maincode

December 28, 2023

[334]:
```
# Nouran Ahmed 20200609
# Mariam Tarek 20200523
# Nada Ashraf  20200587
# Fatma Salah  20200376
# Farah Tawfiq 20200378
```

[335]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
import statistics
from collections import Counter
from matplotlib.backends.backend_pdf import PdfPages
from nltk.corpus import stopwords
import nltk
from nltk.tokenize import word_tokenize
import spacy
import string
import re
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.neural_network import MLPClassifier
from keras.optimizers import Adam
import joblib
import pickle
from nltk.tokenize import word_tokenize
```

```
from google.colab import drive


import tensorflow.keras as keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


import warnings
warnings.filterwarnings("ignore")
```

[336]:
```
# Load the "drug.csv" dataset
data = pd.read_csv("./sentimentdataset (Project 1).csv")
# Print information about the old dataset
print(data.head())
```

```
  Source  ID                                            Message  Target
0   Yelp   0                                Crust is not good.       0
1   Yelp   1          Not tasty and the texture was just nasty.       0
2   Yelp   2  Stopped by during the late May bank holiday of…       1
3   Yelp   3  The selection on the menu was great and so wer…       1
4   Yelp   4     Now I am getting angry and I want my damn pho.       0
```

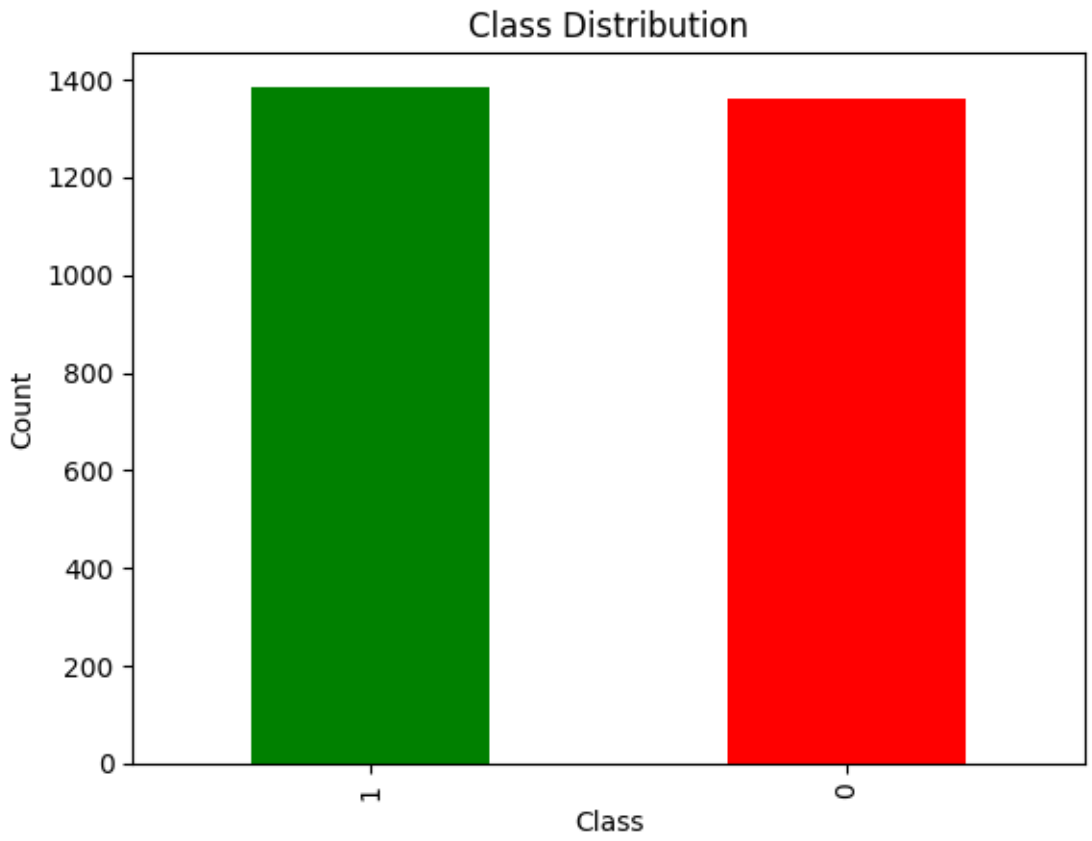
[337]:
```
# Check for missing values in the dataset
missing_values = data.isnull().sum()
# Display the count of missing values for each column
print("\nMissing Values:")
print(missing_values)
# Check if there are any missing values in the entire dataset
if missing_values.sum() == 0:
    print("\nNo missing values in the dataset.")
else:
    print("\nThere are missing values in the dataset.")
```

