000      0.172000      0.184000      0.204000      0.216000
BACC      0.795763      0.794190      0.793715      0.808463      0.816337      0.827893      0.828000      0.816000      0.800448      0.774721
TSS      0.591526      0.588380      0.587430      0.616926      0.632674      0.655786      0.656000      0.632000      0.600896      0.549441
HSS      0.591791      0.587885      0.584426      0.616295      0.632188      0.655912      0.656000      0.632000      0.588683      0.554661
BS      0.171899      0.166021      0.171648      0.178376      0.166142      0.168722      0.163457      0.165513      0.172035      0.174873
BSS      0.687642      0.671347      0.689416      0.713686      0.664739      0.674930      0.653827      0.662053      0.708127      0.714305
NPV      0.811966      0.769912      0.829268      0.831933      0.834711      0.834711      0.836066      0.806202      0.862595      0.767677
FDR      0.218045      0.182482      0.244094      0.213740      0.201550      0.178295      0.179688      0.173554      0.277311      0.205298
ROC_AUC_SCORE 0.872056      0.884252      0.873401      0.846745      0.884386      0.877752      0.888064      0.894144      0.888567      0.859029

Metrics for Random_Forest_Classifier:
      Fold1      Fold2      Fold3      Fold4      Fold5      Fold6      Fold7      Fold8      Fold9      Fold10
TP      83.000000      92.000000      82.000000      89.000000      99.000000      92.000000      91.000000      85.000000      70.000000      101.000000
TN      109.000000      103.000000      116.000000      105.000000      111.000000      106.000000      112.000000      114.000000      133.000000      89.000000
FP      15.000000      9.000000      17.000000      22.000000      16.000000      18.000000      13.000000      11.000000      13.000000      18.000000