```
Missing Values:
Source     0
ID         0
Message    0
Target     0
dtype: int64

No missing values in the dataset.
```

[338]:
```
# Examining the distribution of samples in each class.

# info about the dataset
```

```
print("Info about the dataset: ")
print(data.info())
print("-----------------------------------------")
# count the num of samples in each class
class_distribution = data['Target'].value_counts()
print(class_distribution)
print("-----------------------------------------")
class_distribution.plot(kind='bar', color=['green', 'red'])
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```

```
Info about the dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2745 entries, 0 to 2744
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Source   2745 non-null   object
 1   ID       2745 non-null   int64
 2   Message  2745 non-null   object
 3   Target   2745 non-null   int64
dtypes: int64(2), object(2)
memory usage: 85.9+ KB
None
-----------------------------------------
1    1385
0    1360
Name: Target, dtype: int64
-----------------------------------------
```



[339]:
```
# separating targets and features
target_column = ['Target', 'ID', 'Source']

X = data.drop(target_column, axis=1)
y = pd.DataFrame(data['Target'], columns=['Target'])

print(X.head()) #features
print(y.head()) #target
```

```
                                             Message
0                                Crust is not good.
1         Not tasty and the texture was just nasty.
2  Stopped by during the late May bank holiday of…
3  The selection on the menu was great and so wer…
4    Now I am getting angry and I want my damn pho.
   Target
0       0
1       0
2       1
3       1
```

```
4       0
```

[340]:
```
nltk.download('stopwords')
stopWords = set(stopwords.words('english'))
negationWords = set(["hadn't", "wouldn't", "doesn't", "mightn't", "won't",␣
 ↪"shouldn't", 'haven', 'aren' , 'doesn', 'couldn', 'didn', 'isn', 'wouldn',␣
 ↪'mustn', "isn't", "shan't", "didn't", 'shan', 'hadn', 'wasn', 'weren',␣
 ↪"hasn't", 'mightn', "couldn't", "needn't", "haven't", "weren't", "aren't",␣
 ↪'needn', 'not', 'shouldn', 'hasn', "mustn't", "wasn't", "don't", 'don'])
stopWords = stopWords - negationWords
print(stopWords)
```

```
{'herself', 'here', 'about', 'yourself', 'very', 'which', 'yourselves', 'once',
'in', 'nor', 'each', 'but', 'more', 'ourselves', 'his', 'does', 'whom', 'with',
'theirs', 'further', 'any', 'into', 'themselves', 'the', 'that', 'and', 'me',
'has', 'ours', 'my', 'be', 'hers', 's', 'on', 'these', 'through', 've', 'how',
'between', 'them', 'before', 'when', 'against', 'been', 'while', 'your',
'below', 'will', 'm', 'if', 'both', 'down', "should've", 'i', 'y', 'being',
'so', 'll', "she's", 'other', 'am', 'itself', 'now', 'our', 'do', 'under',
'won', 'no', 'by', 'as', 'off', 'than', 'until', 'those', 'over', 'her',
"you've", "it's", 'an', 'doing', 'they', 'a', 'had', 'him', 're', 'few', 'are',
"that'll", 'why', 'was', 'she', 'then', 'just', 'own', 'of', 'who', 'same',
'for', 'can', 'some', 'we', 'from', 'there', 'should', 'where', 'because',
'again', 'ain', 'yours', 'above', "you'll", 'or', 'is', 'have', 'himself', 'd',
'most', 'this', 'during', "you'd", 'were', 'it', 'ma', 'out', 'to', "you're",
'did', 't', 'only', 'what', 'too', 'its', 'their', 'having', 'after', 'such',
'all', 'at', 'up', 'myself', 'you', 'o', 'he'}
```