FN      43.000000      46.000000      35.000000      34.000000      24.000000      34.000000      34.000000      40.000000      34.000000      42.000000
TPR      0.658730      0.666667      0.700855      0.723577      0.804878      0.730159      0.728000      0.680000      0.673077      0.706294
TNR      0.879032      0.919643      0.872180      0.826772      0.874016      0.854839      0.896000      0.912000      0.910959      0.831776
FPR      0.120968      0.080357      0.127820      0.173228      0.125984      0.145161      0.104000      0.088000      0.089041      0.168224
FNR      0.341270      0.333333      0.299145      0.276423      0.195122      0.269841      0.272000      0.320000      0.326923      0.293706
Precision 0.846939      0.910891      0.828283      0.801802      0.860870      0.836364      0.875000      0.885417      0.843373      0.848739
F1_measure 0.741071      0.769874      0.759259      0.760684      0.831933      0.779661      0.794760      0.769231      0.748663      0.770992
Accuracy 0.768000      0.780000      0.792000      0.776000      0.840000      0.792000      0.812000      0.796000      0.812000      0.760000
Error_rate 0.232000      0.220000      0.208000      0.224000      0.160000      0.208000      0.188000      0.204000      0.188000      0.240000
BACC      0.768881      0.793155      0.786518      0.775174      0.839447      0.792499      0.812000      0.796000      0.792018      0.769035
TSS      0.537762      0.586310      0.573035      0.550349      0.678894      0.584997      0.624000      0.592000      0.584036      0.538069
HSS      0.536800      0.568614      0.578387      0.551196      0.679590      0.584399      0.624000      0.592000      0.601506      0.523295
BS      0.162068      0.169317      0.139592      0.160756      0.125703      0.160055      0.144383      0.142711      0.134333      0.167358
BSS      0.648313      0.684673      0.560664      0.643187      0.502942      0.640260      0.577531      0.570845      0.552937      0.683606
NPV      0.717105      0.691275      0.768212      0.755396      0.822222      0.757143      0.767123      0.740260      0.796407      0.679389
FDR      0.153061      0.089109      0.171717      0.198198      0.139130      0.163636      0.125000      0.114583      0.156627      0.151261
ROC_AUC_SCORE 0.854455      0.875291      0.889146      0.847193      0.909993      0.858871      0.883648      0.905536      0.893967      0.851055

```



Metrics for Naive_Bayes:										
	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6	Fold7	Fold8	Fold9	Fold10
TP	112.000000	128.000000	100.000000	111.000000	109.000000	111.000000	115.000000	113.000000	94.000000	127.000000
TN	79.000000	80.000000	84.000000	88.000000	84.000000	87.000000	90.000000	87.000000	99.000000	63.000000
FP	45.000000	32.000000	49.000000	39.000000	43.000000	37.000000	35.000000	38.000000	47.000000	44.000000
FN	14.000000	10.000000	17.000000	12.000000	14.000000	15.000000	10.000000	12.000000	10.000000	16.000000
TPR	0.888889	0.927536	0.854701	0.902439	0.886179	0.880952	0.920000	0.904000	0.903846	0.888112
TNR	0.637097	0.714286	0.631579	0.692913	0.661417	0.701613	0.720000	0.696000	0.678082	0.588785
FPR	0.362903	0.285714	0.368421	0.307087	0.338583	0.298387	0.280000	0.304000	0.321918	0.411215
FNR	0.111111	0.072464	0.145299	0.097561	0.113821	0.119048	0.080000	0.096000	0.096154	0.111888
Precision	0.713376	0.800000	0.671141	0.740000	0.717105	0.750000	0.766667	0.748344	0.666667	0.742690