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

[341]:
```
# remove punctuation
def removePunctuation(sentence):
    sentenceWithoutPunc = ""
    # iterates over each char and chick if it's punctuation char, not include␣
 ↪it, then joins the other char whitout the punctuation
    sentenceWithoutPunc = "".join(i for i in sentence if i not in string.
 ↪punctuation)
    return sentenceWithoutPunc

# apply the removePunctuation function on the dataframe
X['Message'] = X['Message'].apply(lambda sentence : removePunctuation(sentence))

# print(removePunctuation("Stopped by during don't the late May bank holiday␣
 ↪off Rick Steve recommendation and loved it......"))
```

[342]:
```
# lower casing the sentences
X['Message'] = X['Message'].apply(lambda char: char.lower())
print(X.head())
```

```
                                             Message
0                                  crust is not good
1           not tasty and the texture was just nasty
2  stopped by during the late may bank holiday of…
3  the selection on the menu was great and so wer…
4      now i am getting angry and i want my damn pho
```

[343]:
```
# Tokenization
def Tokenization(sentence):
    # split function is used to split the sentence into tokens
    tokens = re.split(r'\W+', sentence)
    return tokens

# apply the Tokenization function on the dataframe
X['Message'] = X['Message'].apply(lambda sentence: Tokenization(sentence))
print(X.head())
# print(Tokenization('Hello hell hhhh'))
```

```
                                             Message
0                            [crust, is, not, good]
1  [not, tasty, and, the, texture, was, just, nasty]
2  [stopped, by, during, the, late, may, bank, ho…
3  [the, selection, on, the, menu, was, great, an…
4  [now, i, am, getting, angry, and, i, want, my,…
```

[344]:
```
# Remove stop words

# spacy.cli.download("en_core_web_sm")
nlp = spacy.load('en_core_web_sm')
default_stop_words = set(nlp.Defaults.stop_words)
negationWords = set(["hadn't", "wouldn't", "doesn't", "mightn't", "won't",␣
 ↪"shouldn't", 'haven', 'aren' , 'doesn', 'couldn', 'didn', "didnt",'isn',␣
 ↪'wouldn', 'mustn', "isn't", "shan't", "didn't", 'shan', 'hadn', 'wasn',␣
 ↪'weren', "hasn't", 'mightn', "couldn't", "needn't", "haven't", "weren't",␣
 ↪"aren't", 'needn', 'not', 'shouldn', 'hasn', "mustn't", "wasn't", "don't",␣
 ↪'don'])

custom_stop_words = default_stop_words - negationWords
nlp.Defaults.stop_words = custom_stop_words
# print(nlp)
X_filter =X
X_filter = pd.DataFrame(X)
#X_filter = pd.DataFrame(data)
```

```
def stopWordsRemoval(sentenceTokenized):
    allInfo = nlp(' '.join(sentenceTokenized))
    filtered_tokens = []

    for token in allInfo:
      # check if the lowercase of the text is not in the stop words
      if token.text.lower() not in map(str.lower, custom_stop_words):
        filtered_tokens.append(token.text)

    return filtered_tokens


# apply the stopWordsRemoval function on the dataframe
X_filter['Message'] = X_filter['Message'].apply(lambda sentence:␣
 ↪stopWordsRemoval(sentence))
print(X_filter.head())
# print(stopWordsRemoval(["HI i not dokey do "]))
```

```
                                             Message
0                                  [crust, not, good]
1                        [not, tasty, texture, nasty]
2  [stopped, late, bank, holiday, rick, steve, re…
3                     [selection, menu, great, prices]
4                    [getting, angry, want, damn, pho]
```

[345]:
```
# Lemmatization using spaCy

X_filter = pd.DataFrame(X)
#print(X_filter)
def lemmatize(tokens):
    doc = nlp(' '.join(tokens))
    # Extract lemmatized tokens
    lemmatized_tokens = [token.lemma_ for token in doc]
    return lemmatized_tokens

# apply lemmatize function on the dataframe
X_filter['Message'] = X_filter['Message'].apply(lambda sentence:␣
 ↪lemmatize(stopWordsRemoval(sentence)))
print(X_filter.head())
# print(lemmatize([['Now I am getting angry and I want my damn pho.']))
```

```
                                             Message
0                                  [crust, not, good]
1                        [not, tasty, texture, nasty]
2  [stop, late, bank, holiday, rick, steve, recom…
3                     [selection, menu, great, price]
4                       [get, angry, want, damn, pho]
```

[346]:
```
# Generate sentence embeddings
X = X_filter['Message'].apply(lambda sentence: ' '.join(sentence))

count_vectorizer = CountVectorizer()

X_vectorized = count_vectorizer.fit_transform(X)

# Save the CountVectorizer model to a file, we'll use it in the test file
joblib.dump(count_vectorizer, 'count_vectorizer.pkl')

# Convert X_vectorized to a DataFrame
X_df = pd.DataFrame(X_vectorized.toarray(), columns=count_vectorizer.
 ↪get_feature_names_out())

print(X_df)
```

```
      010  10  100  1010  11  110  1199  12  13  15  …  yukon  yum  yummy  \
0       0   0    0     0   0    0     0   0   0   0  …      0    0      0
1       0   0    0     0   0    0     0   0   0   0  …      0    0      0
2       0   0    0     0   0    0     0   0   0   0  …      0    0      0
3       0   0    0     0   0    0     0   0   0   0  …      0    0      0
4       0   0    0     0   0    0     0   0   0   0  …      0    0      0
…     …  ..  …    …  ..  …    …  ..  ..  ..  …    …  …    …
2740    0   0    0     0   0    0     0   0   0   0  …      0    0      0
2741    0   0    0     0   0    0     0   0   0   0  …      0    0      0
2742    0   0    0     0   0    0     0   0   0   0  …      0    0      0
2743    0   0    0     0   0    0     0   0   0   0  …      0    0      0
2744    0   0    0     0   0    0     0   0   0   0  …      0    0      0

      yun  z500a  zero  zillion  zombie  zombiestudent  zombiez
0       0      0     0        0       0              0        0
1       0      0     0        0       0              0        0
2       0      0     0        0       0              0        0
3       0      0     0        0       0              0        0
4       0      0     0        0       0              0        0
…     …    …   …       …     …            …       …
2740    0      0     0        0       0              0        0
2741    0      0     0        0       0              0        0
2742    0      0     0        0       0              0        0
2743    0      0     0        0       0              0        0
2744    0      0     0        0       0              0        0

[2745 rows x 4328 columns]
```

[347]:
```
# Split into training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(X_df, y, test_size = 0.2,␣
 ↪random_state = 42)
```

```
# print(X_train)
# print("_________________________")
# print(X_test)
# print("_________________________")
# print(y_train)
# print("_________________________")
# print(y_test)
# print("_________________________")
```