F1_measure	0.791519	0.859060	0.751880	0.813187	0.792727	0.810219	0.836364	0.818841	0.767347	0.808917
Accuracy	0.764000	0.832000	0.736000	0.796000	0.772000	0.792000	0.820000	0.800000	0.772000	0.760000
Error_rate	0.236000	0.168000	0.264000	0.204000	0.228000	0.208000	0.180000	0.200000	0.228000	0.240000
BACC	0.762993	0.820911	0.743140	0.797676	0.773798	0.791283	0.820000	0.800000	0.790964	0.738448
TSS	0.525986	0.641822	0.486280	0.595352	0.547596	0.582565	0.640000	0.600000	0.581928	0.476897
HSS	0.527031	0.653922	0.478409	0.593301	0.545571	0.583387	0.640000	0.600000	0.553599	0.493141
BS	0.171762	0.158891	0.173728	0.169174	0.168322	0.162347	0.161681	0.159401	0.172657	0.166050
BSS	0.687091	0.642512	0.697770	0.676871	0.673461	0.649429	0.646723	0.637604	0.710687	0.678265
NPV	0.849462	0.888889	0.831683	0.880000	0.857143	0.852941	0.900000	0.878788	0.908257	0.797468
FDR	0.286624	0.200000	0.328859	0.260000	0.282895	0.250000	0.233333	0.251656	0.333333	0.257310
ROC_AUC_SCORE	0.864183	0.888975	0.869481	0.866590	0.878113	0.891897	0.884992	0.899840	0.891596	0.869290

Metrics for Bi-LSTM:										
	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6	Fold7	Fold8	Fold9	Fold10
TP	111.000000	29.000000	110.000000	102.000000	120.000000	2.000000	112.000000	14.000000	104.000000	133.000000
TN	47.000000	99.000000	39.000000	76.000000	18.000000	123.000000	46.000000	124.000000	0.000000	44.000000
FP	77.000000	13.000000	94.000000	51.000000	109.000000	1.000000	79.000000	1.000000	146.000000	63.000000
FN	15.000000	109.000000	7.000000	21.000000	3.000000	124.000000	13.000000	111.000000	0.000000	10.000000
TPR	0.880952	0.210145	0.940171	0.829268	0.975610	0.015873	0.896000	0.112000	1.000000	0.930070
TNR	0.379032	0.883929	0.293233	0.598425	0.141732	0.991935	0.368000	0.992000	0.000000	0.411215
FPR	0.620968	0.116071	0.706767	0.401575	0.858268	0.008065	0.632000	0.008000	1.000000	0.588785