[348]:
```
# Initial Experiment:
svm = LinearSVC()

# Set up parameter grid for Grid Search
param_grid = {'C': [0.01, 0.1, 10, 100]}

# GridSearchCV to find the best params
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)


print("Best Parameters:", grid_search.best_params_)

# predicting the targets of the x_test
y_pred_svc = grid_search.predict(X_test)

# Get the best model to use it to predict the answers in the test file
best_model = grid_search.best_estimator_

# Save the best model to a file, as we'll use it in the test file
joblib.dump(best_model, 'linear_svc_model.pkl')


# find the accuracy of the LinearSVC model
accuracy = accuracy_score(y_test, y_pred_svc)
print("Accuracy on Testing Set:", accuracy)

# Classification report of the LinearSVC model
print("_____________________________________________")
print("Classification Report of The LinearSVC Model:")
print(classification_report(y_test, y_pred_svc))
```

```
Best Parameters: {'C': 0.1}
Accuracy on Testing Set: 0.8469945355191257

---------------------------------------------
Classification Report of The LinearSVC Model:
              precision    recall  f1-score   support
```

```
              0       0.82      0.88      0.85       265
              1       0.88      0.82      0.85       284

       accuracy                           0.85       549
      macro avg       0.85      0.85      0.85       549
   weighted avg       0.85      0.85      0.85       549
```