FNR	0.119048	0.789855	0.059829	0.170732	0.024390	0.984127	0.104000	0.888000	0.000000	0.069930
Precision	0.590426	0.690476	0.539216	0.666667	0.524017	0.666667	0.586387	0.933333	0.416000	0.678571
F1_measure	0.707006	0.322222	0.685358	0.739130	0.681818	0.031008	0.708861	0.200000	0.587571	0.784661
Accuracy	0.632000	0.512000	0.596000	0.712000	0.552000	0.500000	0.632000	0.552000	0.416000	0.708000
Error_rate	0.368000	0.488000	0.404000	0.288000	0.448000	0.500000	0.368000	0.448000	0.584000	0.292000
BACC	0.629992	0.547037	0.616702	0.713847	0.558671	0.503904	0.632000	0.552000	0.500000	0.670642
TSS	0.259985	0.094073	0.233404	0.427693	0.117342	0.007808	0.264000	0.104000	0.000000	0.341285
HSS	0.261020	0.087045	0.223411	0.426057	0.115771	0.007748	0.264000	0.104000	0.000000	0.363979
BS	0.248684	0.249343	0.248753	0.248282	0.249023	0.248908	0.248458	0.248558	0.250559	0.248210
BSS	0.994799	1.008276	0.999105	0.993381	0.996348	0.995695	0.993834	0.994230	1.031343	1.013863
NPV	0.758065	0.475962	0.847826	0.783505	0.857143	0.497976	0.779661	0.527660	0.000000	0.814815
FDR	0.409574	0.309524	0.460784	0.333333	0.475983	0.333333	0.413613	0.066667	0.584000	0.321429
ROC_AUC_SCORE	0.693996	0.661297	0.721547	0.770117	0.687344	0.713198	0.728192	0.714432	0.784444	0.739265

Displaying Averaged Model Metrics:

Average Metrics by Model				
	Logistic_Regression	Random_Forest_Classifier	Naive_Bayes	Bi-LSTM
TP	103.500000	88.400000	112.000000	83.700000
TN	98.000000	109.800000	84.100000	61.600000
FP	27.000000	15.200000	40.900000	63.400000
FN	21.500000	36.600000	13.000000	41.300000
TPR	0.827965	0.707224	0.895665	0.679009
TNR	0.783140	0.877722	0.672177	0.505950
FPR	0.216860	0.122278	0.327823	0.494050
FNR	0.172035	0.292776	0.104335	0.320991
Precision	0.792594	0.853768	0.731599	0.629176