[349]:
```
model = Sequential()

# 10 is the number of neorens in the hidden layer, input_dim is the size of the␣
 ↪input lauer(num of featueres)
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu'))

# the output layer
model.add(Dense(1, activation='sigmoid'))

# set the new_learning_rate of the adam optimizer
new_learning_rate=0.0001
optimizer=Adam(learning_rate=new_learning_rate)

# compile the model
model.compile(loss='binary_crossentropy', optimizer=optimizer,␣
 ↪metrics=['accuracy'])
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test,␣
 ↪y_test))
y_pred = model.predict(X_test)
y_pred_binary = np.round(y_pred)

# calculate the accuracy of the ANN model
accuracy = accuracy_score(y_test, y_pred_binary)
print("Accuracy:", accuracy)


# Classification report of the (ANN model
print("_____________________________________________")
print("Classification Report of The (ANN Model:")
report = classification_report(y_test, y_pred_binary)
print(report)
```

```
Epoch 1/50
69/69 [==============================] - 1s 8ms/step - loss: 0.6923 - accuracy:
0.5250 - val_loss: 0.6920 - val_accuracy: 0.5264
Epoch 2/50
69/69 [==============================] - 0s 6ms/step - loss: 0.6895 - accuracy:
0.5893 - val_loss: 0.6902 - val_accuracy: 0.5610
Epoch 3/50
```

```
69/69 [==============================] - 0s 5ms/step - loss: 0.6862 - accuracy:
0.6475 - val_loss: 0.6879 - val_accuracy: 0.5993
Epoch 4/50
69/69 [==============================] - 0s 5ms/step - loss: 0.6823 - accuracy:
0.6981 - val_loss: 0.6852 - val_accuracy: 0.6393
Epoch 5/50
69/69 [==============================] - 0s 6ms/step - loss: 0.6778 - accuracy:
0.7341 - val_loss: 0.6820 - val_accuracy: 0.6812
Epoch 6/50
69/69 [==============================] - 0s 5ms/step - loss: 0.6725 - accuracy:
0.7696 - val_loss: 0.6781 - val_accuracy: 0.7049
Epoch 7/50
69/69 [==============================] - 0s 5ms/step - loss: 0.6664 - accuracy:
0.7942 - val_loss: 0.6738 - val_accuracy: 0.7322
Epoch 8/50
69/69 [==============================] - 0s 6ms/step - loss: 0.6595 - accuracy:
0.8174 - val_loss: 0.6689 - val_accuracy: 0.7468
Epoch 9/50
69/69 [==============================] - 0s 5ms/step - loss: 0.6520 - accuracy:
0.8329 - val_loss: 0.6636 - val_accuracy: 0.7668
Epoch 10/50
69/69 [==============================] - 0s 5ms/step - loss: 0.6438 - accuracy:
0.8506 - val_loss: 0.6578 - val_accuracy: 0.7741
Epoch 11/50
69/69 [==============================] - 0s 5ms/step - loss: 0.6352 - accuracy:
0.8607 - val_loss: 0.6518 - val_accuracy: 0.7796
Epoch 12/50
69/69 [==============================] - 0s 5ms/step - loss: 0.6263 - accuracy:
0.8689 - val_loss: 0.6457 - val_accuracy: 0.7978
Epoch 13/50
69/69 [==============================] - 0s 5ms/step - loss: 0.6171 - accuracy:
0.8739 - val_loss: 0.6395 - val_accuracy: 0.8051
Epoch 14/50
69/69 [==============================] - 0s 5ms/step - loss: 0.6078 - accuracy:
0.8793 - val_loss: 0.6332 - val_accuracy: 0.8033
Epoch 15/50
69/69 [==============================] - 0s 5ms/step - loss: 0.5984 - accuracy:
0.8866 - val_loss: 0.6269 - val_accuracy: 0.8106
Epoch 16/50
69/69 [==============================] - 0s 5ms/step - loss: 0.5888 - accuracy:
0.8903 - val_loss: 0.6206 - val_accuracy: 0.8142
Epoch 17/50
69/69 [==============================] - 0s 5ms/step - loss: 0.5794 - accuracy:
0.8975 - val_loss: 0.6143 - val_accuracy: 0.8160
Epoch 18/50
69/69 [==============================] - 0s 5ms/step - loss: 0.5699 - accuracy:
0.9007 - val_loss: 0.6080 - val_accuracy: 0.8197
Epoch 19/50
```

```
69/69 [==============================] - 0s 5ms/step - loss: 0.5605 - accuracy:
0.9035 - val_loss: 0.6019 - val_accuracy: 0.8233
Epoch 20/50
69/69 [==============================] - 0s 5ms/step - loss: 0.5512 - accuracy:
0.9089 - val_loss: 0.5959 - val_accuracy: 0.8233
Epoch 21/50
69/69 [==============================] - 0s 5ms/step - loss: 0.5419 - accuracy:
0.9103 - val_loss: 0.5898 - val_accuracy: 0.8215
Epoch 22/50
69/69 [==============================] - 0s 6ms/step - loss: 0.5327 - accuracy:
0.9121 - val_loss: 0.5840 - val_accuracy: 0.8215
Epoch 23/50
69/69 [==============================] - 0s 5ms/step - loss: 0.5237 - accuracy:
0.9153 - val_loss: 0.5782 - val_accuracy: 0.8233
Epoch 24/50
69/69 [==============================] - 1s 8ms/step - loss: 0.5148 - accuracy:
0.9162 - val_loss: 0.5723 - val_accuracy: 0.8233
Epoch 25/50
69/69 [==============================] - 1s 9ms/step - loss: 0.5060 - accuracy:
0.9203 - val_loss: 0.5667 - val_accuracy: 0.8270
Epoch 26/50
69/69 [==============================] - 1s 8ms/step - loss: 0.4974 - accuracy:
0.9221 - val_loss: 0.5611 - val_accuracy: 0.8324
Epoch 27/50
69/69 [==============================] - 1s 9ms/step - loss: 0.4889 - accuracy:
0.9249 - val_loss: 0.5559 - val_accuracy: 0.8306
Epoch 28/50
69/69 [==============================] - 1s 8ms/step - loss: 0.4806 - accuracy:
0.9258 - val_loss: 0.5505 - val_accuracy: 0.8306
Epoch 29/50
69/69 [==============================] - 1s 8ms/step - loss: 0.4724 - accuracy:
0.9262 - val_loss: 0.5453 - val_accuracy: 0.8324
Epoch 30/50
69/69 [==============================] - 1s 9ms/step - loss: 0.4644 - accuracy:
0.9276 - val_loss: 0.5402 - val_accuracy: 0.8324
Epoch 31/50
69/69 [==============================] - 1s 9ms/step - loss: 0.4565 - accuracy:
0.9299 - val_loss: 0.5351 - val_accuracy: 0.8306
Epoch 32/50
69/69 [==============================] - 1s 8ms/step - loss: 0.4487 - accuracy:
0.9303 - val_loss: 0.5302 - val_accuracy: 0.8361
Epoch 33/50
69/69 [==============================] - 1s 10ms/step - loss: 0.4412 - accuracy:
0.9308 - val_loss: 0.5254 - val_accuracy: 0.8342
Epoch 34/50
69/69 [==============================] - 0s 5ms/step - loss: 0.4337 - accuracy:
0.9326 - val_loss: 0.5208 - val_accuracy: 0.8361
Epoch 35/50
```

```
69/69 [==============================] - 0s 6ms/step - loss: 0.4265 - accuracy:
0.9340 - val_loss: 0.5160 - val_accuracy: 0.8361
Epoch 36/50
69/69 [==============================] - 0s 5ms/step - loss: 0.4193 - accuracy:
0.9344 - val_loss: 0.5116 - val_accuracy: 0.8379
Epoch 37/50
69/69 [==============================] - 0s 5ms/step - loss: 0.4123 - accuracy:
0.9349 - val_loss: 0.5073 - val_accuracy: 0.8397
Epoch 38/50
69/69 [==============================] - 0s 5ms/step - loss: 0.4055 - accuracy:
0.9367 - val_loss: 0.5029 - val_accuracy: 0.8379
Epoch 39/50
69/69 [==============================] - 0s 5ms/step - loss: 0.3988 - accuracy:
0.9358 - val_loss: 0.4987 - val_accuracy: 0.8379
Epoch 40/50
69/69 [==============================] - 0s 5ms/step - loss: 0.3922 - accuracy:
0.9362 - val_loss: 0.4946 - val_accuracy: 0.8379
Epoch 41/50
69/69 [==============================] - 0s 5ms/step - loss: 0.3857 - accuracy:
0.9367 - val_loss: 0.4907 - val_accuracy: 0.8379
Epoch 42/50
69/69 [==============================] - 0s 5ms/step - loss: 0.3794 - accuracy:
0.9372 - val_loss: 0.4866 - val_accuracy: 0.8434
Epoch 43/50
69/69 [==============================] - 0s 5ms/step - loss: 0.3732 - accuracy:
0.9390 - val_loss: 0.4829 - val_accuracy: 0.8452
Epoch 44/50
69/69 [==============================] - 0s 5ms/step - loss: 0.3672 - accuracy:
0.9399 - val_loss: 0.4791 - val_accuracy: 0.8470
Epoch 45/50
69/69 [==============================] - 0s 5ms/step - loss: 0.3613 - accuracy:
0.9408 - val_loss: 0.4754 - val_accuracy: 0.8470
Epoch 46/50
69/69 [==============================] - 0s 5ms/step - loss: 0.3554 - accuracy:
0.9413 - val_loss: 0.4718 - val_accuracy: 0.8488
Epoch 47/50
69/69 [==============================] - 0s 5ms/step - loss: 0.3498 - accuracy:
0.9426 - val_loss: 0.4684 - val_accuracy: 0.8488
Epoch 48/50
69/69 [==============================] - 0s 5ms/step - loss: 0.3442 - accuracy:
0.9431 - val_loss: 0.4651 - val_accuracy: 0.8506
Epoch 49/50
69/69 [==============================] - 0s 5ms/step - loss: 0.3388 - accuracy:
0.9440 - val_loss: 0.4616 - val_accuracy: 0.8506
Epoch 50/50
69/69 [==============================] - 0s 5ms/step - loss: 0.3334 - accuracy:
0.9449 - val_loss: 0.4585 - val_accuracy: 0.8488
18/18 [==============================] - 0s 4ms/step
```

```
Accuracy: 0.848816029143898

-----------------------------------------------
Classification Report of The (ANN Model:
              precision    recall  f1-score   support

           0       0.80      0.91      0.85       265
           1       0.91      0.79      0.84       284

    accuracy                           0.85       549
   macro avg       0.85      0.85      0.85       549
weighted avg       0.86      0.85      0.85       549
```

[350]:
```
# Mount Google Drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```