F1_measure	0.809505	0.772613	0.805006	0.544764
Accuracy	0.806000	0.792800	0.784400	0.581200
Error_rate	0.194000	0.207200	0.215600	0.418800
BACC	0.805553	0.792473	0.783921	0.592480
TSS	0.611106	0.584945	0.567843	0.184959
HSS	0.609984	0.583979	0.566836	0.185303
BS	0.169869	0.150627	0.166401	0.248878
BSS	0.684007	0.606496	0.670041	1.002087
NPV	0.818504	0.749453	0.864463	0.634261
FDR	0.207406	0.146232	0.268401	0.370824
ROC_AUC_SCORE	0.876840	0.876916	0.880496	0.721383

Please close the graphic to continue  
Please close the graphic to continue  
Please close the graphic to continue  
Please close the graphic to continue  
Displaying Averaged Model Metrics:

	Logistic_Regression	Random_Forest_Classifier	Naive_Bayes	Bi-LSTM
TP	103.500000	88.400000	112.000000	83.700000
TN	98.000000	109.800000	84.100000	61.600000
FP	27.000000	15.200000	40.900000	63.400000
FN	21.500000	36.600000	13.000000	41.300000
TPR	0.827965	0.707224	0.895665	0.679009
TNR	0.783140	0.877722	0.672177	0.505950
FPR	0.216860	0.122278	0.327823	0.494050
FNR	0.172035	0.292776	0.104335	0.320991
Precision	0.792594	0.853768	0.731599	0.629176
F1_measure	0.809505	0.772613	0.805006	0.544764
Accuracy	0.806000	0.792800	0.784400	0.581200
Error_rate	0.194000	0.207200	0.215600	0.418800
BACC	0.805553	0.792473	0.783921	0.592480
TSS	0.611106	0.584945	0.567843	0.184959
HSS	0.609984	0.583979	0.566836	0.185303
BS	0.169869	0.150627	0.166401	0.248878
BSS	0.684007	0.606496	0.670041	1.002087
NPV	0.818504	0.749453	0.864463	0.634261
FDR	0.207406	0.146232	0.268401	0.370824
ROC_AUC_SCORE	0.876840	0.876916	0.880496	0.721383

Figure 1

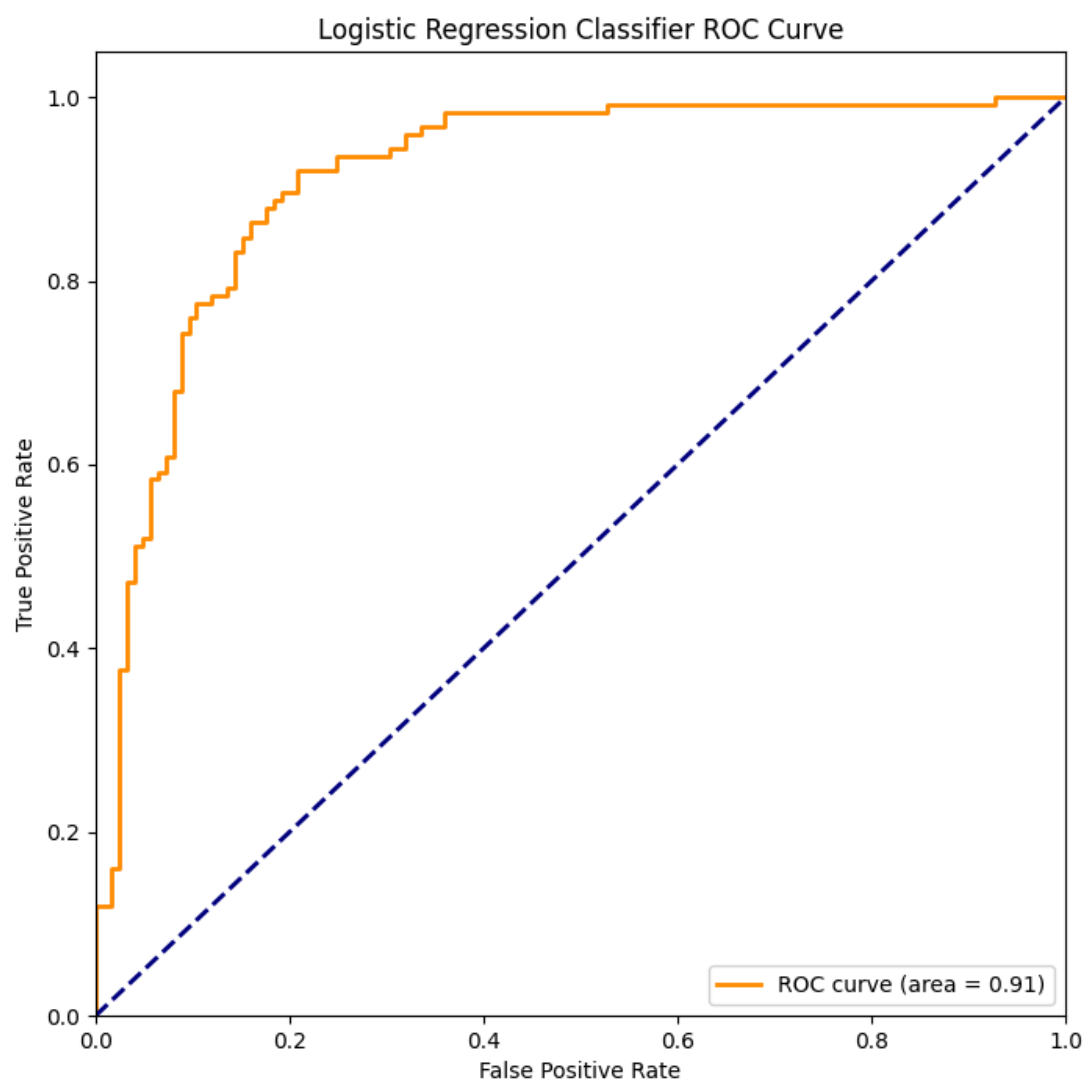
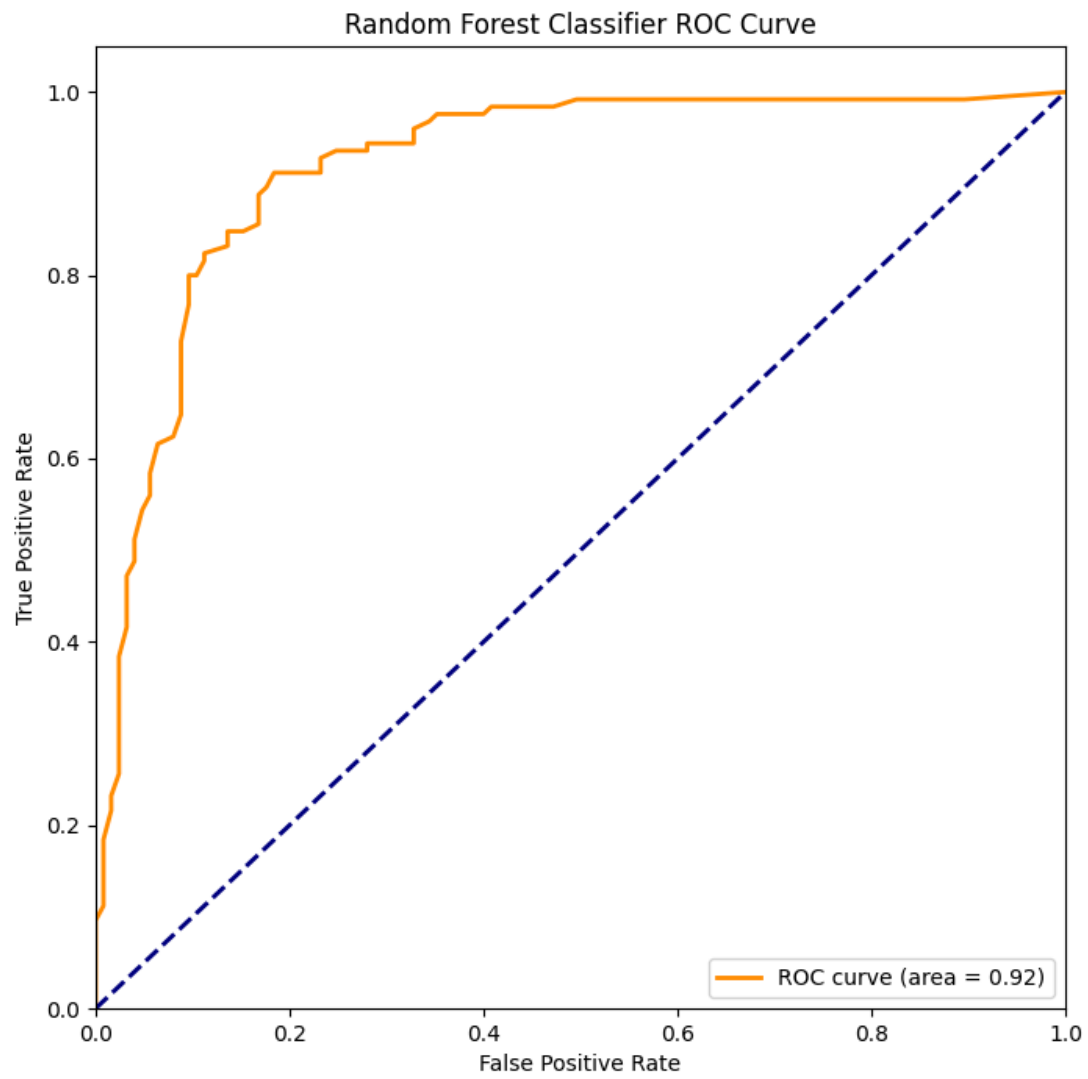


Figure 1



x=0.779 y=0.762



Figure 1

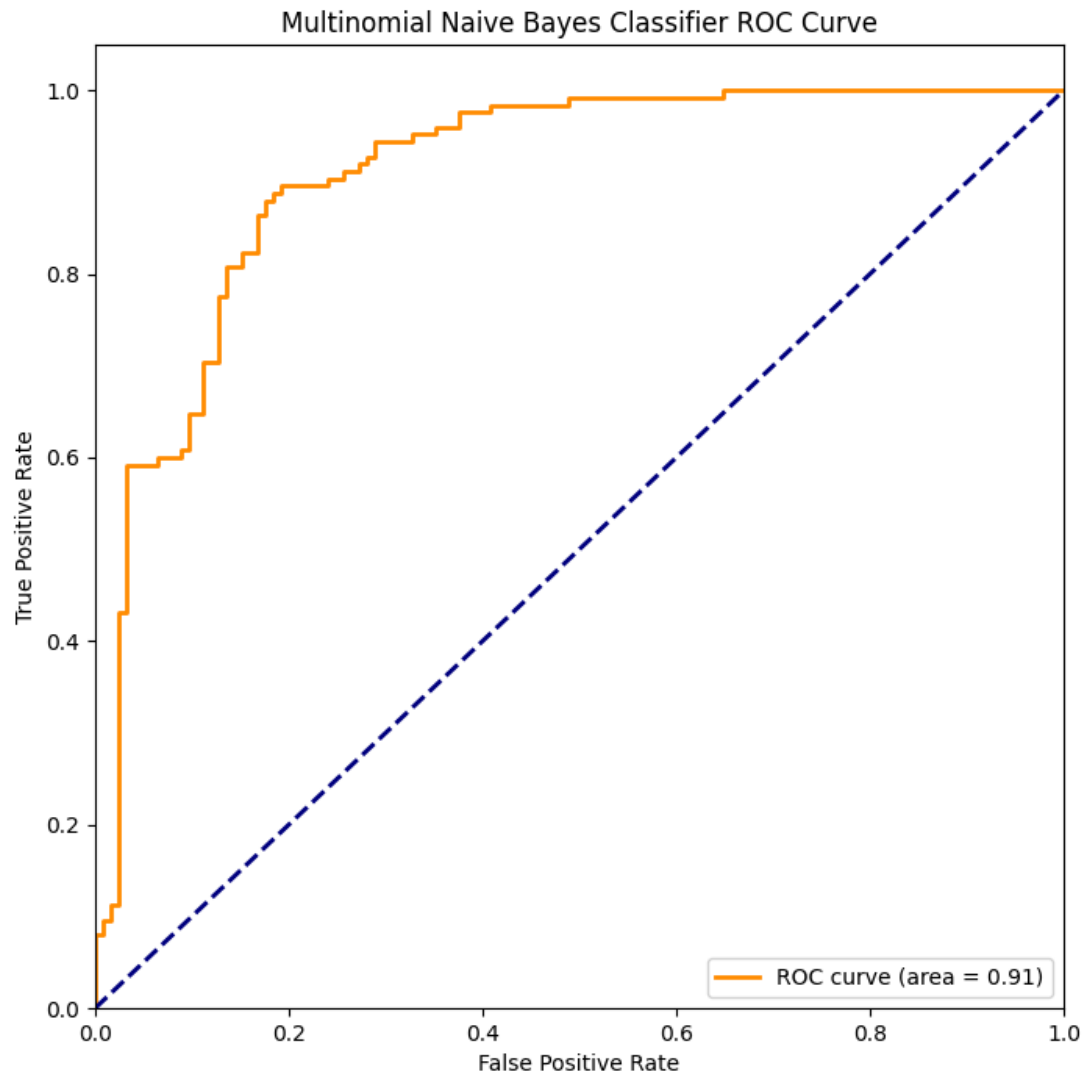


Figure 1

